$\omega(1/\lambda)$-Rate Boolean Garbling Scheme from Generic Groups

Geoffroy Couteau
Université Paris Cité, CNRS, IRIF

Carmit Hazay
Bar-Ilan University

Aditya Hegde
Johns Hopkins University

Naman Kumar
Oregon State University

Abstract

Garbling schemes are a fundamental cryptographic tool for enabling private computations and ensuring that nothing leaks beyond the output. As a widely studied primitive, significant efforts have been made to reduce their size. Until recently, all such schemes followed the Lindell and Pinkas paradigm for Boolean circuits (JoC 2009), where each gate is represented as a set of ciphertexts computed using only symmetric-key primitives. However, this approach is inherently limited to $O(\lambda)$ bits per gate, where λ is the security parameter. Recently, it has been shown that achieving smaller garbled circuit size is possible under stronger assumptions, such as variants of Learning with Errors (LWE) or Indistinguishability Obfuscation (i$\mathcal{O}$). In addition to requiring high-end cryptography, none of these constructions is black-box in the underlying cryptographic primitives, a key advantage of prior work. In this paper, we present the first approach to garbling Boolean circuits that makes a black-box use of a group and uses $o(\lambda)$ bits per gate.

Building on a novel application of the Reverse Multiplication-Friendly Embeddings (RMFE) paradigm (Cascudo et al., CRYPTO 2018), we introduce a new packing mechanism for garbling schemes, that packs boolean values into integers and leverage techniques for arithmetic garbling over integer rings. Our results introduce two new succinct schemes that achieve improved rates by a factor of $\sqrt{\log \lambda}$, retaining the black-box usage. (1) Our first scheme is proven in the Generic Group model (GGM) for circuits with $\Omega(\sqrt{\log \lambda})$ width, obtaining a garbled circuit size of $\lambda \cdot |C|/\sqrt{\log(\lambda)}$. (2) Our second scheme is proven in the plain model under the Power-DDH assumption, attaining a garbled circuit size of $\lambda \cdot (|C|/\sqrt{\log(\lambda)} + \mathsf{poly}(\lambda) \cdot \mathsf{depth}(C)$, but is restricted to layered circuits. *Our schemes are the first to achieve sublinear (in λ) cost per gate under assumptions that do not imply fully homomorphic encryption*; in addition, our scheme is also the first to achieve this while making a black-box use of cryptography.

1 Introduction

Garbling schemes [Yao86, LP09] are a cryptographic object that enables the oblivious evaluation of computations, ensuring that nothing beyond the output is revealed from the computation's flow. This strong privacy guarantee has established garbling schemes as a fundamental cryptographic tool with a wide range of applications. One of their most prominent applications is in secure two-party computation. In this setting, a garbler creates an encoded version of the function along with encoded inputs. It provides these to an evaluator, who privately evaluates the encoded computation and learns only the output. The correctness property of the garbling scheme guarantees that the right output is obtained. The ability to implement garbling schemes using only symmetric-key cryptography, combined with the garbling algorithm's depth complexity independent of the computed circuit's depth complexity, has made them a highly competitive technique for achieving constant round secure computation.

Since their introduction by Yao [Yao86] and formalization by Lindell and Pinkas [LP09], extensive research has focused on understanding the concrete efficiency of Boolean circuit garbling, e.g., [KS08, PSSW09, KMR14, ZRE15, RR21]. Building on the gate-by-gate approach introduced in [LP09], this research culminated in achieving 1.5 times λ bits per AND gate [RR21], with no communication cost for XOR gates [KS08], where λ is the security parameter. This resulted in an $O(\lambda)$ inflation between the circuit representation C and the length of the garbled circuit $\hat{C}$. An important feature of this line of work is the black-box access to the underlying symmetric-key primitive. This abstraction treats cryptographic operations as an oracle, allowing them to be instantiated with a pseudorandom function (PRF) or a hash function, resulting in highly practical schemes.

Using black-box access to the underlying cryptographic primitive allows for a modular design approach, ensuring that the constructions remain independent of specific implementations and rely only on the input-output behavior of the primitive. This flexibility allows for improved instantiations under different hardness assumptions. As a result, the construction of black-box schemes is both theoretically appealing and practically beneficial and has therefore been extensively studied with the goal of understanding its power and limitations; see [DI05, PW09, HIMV19, IKSS22] for a few examples. This feature is also at the focus of our work.

Attempts to explore the limitations of these constructions have provided evidence that current techniques have reached their limits, suggesting that breaking these barriers will require entirely new approaches [ZRE15]. The source of this limitation lies in the technique from [LP09], which assigns two labels to each wire and treats each gate as a set of four ciphertexts encrypting the labels associated with the output wire. Since each label functions as a key to the symmetric-key primitive, its length must scale with the security parameter to maintain privacy; otherwise, the scheme's security would be compromised. Therefore, the question of reducing the rate, the ratio between C and $\hat{C}$, while basing security only on symmetric key primitives, remained unresolved until recently.

A recent and exciting result by Liu et al. [LWYY24] presents the first rate-1 Boolean garbling scheme based on the Ring Learning with Errors (RLWE) or NTRU assumption, leveraging these to define a somewhat homomorphic encryption scheme. This encryption scheme is used to evaluate a low-depth pseudorandom generator (PRG) seed, which is subsequently used to derive the garbling material for each gate. The ciphertext is then decrypted using a key-dependent message (KDM)-secure encryption scheme. The bulk of the computational complexity arises from homomorphically evaluating the PRG seed, necessitating the use of a low-depth PRG. Additionally, the scheme is inherently non-black-box in its reliance on the details of the underlying PRG construction.

Leveraging stronger primitives allows for improved garbling schemes using laconic function evaluation (LFE), a dual primitive to fully homomorphic encryption (FHE), achieving sublinear rates, dependent on circuit depth, based on LWE or subexponential indistinguishability obfuscation (i$\mathcal{O}$). This is done by adding a garbling layer on top of the LFE protocol, garbling the LFE encoding function. The combination of LFE's succinctness and the privacy of garbling results in a succinct garbling scheme. Recent advancements in LFE [DGM23] eliminate the dependency on the circuit's depth. This is achieved through standard (polynomially secure) i$\mathcal{O}$ and somewhere statistically binding (SSB) hash functions, instantiated from various number-theoretic assumptions. In another recent work [HLL23], Hsieh et al. introduced a compact reusable garbling scheme based on a new circular-secure variant of LWE. Unlike previous constructions, which are limited to one-time use, their approach enables multiple uses, making it a stronger form of garbling. Finally, succinct randomized encoding for Turing machines [BGL+15], based on i$\mathcal{O}$ for P/poly and one-way functions, grows only polylogarithmically with the program's running time. However, extending this technique to a circuit representation of the computed function remains unclear. In Table 1, we summarize the current landscape of Boolean garbling schemes.

Ref.	$\lvert\hat{C}\rvert$	Hardness Assumption	Black-Box
[ZRE15]	$2\lambda \cdot \lvert C\rvert$	RO / RTCCR	✓
[RR21]	$1.5\lambda \cdot \lvert C\rvert$	RO / RTCCR	✓
[QWW18]	$\mathsf{poly}(\lambda) \cdot \mathsf{depth}(C)$	LWE / Subexponential i$\mathcal{O}$	✗
[DGM23]	$\mathsf{poly}(\lambda)$	i$\mathcal{O}$ + SSB	✗
[HLL23]	$\mathsf{poly}(\lambda)$	circular LWE[1]	✗
[LWYY24]	$(1 + o(1)) \cdot \lvert C\rvert$	RLWE / NTRU	✗
This Work	$\lambda \cdot O(\lvert C\rvert)/\sqrt{\log(\lambda)} + \mathsf{poly}(\lambda)$	GGM / ap-eTCCR[3]	✓
This Work[2]	$\lambda \cdot \frac{O(\lvert C\rvert)}{\sqrt{\log(\lambda)}} + \mathsf{poly}(\lambda) \cdot \mathsf{depth}(\lvert C\rvert)$	Power-DDH + eTCR[4]	✓

[1] Requires a new circular variant of LWE.
[2] Restricted to layered circuits.
[3] ap-eTCCR stands for Tweakable Circular Correlation Robustness for exponential correlations with auxiliary powers.
[4] eTCR stands for Tweakable Correlation Robustness for exponential correlations.

Table 1: The landscape of garbling schemes for Boolean circuits.

As it stands today, our understanding of Boolean garbling schemes with $\omega(1/\lambda)$ rates, based on assumptions that do not imply FHE, remains highly limited, even without the black-box requirement. This paper seeks to advance research in this direction by presenting two new sublinear-rate schemes that are black-box in their use of cryptographic primitives and rely on group-based assumptions for security, reducing the gap toward existing non-succinct schemes.

It is worth noting that when going beyond Boolean computations into the arithmetic regimes, the problem becomes simpler for the bounded setting, where the computation is performed over the integers while a bound B bounds the length of the wire values [BLLL23, MORS24, CHHK25]. This simplification arises from using stronger (non-symmetric-key) tools such as constant-rate additive encryption schemes or Homomorphic Secret Sharing (HSS). Specifically, the gap between the plain and the encoded data becomes smaller for larger computation domains, facilitating the construction of primitives with smaller rates. Our techniques are inspired by this sequence of works for designing arithmetic garbling [BLLL23, MORS24, CHHK25], building on [AIK11]. In particular, we demonstrate that the techniques developed in [CHHK25] can also be applied to Boolean circuits when packing the gates into batches of size $\sqrt{\log \lambda}$. Our packing mechanism is based on a novel application of the Reverse Multiplication-Friendly Embeddings (RMFE) paradigm from [CCXY18] that supports computations over tuples of binary values, where addition and multiplication are performed coordinate-wise, to be embedded into computations over an extension field. This technique has previously been only used in secure multi-party computation (MPC) e.g., [CG20, PS21, EHL+23], which is inherently interactive. To the best of our knowledge, our work is the first to apply RMFE-based embedding techniques to garbling schemes, which are non-interactive primitives.

More concretely, our first construction is proven under an assumption that is reminiscent of the Tweakable Circular Correlation Robust (TCCR) assumption used in [RR21] except that it considers exponential correlations [BCM+24, CHHK25] (namely, s^x where $s \in \mathbb{G}$ and $x \in \mathbb{Z}_{\mathsf{ord}(\mathbb{G})}$). We prove our construction in the generic group model (GGM) and the random oracle. Given that the random oracle can be instantiated in the GGM setting, our result presents the first succinct garbling scheme in the GGM model that retains black-box usage, reducing the garbling size by a factor of $O(\sqrt{\log(\lambda)})$. This scheme is proven for circuits that are not too narrow, requiring a width of $\sqrt{\log(\lambda)}$, a milder restriction than prior MPC work, which requires in some settings greater width to support packed secret sharing or is restricted to SIMD circuits[1]. Informally, we prove that,

Theorem 1 (Informal). *In the generic group model, there exists a Boolean garbling scheme* GC *that garbles any Boolean circuit C into a garbling $\hat{C}$ such that*

$$|\hat{C}| = \frac{\lambda}{\sqrt{\log(\lambda)}} \cdot O(|C|) + \mathsf{poly}(\lambda).$$

Our second construction is proven in the standard model under the power-DDH hardness assumption together with the existence of tweakable correlation robust hash functions for a family of exponential correlations, which implies a layered version of the TCCR assumption. This variant applies to layered circuits, where the gate set is partitioned into D levels such that gates at level i receive inputs from level $i-1$. Such circuits have been previously studied in the context of HSS [BGI16] and shown to be effective in overcoming communication barriers. Informally, we prove that.

Theorem 2 (Informal). *If there exists a TCR hash for the exponential correlation with respect to a group over which the power-DDH assumption holds, then there exists a Boolean garbling scheme that garbles layered circuits C into a garbling $\hat{C}$ such that*

$$|\hat{C}| = \frac{\lambda}{\sqrt{\log(\lambda)}} \cdot O(|C|) + \mathsf{poly}(\lambda) \cdot \mathsf{depth}(C).$$

2 Technical Overview

2.1 Overview of [CHHK25]

Our starting point is the recent work of [CHHK25], that described the first construction (without resorting to FHE or iO) of a garbling scheme for *arithmetic* circuits over a small (polynomial) integer ring $\mathbb{Z}_B$ with $O(\lambda)$ bits per gate (independently of B). At the heart of their construction are two efficient one-round protocols for computing on authenticated shares.

Concretely, fix a group $\mathbb{G}$ of prime order p with $|p| = O(\lambda)$ and assume that the following assumption holds: pick a random (secret) $h \leftarrow_\$ \mathbb{G}$, a secret exponent $\alpha \leftarrow_\$ \mathbb{Z}_p$, and compute h^{α^i} for all *nonzero* i from $-B$ to B. Then no efficient adversary should, given these values, be able to distinguish h from random. Under this variant of the power-DDH assumption, [CHHK25] showed the following:

Lemma 3 (informal). *Let* G *and* E *be two parties holding additive shares of $\Delta \cdot x$ over $\mathbb{Z}_{p-1}$, where Δ is a random MAC key (from $\mathbb{Z}_{p-1}$) known to* G*, and $x \in \{0, \cdots, B\}$ is a value known to* E*. Fix any (public) function $f : \{0, \cdots, B\} \to \mathbb{Z}_{p-1}$. Then there exists a one-message secure protocol where* G*, holding an input $y \in \mathbb{Z}_{p-1}$, sends $O(\lambda)$ bits to* E*, and both parties obtain additive shares of $y \cdot f(x) \bmod p-1$.*

Let us overview briefly how [CHHK25] uses this protocol to garbled a circuit C over $\mathbb{Z}_B$. The parties always maintain the following invariant: for any gate u of C carrying a value x_u during the computation of C on an input x, the parties will hold

[1] Circuits that possess multiple repetitions of smaller subcircuits.

- shares k_u, ℓ_u of x_u over B ($\ell_u \in \{0, \cdots, B\}$ is E's share), and
- shares K_u, L_u of $\Delta \cdot \ell_u$ over $\mathbb{Z}_{p-1}$.

Now, let us look at a multiplication gate w (additions are simpler). Let x_u and x_v denote the values on the wires entering the gate, with respective garbler shares (the *keys*) $(k_u, K_u), (k_v, K_v)$ and evaluator shares (the *labels*) $(\ell_u, L_u), (\ell_v, L_v)$. We have:

$$x_u x_v = k_u k_v + \ell_u \ell_v + k_u \ell_v + k_v \ell_u.$$

Above, the values in blue are known to one party and can be locally added to their share; the important terms are the cross terms in red. Given that they hold (by assumption) shares of $\Delta \cdot \ell_u$ and of $\Delta \cdot \ell_v$, the parties will simply use two invocations of the protocol from Lemma 3 (setting f to the identity function). The two messages from G will be part of the *garbling material* attached to this gate in the garbled circuit. Let us write $(\langle k_u \ell_v \rangle_{\mathsf{G}}, \langle k_v \ell_u \rangle_{\mathsf{G}})$ and $(\langle k_u \ell_v \rangle_{\mathsf{E}}, \langle k_v \ell_u \rangle_{\mathsf{E}})$ the shares obtained by G and E respectively. We define

$$\begin{aligned} k_w &:= \langle k_u \ell_v \rangle_{\mathsf{G}} + \langle k_v \ell_u \rangle_{\mathsf{G}} + k_u k_v \\ \ell_w &:= \langle k_u \ell_v \rangle_{\mathsf{E}} + \langle k_v \ell_u \rangle_{\mathsf{E}} + \ell_u \ell_v. \end{aligned}$$

We now turn our attention to the task of building shares of $\Delta \cdot \ell_w$. We have

$$\begin{aligned} \Delta \cdot \ell_w &= \Delta \cdot (\langle k_u \ell_v \rangle_{\mathsf{E}} + \langle k_v \ell_u \rangle_{\mathsf{E}} + \ell_u \ell_v) \\ &= \Delta \cdot (k_u \ell_v - \langle k_u \ell_v \rangle_{\mathsf{G}} + k_v \ell_u - \langle k_v \ell_u \rangle_{\mathsf{G}}) + (K_u + L_u)\ell_v \\ &= (\Delta k_u) \cdot \ell_v + (\Delta k_v) \cdot \ell_u + K_u \cdot \ell_v + L_u \ell_v - \Delta \cdot (\langle k_u \ell_v \rangle_{\mathsf{G}} + \langle k_v \ell_u \rangle_{\mathsf{G}}), \end{aligned}$$

where again the terms in blue are known to one of the parties, while the terms in red are the product of a value known to G with a value in $\{0, \cdots, B\}$ (ℓ_u or ℓ_v) known to E. Hence, these three cross terms are shared using three more instances of the protocol of Lemma 3. Then, G defines K_w as the sum of the shares of these cross terms minus $\Delta \cdot (\langle k_u \ell_v \rangle_{\mathsf{G}} + \langle k_v \ell_u \rangle_{\mathsf{G}})$, and E defines L_w as the sum of its shares plus $L_u \ell_v$, giving $K_w + L_w = \Delta \cdot \ell_w$, as required.

This overview overlooks important technicalities, which we briefly sketch as they will also show up in our work. First, there is a size issue: the value ℓ_w computed above does not belong to $\{0, \cdots, B\}$, which is crucial (otherwise, ℓ_w cannot be used as input in the protocol of Lemma 3). Of course, a simple fix is to reduce it modulo B (it is not too hard to guarantee that the shares of the cross terms are shares over the integers). The problem is that now, denoting $\tilde{\ell}_w$ the original value (before reduction modulo B) and $\ell_w = [\tilde{\ell}_w \mod B]$ the reduced value, we have a modulus mismatch: K_w, L_w form shares of $\Delta \cdot \tilde{\ell}_w$, while we need them to form shares of $\Delta \cdot \ell_w = \Delta \cdot [\tilde{\ell}_w \mod B]$.

This is where the protocol of Lemma 3 comes to the rescue: the parties will use one last invocation of this protocol, where E inputs $\tilde{\ell}_w$ and G inputs Δ, setting f to be the function "reduction mod B", to obtain shares of $\Delta \cdot f(\tilde{\ell}_w) = \Delta \cdot \ell_w$. However, for this to work, we crucially need $\tilde{\ell}_w$ to be small in the first place! In [CHHK25], this is solved by introducing a (more complex) variant of the protocol that guarantees that the shares of $k_u \ell_v$ and $k_v \ell_u$ are actually *small integers* (roughly bounded by B^3). This requires care, as G's message will now contain its input (say, k_u) masked over the integers by small values, which introduces some leakage on k_u. A core technical contribution of [CHHK25] is a way to add a carefully crafted *noise* to k_u to guarantee that the leakage remains harmless with overwhelming probability while letting the function f evaluated on $\tilde{\ell}_w$ remove the noise in the end.

2.2 Computing on Batches via RMFE

Unfortunately, the methodology of [CHHK25] does not improve over traditional garbling schemes (such as Yao's) when the ring is $\mathbb{Z}_2$, since it still requires $\Omega(\lambda)$ bits for each gate of the circuit. To improve the garbled circuit size over $\mathbb{Z}_2$, our high-level approach is the following: we devise a methodology to *pack* the bits carried on multiple values into a single element of a larger ring, and rely on the approach of [CHHK25] to operate on these "packed ring elements". Then, to ensure that the computation proceeds according to the topology of the circuit, we also devise methods to efficiently *unpack* a batch of wire values and reorder them cheaply into new batches.

The core ingredient of our approach is the notion of Reverse Multiplication-Friendly Embeddings (RMFE) [CCXY18]. A (t, m)-RMFE is a pair of $\mathbb{F}_2$-linear maps $\Phi : \mathbb{F}_2^t \longrightarrow \mathbb{F}_{2^m}$ and $\Psi : \mathbb{F}_{2^m} \longrightarrow \mathbb{F}_2^t$ satisfying $\mathbf{x} \odot \mathbf{y} = \Psi(\Phi(\mathbf{x}) \cdot \Phi(\mathbf{y}))$ for all $\mathbf{x}, \mathbf{y} \in \mathbb{F}_2^t$, where $\odot$ denotes the component-wise product. It is not too hard to see that RMFEs also come with an inverse map Φ^{-1} (which is not the same as Ψ). The main result on RMFE that we use is a central lemma from [CCXY18] (restated here as Lemma 4): there exists a family of (t, m)-RMFE where $m = \Theta(t)$.

For the sake of exposition, assume that we have at hand a "magic protocol" identical to that of Lemma 3, but that would operate natively over elements of any (small) extension field of $\mathbb{F}_2$, such as $\mathbb{F}_{2^m}$. That is, given inputs $x \in \mathbb{F}_{2^m}$ from E and $y \in \mathbb{F}_{2^\lambda}$ from G, a public function f, and shares of $\Delta \cdot y$ with $\Delta \in \mathbb{F}_{2^\lambda}$ known to G (we stress that this is a thought experiment – we do not actually have such a scheme), the parties could, using one $O(\lambda)$-bit message from the garbler to the sender, obtain shares of $y \cdot f(x)$. Then, we could apply the following approach:

First, divide the Boolean circuit into layers, such that each layer contains gates of a single type (either AND or XOR) and takes its inputs from previous layers. Any Boolean circuit can be converted into one of this form with a constant factor blowup [DIK10]. Fix a batch size t and break each layer into blocks of t bits. Assume that the parties maintain the following invariant: for any gate u with a bit x_u, they will hold shares $k_u \oplus \ell_u = x_u$ and shares $K_u + L_u = \Delta \cdot \ell_u$. Now, consider a layer of AND gates. For a given batch $\mathcal{B}$ of t gates in the layer, let Left and Right denote the size-t subset of the left-parents and right-parents of the node in $\mathcal{B}$. The parties G, E execute the following steps:

Packing. Aggregate the shares of x_u for $u \in \mathsf{Left}$ into a share of an element $x_\mathsf{l} \in \mathbb{F}_{2^t}$ whose bits are the x_u's, and the shares of $\Delta \cdot \ell_u$ into a share of $\Delta \cdot \ell_\mathsf{l}$, where the bits of $\ell_\mathsf{l} \in \mathbb{F}_{2^t}$ are the ℓ_u's. Do the same thing for Right, getting shares of x_r and $\Delta \cdot \ell_\mathsf{r}$.

RME Encoding. Compute the RMFEs $\Phi(x_\mathsf{l}), \Phi(x_\mathsf{r})$. Using (two calls to) the "magic protocol" with input Δ from G and $\ell_\mathsf{l}, \ell_\mathsf{r}$ from E and function $f = \Phi$, the garbler adds $O(\lambda)$ bits to the garbling material of the batch $\mathcal{B}$ and the parties obtain shares $(K_\mathsf{l}, L_\mathsf{l}), (K_\mathsf{r}, L_\mathsf{r})$ of $\Delta \cdot \Phi(\ell_\mathsf{l})$ and $\Delta \cdot \Phi(\ell_\mathsf{r})$.

Product. The parties use the same equations as before:

$$\Phi(x_\mathsf{l}) \cdot \Phi(x_\mathsf{r}) = \Phi(\ell_\mathsf{l}) \cdot \Phi(\ell_\mathsf{r}) + \Phi(k_\mathsf{l}) \cdot \Phi(k_\mathsf{r}) + \Phi(\ell_\mathsf{l}) \cdot \Phi(k_\mathsf{r}) + \Phi(k_\mathsf{l}) \cdot \Phi(\ell_\mathsf{r}).$$

The red cross terms are computed via two calls to the "magic protocol" (with $O(\lambda)$ bits of added material) and the evaluator sets

$$\tilde{\ell} := \langle \Phi(\ell_\mathsf{l}) \cdot \Phi(k_\mathsf{r}) \rangle_\mathsf{E} + \langle \Phi(k_\mathsf{l}) \cdot \Phi(\ell_\mathsf{r}) \rangle_\mathsf{E} + \Phi(\ell_\mathsf{l}) \cdot \Phi(\ell_\mathsf{r}).$$

The garbler computes a corresponding $\tilde{k}$. Then, the parties compute shares of $\Delta \cdot \tilde{\ell}$ using

$$\begin{aligned}\Delta \cdot \tilde{\ell} =& (\Delta\Phi(\ell_\mathsf{l})) \cdot \Phi(k_\mathsf{r}) + (\Delta\Phi(\ell_\mathsf{r})) \cdot \Phi(k_\mathsf{l}) + K_\mathsf{l} \cdot \Phi(\ell_\mathsf{r}) \\ & - \Delta \cdot (\langle \Phi(\ell_\mathsf{l}) \cdot \Phi(k_\mathsf{r}) \rangle_\mathsf{G} + \langle \Phi(k_\mathsf{l}) \cdot \Phi(\ell_\mathsf{r}) \rangle_\mathsf{G}) + L_\mathsf{l}\Phi(\ell_\mathsf{r}),\end{aligned}$$

using three calls to the magic protocol to share the cross terms.

Unpacking. Eventually, the parties compute the shares (k_u, ℓ_u) for $u \in \mathcal{B}$ by using the RMFE mapping $\Psi(\tilde{k}), \Psi(\tilde{\ell})$ (recall that the mapping is $\mathbb{F}_2$-linear), obtaining shares of $(x_u)_{u \in \mathsf{Left}} \odot (x_u)_{u \in \mathsf{Right}}$ (which form the t outputs of the batch of gates). It remains to obtain shares of $\Delta \cdot \ell_u$ for each $u \in \mathcal{B}$; this is done using more calls to the magic protocol with input Δ from G, $\tilde{\ell}$ from E, and for all functions $f_i = \mathsf{Bit}_i \circ \Psi$, where Bit_i outputs the i-th bit of its input.

The above high-level description successfully maintains the invariant, and circumvents the issue of handling the topology of the circuit, since all shares are projected back to bitwise-authenticated shares after each batch of operations. Nevertheless, it suffers from two annoying downside:

1. We are not aware of any "magic protocol" satisfying the requirements listed above, and

2. The projection of $\Delta \cdot \tilde{\ell}$ to $\Delta \cdot \ell_u$ for all $u \in \mathcal{B}$ requires t calls to the protocol, which incurs an $\Omega(t \cdot \lambda)$ overhead in the size of the garbling material, which is too much (as it does not improve over classical Yao-style garbling).

Below, we explain how to deal with each issue in turn.

2.3 Replacing the "Magic Protocol"

Our strategy is to emulate the features of the magic protocol while relying instead on the protocol from Lemma 3 (since we *do* know of an instantiation of this one). The core component of our strategy is the following (natural) embedding of $\mathbb{F}_{2^k}$ over the integers (for any k): view $\mathbb{F}_{2^k}$ as $\mathbb{F}_2[X]/P(X)$ where P is an irreducible polynomial of degree k, and parse elements of $\mathbb{F}_{2^k}$ as $\mathbb{F}_2$-polynomials of degree at most k (we write $\mathbb{F}_2[X;k]$ to denote this set). For any $x = \sum_{i=0}^{k-1} x_i \cdot X^i \in \mathbb{F}_2[X;k]$, we embed x over the integers by computing $x(N) = \sum_{i=0}^{k-1} x_i \cdot N^i \in \mathbb{N}$ (for an integer N to be specified later), and we further view $x(N)$ as an element of $\mathbb{Z}_{p-1}$ via the canonical embedding.

A useful feature of this embedding is that it preserves operations over $\mathbb{F}_{2^k}$ to some extent. Given $x, y \in \mathbb{F}_{2^k}$, if $N > k$, then $x(N) \cdot y(N)$ *encodes* $x \cdot y$ in the following sense: denote $\mathsf{Mod}_N(n, 2)$ the function that, on an integer u, writes u in N-arry as $u = \sum_i u_i \cdot N^i$ and returns $\sum_i [u_i \bmod 2] \cdot N^i$ (that is, it reduces each coefficient of the N-ary decomposition of u modulo 2). Then, provided that $N > k$, we have $x \cdot y = \mathsf{Mod}_N(x(N) \cdot y(N), 2)$. Using this embedding, our strategy to emulate the magic protocol is the following:

- For each gate u, in addition to XOR-share (k_u, ℓ_u) of x_u, the parties will hold shares of $\Delta \cdot \ell_u$ *over* $\mathbb{Z}_{p-1}$.

- When packing, the parties will compute linear combinations of their shares of values $\Delta \cdot \ell_i$ with powers of N, to obtain shares of $\Delta \cdot (\sum_i \ell_i \cdot N^i)$.

- We will make heavy use of the fact that the one-message protocol from Lemma 3 can evaluate arbitrary functions. For instance, the parties will use an invocation of this protocol to get $\Delta \cdot f(\sum_i \ell_i \cdot N^i)$, where f is the function that (1) extracts all the bits $(\ell_i)_i$ from this encoding; (2) compute $\Phi((\ell_i)_i)$; (3) re-embed this value onto $\mathbb{Z}_{p-1}$ by viewing it as a polynomial and computing $\Phi((\ell_i)_i)(N)$. After computing a product of embedded values, the parties will again use this feature to evaluate the $\mathsf{Mod}_N(\cdot, 2)$ function, but also to reduce the (embedded) polynomial modulo P (in order to obtain an embedding of the correct product over $\mathbb{F}_{2^m}$).

The above sketch hides some very important technicalities. The most important one is the fact that, as in [CHHK25], maintaining the invariant requires ensuring that the shares of the (embeddings of the) cross terms $\Phi(\ell_\mathsf{l}) \cdot \Phi(k_\mathsf{r}), \Phi(k_\mathsf{l}) \cdot \Phi(\ell_\mathsf{r})$ are small, but also, crucially, *that the sharing produced by the protocol of Lemma 3 remains compatible with the (limited) homomorphic properties of the embedding*. For instance, given

$$\Phi(x_\mathsf{l}) \cdot \Phi(x_\mathsf{r}) = \Phi(\ell_\mathsf{l}) \cdot \Phi(\ell_\mathsf{r}) + \Phi(k_\mathsf{l}) \cdot \Phi(k_\mathsf{r}) + \Phi(\ell_\mathsf{l}) \cdot \Phi(k_\mathsf{r}) + \Phi(k_\mathsf{l}) \cdot \Phi(\ell_\mathsf{r}),$$

The parties will operate only on *integer embeddings* ($\Phi(\ell_\mathsf{l})(N), \Phi(\ell_\mathsf{r})(N)$, etc) and will apply the $\mathsf{Mod}_N(\cdot, 2)$ and the $\mod P$ operations on the shares of the red terms via calls to the protocol. But for this to work, we need that the outputs indeed sum to (the integer embedding of) $\mathsf{ModP}(\mathsf{Mod}(\Phi(x_\mathsf{l})(N) \cdot \Phi(x_\mathsf{r})(N), 2))$ (where ModP is the function that extracts the embedded polynomials, reduces it modulo P, and embeds the result). This, in turn, depends on how much the protocol from Lemma 3 blows up the size of the shares.

In fact, the protocol does incur a significant blowup – the shares are computed as a sum of 2^m terms, where each term is a product of embeddings, and one of the embeddings has been perturbated with noise to protect against leakage. Nevertheless, a careful choice of N ensures that the homomorphic properties are sufficient to support these computations. Increasing the size of N this much has a cost, though: the computational complexity of the protocol will grow as much as 2^{m^2} (instead of the naive 2^m one could have hoped for). This is the main reason why our result is limited to batching up to $t = \sqrt{\log \lambda}$ since 2^{m^2} must remain polynomial (and $m = O(t)$). We defer the remaining technical details on these issues to the main body.

2.4 Batch Function Evaluation

We now turn our attention to the second downside of our template: the unpacking procedure has a cost scaling as $O(t \cdot \lambda)$. Here, we make a simple but crucial observation: in the protocol from [CHHK25], when computing shares of $y \cdot f(x)$ the message from the garbler to the evaluator depends *solely on* y, and not on the function f! The only dependency on f appears in the *local computation* of the parties. A consequence of this observation is that when y stays the same across multiple instances, G and E can reuse the *same* $O(\lambda)$-bit message to compute shares of $y \cdot f_i(x)$ for an arbitrary number of functions f_i. This simple but powerful observation immediately allows to reduce the cost of unpacking from $O(t\lambda)$ to $O(\lambda)$.

2.5 Dealing with Circular Security

We now discuss an important aspect that we have glossed over so far. The protocol claimed in Lemma 3 can be proven secure under power-DDH only if the garbler input y is *independent of* Δ. Indeed, abstracting out some details, recall that power-DDH says that given a secret $h \leftarrow_\$ \mathbb{G}$, a secret exponent $\alpha \leftarrow_\$ \mathbb{Z}_p$, no efficient adversary should, given the h^{α^i} for all *nonzero* i from $-B$ to B, be able to distinguish h from random. In the protocol of [CHHK25], the evaluator will learn a value of the form $y + H(h^z)$, where z is some value known to the evaluator and H is a suitable hash function from $\mathbb{G}$ to $\mathbb{Z}_{p-1}$. Then, security is shown by using power-DDH to replace h with a random group element, effectively replacing $H(h^z)$ with a random value, hence hiding y. However, the secret exponent α is tied to the MAC Δ used in the protocol: given a generator G of $\mathbb{Z}_p$, they satisfy the relation $\alpha = G^\Delta \mod p$. Hence, whenever we decide to set the garbler's input to Δ in this protocol (which we do in several steps), we leak $\Delta + H(h^z)$ to the evaluator, where the masked value is now a function of the secret exponent α itself, making it impossible to invoke power-DDH to randomize this term!

We provide two alternatives to deal with this issue. First, we formalize the exact security notion required to prove the security of our schemes. In essence, the notion states that terms of the form $H(h^z)+\Delta\cdot v$ for known (z,v) should look jointly indistinguishable from random to an adversary knowing the h^{α^i} for $i\neq 0$. Then, we prove that when modeling the group as a generic group (in Shoup's variant of the GGM) and the hash function H as a random oracle, this assumption holds unconditionally. Because a random oracle can be constructed from Shoup's GGM, this implies that our entire garbling scheme can be proven secure in the GMM.

Second, we use the same strategy as [CHHK25]: we rely instead on a *leveled* version, where we use a different Δ_j for each layer and only use the secret h_j associated with the Δ_j from a layer to mask Δ_{j+1}. With this change, security can be proven under the power-DDH assumption and the (non-circular) correlation robustness of the hash function for a suitable family of "exponential correlations". The price to pay is twofold: first, we must now include in the garbled circuit a tuple of the form $(h^{\alpha^i})_i$ for each layer, adding a term $\mathsf{depth}(C)\times\mathsf{poly}(\lambda)$ to the size of the garbled circuit. Second, and more annoyingly, our packing procedure crucially requires that all the values being packed are authenticated *under the same* Δ. This constrains us to restrict our attention to *layered* Boolean circuits, where all the parent nodes of a layer are in the previous layer, ensuring that when evaluating the j-th layer, all the nodes to be packed are authenticated with the same Δ_{j-1}.

3 Preliminaries

We begin by introducing the notation that will be used throughout the subsequent sections.

General notation. Given a distribution $\mathcal{D}$ (resp. a set S), we write $x \leftarrow\!\!\$\,\mathcal{D}$ (resp. $x \leftarrow\!\!\$\,S$) to denote that x is sampled from $\mathcal{D}$ (resp. that x is sampled uniformly over S). Given an integer B, we denote by $[B]$ the set $\{0,\cdots,B\}$, by $[\pm B]$ the set $\{-B,-B+1,\cdots,0,\cdots,B-1,B\}$, and by $[B]^*$, $[\pm B]^*$ the sets sets $[B]$, $[\pm B]$ without 0. When convenient, we let poly denote an unspecified polynomial.

Arithmetic. Given integers u,n, we write $[u \bmod n]$ to denote the representative of $u \bmod n$ as an element of $[n-1]\subset\mathbb{N}$. More generally, if u denotes a polynomial, n an integer, and P a polynomial, we write $[u \bmod 2, P]$ to denote the representative of $(u \bmod 2) \bmod P$ as an element of $\mathbb{N}[X]$ of degree at most $\deg(P)-1$ and with coefficients in $[n-1]$.

Polynomials. Given a parameter m, we let P_m denote an irreducible degree-m over $\mathbb{F}_2$. We view elements of $\mathbb{F}=\mathbb{F}_{2^m}$ as polynomials over $\mathbb{F}_2[X]/P_m(X)$. For any ring $\mathcal{R}$, we write $\mathcal{R}[X;m]$ to denote the set of polynomials $a\in\mathcal{R}[X]$ with $\deg(a)\leq m$. Given a polynomial $r=\sum_i r[i]\cdot X^i$, let Eval_N denote the procedure that, on input r, returns $r(N)=\sum_i r[i]\cdot N^i$. By default, addition ("+") refers to the addition over the structure the operands live in. As we often switch between interpretations (e.g., viewing an element of $\mathbb{F}_{2^m}\equiv\mathbb{F}_2[X]/P(X)$ as an element of $\mathbb{Z}[X]$), we add clarification whenever there is an ambiguity. We also sometimes write $\oplus$ to denote the bitwise-XOR to make it clear that the coefficient-wise addition is done modulo 2.

Garbling. Throughout this paper, we let G denote the garbler, and E denote the evaluator. We use the notation $\langle x\rangle$ for additive (or subtractive) shares of x. Since this sharing is frequently between a garbler and an evaluator, we will use $\langle x\rangle_{\mathsf{G}}$ to denote the garbler's share of x and $\langle x\rangle_{\mathsf{E}}$ to denote the evaluator's share of x.

We use the standard definition of garbling schemes from [BHR12], specialized as in previous works [ZRE15], to the setting of Boolean circuits.

Definition 1 (Garbling Scheme)**.** A *garbling scheme* GC for Boolean circuits consists of the following algorithms.

- $\mathsf{GC.Garble}(1^\lambda, C)$: A PPT algorithm that on input 1^λ and a Boolean circuit C, outputs $(\hat{C}, \mathsf{e}, \mathsf{d})$ where $\hat{C}$ is a *garbled circuit*, e is an *encoding information*, and d is a *decoding information*.

- $\mathsf{GC.Enc}(\mathsf{e}, x)$: A polynomial time algorithm that on input e and $x \in \{0,1\}^{|I(C)|}$, output a *garbled input* $\hat{x}$.

- $\mathsf{GC.Eval}(\hat{C}, \hat{x})$: A polynomial time algorithm that on input a garbled circuit and a garbled input, outputs a *garbled output* p.

- $\mathsf{GC.Dec}(\mathsf{d}, \hat{y})$: A polynomial time algorithm that on input the decoding information and the garbled output, outputs $y \in \{0,1\}^{|O(C)|}$.

A garbling scheme is *correct* if for every Boolean circuit C, every $(\hat{C}, \mathsf{e}, \mathsf{d})$ in the support of $\mathsf{GC.Garble}(1^\lambda, C)$, and every input $x \in \{0,1\}^{|I(C)|}$, it holds that

$$\mathsf{GC.Dec}(\mathsf{d}, \mathsf{GC.Eval}(\hat{C}, \mathsf{GC.Enc}(\mathsf{e}, x))) = C(x).$$

Furthermore, a garbling scheme is *private* if there exists a simulator SimGC such that for every infinite family $\{C_\lambda\}_{\lambda\in\mathbb{N}}$ of Boolean circuits with $|C_\lambda| \leq \mathrm{poly}(\lambda)$ and every infinite family of inputs $\{x_\lambda\}_{\lambda\in\mathbb{N}}$ with $|x_\lambda| = |I(C_\lambda)|$ for every $\lambda \in \mathbb{N}$, the following distribution families (parameterized with λ) are indistinguishable:

$$\{(\hat{C}, \hat{x}, \mathsf{d}) \leftarrow\$ \mathsf{SimGC}(1^\lambda, C_\lambda, C(x_\lambda))\}$$
$$\{(\hat{C}, \hat{x}, \mathsf{d}) : (\hat{C}, \mathsf{e}, \mathsf{d}) \leftarrow\$ \mathsf{GC.Garble}(1^\lambda, C),\ \hat{x} \leftarrow\$ \mathsf{GC.Enc}(\mathsf{e}, x_\lambda)\}$$

3.1 Generic Group Model

We rely on Shoup's generic group model [Sho97] (GGM). For simplicity, we restrict our attention to prime order groups.

Definition 2 (Shoup's GGM)**.** Let p denote a prime. Fix a set $\mathbf{G}$ of cardinality p and let σ denote a random bijective mapping from $\mathbb{Z}_p$ to $\mathbf{G}$. Given p (available to all parties), in Shoup's GGM, all parties have access to a *group oracle* $O_{\mathbb{G}}$ with the following queries:

- $\mathsf{Encode}(x)$: given $x \in \mathbb{Z}_p$, return $\sigma(x)$.

- $\mathsf{Add}(a, b, \sigma(x), \sigma(y))$: given $a, b \in \mathbb{Z}_p$ and $\sigma(x), \sigma(y) \in \mathbf{G}$, set $z \coloneqq a \cdot x + b \cdot y \bmod p$ and return $\sigma(z)$.

We let $g \coloneqq \mathsf{Encode}(1)$ denote a fixed generator of $\mathbf{G}$. Note that any party can sample uniformly from $\mathbf{G}$ given one call to Encode. As a shorthand for exponentiations, given $x \in \mathbb{Z}_p$ and $h \in \mathbf{G}$, we write $x \bullet h$ to denote $\mathsf{Add}(x, 0, h, \mathsf{Encode}(0)) = \mathsf{Encode}(x \cdot y)$ where $h = \mathsf{Encode}(y)$.

3.2 Boolean Circuits

A circuit is a directed acyclic graph. Internal nodes are called *gates*, nodes of indegree 0 are called *input gates*, and nodes of outdegree 0 are called *output gates*. Edges are called *wires*. In this work, we consider polynomial-size Boolean circuits of fan-in 2 over the basis $\{\oplus, \wedge\}$. Given a circuit C, we let $D \coloneqq \mathsf{depth}(C)$ denote the depth of C (the length of the longest path from an input to an output), $I(C)$ denote the set of input wires, $W(C)$ denote the set of all wires, $O(C)$ denote the set of output wires, and $\Gamma(C)$ denote the set of gates. We write $|C|$ to denote the size of C (the number of gates in C).

Layered circuits. In a layered boolean circuit C, the set of gates $\Gamma(C)$ can be partitioned into $D = \mathsf{depth}(C)$ layers $(\mathcal{L}_1, \ldots, \mathcal{L}_D)$ such that every wire connects adjacent layers i.e., every edge $(u, v) \in C$ is such that $u \in \mathcal{L}_i$ and $v \in \mathcal{L}_{i+1}$ for some $i \in [D-1]$. Thus, the inputs to gates in $\mathcal{L}_1$ consist only of the circuit inputs. Any boolean circuit of size s and depth D can be computed by a layered circuit of size sD, with a lower bound of $s \log s$.

Rate of Boolean garbling. The rate of a boolean garbling scheme is a measure of the efficiency of the scheme.

Definition 3 (Rate of a boolean garbling scheme)**.** Let $\mathcal{C}$ be a class of boolean circuits, and let GC be a boolean garbling scheme for $\mathcal{C}$. The rate of GC is defined as

$$\liminf_{C \in \mathcal{C}} \min_{\hat{C} \in \mathcal{S}} \min_{x} \frac{|C| + |I(C)|}{|\hat{C}| + |\mathsf{e}|}$$

where $\mathcal{S} = \mathsf{Sup}(\mathsf{GC.Garble}(1^\lambda, C))$, the minimum is taken over all admissible inputs to C, and the limit infimum is taken over $\mathcal{C}$ partially ordered by subcircuit inclusion.

For garbling schemes in this work, the size of the garbled circuit $\hat{C}$ primarily depends on the size (and depth) of the circuit C and the size of the encoding information e is $O(\lambda \cdot I(C))$. In this case, it suffices to consider the rate as

$$\min_{\substack{C \in \mathcal{C} \\ x}} \frac{|C|}{|\hat{C}|}$$

where $\hat{C}$ is the size of the garbled circuit corresponding to C.

3.3 Preliminaries on Power-DDH

We let $\mathsf{GrpGen}(1^\lambda)$ denote a deterministic algorithm that, on input 1^λ, outputs a tuple $(\mathbb{G}, p, g)$ where $\mathbb{G}$ is a cyclic group of order a prime p of length $O(\lambda)$ bits, and g is a generator of $\mathbb{G}$. We recall the variant of power-DDH introduced in [CHHK25]:

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

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

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

As shown in [CHHK25], this formulation of power-DDH is equivalent to the other traditional formulation of the power-DDH assumption [GJM03, CNs07] (where the last term h^{α^B} is the one that should be indistinguishable from random), and more generally to the variant where all terms h^{α^i} are replaced by random, up to a factor-2 loss in the size of B.

3.4 Reverse Multiplication-Friendly Embeddings

Reverse Multiplication-Friendly Embeddings (RMFE) [CCXY18] allow computations over tuples of binary values, where addition and multiplication are performed coordinate-wise, to be embedded into computations over an extension field. Ideally, such an embedding would be a ring isomorphism that preserves both addition and multiplication i.e., the embedding would be a map $\Phi : \mathbb{F}_2^t \to \mathbb{F}_{2^t}$ such that $\Phi(\mathbf{x}+\mathbf{y}) = \Phi(\mathbf{x})+\Phi(\mathbf{y})$ and $\Phi(\mathbf{x} \cdot \mathbf{y}) = \Phi(\mathbf{x}) \cdot \Phi(\mathbf{y})$. However, such an isomorphism cannot exist: while $\mathbb{F}_2^t$ and $\mathbb{F}_{2^t}$ are isomorphic as additive groups, their multiplicative structures differ (e.g., $\mathbb{F}_2^t$ has zero divisors while $\mathbb{F}_{2^t}$ does not). To circumvent this issue, RMFEs provide a mapping $\Phi : \mathbb{F}_2^t \to \mathbb{F}_{2^m}$ with a weaker guarantee: it allows embedding a *single* multiplication over $\mathbb{F}_2^t$ as multiplication over $\mathbb{F}_{2^m}$. After each multiplication, the result in $\mathbb{F}_{2^m}$ must be mapped back to $\mathbb{F}_2^t$ using $\Psi : \mathbb{F}_{2^m} \to \mathbb{F}_2^t$, then re-embedded into $\mathbb{F}_{2^m}$ using Φ before another multiplication can be performed. We next recall the definition of RMFEs.

Definition 5 (Reverse Multiplication Friendly Embeddings [CCXY18])**.** Let q be a prime power, $\mathbb{F}_q$ be a field of q elements, and let $m, t \geq 1$ be integers. A pair (Φ, Ψ) is called a $(t, m)_q$-reverse multiplication friendly embedding (RMFE) if $\Phi : \mathbb{F}_q^t \to \mathbb{F}_{q^m}$ and $\Psi : \mathbb{F}_{q^m} \to \mathbb{F}_q^t$ are two $\mathbb{F}_q$-linear maps satisfying

$$\mathbf{x} \cdot \mathbf{y} = \Psi\left(\Phi(\mathbf{x}) \cdot \Phi(\mathbf{y})\right)$$

for all $\mathbf{x}, \mathbf{y} \in \mathbb{F}_q^t$.

Let $\mathbf{1} = (1, \ldots, 1) \in \mathbb{F}_2^t$ and let $\Phi^{-1} : \mathbb{F}_{2^m} \to \mathbb{F}_2^t$ be defined as $\Phi^{-1}(x) = \Psi(\Phi(\mathbf{1}) \cdot x)$. It then follows that for all $\mathbf{x} \in \mathbb{F}_2^t$, $\Phi^{-1}(\Phi(\mathbf{x})) = \Psi(\Phi(\mathbf{1}) \cdot \Phi(\mathbf{x})) = \mathbf{x}$, and thus, Φ^{-1} can be used for decoding an embedded value $x \in \mathbb{F}_{2^m}$. Moreover, it is easy to see that Φ^{-1} is also $\mathbb{F}_2$-linear.

Our use of RMFEs is motivated by the fact that computation over the extension field $\mathbb{F}_{2^m}$ can be expressed as an arithmetic circuit, making it compatible with efficient techniques for arithmetic garbling. Consequently, the rate of the embedding, t/m, is crucial for ensuring that the efficiency of computation over $\mathbb{F}_{2^m}$ translates to computation over $\mathbb{F}_2^t$. The following lemma from [CCXY18] establishes the existence of constant rate RMFEs.

Lemma 4 (Constant rate RMFE [CCXY18])**.** *For every finite prime power q, there exists a family of $(t, m)_q$-RMFE where $m = \Theta(t)$.*

4 Correlation-Robustness for Exponential Correlations

Given a secret s, a hash function H is said to be *correlation-robust* for a class of correlations C if (informally) samples of the form $\mathsf{H}(C(x, s))$ for public inputs x's are indistinguishable from random. Several classes of correlations have been commonly used in the literature, such as additive correlations [IKNP03] ($C(x, s) = x + s$), affine correlations [SS24] ($C((x_0, x_1), s) = x_0 \cdot s + x_1$), group-induced correlations [AMN+18] ($C(x, s) = x \cdot s$ where x, s belong to some group $(\mathbb{G}, \cdot)$) and exponential correlations [BCM+24, CHHK25] ($C(x, s) = s^x$ where $s \in \mathbb{G}$ and $x \in \mathbb{Z}_{\mathrm{ord}(\mathbb{G})}$).

Tweakable circular correlation-robustness (TCCR). Previous works on garbled circuit typically require a strengthening of the notion of correlation-robustness:

- A hash function H is *tweakable* correlation-robust for a class of correlations C if (informally) samples of the form $\mathsf{H}(C(x, s), y)$ are indistinguishable from random given public inputs x's and public *tweaks* y's (where all pairs (x, y) are distinct).

- A hash function H is *circularly* correlation-robust for C if (informally) samples of the form $\mathsf{H}(C(x, s)) + L(s)$ are indistinguishable from random, where the x's are public inputs and the L's are public linear functions. In other words, the hash outputs can be used to mask (linear functions of) the secret key s. If the hash can additionally take a tweak y and samples of the form $H(C(x, s), y) + L(s)$ are indistinguishable from random, we say that H is *tweakable circular correlation-robust* (TCCR) for the correlation C.

Circular correlation-robustness was first defined and studied in [CKKZ12]. Several variants have been used and refined in subsequent works [ZRE15, GKWY20]. The variant used in our work is closer in spirit to the notion of tweakable circular correlation-robustness (TCCR) used in [RR21].

Tweakable correlation-robust hashing for exponential correlations. Given a security parameter λ, fix group parameters $(\mathbb{G}, p, g, G) \coloneqq \mathsf{GrpGen}^*(1^\lambda)$ (recall that g generates $\mathbb{G}$, and G generates $\mathbb{Z}_p^*$). We start by defining the simplest variant of correlation-robustness considered in this work, where no circular security is required, and where the adversary is not given access to auxiliary inputs.

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

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

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

where the probability is taken over the random choice of $h \leftarrow\!\!\$\ \mathbb{G}$ and of a random oracle $\mathcal{R} : \mathbb{Z}_{p-1} \times \{0, 1\}^* \to \mathbb{Z}_{p-1}$.

The above assumption refers to (tweakable) correlation-robustness for the same correlation as [CHHK25], where the secret is a random $h \leftarrow\!\!\$\ \mathbb{G}$, and the correlation is given by $C(x, h) = h^x$ for $x \in \mathbb{Z}_{p-1}$. We call this correlation the *exponential correlation over* $\mathbb{G}$.

4.1 Circular Correlation-Robustness in the Generic Group Model

In this work, we rely on (a form of) tweakable *circular* correlation-robust hash for the exponential correlation over $\mathbb{G}$. In addition, we need a strengthening of the notion where the adversary is given *auxiliary information* in the form of group elements $h_i \coloneqq h^{G^{i\cdot\Delta}}$ for a random $\Delta \leftarrow\!\!\$\ \mathbb{Z}_{p-1}$ and various $i \neq 0$, and where circular security must hold with respect to linear functions of Δ.

Definition 7 (TCCR hashing for exponential correlation with auxiliary powers over $\mathbb{G}$)**.** Given a security parameter λ, let $(\mathbb{G}, p, g, G) \coloneqq \mathsf{GrpGen}^*(1^\lambda)$. Let $\mathsf{H} = \{\mathsf{H}_\lambda\}_{\lambda\in\mathbb{N}}$ be a family of hash functions $\mathsf{H}_\lambda : \mathbb{G} \to \mathbb{Z}_{p-1}$. Given $\Delta \in \mathbb{Z}_{p-1}$ and $h \in \mathbb{G}$, let $\mathcal{O}_{\mathsf{H},h,\Delta}$ denote the oracle that, on input $(x, y, z) \in \mathbb{Z}_{p-1} \times \{0, 1\}^* \times \mathbb{Z}_{p-1}$, returns $\mathsf{H}(h^x, y) + z \cdot \Delta \bmod p - 1$. We say that a list of queries to $\mathcal{O}_{\mathsf{H},h,\Delta}$ is *admissible* if for every pair of queries $(x, y, z), (x', y', z')$, if $(x, y) = (x', y')$, then $z = z'$.

Given a polynomial bound $B = B(\lambda)$, we say that the hash family $\mathsf{H} = \{\mathsf{H}_\lambda\}_{\lambda\in\mathbb{N}}$ is a *TCCR hash for exponential correlation with* $2B - 1$ *auxiliary powers over* $\mathbb{G}$, denoted $(B, \mathbb{G})$-ap-eTCCR, if for every probabilistic polynomial-time adversary $\mathcal{A}$ that makes admissible queries, it holds that

$$\mathsf{Adv}^{\text{ap-eTCCR}}_{\mathcal{A},1^\lambda} \coloneqq \left|\Pr[\mathcal{A}^{\mathsf{H},\mathcal{O}_{\mathsf{H},h,\Delta}}((h_i)_{i\in[\pm B]\setminus\{0\}}) = 1] - \Pr[\mathcal{A}^{\mathsf{H},\mathcal{R}}((h_i)_{i\in[\pm B]\setminus\{0\}}) = 1]\right|$$

is negligible, where the probability is taken over the random choice of $(\Delta, h) \leftarrow\!\$\, \mathbb{Z}_{p-1} \times \mathbb{G}$ and of a random oracle $\mathcal{R} : \mathbb{Z}_{p-1} \times \{0,1\}^* \times \mathbb{Z}_{p-1} \to \mathbb{Z}_{p-1}$, and where the h_i's are defined as follows: set $\alpha \coloneqq G^\Delta \bmod p$ and define $h_i \coloneqq h^{\alpha^i}$ for all $i \in [\pm B]$.

The theorem below shows that, when modeling H as a random oracle and $\mathbb{G}$ as a generic group, the assumption of Definition 7 holds unconditionally:

Theorem 5. *Let $\mathbb{G}$ be modeled via a generic group oracle $\mathcal{O}_\mathbb{G}$ (Definition 2), H be modeled as a random oracle, and $\mathcal{A}$ be an adversary making at most $Q_\mathbb{G}$ queries to $\mathcal{O}_\mathbb{G}$, Q_H queries to H, and Q_O queries to $\mathcal{O}_{\mathsf{H},h,\Delta}$. Then*

$$\mathsf{Adv}^{\mathsf{ap\text{-}eTCCR}}_{\mathcal{A},1^\lambda} \leq \frac{(Q_\mathbb{G} + Q_O)^2 + (Q_O + 1) \cdot (2B \cdot (Q_\mathbb{G} + 1)^2 + (Q_O - 1) \cdot Q_\mathsf{H}/2)}{p}$$

We note that since a random oracle can be implemented unconditionally in Shoup's GGM, one can choose a H such that the resulting assumption can be proven solely in Shoup's GGM.

Proof. We prove Theorem 5 via a sequence of hybrids.

Hybrid_0**.** This is the real game (in the GGM): the experiment samples $\Delta \leftarrow\!\$\, \mathbb{Z}_{p-1}$ and $h \leftarrow\!\$\, \mathbf{G}$. It sets $\alpha \coloneqq G^\Delta \bmod p$ and defined $h_i \coloneqq \alpha^i \bullet h$ for all $i \in [\pm B]$. We let the experiment sample the random oracle H lazily upon queries of either $\mathcal{A}$ or $\mathcal{O}_{\mathsf{H},h,\Delta}$ to H. On input $(h_i)_{i\in[\pm B]\setminus\{0\}}$, $\mathcal{A}$ makes queries to $\mathcal{O}_\mathbb{G}$, H, and $\mathcal{O}_{\mathsf{H},h,\Delta}$, and returns a bit b. We let $\Pr[\mathsf{Hybrid}_0]$ denote the probability that $b = 1$ in this hybrid.

Hybrid_1**.** In this game, we modify the sampling of σ (see Definition 2). Instead of sampling an injective mapping σ before the game, the experiment maintains a list L of pairs $(x, \sigma(x))$. Upon receiving any query $\mathsf{Encode}(x)$ from either $\mathcal{A}$ or $\mathcal{O}_{\mathsf{H},h,\Delta}$, if L contains a pair $(x, \sigma(x))$, the experiment returns $\sigma(x)$. Otherwise, the experiment samples $y \leftarrow\!\$\, S$, adds (x, y) to L, and returns y. Conditioned on no collision occurring during this process, the mapping σ sampled this way is a uniform injective mapping, and this hybrid is perfectly indistinguishable from the previous one. Bounding the probability of collisions (pairs (x, x') with $\sigma(x) = \sigma(x')$), we have:

$$|\Pr[\mathsf{Hybrid}_1] - \Pr[\mathsf{Hybrid}_0]| \leq \frac{(Q_\mathbb{G} + Q_O)^2}{p}.$$

Hybrid_2**.** In this game, we slightly modify the sampling of the h_i's. Namely, we let the experiment sample $h_{-B} \leftarrow\!\$\, \mathbf{G}$ and define $h_i \coloneqq (\alpha^{B+i}) \bullet h_{-B}$ for $i = -B + 1$ to B. For convenience, we rename h_{-B} as f and h_i as f_{B+i}, so that $f_i = \alpha^i \bullet f$ for $i = 0$ to $2B$. Note that $h = f_B$ is the secret group element. This change is purely syntactical, and we have

$$\Pr[\mathsf{Hybrid}_2] = \Pr[\mathsf{Hybrid}_1].$$

Hybrid_3**.** In this game, we delay the choice of Δ (and α) until the first query of $\mathcal{A}$ to $\mathcal{O}_{\mathsf{H},h,\Delta}$. This is achieved by simulating the generic group via symbolic computations. At the start of the game, the experiment defines a formal variable X. It samples $(f_0, f_1, \cdots, f_{B-1}, f_{B+1}, \cdots, f_{2B}) \leftarrow\!\$\, \mathbf{G}^{2B+1}$ (note that f_B is not sampled) and stores the pairs (X^i, f_i) for all $i \neq B$, as well as the pair $(1, g)$ (recall that g is defined as $\mathsf{Encode}(1)$) in the list L. Then, it proceeds as follows:

Before the first query to $\mathcal{O}_{\mathsf{H},h,\Delta}$. The experiment maintains the invariant that L is a list of pairs $(P, u \in \mathbf{G})$ where P is a multivariate polynomial (with coefficients over $\mathbb{Z}_p$) of the form $P'(X, X_1, \cdots, X_n) = \sum_{i=0}^{2B} c_i \cdot X^i + \mathsf{Lin}(X_1, \cdots, X_n)$, where $c_B = 0$ and Lin is a linear multivariate polynomial.

- Upon receiving a query $\mathsf{Encode}(x)$, it behaves as in Hybrid_2: if L contains a pair $(x, \sigma(x))$, it returns $\sigma(x)$; else, it returns $y \leftarrow\$ \mathbf{G}$ and adds (x, y) to L.
- Upon receiving a query $\mathsf{Add}(a, b, u, v)$ with $a, b \in \mathbb{Z}_p$ and $u, v \in \mathbf{G}$, if L does not contain a pair (P_u, u) (resp. a pair (P_v, v)), it defines a new formal variable X_u and adds $(P_u \coloneqq X_u, u)$ to L (resp. it defines X_v and stores $(P_v \coloneqq X_v, v)$). If L contains a pair $(a \cdot P_u + b \cdot P_v, w)$, it returns w. Else, it returns $w \leftarrow\$ \mathbf{G}$ and adds $(a \cdot P_u + b \cdot P_v, w)$ to L.

Upon receiving the first query to $O_{\mathsf{H},h,\Delta}$. Upon receiving a query (x, y, z) to $O_{\mathsf{H},h,\Delta}$, the experiment samples $\Delta \leftarrow\$ \mathbb{Z}_{p-1}$ and sets $\alpha \coloneqq G^{\Delta} \bmod p$. Then, it does the following:

Storing $x \bullet f_B$. The experiment samples $v \leftarrow\$ \mathbf{G}$ and adds $(x \cdot X^B, v)$ to L.

Collapsing L. Let $X, X_1, \cdots, X_n$ denote all formal variables appearing in L (where $n \leq Q_{\mathbb{G}}$). The experiment samples $(x_1, \cdots, x_n) \leftarrow\$ \mathbb{Z}_p^n$ and substitutes $P_u(X, X_1, \cdots, X_n)$ with $P_u(\alpha, x_1, \cdots, x_n)$ across all pairs (P_u, u) in L. If a collapse happens, *i.e.* $P_u(\alpha, x_1, \cdots, x_n) = P_v(\alpha, x_1, \cdots, x_n)$ for two distinct polynomials $P_u, P_v \in L$, it raises a flag Fail_α.

Sampling the answer. If a previous query to H of the form $\mathsf{H}(v, y)$ had been made, it raises a flag $\mathsf{Fail}_{\mathsf{H}}$ and returns this answer. Else, it sample $a \leftarrow\$ \mathbb{Z}_{p-1}$, returns a, and stores $\mathsf{H}(v, y) \coloneqq a - z \cdot \Delta \bmod p - 1$.

For all subsequent queries. The experiment behaves as in Hybrid_2.

We prove the following:

Lemma 6.

$$|\Pr[\mathsf{Hybrid}_3] - \Pr[\mathsf{Hybrid}_2]| \leq \frac{2B \cdot (Q_{\mathbb{G}} + 1)^2 + Q_{\mathsf{H}}}{p}.$$

Proof. First, we observe that conditioned on Hybrid_3 not failing (*i.e.* not raising a flag Fail_α or $\mathsf{Fail}_{\mathsf{H}}$), Hybrid_2 and Hybrid_3 are perfectly indistinguishable: all answers of the experiment to queries to $O_{\mathbb{G}}, \mathsf{H}$ before the first query to $O_{\mathsf{H},h,\Delta}$ are distributed identically to Hybrid_2 conditioned on Fail_α not being raised, and the distribution of $\mathsf{H}(v, y) \coloneqq a - z \cdot \Delta \bmod p - 1$ for a uniform a (conditioned on $\mathsf{Fail}_{\mathsf{H}}$ not being raised) is identical to the distribution of $\mathsf{H}(v, y)$ in Hybrid_2. It remains to bound the probability of the failure events.

First, note that $\alpha = G^{\Delta} \bmod p$ for $\Delta \leftarrow\$ \mathbb{Z}_{p-1}$ is uniformly distributed over $\mathbb{Z}_p$. Fix two distinct polynomials P_u, P_v from L. Write

$$\begin{aligned} P_u(X, X_1, \cdots, X_n) &= P'_u(X) + \mathsf{Lin}_u(X_1, \cdots, X_n), \\ P_v(X, X_1, \cdots, X_n) &= P'_v(X) d_i \cdot X^i + \mathsf{Lin}_v(X_1, \cdots, X_n), \end{aligned}$$

where P'_u, P'_v are univariate polynomials of degree at most $2B$. Then, let $P' \coloneqq P'_u - P'_v$ and $\mathsf{Lin} \coloneqq \mathsf{Lin}_v - \mathsf{Lin}_u$. We have

$$\begin{aligned} & P_u(\alpha, x_1, \cdots, x_n) = P_v(\alpha, x_1, \cdots, x_n) \\ \iff\ & P'(\alpha) = \mathsf{Lin}(x_1, \cdots, x_n). \end{aligned}$$

Now, two cases can occur: either $P' = 0$ (as a polynomial), in which case $\mathsf{Lin} \neq 0$ (as $P_u \neq P_v$) and the probability (over a random choice of $(x_1, \cdots, x_n)$) that $\mathsf{Lin}(x_1, \cdots, x_n) = 0$ is exactly $1/p$. Or $P' \neq 0$, in which case P' has at most $2B$ roots (as it has degree at most $2B$) and for any $(x_1, \cdots, x_n)$, the probability (over the random choice of α) that $P'(\alpha) = \mathsf{Lin}(x_1, \cdots, x_n)$ is at most $2B/p$. Given that L contains at most $Q_{\mathbb{G}} + 1$

entries (one for each query of $\mathcal{A}$ to $\mathbb{G}$, and the pair $(x \cdot X^B, v)$), it follows from a straightforward union bound that $\Pr[\mathsf{Fail}_\alpha] \leq (Q_\mathbb{G} + 1)^2 \cdot 2B/p$. For Fail_H, observe that v is sampled uniformly at random from $\mathbf{G}$, and the probability that any query of $\mathcal{A}$ to H is of the form (v, y) is therefore at most Q_H/p. This concludes the proof. ■

$\mathsf{Hybrid}_{3.2}$. This hybrid is defined identically to Hybrid_3, except that it uses symbolic computations before the *second* query to $\mathcal{O}_{\mathsf{H},h,\Delta}$. Upon receiving the first query (x_1, y_1, z_1) to $\mathcal{O}_{\mathsf{H},h,\Delta}$, it does the following:

- It samples $v_1 \leftarrow\$ \mathbf{G}$ and adds $(x_1 \cdot X^B, v_1)$ to L.
- It returns $a_1 \leftarrow\$ \mathbb{Z}_{p-1}$ and proceeds with the symbolic emulation of the generic group as before.

Upon receiving the *second* query (x_2, y_2, z_2) to $\mathcal{O}_{\mathsf{H},h,\Delta}$, it samples $\Delta \leftarrow\$ \mathbb{Z}_{p-1}$ and executes the steps **storing** $x_2 \bullet f_B$, **collapsing** L, and **sampling the answers** from Hybrid_3, with the following modification to the first and last steps:

- In the first step, if $x_2 = x_1$, it simply sets $v_2 = v_1$.
- In the last step, if $(x_1, y_1) = (x_2, y_2)$, it sets $a_2 \coloneqq a_1$ (note that the security game forces $z_1 = z_2$ in this case, as $\mathcal{A}$ is restricted to making admissible queries).

It behaves as Hybrid_2 (and Hybrid_3) for all subsequent queries. Until (and including) the first query to $\mathcal{O}_{\mathsf{H},h,\Delta}$, the answers of $\mathsf{Hybrid}_{3.2}$ are distributed identically to that of Hybrid_3 (in particular, the answer a_1 to the first query is sampled uniformly in both hybrids). The event Fail_α is defined as in Hybrid_3. We modify the definition of the event Fail_H as follows:

Fail_H: the experiment raises Fail_H if the list of all queries made by $\mathcal{A}$ to H contains either a tuple (v_1, y') or (v_2, y') *for any* y'.

Due to the symbolic evaluation, the answers all queries of $\mathcal{A}$ to $\mathcal{O}_\mathbb{G}$ up to the second query to $\mathcal{O}_{\mathsf{H},h,\Delta}$ are totally independent of v_1, v_2, since all queries of $\mathcal{A}$ are with respect to tuples (P_u, u) where the coefficient of X^B in P_u is 0 (since $\mathcal{A}$ is not given access to the tuple (X^B, f)). Hence, v_1 and v_2 are perfectly random and independent from $\mathcal{A}$'s view (with the exception that $v_1 = v_2$ when $x_1 = x_2$), and we can bound Fail_H by $2Q_\mathsf{H}/p$.

Conditioned on Fail_α and Fail_H not being raised after the second $\mathcal{O}_{\mathsf{H},h,\Delta}$ query, Hybrid_3 and $\mathsf{Hybrid}_{3.2}$ are perfectly indistinguishable, and the probability of the failure events can be bounded by the same analysis as Lemma 24. Note that when Fail_H is not raised, the answer a_2 to the second query is either equal to a_1, or uniformly random (and independent of Δ). We get

$$|\Pr[\mathsf{Hybrid}_{3.2}] - \Pr[\mathsf{Hybrid}_3]| \leq \frac{2B \cdot (Q_\mathbb{G} + 1)^2 + 2Q_\mathsf{H}}{p}.$$

$\mathsf{Hybrid}_{3.i}$. For $i = 3$ to $Q_\mathcal{O}$, we let $\mathsf{Hybrid}_{3.i}$ be defined as follows: it uses symbolic evaluation up to the i-th query to $\mathcal{O}_{\mathsf{H},h,\Delta}$. For $j = 1$ to i, it answers queries to $\mathcal{O}_{\mathsf{H},h,\Delta}$ as follows: if $(x_j, y_j, z_j) = (x_k, y_k, z_k)$ for some $k < j$, it sets $v_j \coloneqq v_k$ and returns a_k. Else, it samples $v_j \leftarrow\$ \mathbf{G}$, adds $(x_j \cdot X^B, v_j)$ to L, and returns $a_j \leftarrow\$ \mathbb{Z}_{p-1}$. After the **collapsing** L and **sampling the answers** steps, it raises a flag Fail_α if a collision occured during collapsing, and a flag $\mathsf{Fail}_{\mathsf{H}}$ if the list of $\mathcal{A}$'s queries to H contain a pair (v_j, y') for any y'. It behaves as $\mathsf{Hybrid}_{3.(i-1)}$ afterwards. The same analysis as before shows

$$|\Pr[\mathsf{Hybrid}_{3.i}] - \Pr[\mathsf{Hybrid}_{3.(i-1)}]| \leq \frac{2B \cdot (Q_{\mathbb{G}} + 1)^2 + j \cdot Q_{\mathsf{H}}}{p}.$$

Hybrid_4**.** Observe that from $\mathsf{Hybrid}_{3.Q_O}$, the oracle $O_{\mathsf{H},h,\Delta}$ behaves identically to a random oracle. In Hybrid_4, we replace $O_{\mathsf{H},h,\Delta}$ with a random oracle $\mathcal{R}$ (this is just a syntactic change at this step). In addition, the experiment uses the true generic oracle $O_{\mathbb{G}}$ instead of using symbolic evaluation. By the same argument as before, this hybrid is indistinguishable from the previous one unless the event Fail_α is raised, hence

$$|\Pr[\mathsf{Hybrid}_4] - \Pr[\mathsf{Hybrid}_{3.Q_O}]| \leq \frac{2B \cdot (Q_{\mathbb{G}} + 1)^2}{p},$$

which concludes the proof. ■

4.2 Leveled Circular Correlation-Robustness

We will also consider a *leveled* version of the assumption. In this version, there are d levels. For each $j \leq d+1$, the security experiment samples $(\Delta_j, h_j) \leftarrow\$\ \mathbb{Z}_{p-1} \times \mathbb{G}$ and sets $\alpha_j \coloneqq G^{\Delta_j} \bmod p$ and $h_{j,i} \coloneqq h_j^{\alpha_j^i}$ for all $i \in [\pm B]$. The adversary $\mathcal{A}$ is restricted to make at most d adaptive batches of queries to the oracle, and the oracle answer the j-th batch of queries with answers of the form $\mathsf{H}(h_j^x, y) + z \cdot \Delta_{j+1} \bmod p-1$.

Definition 8 (Leveled TCCR hashing for exponential correlation with auxiliary powers over $\mathbb{G}$)**.** Given a security parameter λ, let $(\mathbb{G}, p, g, G) \coloneqq \mathsf{GrpGen}^*(1^\lambda)$. Let $\mathsf{H} = \{\mathsf{H}_\lambda\}_{\lambda \in \mathbb{N}}$ be a family of hash functions $\mathsf{H}_\lambda : \mathbb{G} \longrightarrow \mathbb{Z}_{p-1}$. Fix a (polynomial) depth parameter $d = d(\lambda)$. Let $O_{\mathsf{H},\mathbf{h},\mathbf{\Delta}}$ denote the following stateful oracle:

$O_{\mathsf{H},\mathbf{h},\mathbf{\Delta}}$

Initialize $\eta \coloneqq 0$

parse $\mathbf{h} = (h_1, \ldots, h_{d+1}) \in \mathbb{G}^{d+1}$

parse $\mathbf{\Delta} = (\Delta_1, \ldots, \Delta_{d+1}) \in (\mathbb{Z}_{p-1})^{d+1}$

On input $\mathcal{S} = \left\{(x_j, y_j, z_j)\right\}_{j=1}^{|\mathcal{S}|} \subset \mathbb{Z}_{p-1} \times \{0,1\}^* \times \mathbb{Z}_{p-1}$:

  $\eta \coloneqq \eta + 1$

  if $\eta \leq d$, **return** $\left(\mathsf{H}(h_\eta^{x_j}, y_j) + z_j \cdot \Delta_{\eta+1} \bmod p-1\right)_{j=1}^{|\mathcal{S}|}$

Given a polynomial bound $B = B(\lambda)$, we say that the hash family $\mathsf{H} = \{\mathsf{H}_\lambda\}_{\lambda \in \mathbb{N}}$ is a *d-leveled TCCR hash for exponential correlation with* $2B-1$ *auxiliary powers per level over* $\mathbb{G}$ if for every probabilistic polynomial-time adversary $\mathcal{A}$, it holds that

$$\left|\Pr\left[\mathcal{A}^{\mathsf{H},O_{\mathsf{H},\mathbf{h},\mathbf{\Delta}}}\left((h_{i,j})_{\substack{i \leq d+1 \\ j \in [\pm B]^*}}\right)\right] - \Pr\left[\mathcal{A}^{\mathsf{H},\mathcal{R}}\left((h_{i,j})_{\substack{i \leq d+1 \\ j \in [\pm B]^*}}\right)\right]\right| \leq \mathsf{negl}(\lambda),$$

where the probability is taken over the random choice of $(\Delta_i, h_i) \leftarrow\$\ \mathbb{Z}_{p-1} \times \mathbb{G}$ for $i \in [d]$ and of a random oracle $\mathcal{R}$ that on input a set S of tuples $(x, y, z) \in \mathbb{Z}_{p-1} \times \{0,1\}^* \times \mathbb{Z}_{p-1}$, outputs $(r_1, \cdots, r_{|S|}) \leftarrow\$\ (\mathbb{Z}_{p-1})^{|S|}$, and where the $h_{i,j}$'s are defined as follows: for $i \in [d]$, set $\alpha_i \coloneqq G^{\Delta_i} \bmod p$ and define $h_{i,j} \coloneqq h_i^{\alpha_i^j}$ for all $j \in [\pm B]^*$.

While the above definition is somewhat involved, the theorem below shows that under the power-DDH assumption, any tweakable correlation-robust hash for the exponential correlation over $\mathbb{G}$ is also a leveled TCCR hash for the exponential correlation with auxiliary powers over $\mathbb{G}$.

Theorem 7. *Let* $(\mathbb{G}, p, g, G) \coloneqq \mathsf{GrpGen}^*(1^\lambda)$*, let* H *be a family of TCR hash functions for exponential correlation over* $\mathbb{G}$ *(Definition 6), and let* $B = B(\lambda)$ *be a polynomial bound. Then if the* B*-power-DDH assumption (Definition 4) holds with respect to* GrpGen^**, for all polynomial* $d = d(\lambda)$*,* H *is a* d*-leveled TCCR hash for exponential correlation with* $2B - 1$ *auxiliary powers per level over* $\mathbb{G}$ *(Definition 8).*

Proof. Consider an arbitrary PPT adversary $\mathcal{A}$ against the d-leveled TCCR security of H. We will show that $\mathcal{A}$ has negligible advantage by a straightforward hybrid argument.

Let $\mathcal{O}^{(i)}_{\mathsf{H},\mathbf{h},\Delta}$ be a stateful oracle that is identical to $\mathcal{O}_{\mathsf{H},\mathbf{h},\Delta}$ except for the following: for all queries $\mathcal{S}$, it returns $|\mathcal{S}|$ uniformly random elements from $\mathbb{Z}_{p-1}$ if $\eta \le i$; otherwise, it responds as in $\mathcal{O}_{\mathsf{H},\mathbf{h},\Delta}$. Consider a sequence of hybrids

$$\mathsf{Hybrid}_{0,0}\ ,\ \mathsf{Hybrid}_{0,1}\ ,\ \ldots\ ,\ \mathsf{Hybrid}_{i,0}\ ,\ \mathsf{Hybrid}_{i,1}\ ,\ \mathsf{Hybrid}_{i+1,0}\ ,\ \ldots\ ,\ \mathsf{Hybrid}_{d,1}$$

defined as follows.

- $\mathsf{Hybrid}_{0,0}$: This hybrid denotes $\mathcal{A}^{\mathsf{H},\mathcal{O}_{\mathsf{H},\mathbf{h},\Delta}}((h_{i,j})_{i,j})$ as described in Definition 8.

- $\mathsf{Hybrid}_{i,1}$: For each $i \in [d]$, $\mathsf{Hybrid}_{i,1}$ denotes $\mathcal{A}^{\mathsf{H},\mathcal{O}^{(i)}_{\mathsf{H},\mathbf{f}_i,\Delta}}((h_{i,j})_{i,j})$ where
$$\mathbf{f}_i = (f_1, \ldots, f_{i+1}, h_{i+2}, \ldots, h_{d+1}),$$
$(f_1, \ldots, f_{i+1}) \xleftarrow{\$} \mathbb{G}^{i+1}$, and $(\mathbf{h}, \Delta, (h_{i,j})_{i,j})$ are distributed as in $\mathsf{Hybrid}_{i,0}$. In other words, $\mathsf{Hybrid}_{i,1}$ corresponds to the output of the adversary when the oracle returns uniformly random elements from $\mathbb{Z}_{p-1}$ for the first i queries and computes the output as in $\mathcal{O}_{\mathsf{H},\mathbf{h},\Delta}$ for the last $d - i - 1$ queries. In the $(i+1)$-th query, it returns $\mathsf{H}(f_{i+1}^x, y) + z \cdot \Delta_{i+1} \bmod p - 1$ for every (x, y, z) in the queried input $\mathcal{S}$.

- $\mathsf{Hybrid}_{i,0}$: For each $i \in \{1, \ldots, d\}$, $\mathsf{Hybrid}_{i,1}$ denotes $\mathcal{A}^{\mathsf{H},\mathcal{O}^{(i)}_{\mathsf{H},\mathbf{f}_i,\Delta}}((h_{i,j})_{i,j})$ where
$$\mathbf{f}_i = (f_1, \ldots, f_i, h_{i+1}, \ldots, h_{d+1}),$$
$(f_1, \ldots, f_i) \xleftarrow{\$} \mathbb{G}^i$, and $(\mathbf{h}, \Delta, (h_{i,j})_{i,j})$ are distributed as in $\mathsf{Hybrid}_{i-1,1}$. In other words, $\mathsf{Hybrid}_{i,0}$ corresponds to the output of the adversary when the oracle returns uniformly random elements from $\mathbb{Z}_{p-1}$ for the first i queries and computes the output as in $\mathcal{O}_{\mathsf{H},\mathbf{h},\Delta}$ for the last $d - i$ queries.

We next show that, in the sequence of hybrids above, each hybrid is indistinguishable from the next using the following two claims.

Claim. *If the* B*-power-DDH assumption holds with respect to* GrpGen^* *then for every* $i \in [d]$*,* $\mathsf{Hybrid}_{i,0} \overset{c}{\approx} \mathsf{Hybrid}_{i,1}$.

Proof. Note that Hybrid_0 is identical to $\mathcal{A}^{\mathsf{H},\mathcal{O}^{(0)}_{\mathsf{H},\mathbf{h},\Delta}}((h_{i,j})_{i,j})$ since the oracles $\mathcal{O}_{\mathsf{H},\mathbf{h},\Delta}$ and $\mathcal{O}^{(0)}_{\mathsf{H},\mathbf{h},\Delta}$ are equivalent in this case. Thus, for any $i \in [d]$, the only difference between $\mathsf{Hybrid}_{i,0}$ and $\mathsf{Hybrid}_{i,1}$ is that the output of the $(i+1)$-th query is computed as $\mathsf{H}(h_{i+1}^x, y) + z \cdot \Delta_{i+1} \bmod p - 1$ in $\mathsf{Hybrid}_{i,0}$ and as $\mathsf{H}(f_{i+1}^x, y) + z \cdot \Delta_{i+1} \bmod p - 1$ in $\mathsf{Hybrid}_{i,1}$ for every $(x, y, z) \in \mathcal{S}$. However, under the B-power-DDH assumption, the uniformly random f_{i+1} is indistinguishable from h_{i+1} given $(h_{i,j})_j$. It then immediately follows that $\mathsf{Hybrid}_{i,0} \overset{c}{\approx} \mathsf{Hybrid}_{i,1}$. $\square$

Claim. *If* H *is a TCR hash for exponential correlation over* $\mathbb{G}$ *then for every* $i \in [d-1]$, $\mathsf{Hybrid}_{i,1} \overset{c}{\approx} \mathsf{Hybrid}_{i+1,0}$.

Proof. For any $i \in [d-1]$, the only difference between $\mathsf{Hybrid}_{i,1}$ and $\mathsf{Hybrid}_{i+1,0}$ is that the output of the $(i+1)$-th query is computed as $\mathsf{H}(f_{i+1}^x, y) + z \cdot \Delta_{i+1} \bmod p-1$ for a random group element f_{i+1} in $\mathsf{Hybrid}_{i,1}$ while in $\mathsf{Hybrid}_{i+1,0}$, the $(i+1)$-th query's output is a set of uniformly random elements from $\mathbb{Z}_{p-1}$. However, if H is a TCR for exponential correlation then $\mathsf{H}(f_{i+1}^x, y)$ and hence the output of the oracle in $\mathsf{Hybrid}_{i,1}$ are indistinguishable from random values in $\mathbb{Z}_{p-1}$. This implies that $\mathsf{Hybrid}_{i,1}$ is indistinguishable from $\mathsf{Hybrid}_{i+1,0}$. □

Since d is polynomial in the security parameter, it follows from the above claims that $\mathsf{Hybrid}_{0,0} \overset{c}{\approx} \mathsf{Hybrid}_{d,1}$. Moreover, observe that the oracle provided to the adversary in $\mathsf{Hybrid}_{d,1}$ is identical to $\mathcal{R}$, as defined in Definition 8, since it does not use $\mathbf{h} = (h_1, \ldots, h_{d+1})$ and returns uniformly random elements from $\mathbb{Z}_{p-1}$ for all d queries. Since $\mathsf{Hybrid}_{0,0} \overset{c}{\approx} \mathsf{Hybrid}_{d,1}$, we have

$$\left|\Pr\left[\mathcal{A}^{\mathsf{H},O_{\mathsf{H},\mathbf{h},\Delta}}\left((h_{i,j})_{\substack{i\le d+1\\ j\in[\pm B]^*}}\right)\right] - \Pr\left[\mathcal{A}^{\mathsf{H},\mathcal{R}}\left((h_{i,j})_{\substack{i\le d+1\\ j\in[\pm B]^*}}\right)\right]\right| \le \mathsf{negl}(\lambda).$$

■

5 Building Blocks

5.1 Embedding Polynomials into $\mathbb{Z}_p$

Given a degree-m polynomial $a \in \mathbb{F}_2[X]$, we let $(a[i])_{i\in[m]} \in \mathbb{F}_2^{m+1}$ denote the list of its coefficients. In this work, we manipulate embeddings of polynomials over $\mathbb{Z}_p$. Given parameters $N \ge 2$ and $m \in \mathbb{N}$, let $\mathsf{B}(N,m) \coloneqq (N^{m+1}-1)/(N-1)$. We encode $a \in \mathbb{F}_2[X]$ into $\tilde{a} \in [\mathsf{B}(N, \deg(a))]$ by interpreting a as an element of $\mathbb{Z}[X]$ and computing $\tilde{a} \coloneqq a(N)$. When $|\mathsf{B}(N, \deg(a))| < p$ (looking ahead, this will always hold in our constructions), we slightly abuse the notation and view $\tilde{a}$ as an element of $\mathbb{Z}_p$ via the natural embedding from $[p]$ to $\mathbb{Z}_p$. For any $N \ge 2, m \in \mathbb{N}$, we let $\mathsf{I}_{N,m} \coloneqq \{\tilde{a} \in \mathbb{Z}_p : \exists a \in \mathbb{F}_2[X], \deg(a) \le m, \tilde{a} = a(N)\}$ denote the subset of all valid embeddings of degree-at-most-m polynomials to $\mathbb{Z}_p$.

Procedures. We introduce below two procedures that are used to manipulate embeddings of polynomials into $\mathbb{Z}_p$. In the procedures below, $\mathbb{Z}_p$ is identified with the subset of integers $\{0, \cdots, p-1\}$.

- $\mathsf{toPoly}_N(\tilde{a})$: On input $\tilde{a} \in \mathbb{Z}_p$, parse $\tilde{a} = \sum_{i=0}^m a[i] \cdot N^i$ (the N-ary decomposition of $\tilde{a}$) and return $a \coloneqq \sum_{i=0}^m a[i] \cdot X^i \in \mathbb{Z}[X]$, a degree-$m$ integer polynomial with coefficients in $\{0, N-1\}$. When N is clear from the context, we write $a \coloneqq \mathsf{toPoly}(\tilde{a})$.
- $\mathsf{Mod}_N(\tilde{a}, M)$: On input $\tilde{a} \in \mathbb{Z}_p$ and a modulus M, compute $b \coloneqq [\mathsf{toPoly}_N(\tilde{a}) \bmod M]$ and return $\tilde{b} \coloneqq b(N)$. When N is clear from the context, we write $\tilde{b} \coloneqq \mathsf{Mod}(\tilde{a}, M)$.

Above, we let all procedures take variable length inputs: the degree m is inferred from the input and N. Furthermore, we let Mod take as second input either an integer $M \in \mathbb{N}$ (in which case $[a \bmod M] = b \in \mathbb{Z}_M[X]$) or a polynomial $M \in Z[X]$ (in which case $[a \bmod M] = b \in \mathbb{Z}[X]/M$). We slightly abuse the notation and write, given an integer n and a polynomial P, $\mathsf{Mod}_N(\tilde{a}, (n, P))$ to denote $\mathsf{Mod}_N(\mathsf{Mod}_N(\tilde{a}, n), P)$ (that is, the function that computes the polynomial associated to $\tilde{a}$, reduces it modulo n and P, and embeds back the polynomial in $\mathbb{Z}_n[X]/P$ over the integers).

Input perturbation. We introduce a procedure that *perturbates* an element $a \in \mathbb{F}_2[X]$ in the following sense: the perturbation maps a to a polynomial $a' \in \mathbb{Z}[X]$ such that $a(N) = \mathsf{Mod}_N(a'(N), 2)$ (*i.e.*, a' preserves the parity of the value of the coefficients of a, and its coefficients do not overflow N), but a can be masked by a random $\mathbb{Z}[X]$-polynomial *with small coefficients*.

- $\mathsf{Pert}_c(a)$: On input $a \in \mathbb{F}_2[X]$, sample c uniformly random shares $(a_1, \cdots, a_c)$ of a over $\mathbb{F}_2[X; \deg(a)]$. Interpret each a_i as a polynomial over $\mathbb{Z}[X]$ and return $a' = \sum_{i=1}^{c} a_i$ (where the sum is computed over $\mathbb{Z}[X]$).

It is immediate to check that for any N such that $c < N$, it holds that $a(N) = \mathsf{Mod}_N(a'(N), 2)$. The following lemma shows that one can statistically hide a by masking $a' \leftarrow\!\!\$\ \mathsf{Pert}_c(a)$ with a carefully chosen element $r \in \mathbb{Z}[X]$ with *small* coefficients such that $\|r\|_\infty \leq c$. We first define the appropriate distribution over polynomials with small coefficients for integers m, c:

- $\mathsf{RandSum}_{m,c}$: Sample $(r_1, \cdots, r_c) \leftarrow\!\!\$\ \mathbb{F}_2[X; m]^c$ and set $r \coloneqq \sum_{i=1}^{c} r_i$, where the r_i's are interpreted as polynomials over $\mathbb{Z}[X; m]$. Output r.

- $\mathsf{RandSum}_{m,c}(N)$: Sample $r \leftarrow\!\!\$\ \mathsf{RandSum}_{m,c}$ and output $r(N)$.

It is clear from the definition that for any r in the support of $\mathsf{RandSum}_{m,c}$, we have $\|r\|_\infty \leq c$. Equipped with the above definition, we have the following lemma:

Lemma 8. *For any $m, c \in \mathbb{N}$ and $a \in \mathbb{F}_2[X; m]$, denote $\mathcal{D}^{(a)}_{m,c} \coloneqq \{a' + r : a' \leftarrow\!\!\$\ \mathsf{Pert}_c(a),\ r \leftarrow\!\!\$\ \mathsf{RandSum}_{m,c}\}$ and $\mathcal{D}_{m,c} \coloneqq \{a' + r : a', r \leftarrow\!\!\$\ \mathsf{RandSum}_{m,c}\}$. Then:*

$$\mathsf{SD}(\mathcal{D}^{(a)}_{m,c}, \mathcal{D}_{m,c}) \leq \frac{m}{2^c}.$$

Using the above procedures, integer encodings can support a limited number of homomorphic additions and multiplications. Concretely, we will use the following simple lemma:

Lemma 9. *Let $m, T, c, N \in \mathbb{N}$ be integers. Fix any tuple $(a_1, \cdots, a_T, b_1, \cdots, b_T) \in \mathbb{F}_2[X; m]^{2T}$ and let $b'_i \leftarrow\!\!\$\ \mathsf{Pert}_c(b_i)$ for $i = 1$ to T. Define*

$$v \coloneqq \sum_{i=1}^{T} a_i \cdot b_i, \qquad \tilde{v} \coloneqq \sum_{i=1}^{T} a_i(N) \cdot b'_i(N),$$

where the left sum is computed over $\mathbb{F}_2[X]$, and the right sum is computed over $\mathbb{N}$. Then, if $N > T \cdot c \cdot m$, it holds that

$$v = \mathsf{toPoly}_N\left(\mathsf{Mod}_N(\tilde{v}, 2)\right).$$

Mapping to $\mathsf{RandSum}$ **samples.** Given a set S, we write $\mathbb{U}_S$ to denote the uniform distribution over S.

Definition 9. We denote by $\mathsf{map} : \mathbb{Z}_{p-1} \to \mathbb{Z}_{p-1}$ a mapping such that $\mathsf{map}(\mathbb{U}_{\mathbb{Z}_{p-1}}) \approx_{\mathrm{c}} \mathsf{RandSum}_{m,c}(N)$.

Note that map is implicitly parametrized by (N, m, c); we write $\mathsf{map}_{N,m,c}$ when we want to make the dependency explicit. Concretely, constructing map is done as follows:

- On input $x \in \mathbb{Z}_{p-1}$, set $x' \coloneqq [x \bmod 2^\lambda]$, and parse x' as an element of $\{0,1\}^\lambda$. Note that over a uniform choice of $x \leftarrow\!\!\$\ \mathbb{Z}_{p-1}$, where p is a 2λ-bit prime, the induced distribution of x' is $2^{-\lambda}$-close to uniform over $\{0,1\}^\lambda$.

- Let $\mathsf{PRG} : \{0,1\}^\lambda \to \{0,1\}^{c \cdot (m+1)}$ denote a pseudorandom generator. Compute $y = \mathsf{PRG}(x')$ and parse the output as a c-tuple $(y_1, \cdots, y_c)$ of degree-m $\mathbb{F}_2$-polynomials $y_i \in \mathbb{F}_2[X; m]$.

- Run the $\mathsf{RandSum}_{m,c}(N)$ procedure: compute $r \coloneqq \sum_{i=1}^{c} y_i$ over $\mathbb{Z}[X; m]$ and output $r(N)$.

5.2 VOLE to OLE Procedure

We start by recalling a power-DDH-based puncturable pseudorandom function (PPRF) introduced recently in [CHHK25], which is at the heart of our construction (we do not recall the formal definition of PPRFs, as we will directly use the construction below rather than abstracting it out as a PPRF). Let $(\mathbb{G}, p, g, G) \coloneqq \mathsf{GrpGen}^*(1^\lambda)$. We let $\mathsf{H} = \{\mathsf{H}_\lambda : \mathbb{G} \times \{0,1\}^* \to \mathbb{Z}_{p-1}\}$ denote a family of hash functions over $\mathbb{G} \times \{0,1\}^*$.

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

$(\mathbb{G}, p, g, G) \coloneqq \mathsf{GrpGen}^*(1^\lambda)$
$(\Delta, h) \leftarrow\!\!\$\ \mathbb{Z}_{p-1} \times \mathbb{G}$
$\alpha \coloneqq G^\Delta \bmod p$
for $i \in [\pm B], h_i \coloneqq h^{\alpha^i}$
$\mathsf{mpk} \coloneqq \left(\mathbb{G}, p, G, (h_i)_{i \in [\pm B]^*}\right)$
$\mathsf{msk} \coloneqq (\mathsf{mpk}, \Delta, h_0)$
return $(\mathsf{mpk}, \mathsf{msk})$

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

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

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

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

$\underline{\mathsf{F}^\mathsf{H}(\mathsf{k}, x, \mathsf{salt}) \coloneqq \mathsf{F.Eval}^\mathsf{H}(\mathsf{k}, x, \mathsf{salt})}$

parse k as $(\mathsf{msk}, \mathsf{sk})$
parse $(h_i)_{i \in [\pm B]}$ from msk
return $\mathsf{H}\left(h_x^{\mathsf{sk}}, \mathsf{salt}\right)$

$\underline{\mathsf{pF}^\mathsf{H}(\mathsf{k}^*, z, x, \mathsf{salt}) \coloneqq \mathsf{F.PEval}^\mathsf{H}(\mathsf{k}^*, z, x, \mathsf{salt})}$

parse k^* as $(\mathsf{mpk}, \mathsf{psk})$
parse $(h_i)_{i \neq 0}$ from mpk
return $\mathsf{H}\left(h_{x-z}^{\mathsf{psk}}, \mathsf{salt}\right)$

Lemma 10 (Correctness of the PPRF). *Fix an arbitrary polynomial modulus B and let $\mathsf{H} = \{\mathsf{H}_\lambda : \mathbb{G} \times \{0,1\}^* \to \mathbb{Z}_{p-1}\}$ denote a family of hash functions over $\mathbb{G}$. Let $(\mathsf{mpk}, \mathsf{msk})$ be a master key pair in the support of $\mathsf{F.Setup}(1^\lambda, B)$. Parse msk as $(\mathsf{mpk}, \Delta, h_0)$. Fix any constraint $z \in \{0, \cdots, B\}$. Then for any $\Gamma_\mathsf{E}, \Gamma_\mathsf{G} \in \mathbb{Z}_{p-1}$ such that $\Gamma_\mathsf{E} - \Gamma_\mathsf{G} = \Delta \cdot z$, denoting $\mathsf{sk} \coloneqq \mathsf{F.KeyGen}(\mathsf{mpk}, \Gamma_\mathsf{G})$, $\mathsf{psk} \coloneqq \mathsf{F.Punct}(\mathsf{mpk}, \Gamma_\mathsf{E})$, $\mathsf{k} \coloneqq (\mathsf{msk}, \mathsf{sk})$, and $\mathsf{k}^* \coloneqq (\mathsf{mpk}, \mathsf{psk})$, it holds that for any input $x \in \{0, \cdots, B\} \setminus \{z\}$ and salt $\mathsf{salt} \in \{0,1\}^*$,*

$$\mathsf{F}^\mathsf{H}(\mathsf{k}, x, \mathsf{salt}) = \mathsf{pF}^\mathsf{H}(\mathsf{k}^*, z, x, \mathsf{salt}).$$

In [CHHK25], it is shown that if H is a tweakable correlation-robust hash function for exponential correlations over $\mathbb{G}$ and if the power-DDH assumption holds, then the above PPRF satisfies a strong notion of pseudorandomness: all evaluations $\mathsf{F}^\mathsf{H}(\mathsf{k}, z, \mathsf{salt})$ at the punctured point look pseudorandom given k^{**}, even when the adversary is allowed to request multiple pairs $(\mathsf{k}, \mathsf{k}^*)$ for the same master secret key msk. We do not formally state this security notion here: we will instead directly prove the security of our garbling scheme constructed from this primitive.

A VOLE-to-OLE procedure. Given a security parameter λ, fix integers $N = N(\lambda)$, $m = m(\lambda)$ such that $N^m = \mathsf{poly}(\lambda)$. Let $(\mathbb{G}, p, g, G) \coloneqq \mathsf{GrpGen}^*(1^\lambda)$ and $\mathsf{H} = \{\mathsf{H}_\lambda : \mathbb{G} \times \{0,1\}^* \to \mathbb{Z}_{p-1}\}$. Fix a master key pair $(\mathsf{mpk}, \mathsf{msk}) \leftarrow\!\!\$\ \mathsf{F.Setup}(1^\lambda, N^m)$ and parse msk as $(\mathsf{mpk}, \Delta, h_0)$. A core building block of our garbling scheme is a one-message (from G to E) VOLE-to-OLE protocol where G and E hold respective inputs $(v_\mathsf{G}, v_\mathsf{E}) \in \mathbb{Z}_{p-1} \times \mathsf{I}_{N,m}$ and shares of $\Delta \cdot v_\mathsf{E}$, and obtain as output shares of $v_\mathsf{G} v_\mathsf{E}$.

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

$\mathsf{sk} \coloneqq \mathsf{F.KeyGen}(\mathsf{msk}, \langle \Delta v_{\mathsf{E}} \rangle_{\mathsf{G}})$

$\mathsf{k} \coloneqq (\mathsf{msk}, \mathsf{sk})$

$\mathsf{shift} \coloneqq \sum_{x \in \mathsf{I}_{N,m}} \mathsf{F}^{\mathsf{H}}(\mathsf{k}, x, \mathsf{salt}) + v_{\mathsf{G}}$

$z_{\mathsf{G}} \coloneqq \sum_{x \in \mathsf{I}_{N,m}} x \cdot \mathsf{F}^{\mathsf{H}}(\mathsf{k}, x, \mathsf{salt})$

return $(\mathsf{shift}, z_{\mathsf{G}})$

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

$\mathsf{psk} \coloneqq \mathsf{F.Punct}(\mathsf{mpk}, \langle \Delta v_{\mathsf{E}} \rangle_{\mathsf{E}})$

$\mathsf{k}^* \coloneqq (\mathsf{mpk}, \mathsf{psk})$

$z_{\mathsf{E}} \coloneqq \sum_{x \in \mathsf{I}_{N,m}} (x - v_{\mathsf{E}}) \cdot \mathsf{pF}^{\mathsf{H}}(\mathsf{k}^*, v_{\mathsf{E}}, x, \mathsf{salt}) + v_{\mathsf{E}} \cdot \mathsf{shift}$

return z_{E}

We will consider two variants of the above procedure. In the first variant, the computation of $(\mathsf{shift}, z_{\mathsf{G}}, z_{\mathsf{E}})$ is done *over the integers*, while in the second variant, the computation is done *over* $\mathbb{Z}_{p-1}$. In the first case, $v_{\mathsf{G}}, v_{\mathsf{E}}$, and all outputs of H are treated as positive integers in $[p-2] \subset \mathbb{Z}$. We will write $\mathcal{R}\text{-}\mathsf{VtO}^{\mathsf{H}}_{\mathsf{G}}(\mathsf{msk}, v_{\mathsf{G}}, \langle \Delta v_{\mathsf{E}} \rangle_{\mathsf{G}}, \mathsf{salt})$ and $\mathcal{R}\text{-}\mathsf{VtO}^{\mathsf{H}}_{\mathsf{G}}(\mathsf{msk}, v_{\mathsf{G}}, \langle \Delta v_{\mathsf{E}} \rangle_{\mathsf{G}}, \mathsf{salt})$ to explicitly indicate that computation takes place over a ring $\mathcal{R} \in \{\mathbb{Z}, \mathbb{Z}_{p-1}\}$. By default, if no ring is indicated, the computation happens over $\mathbb{Z}_{p-1}$.

Remark 1. The VtO procedure is identical to the VOLE-to-OLE procedure introduced in [CHHK25], up to a minor difference: instead of summing over all possible inputs x between 0 and the bound $B = N^m$, we leverage the fact that the evaluator input v_{E} will always be the embedding $v(N) \in \mathsf{I}_{N,m}$ of some $\mathbb{F}_2$-polynomial $v \in \mathbb{F}_2[X; m]$. Since this is known to both parties, they can restrict the sums in the VtO procedures to be over valid embeddings, which reduces the number of terms in the sum from N^m to 2^m. This minor modification is crucial to control the growth of the size of the output share, which in turns influences the amount of garbling material per gate in our Boolean garbling scheme.

Correctness. The following lemma establishes perfect correctness of VtO:

Lemma 11. *For every $\mathcal{R} \in \{\mathbb{Z}, \mathbb{Z}_{p-1}\}$, every $(\mathsf{mpk}, \mathsf{msk})$ in the support of $\mathsf{F.Setup}$ with $\mathsf{msk} \coloneqq (\mathsf{mpk}, \Delta, h_0)$, every $v_{\mathsf{E}} \in \mathsf{I}_{N,m}$, every $v_{\mathsf{G}} \in \mathbb{Z}_{p-1}$, every $\mathsf{salt} \in \{0,1\}^*$, and every $\langle \Delta v_{\mathsf{E}} \rangle_{\mathsf{G}}, \langle \Delta v_{\mathsf{E}} \rangle_{\mathsf{E}}$ such that $\langle \Delta v_{\mathsf{E}} \rangle_{\mathsf{E}} - \langle \Delta v_{\mathsf{E}} \rangle_{\mathsf{G}} = \Delta_{\mathsf{E}}$, denoting $(\mathsf{shift}, z_{\mathsf{G}}) \coloneqq \mathcal{R}\text{-}\mathsf{VtO}^{\mathsf{H}}_{\mathsf{G}}(\mathsf{msk}, v_{\mathsf{G}}, \langle \Delta v_{\mathsf{E}} \rangle_{\mathsf{G}}, \mathsf{salt})$ and $z_{\mathsf{E}} \coloneqq \mathcal{R}\text{-}\mathsf{VtO}^{\mathsf{H}}_{\mathsf{E}}(\mathsf{mpk}, v_{\mathsf{E}}, \langle \Delta v_{\mathsf{E}} \rangle_{\mathsf{E}}, \mathsf{salt}, \mathsf{shift})$, it holds that $z_{\mathsf{E}} - z_{\mathsf{G}} = v_{\mathsf{E}} v_{\mathsf{G}}$ (over $\mathcal{R}$). Furthermore, if $\mathcal{R} = \mathbb{Z}$, then $z_{\mathsf{G}}, z_{\mathsf{E}}$ are positive integers.*

The last part of the lemma follows immediately from the definition of z_{G} and the first part of the lemma. The proof of the first part is a routine check:

$$\begin{aligned} z_{\mathsf{E}} &= \sum_{x \in \mathsf{I}_{N,m}} (x - v_{\mathsf{E}}) \cdot \mathsf{pF}^{\mathsf{H}}(\mathsf{k}^*, v_{\mathsf{E}}, x, \mathsf{salt}) + v_{\mathsf{E}} \cdot \mathsf{shift} \\ &= \sum_{x \in \mathsf{I}_{N,m}} (x - v_{\mathsf{E}}) \cdot \mathsf{F}^{\mathsf{H}}(\mathsf{k}, x, \mathsf{salt}) + v_{\mathsf{E}} \cdot \mathsf{shift} \qquad \triangleright \text{ via Lemma 10} \\ &= z_{\mathsf{G}} - v_{\mathsf{E}} \cdot (\mathsf{shift} - v_{\mathsf{G}}) + v_{\mathsf{E}} \cdot \mathsf{shift} = z_{\mathsf{G}} + v_{\mathsf{E}} v_{\mathsf{G}}. \end{aligned}$$

Simulating shifts using $\mathcal{O}_{\mathsf{H},h_0,\Delta}$. We define a simulator for VtO, that will be used in our security analysis, that simulates shift and z_{E} using only mpk, psk, and calls to the oracle $\mathcal{O}_{\mathsf{H},h,\Delta}$ from Definition 7. To handle the case where the garbler input v_{G} is an affine function of Δ (as this is the case in some parts of the garbling procedure), the simulator takes two additional inputs (a, b) such that $v_{\mathsf{G}} = a \cdot \Delta + b$. Intuitively, the $\Delta \cdot a$ term will be computed within $\mathcal{O}_{\mathsf{H},h_0,\Delta}$, and the simulator adds b to the output.

Some of our procedures also use $\mathsf{VtO}^{\mathsf{H}_0}$ with a hash function H_0 defined as $\mathsf{H}_0 \coloneqq \mathsf{map} \circ \mathsf{H}$ for a suitable mapping map. To handle this usecase, we also consider a variant of SimVtO, denoted Sim′VtO, that takes as input the map map. As a will always be equal to 0 when using this variant, we omit it from the inputs.

Fix integers $N = N(\lambda)$, $m = m(\lambda)$ such that $N^m = \mathrm{poly}(\lambda)$. Let $(\mathbb{G}, p, g, G) \coloneqq \mathsf{GrpGen}^*(1^\lambda)$ and $\mathsf{H} = \{\mathsf{H}_\lambda : \mathbb{G} \times \{0,1\}^* \to \mathbb{Z}_{p-1}\}_\lambda$. Fix a master key pair $(\mathsf{mpk}, \mathsf{msk}) \leftarrow\$ \mathsf{F.Setup}(1^\lambda, N^m)$ and parse msk as $(\mathsf{mpk}, \Delta, h_0)$. Let $\mathcal{O} \coloneqq \mathcal{O}_{\mathsf{H},h_0,\Delta}$ be the oracle defined in Definition 7.

$\underline{\mathsf{SimVtO}^{\mathsf{H},\mathcal{O}}(\mathsf{mpk}, v_\mathsf{E}, \langle\Delta v_\mathsf{E}\rangle_\mathsf{E}, a, b, \mathsf{salt})}$	$\underline{\mathsf{Sim'VtO}^{\mathsf{H},\mathcal{O}}(\mathsf{mpk}, v_\mathsf{E}, \langle\Delta v_\mathsf{E}\rangle_\mathsf{E}, \mathsf{map}, b, \mathsf{salt})}$
$\mathsf{psk} \coloneqq \mathsf{F.Punct}(\mathsf{mpk}, \langle\Delta v_\mathsf{E}\rangle_\mathsf{E})$	$\mathsf{psk} \coloneqq \mathsf{F.Punct}(\mathsf{mpk}, \langle\Delta v_\mathsf{E}\rangle_\mathsf{E})$
$\mathsf{k}^* \coloneqq (\mathsf{mpk}, \mathsf{psk})$	$\mathsf{k}^* \coloneqq (\mathsf{mpk}, \mathsf{psk})$
$\textbf{for } x \in \mathsf{I}_{N,m} \setminus \{v_\mathsf{E}\}:$	$\textbf{for } x \in \mathsf{I}_{N,m} \setminus \{v_\mathsf{E}\}:$
$\quad y_x \coloneqq \mathsf{pF}^\mathsf{H}(\mathsf{k}^*, v_\mathsf{E}, x, \mathsf{salt})$	$\quad y_x \coloneqq \mathsf{map}\left(\mathsf{pF}^\mathsf{H}(\mathsf{k}^*, v_\mathsf{E}, x, \mathsf{salt})\right)$
$\mathsf{shift} \coloneqq \sum_{x \in \mathsf{I}_{N,m} \setminus \{v_\mathsf{E}\}} y_x + \mathcal{O}(\mathsf{psk}, \mathsf{salt}, a) + b$	$\mathsf{shift} \coloneqq \sum_{x \in \mathsf{I}_{N,m} \setminus \{v_\mathsf{E}\}} y_x + \mathsf{map}\left(\mathcal{O}(\mathsf{psk}, \mathsf{salt}, 0)\right) + b$
$z_\mathsf{E} \coloneqq \mathsf{VtO}^\mathsf{H}_\mathsf{E}(\mathsf{mpk}, v_\mathsf{E}, \langle\Delta v_\mathsf{E}\rangle_\mathsf{E}, \mathsf{salt}, \mathsf{shift})$	$z_\mathsf{E} \coloneqq \mathsf{VtO}^{\mathsf{H}_0}_\mathsf{E}(\mathsf{mpk}, v_\mathsf{E}, \langle\Delta v_\mathsf{E}\rangle_\mathsf{E}, \mathsf{salt}, \mathsf{shift})$
$\textbf{return } (\mathsf{shift}, z_\mathsf{E})$	$\textbf{return } (\mathsf{shift}, z_\mathsf{E})$

As for VtO, the computation can be performed either over $\mathbb{Z}$ or $\mathbb{Z}_{p-1}$; we write $\mathcal{R}$-SimVtO to indicate the ring explicitly. The following lemma states that SimVtO outputs the exact same shift as $\mathsf{VtO}^\mathsf{H}_\mathsf{G}$ and the same z_E as $\mathsf{VtO}^\mathsf{H}_\mathsf{E}$:

Lemma 12 (Perfect simulation)**.** *For every* $(\mathsf{mpk}, \mathsf{msk})$ *in the support of* $\mathsf{F.Setup}(1^\lambda, N^m)$ *with* $\mathsf{msk} \coloneqq (\mathsf{mpk}, \Delta, h_0)$*, every* $v_\mathsf{E} \in \mathsf{I}_{N,m}$*, every* $v_\mathsf{G} \in \mathbb{Z}_{p-1}$*, every pair* $(a, b) \in \mathbb{Z}^2_{p-1}$ *such that* $v_\mathsf{G} = a \cdot \Delta + b$*, every* $\mathsf{salt} \in \{0,1\}^*$*, and every* $\langle\Delta v_\mathsf{E}\rangle_\mathsf{G}, \langle\Delta v_\mathsf{E}\rangle_\mathsf{E}$ *such that* $\langle\Delta v_\mathsf{E}\rangle_\mathsf{E} - \langle\Delta v_\mathsf{E}\rangle_\mathsf{G} = \Delta v_\mathsf{E}$*, denoting* $\mathcal{O} \coloneqq \mathcal{O}_{\mathsf{H},h_0,\Delta}$*, denoting* $(\mathsf{shift}, z_\mathsf{E}) \coloneqq \mathsf{SimVtO}^{\mathsf{H},\mathcal{O}}(\mathsf{mpk}, v_\mathsf{E}, \langle\Delta v_\mathsf{E}\rangle_\mathsf{E}, a, b, \mathsf{salt})$*, it holds that*

$$\begin{aligned}(\mathsf{shift}, \_) &= \mathsf{VtO}^\mathsf{H}_\mathsf{G}(\mathsf{msk}, v_\mathsf{G}, \langle\Delta v_\mathsf{E}\rangle_\mathsf{G}, \mathsf{salt}), \ \textit{and} \\ z_\mathsf{E} &= \mathsf{VtO}^\mathsf{H}_\mathsf{E}(\mathsf{mpk}, v_\mathsf{E}, \langle\Delta v_\mathsf{E}\rangle_\mathsf{E}, \mathsf{salt}, \mathsf{shift}).\end{aligned}$$

Furthermore, for any $\mathsf{map} : \mathbb{Z}_{p-1} \to \{0,1\}^*$*, setting* $\mathsf{H}_0 \coloneqq \mathsf{map} \circ \mathsf{H}$ *and* $(\mathsf{shift}', z'_\mathsf{E}) \coloneqq \mathsf{Sim'VtO}^{\mathsf{H},\mathcal{O}}(\mathsf{mpk}, v_\mathsf{E}, \langle\Delta v_\mathsf{E}\rangle_\mathsf{E}, \mathsf{map}, v_\mathsf{G}, \mathsf{salt})$*, it holds that*

$$\begin{aligned}(\mathsf{shift}', \_) &= \mathsf{VtO}^{\mathsf{H}_0}_\mathsf{G}(\mathsf{msk}, v_\mathsf{G}, \langle\Delta v_\mathsf{E}\rangle_\mathsf{G}, \mathsf{salt}), \ \textit{and} \\ z'_\mathsf{E} &= \mathsf{VtO}^{\mathsf{H}_0}_\mathsf{E}(\mathsf{mpk}, v_\mathsf{E}, \langle\Delta v_\mathsf{E}\rangle_\mathsf{E}, \mathsf{salt}, \mathsf{shift}).\end{aligned}$$

Proof. Let $k^* \coloneqq (\mathsf{mpk}, \mathsf{psk})$ where $\mathsf{psk} \coloneqq \mathsf{F.Punct}(\mathsf{mpk}, \langle\Delta v_\mathsf{E}\rangle_\mathsf{E})$. Observe that for shift computed using $\mathsf{VtO}^\mathsf{H}_\mathsf{G}(\mathsf{msk}, v_\mathsf{G}, \langle\Delta v_\mathsf{E}\rangle_\mathsf{G}, \mathsf{salt})$, we have,

$$\begin{aligned}\mathsf{shift} &= \sum_{x \in \mathsf{I}_{N,m}} \mathsf{F}^\mathsf{H}(\mathsf{k}, x, \mathsf{salt}) + v_\mathsf{G} \\ &= \sum_{x \in \mathsf{I}_{N,m} \setminus \{v_\mathsf{E}\}} \mathsf{pF}^\mathsf{H}(\mathsf{k}^*, v_\mathsf{E}, x, \mathsf{salt}) + \mathsf{F}^\mathsf{H}(\mathsf{k}, x, \mathsf{salt}) + a\Delta + b && \triangleright \text{ via Lemma 10} \\ &= \sum_{x \in \mathsf{I}_{N,m} \setminus \{v_\mathsf{E}\}} \mathsf{pF}^\mathsf{H}(\mathsf{k}^*, v_\mathsf{E}, x, \mathsf{salt}) + \mathcal{O}(\mathsf{psk}, \mathsf{salt}, a) + b && \triangleright \text{ via Definition 7}\end{aligned}$$

where the last expression is equal to shift as computed in $\mathsf{SimVtO}^{\mathsf{H},\mathcal{O}}(\mathsf{mpk}, v_\mathsf{E}, \langle\Delta v_\mathsf{E}\rangle_\mathsf{E}, a, b, \mathsf{salt})$. It then immediately follows that z_E, as computed by $\mathsf{SimVtO}^{\mathsf{H},\mathcal{O}}$, is equal to z_E output by $\mathsf{VtO}^\mathsf{H}_\mathsf{E}$, since $\mathsf{SimVtO}^{\mathsf{H},\mathcal{O}}$ simply runs $\mathsf{VtO}^\mathsf{H}_\mathsf{E}$ using the shift it computes. It is easy to see that a similar argument shows that shift' and z'_E, as output by $\mathsf{Sim'VtO}^{\mathsf{H},\mathcal{O}}$, are identical to shift' and z'_E output by $\mathsf{VtO}^{\mathsf{H}_0}_\mathsf{G}$ and $\mathsf{VtO}^{\mathsf{H}_0}_\mathsf{E}$ respectively. ■

5.3 Batch Function Evaluation on Authenticated Shares

We now introduce a second core procedure. It builds upon the following observation from [CHHK25] (a similar observation was also made in previous works, e.g., [Hea24]): the VtO procedure can be modified to convert shares of $\Delta \cdot v_{\mathsf{E}}$ into shares of $v_{\mathsf{G}} \cdot f(v_{\mathsf{E}})$ *for an arbitrary function* f (known to the parties): it suffices to compute instead $z_{\mathsf{G}} \coloneqq \sum_x f(x) \cdot \mathsf{F}^{\mathsf{H}}(\mathsf{k}, x, \mathsf{salt})$ and $z_{\mathsf{E}} \coloneqq \sum_x (f(v_{\mathsf{E}}) - f(x)) \cdot \mathsf{pF}^{\mathsf{H}}(\mathsf{k}^*, v_{\mathsf{E}}, x, \mathsf{salt}) + f(v_{\mathsf{E}}) \cdot \mathsf{shift}$.

A useful implication of this *functional* VtO procedure is that the parties can evaluate an arbitrary function on authenticated shares: given shares of $\Delta \cdot v_{\mathsf{E}}$, they can obtain shares of $\Delta \cdot f(v_{\mathsf{E}})$ by letting G set Δ to be its input to the functional VtO procedure. In this work, we make an additional observation which, while very simple in hindsight, proves to be very powerful: this "functional authentication" procedure can be generalized to allow the evaluation of an arbitrary-size tuple of functions $f_1, f_2, \cdots$ on an authenticated share *without any penalty in communication*. Indeed, the shift transmitted from G to E depends solely on its input v_{G} (set here to Δ), and *not* on the target function f. Hence, if G and E want to obtain shares of $\Delta \cdot f_i(v_{\mathsf{E}})$ for many functions i, they can run arbitrarily many parallel executions of the functional authentication procedure using the transmission of a single shift from G to E, independent of the number of functions. We describe the procedure below.

The batch functional authentication procedure. For $i = 1$ to q, let $f_i : \mathsf{I}_{N,m} \rightarrow \mathbb{Z}_{p-1}$ denote a public function. We let q unspecified and assume that BatchfAuth takes variable-length inputs.

$\mathsf{BatchfAuth}^{\mathsf{H}}_{\mathsf{G}}(\mathsf{msk}, \Delta', \langle \Delta v_{\mathsf{E}} \rangle_{\mathsf{G}}, (f_1, \cdots, f_q), \mathsf{salt})$

- $\mathsf{sk} \coloneqq \mathsf{F.KeyGen}(\mathsf{msk}, \langle \Delta v_{\mathsf{E}} \rangle_{\mathsf{G}})$
- $\mathsf{k} \coloneqq (\mathsf{msk}, \mathsf{sk})$
- $\mathsf{shift} \coloneqq \sum_{x \in \mathsf{I}_{N,m}} \mathsf{F}^{\mathsf{H}}(\mathsf{k}, x, \mathsf{salt}) + \Delta'$
- **for** $i = 1$ **to** q :
  - $z_{\mathsf{G}}[i] \coloneqq \sum_{x \in \mathsf{I}_{N,m}} f_i(x) \cdot \mathsf{F}^{\mathsf{H}}(\mathsf{k}, x, \mathsf{salt})$
- **return** $(\mathsf{shift}, z_{\mathsf{G}})$

$\mathsf{BatchfAuth}^{\mathsf{H}}_{\mathsf{E}}(\mathsf{mpk}, v_{\mathsf{E}}, \langle \Delta v_{\mathsf{E}} \rangle_{\mathsf{E}}, (f_1, \cdots, f_q), \mathsf{salt}, \mathsf{shift})$

- $\mathsf{psk} \coloneqq \mathsf{F.Punct}(\mathsf{mpk}, \langle \Delta v_{\mathsf{E}} \rangle_{\mathsf{E}})$
- $\mathsf{k}^* \coloneqq (\mathsf{mpk}, \mathsf{psk})$
- **for** $x \in \mathsf{I}_{N,m} \setminus \{v_{\mathsf{E}}\}$:
  - $y_x \coloneqq \mathsf{pF}^{\mathsf{H}}(\mathsf{k}^*, v_{\mathsf{E}}, x, \mathsf{salt})$
- **for** $i = 1$ **to** q :
  - $z_{\mathsf{E}}[i] \coloneqq \sum_{x \in \mathsf{I}_{N,m}} (f_i(x) - f_i(v_{\mathsf{E}})) \cdot y_x + f_i(v_{\mathsf{E}}) \cdot \mathsf{shift}$
- **return** z_{E}

We will also use a variant of BatchfAuth, denoted S-BatchfAuth, where the set $\mathsf{I}_{N,m}$ of the summands is replaced with another set S (that is, the sums for shift, $z_{\mathsf{G}}[i]$, and $z_{\mathsf{E}}[i]$ are over all $x \in S$).

Correctness. The following lemma establishes perfect correctness of the BatchfAuth procedure:

Lemma 13. *For every* $(\mathsf{mpk}, \mathsf{msk})$ *in the support of* $\mathsf{F.Setup}(1^\lambda, N^m)$ *with* $\mathsf{msk} \coloneqq (\mathsf{mpk}, \Delta, h_0)$*, every* $v_{\mathsf{E}} \in \mathsf{I}_{N,m}$*, every* $\Delta' \in \mathbb{Z}_{p-1}$*, every* $\langle \Delta v_{\mathsf{E}} \rangle_{\mathsf{G}}, \langle \Delta v_{\mathsf{E}} \rangle_{\mathsf{E}}$ *such that* $\langle \Delta v_{\mathsf{E}} \rangle_{\mathsf{E}} - \langle \Delta v_{\mathsf{E}} \rangle_{\mathsf{G}} = \Delta \cdot v_{\mathsf{E}} \bmod p-1$*, every* $q \geq 1$*, every* q*-tuple* $(f_1, \cdots, f_q)$ *of functions* $f_i : \mathsf{I}_{N,m} \rightarrow \mathbb{Z}_{p-1}$*, and every* $\mathsf{salt} \in \{0,1\}^*$*, denoting* $(\mathsf{shift}, z_{\mathsf{G}}) \coloneqq \mathsf{BatchfAuth}^{\mathsf{H}}_{\mathsf{G}}(\mathsf{msk}, \Delta', \langle \Delta v_{\mathsf{E}} \rangle_{\mathsf{G}}, (f_1, \cdots, f_q), \mathsf{salt})$ *and* $z_{\mathsf{E}} \coloneqq \mathsf{BatchfAuth}^{\mathsf{H}}_{\mathsf{E}}(\mathsf{mpk}, v_{\mathsf{E}}, \langle \Delta v_{\mathsf{E}} \rangle_{\mathsf{E}}, (f_1, \cdots, f_q), \mathsf{salt}, \mathsf{shift})$*, for all* $i \leq q$*, it holds that*

$$z_{\mathsf{E}}[i] - z_{\mathsf{G}}[i] = \Delta' \cdot f_i(v_{\mathsf{E}}) \bmod p-1.$$

The proof is again routinely checked using the correctness of the PPRF at all points $x \neq v_{\mathsf{E}}$, and using the fact that $f(x) - f(v_\mathsf{E}) = 0$ when $x = v_\mathsf{E}$:

$$\begin{aligned} z_\mathsf{E} &= \sum_{x \in \mathsf{I}_{N,m}} (f(x) - f(v_\mathsf{E})) \cdot \mathsf{pF}^\mathsf{H}(\mathsf{k}^*, v_\mathsf{E}, x, \mathsf{salt}) + f(v_\mathsf{E}) \cdot \mathsf{shift} \\ &= \sum_{x \in \mathsf{I}_{N,m}} (f(x) - f(v_\mathsf{E})) \cdot \mathsf{F}^\mathsf{H}(\mathsf{k}, x, \mathsf{salt}) + f(v_\mathsf{E}) \cdot \mathsf{shift} \qquad \triangleright \text{ via Lemma 10} \\ &= z_\mathsf{G} - v_\mathsf{E} \cdot \sum_{x \in \mathsf{I}_{N,m}} \mathsf{F}^\mathsf{H}(\mathsf{k}, x, \mathsf{salt}) + f(v_\mathsf{E}) \cdot \left(\sum_{x \in \mathsf{I}_{N,m}} \mathsf{F}^\mathsf{H}(\mathsf{k}, x, \mathsf{salt}) + v_\mathsf{G} \right) \\ &= z_\mathsf{G} + v_\mathsf{G} \cdot f(v_\mathsf{E}). \end{aligned}$$

Simulating shifts using $\mathcal{O}_{\mathsf{H},h_0,\Delta}$. We outline a simulator for $(\mathsf{shift}, z_\mathsf{E})$ for the case $\Delta' = \Delta$ (used in our main construction). The case where Δ' is independent of Δ (used for our result in the standard model) is discussed afterwards. Fix integers $N = N(\lambda)$, $m = m(\lambda)$ such that $N^m = \mathrm{poly}(\lambda)$. Let $(\mathbb{G}, p, g, G) \coloneqq \mathsf{GrpGen}^*(1^\lambda)$ and $\mathsf{H} = \{\mathsf{H}_\lambda : \mathbb{G} \times \{0,1\}^* \to \mathbb{Z}_{p-1}\}$. Fix a master key pair $(\mathsf{mpk}, \mathsf{msk}) \leftarrow\!\!\$\ \mathsf{F.Setup}(1^\lambda, N^m)$ and parse msk as $(\mathsf{mpk}, \Delta, h_0)$. Let $\mathcal{O} \coloneqq \mathcal{O}_{\mathsf{H},h_0,\Delta}$ be the oracle defined in Definition 7.

$\mathsf{SimBatchfAuth}^{\mathsf{H},\mathcal{O}}(\mathsf{mpk}, v_\mathsf{E}, \langle \Delta v_\mathsf{E} \rangle_\mathsf{E}, (f_1, \cdots, f_q), \mathsf{salt})$

1 : $\mathsf{psk} \coloneqq \mathsf{F.Punct}(\mathsf{mpk}, \langle \Delta v_\mathsf{E} \rangle_\mathsf{E})$
2 : $\mathsf{k}^* \coloneqq (\mathsf{mpk}, \mathsf{psk})$
3 : $\mathsf{shift} \coloneqq \sum_{x \in \mathsf{I}_{N,m} \setminus \{v_\mathsf{E}\}} \mathsf{pF}^\mathsf{H}(\mathsf{k}^*, v_\mathsf{E}, x, \mathsf{salt}) + \mathcal{O}(\mathsf{psk}, \mathsf{salt}, 1)$
4 : $z_\mathsf{E} \coloneqq \mathsf{BatchfAuth}^\mathsf{H}_\mathsf{E}(\mathsf{mpk}, v_\mathsf{E}, \langle \Delta v_\mathsf{E} \rangle_\mathsf{E}, (f_1, \cdots, f_q), \mathsf{salt}, \mathsf{shift})$
5 : **return** $(\mathsf{shift}, z_\mathsf{E})$

As for BatchfAuth, we let S-$\mathsf{SimBatchfAuth}^{\mathsf{H},\mathcal{O}}$ denote the variant where the sum is computed over $S \setminus \{v_\mathsf{E}\}$. The following lemma states that SimVtO outputs the exact same shift as $\mathsf{VtO}^\mathsf{H}_\mathsf{G}$ and the same z_E as $\mathsf{VtO}^\mathsf{H}_\mathsf{E}$:

Lemma 14 (Perfect simulation). *For every* $(\mathsf{mpk}, \mathsf{msk})$ *in the support of* $\mathsf{F.Setup}(1^\lambda, N^m)$ *with* $\mathsf{msk} \coloneqq (\mathsf{mpk}, \Delta, h_0)$, *every* $v_\mathsf{E} \in \mathsf{I}_{N,m}$, *and every* $\langle \Delta v_\mathsf{E} \rangle_\mathsf{G}, \langle \Delta v_\mathsf{E} \rangle_\mathsf{E}$ *such that* $\langle \Delta v_\mathsf{E} \rangle_\mathsf{E} - \langle \Delta v_\mathsf{E} \rangle_\mathsf{G} = \Delta v_\mathsf{E}$, *every* $q \geq 1$, *every q-tuple* $(f_1, \cdots, f_q)$ *of functions* $f_i : \mathsf{I}_{N,m} \to \mathbb{Z}_{p-1}$, *and every* $\mathsf{salt} \in \{0,1\}^*$, *denoting* $\mathcal{O} \coloneqq \mathcal{O}_{\mathsf{H},h_0,\Delta}$, *denoting* $(\mathsf{shift}, z_\mathsf{E}) \coloneqq \mathsf{SimBatchfAuth}^{\mathsf{H},\mathcal{O}}(\mathsf{mpk}, v_\mathsf{E}, \langle \Delta v_\mathsf{E} \rangle_\mathsf{E}, (f_1, \cdots, f_q), \mathsf{salt})$, *it holds that*

$$\begin{aligned} \mathsf{shift} &= \mathsf{BatchfAuth}^\mathsf{H}_\mathsf{G}(\mathsf{msk}, \Delta, \langle \Delta v_\mathsf{E} \rangle_\mathsf{G}, (f_1, \cdots, f_q), \mathsf{salt}), \ \text{and} \\ z_\mathsf{E} &= \mathsf{BatchfAuth}^\mathsf{H}_\mathsf{E}(\mathsf{mpk}, v_\mathsf{E}, \langle \Delta v_\mathsf{E} \rangle_\mathsf{E}, (f_1, \cdots, f_q), \mathsf{salt}, \mathsf{shift}). \end{aligned}$$

The straightforward proof, which is essentially identical to that of Lemma 12, is omitted.

Remark 2. The case where BatchfAuth uses a Δ' independent of Δ, as in our standard model construction, is obtained as a simple variant of the above procedure by letting $\mathsf{SimBatchfAuth}^{\mathsf{H},\mathcal{O}}$ additionally take Δ' as input and computing instead shift as

$$\mathsf{shift} \coloneqq \sum_{x \in \mathsf{I}_{N,m} \setminus \{v_\mathsf{E}\}} \mathsf{pF}^\mathsf{H}(\mathsf{k}^*, v_\mathsf{E}, x, \mathsf{salt}) + \mathcal{O}(\mathsf{psk}, \mathsf{salt}, 0) + \Delta'.$$

6 $\omega(1/\lambda)$-Rate Boolean Garbling Scheme from Generic Groups

We prove in this section the following theorem:

Theorem 15. *There exists a polynomial* $B(\lambda) = \mathsf{poly}(\lambda)$ *such that, given a group* $\mathbb{G}$ *of order* p *(whose elements are of size* $O(\lambda)$ *bits), if there exists a TCCR hashing* H *for exponential correlation with* B *auxiliary powers over* $\mathbb{G}$*, then there exists a Boolean garbling scheme* $\mathsf{GC} = (\mathsf{GC.Garble}, \mathsf{GC.Enc}, \mathsf{GC.Eval}, \mathsf{GC.Dec})$ *such that for each circuit* C *whose layers contain at least* $\sqrt{\log\lambda}$ *gates, it holds that*

$$|\hat{C}| = \frac{\lambda}{\sqrt{\log(\lambda)}} \cdot O(|C|) + \mathsf{poly}(\lambda).$$

For extremely narrow circuits, the above cost can grow by up to an additive $O(\lambda \cdot D)$ *factor.*

6.1 High Level Structure

For convenience, we first describe the garbling procedure by abstracting out the gadgets used to compute the keys and labels for batches of XOR and AND gates and for packing/unpacking keys and labels into batches. The garbling scheme uses the following parameters:

- t: the number of bits in a batch. Concretely, we will set $t = \sqrt{\log\lambda}$.
- m: the degree of the extension field $\mathbb{F}_{2^m}$ where batches of t bits are embedded via the $(t, m)_2$-RMFE (Φ, Ψ). Using Lemma 4, we have $m = O(t)$.
- c: a statistical security parameter for hiding perturbed polynomials. Our construction guarantees security up to a $\mathsf{poly}(\lambda)/2^c$ statistical leakage probability. Concretely, we will set c to $2^{\sqrt{\log\lambda}}$.
- N: a size parameter for embedding polynomials into integers without overflows. Our construction requires $N > 2c \cdot m \cdot (2^m + 1)$, and $N^m \leq \mathsf{poly}(\lambda)$. Using our parameters $m = O(\sqrt{\log\lambda})$ and $c = 2^{\sqrt{\log\lambda}}$ yields $c^m \cdot (m \cdot (2^m + 1))^m = \mathsf{poly}(\lambda)$, as required.

Parameters. Let $(\mathbb{G}, p, g, G) \coloneqq \mathsf{GrpGen}^*(1^\lambda)$. Let $t = \sqrt{\log\lambda}$ denote the batch parameter and (Φ, Ψ) be a $(t, m)_2$-reverse multiplication friendly embedding with $m = O(t)$. Let $c = \omega(1)$ denote a statistical security parameter with $c \leq 2^{\sqrt{\log\lambda}}$ and set $N = 2c \cdot m \cdot (2^m + 1) + 1$ (note that $N^m = \mathsf{poly}(\lambda)$). Let $\mathsf{H} = \{\mathsf{H}_\lambda : \mathbb{G} \times \{0,1\}^* \to \mathbb{Z}_{p-1}\}_{\lambda\in\mathbb{N}}$ denote a TCCR hash family for exponential correlation with auxiliary powers over $\mathbb{G}$. Let $\mathsf{H}_0 \coloneqq \mathsf{map} \circ \mathsf{H}$, where $\mathsf{map} = \mathsf{map}_{N,m,c}$ is the mapping from Definition 9.

Input. The input to $\mathsf{GC.Garble}$ is a boolean circuit C with $|C| = s$ gates, $n = |I(C)|$ inputs, and depth $\mathsf{depth}(C) = D$. Without loss of generality, we assume that the gates of C are divided into D layers, denoted $\mathcal{L}_1, \cdots, \mathcal{L}_D$, where each layer contains either only AND gates or only XOR gates. All incoming wires of gates in a layer are connected to inputs or to gates from previous layers (in particular, the gates in $\mathcal{L}_1$ are only connected to input gates). Each layer $\mathcal{L}_d$ is partitioned into $n_d \coloneqq \lceil |\mathcal{L}_d|/t \rceil$ batches $(\mathcal{B}_{d,1}, \cdots, \mathcal{B}_{d,n_d})$ containing at most t gates each. For $d = 1$ to D, let $\mathsf{salt}_{d,1}, \cdots, \mathsf{salt}_{d,n_d}$ denote unique identifiers for each batch of gate, and write $\mathsf{salt}_{d,i,j} \coloneqq \mathsf{salt}_{d,i} || j$ for $j = 0, 1, 2, 3$. We assume that all these data can be parsed from the description C of the circuit.

Algorithms. The algorithms $(\mathsf{GC.Garble}, \mathsf{GC.Enc}, \mathsf{GC.Eval}, \mathsf{GC.Dec})$ are represented below. The algorithms $\mathsf{GC.Garble}$ and $\mathsf{GC.Eval}$ rely on subprocedures, respectively $(\mathsf{Pack}^{\mathsf{H}}_{\mathsf{G}}, \mathsf{BatchAND}^{\mathsf{H}}_{\mathsf{G}}, \mathsf{UnpackAND}^{\mathsf{H}}_{\mathsf{G}}, \mathsf{BatchXOR}^{\mathsf{H}}_{\mathsf{G}}, \mathsf{UnpackXOR}^{\mathsf{H}}_{\mathsf{G}})$ and $(\mathsf{Pack}^{\mathsf{H}}_{\mathsf{E}}, \mathsf{BatchAND}^{\mathsf{H}}_{\mathsf{E}}, \mathsf{UnpackAND}^{\mathsf{H}}_{\mathsf{E}}, \mathsf{BatchXOR}^{\mathsf{H}}_{\mathsf{E}}, \mathsf{UnpackXOR}^{\mathsf{H}}_{\mathsf{E}})$, that are described afterwards.

Algorithm GC.Garble($1^\lambda, C$)

Input. A boolean circuit C with $|C| = s$ gates and depth $\mathrm{depth}(C) = D$ represented as described in Section 6.1. The input gates are indexed from 1 to n.

Initialization.

- Sample $(\mathsf{mpk}, \mathsf{msk}) \leftarrow_\$ \mathsf{F.Setup}(1^\lambda, N^m)$. Parse $\mathsf{msk} \coloneqq (\mathsf{mpk}, \Delta, g_0)$.
- For each input wire i, sample $(k_i, K_i) \leftarrow_\$ \{0, 1\} \times \mathbb{Z}_{p-1}$.

Procedure. The garbling proceeds in a layer-by-layer fashion, from $\mathcal{L}_1$ to $\mathcal{L}_D$. After evaluating a layer $\mathcal{L}_d$, it labels each gate u in the layer with a pair (k_u, K_u) and stores a garbling $\hat{\mathcal{L}}_d$ of $\mathcal{L}_d$.

On layer $\mathcal{L}_d$. For $i = 1$ to n_d,

- Let $\mathsf{Left}_{d,i}$ (resp. $\mathsf{Right}_{d,i}$) denote the multisets of gates that are the left parent (resp. right parent) of a gate in $\mathcal{B}_{d,i}$. Retrieve the pairs (k_u, K_u) labeling each $u \in \mathsf{Left}_{d,i} \cup \mathsf{Right}_{d,i}$ and compute

$$(k_\mathsf{l}, K_\mathsf{l}, \mathsf{shift}_{\mathsf{l},d,i}) \coloneqq \mathsf{Pack}^\mathsf{H}_\mathsf{G}(\mathsf{msk}, (k_u, K_u)_{u \in \mathsf{Left}_{d,i}}, \mathsf{salt}_{d,i,0})$$
$$(k_\mathsf{r}, K_\mathsf{r}, \mathsf{shift}_{\mathsf{r},d,i}) \coloneqq \mathsf{Pack}^\mathsf{H}_\mathsf{G}(\mathsf{msk}, (k_u, K_u)_{u \in \mathsf{Right}_{d,i}}, \mathsf{salt}_{d,i,1}).$$

- If $\mathcal{L}_d$ is an AND layer:
  - $(k_\mathsf{out}, K_\mathsf{out}, S_{d,i}) \leftarrow_\$ \mathsf{BatchAND}^\mathsf{H}_\mathsf{G}(\mathsf{msk}, (k_\mathsf{l}, K_\mathsf{l}), (k_\mathsf{r}, K_\mathsf{r}), \Delta, \mathsf{salt}_{d,i,2})$
  - $((k[j], K_j)_{0 \le i \le t-1}, \mathsf{shift}_{\mathsf{out},d,i}) \coloneqq \mathsf{UnpackAND}^\mathsf{H}_\mathsf{G}(\mathsf{msk}, k_\mathsf{out}, K_\mathsf{out}, \mathsf{salt}_{d,i,3})$
  - $(k_u, K_u)_{u \in \mathcal{B}_{d,i}} \coloneqq (k[j], K_j)_{0 \le j \le |\mathcal{B}_{d,i}|-1}$
- If $\mathcal{L}_d$ is a XOR layer:
  - $(k_\mathsf{out}, K_\mathsf{out}, S_{d,i}) \leftarrow_\$ \mathsf{BatchXOR}^\mathsf{H}_\mathsf{G}(\mathsf{msk}, (k_\mathsf{l}, K_\mathsf{l}), (k_\mathsf{r}, K_\mathsf{r}), \Delta, \mathsf{salt}_{d,i,2})$
  - $((k[j], K_j)_{0 \le j \le t-1}, \mathsf{shift}_{\mathsf{out},d,i}) \coloneqq \mathsf{UnpackXOR}^\mathsf{H}_\mathsf{G}(\mathsf{msk}, k_\mathsf{out}, K_\mathsf{out}, \mathsf{salt}_{d,i,3})$
  - $(k_u, K_u)_{u \in \mathcal{B}_{d,i}} \coloneqq (k[j], K_j)_{0 \le j \le |\mathcal{B}_{d,i}|-1}$
- Label each $u \in \mathcal{B}_{d,i}$ with the key pair (k_u, K_u).

Set $\hat{\mathcal{L}}_d \coloneqq (\mathsf{shift}_{\mathsf{l},d,i}, \mathsf{shift}_{\mathsf{r},d,i}, S_{d,i}, \mathsf{shift}_{\mathsf{out},d,i})_{i \le n_d}$.

Output. Return $\mathsf{e} \coloneqq ((k_i, K_i)_{i \le n}, \Delta)$, $\hat{C} \coloneqq (C, \mathsf{mpk}, (\hat{\mathcal{L}}_d)_{d \le D})$, and $\mathsf{d} \coloneqq (k_o)_{o \in O(C)}$.

Algorithm 1: Garbling procedure of the Boolean garbling scheme

Algorithm GC.Enc(e, x)

Input. Encoding information e and input $x \in \{0, 1\}^n$. Parse $\mathsf{e} \coloneqq ((k_i, K_i)_{i \le n}, \Delta)$.

Procedure. For $i = 1$ to n, set $(\ell_i, L_i) \coloneqq (k_i \oplus x_i, K_i + \Delta \cdot \ell_i \bmod p - 1)$.

Output. Return $\hat{x} \coloneqq (\ell_i, L_i)_{i \le n}$.

Algorithm 2: Encoding procedure of the Boolean garbling scheme

We now describe the evaluation procedure. It maintains the invariant that on each gate u carrying a value y_u, the keys and labels computed by the garbler and evaluator respectively satisfy $(\ell_u, L_u) = (k_u \oplus y_u, K_u + \Delta \cdot \ell_u \bmod p-1)$.

Algorithm $\mathsf{GC.Eval}(\hat{C}, \hat{x})$

Inputs. Parse $\hat{C}$ as $(C, \mathsf{mpk}, (\hat{\mathcal{L}}_d)_{d \le D})$ and $\hat{x}$ as $(\ell_i, L_i)_{i \le n} \in (\mathbb{F}_2 \times \mathbb{Z}_{p-1})^n$.

Procedure. The evaluation proceeds in a layer-by-layer fashion, from $\mathcal{L}_1$ to $\mathcal{L}_D$. After evaluating a layer $\mathcal{L}_d$, it labels each gate u in the layer with a pair (ℓ_u, L_u).

On layer $\mathcal{L}_d$. For $i = 1$ to n_d,

- Parse $\hat{\mathcal{L}}_d$ as $\hat{\mathcal{L}}_d \coloneqq (\mathsf{shift}_{\mathsf{l},d,i}, \mathsf{shift}_{\mathsf{r},d,i}, S_{d,i})_{i \le n_d}, \mathsf{shift}_{\mathsf{out},d,i})$.
- Let $\mathsf{Left}_{d,i}$ (resp. $\mathsf{Right}_{d,i}$) denote the multisets of gates that are the left parent (resp. right parent) of a gate in $\mathcal{B}_{d,i}$. Retrieve the pairs (ℓ_u, L_u) labeling each $u \in \mathsf{Left}_{d,i} \cup \mathsf{Right}_{d,i}$ and compute

$$(\ell_\mathsf{l}, L_\mathsf{l}) \coloneqq \mathsf{Pack}^\mathsf{H}_\mathsf{E}(\mathsf{mpk}, (\ell_u, L_u)_{u \in \mathsf{Left}_{d,i}}, \mathsf{salt}_{i,d,0}, \mathsf{shift}_{\mathsf{l},d,i})$$
$$(\ell_\mathsf{r}, L_\mathsf{r}) \coloneqq \mathsf{Pack}^\mathsf{H}_\mathsf{E}(\mathsf{mpk}, (\ell_u, L_u)_{u \in \mathsf{Right}_{d,i}}, \mathsf{salt}_{i,d,1}, \mathsf{shift}_{\mathsf{r},d,i}).$$

- If $\mathcal{L}_d$ is an AND layer:
  - $(\ell_\mathsf{out}, L_\mathsf{out}) \coloneqq \mathsf{BatchAND}^\mathsf{H}_\mathsf{E}(\mathsf{mpk}, (\ell_\mathsf{l}, L_\mathsf{l}), (\ell_\mathsf{r}, L_\mathsf{r}), \mathsf{salt}_{d,i,2}, S_{d,i})$
  - $(\ell[j], L_j)_{0 \le j \le t-1} \coloneqq \mathsf{UnpackAND}^\mathsf{H}_\mathsf{E}(\mathsf{mpk}, \ell_\mathsf{out}, L_\mathsf{out}, \mathsf{salt}_{d,i,3}, \mathsf{shift}_{\mathsf{out},d,i})$
  - $(\ell_u, L_u)_{u \in \mathcal{B}_{d,i}} \coloneqq (\ell[j], L_j)_{0 \le j \le |\mathcal{B}_{d,i}|-1}$
- If $\mathcal{L}_d$ is a XOR layer:
  - $(\ell_\mathsf{out}, L_\mathsf{out}) \coloneqq \mathsf{BatchXOR}^\mathsf{H}(\mathsf{mpk}, (\ell_\mathsf{l}, L_\mathsf{l}), (\ell_\mathsf{r}, L_\mathsf{r}), \mathsf{salt}_{d,i,2}, S_{d,i})$
  - $(\ell[j], L_j)_{0 \le i \le t-1} \coloneqq \mathsf{UnpackXOR}^\mathsf{H}_\mathsf{E}(\mathsf{mpk}, \ell_\mathsf{out}, L_\mathsf{out}, \mathsf{salt}_{d,i,3}, \mathsf{shift}_{\mathsf{out},d,i})$
  - $(\ell_u, L_u)_{u \in \mathcal{B}_{d,i}} \coloneqq (\ell[j], L_j)_{0 \le i \le |\mathcal{B}_{d,i}|-1}$
- Label each $u \in \mathcal{B}_{d,i}$ with (ℓ_u, L_u).

Output. Return $\hat{y} \coloneqq (\ell_o)_{o \in O(C)}$.

Algorithm 3: Evaluator algorithm of the Boolean garbling scheme

Algorithm $\mathsf{GC.Dec}(\mathsf{d}, \hat{y})$

Input. Decoding information d and garbled output $\hat{y}$. Parse $\mathsf{d} \coloneqq (k_o)_{o \in O(C)}$ and $\hat{y} \coloneqq (\ell_o)_{o \in O(C)}$.

Output. Return $y \coloneqq (\ell_o \oplus k_o)_{o \in O(C)}$.

Algorithm 4: Decoding procedure of the Boolean garbling scheme

Padding. For convenience, given values $(v_u)_{u\in\mathcal{B}}$ either in $\mathbb{F}_2^{|\mathcal{B}|}$ or in $\mathbb{Z}_{p-1}^{|\mathcal{B}|}$ where $\mathcal{B}$ is a subset of indices of size at most t, we write $(v[0],\cdots,v[t-1]) \coloneqq \mathsf{pad}_t((v_u)_{u\in\mathcal{B}})$ to denote the procedure that assigns $(v_{u_0},\cdots,v_{u_{|\mathcal{B}|-1}})$ to $(v[0],\cdots,v[|\mathcal{B}|-1])$, where $u_0\cdots u_{|\mathcal{B}|-1}$ is an ordering (e.g., lexicographic) of the elements of $\mathcal{B}$, and assigns 0 to the remaining $v[i]$'s for $i=|\mathcal{B}|$ to $t-1$. That is, pad_t pads a list of values with zeroes to get an element of $\mathbb{F}_2^t$ or $\mathbb{Z}_{p-1}^t$ (we slightly abuse our notations and do not distinguish between padding with $0\in\mathbb{F}_2$ or with $0\in\mathbb{Z}_{p-1}$).

6.2 Efficiency and Correctness

Efficiency. Let $\hat{C} \coloneqq \mathsf{GC.Garble}(1^\lambda, C)$. We have

$$|\hat{C}| = |C| + |\mathsf{mpk}| + \sum_{d=1}^{D} |\hat{\mathcal{L}}_D|.$$

The size of $|\mathsf{mpk}|$ is $2N^m+1$ elements of $\mathbb{G}$, which translates to $O(\lambda\cdot(2cm2^m)^m)$ bits. Using $m=O(\sqrt{\log\lambda})$ yields $|\mathsf{mpk}| = \mathsf{poly}(\lambda)$, where poly is a fixed polynomial independent of C. The size of each garbled layer $\hat{\mathcal{L}}_d$ is $O(\lambda\cdot n_d)$, as it contains n_d constant-length tuples of shifts (4 for a batch of XORs, 9 for a batch of ANDs), where each shift is $O(\lambda)$-bit long. This yields

$$|\hat{C}| = |C| + O(\lambda)\cdot\sum_{d=1}^{D}\lceil|\mathcal{L}_d|/t\rceil + \mathsf{poly}(\lambda).$$

For circuits that are not too narrow (where the layers contain more than $\sqrt{\log\lambda}$ gates), this translates to $O(\lambda/\sqrt{\log(\lambda)})\cdot|C| + \mathsf{poly}(\lambda)$ bits. In the worst-case, for extremely narrow circuits, $|\hat{C}|$ can grow to $O(\lambda/\sqrt{\log(\lambda)})\cdot|C| + O(\lambda\cdot D) + \mathsf{poly}(\lambda)$ bits.

Correctness. Let x denote an input to C. For each gate u, let x_u denote the bit output by this gate in the computation of $C(x)$. Given a batch $\mathcal{B}$ of gates, let $x_{\mathcal{B}} \coloneqq (x_u)_{u\in\mathcal{B}}$. The proof of correctness relies on the fact that all the procedures maintain a suitable invariant throughout the computation. The required invariant for each procedure is guaranteed by a correctness lemma:

- Lemma 16 for $\mathsf{Pack}^{\mathsf{H}}$
- Lemma 17 for $\mathsf{Unpack}^{\mathsf{H}}$
- Lemma 19 for $\mathsf{BatchAND}^{\mathsf{H}}$
- Lemma 18 for $\mathsf{BatchXOR}^{\mathsf{H}}$

At the start of the procedure, for each input gate i, we have $k_i\oplus\ell_i = x_i$ and $L_i - K_i = \Delta\cdot\ell_i \bmod p-1$ by definition of GC.Enc. Then, fix a layer d and a batch $i\le n_d$ and assume that before running the procedures on $\mathcal{L}_d$, it holds that $x_u = k_u\oplus\ell_u$ and $L_u - K_u = \Delta\cdot\ell_u \bmod p-1$ for all $u\in\mathsf{Left}_{d,i}\cup\mathsf{Right}_{d,i}$. Then,

- By Lemma 16, it holds after running the $\mathsf{Pack}^{\mathsf{H}}$ procedures that $k_{\mathsf{l}}+\ell_{\mathsf{l}} = \Phi(\mathsf{pad}_t(x_{\mathsf{Left}_{d,i}}))$, $k_{\mathsf{r}}+\ell_{\mathsf{r}} = \Phi(\mathsf{pad}_t(x_{\mathsf{Right}_{d,i}}))$, $L_{\mathsf{l}}-K_{\mathsf{l}} = \Delta\cdot\ell_{\mathsf{l}}(N) \bmod p-1$, and $L_{\mathsf{r}}-K_{\mathsf{r}} = \Delta\cdot\ell_{\mathsf{r}}(N) \bmod p-1$. Let $x_{\mathsf{l}} \coloneqq \Phi(\mathsf{pad}_t(x_{\mathsf{Left}_{d,i}}))$ and $x_{\mathsf{r}} \coloneqq \Phi(\mathsf{pad}_t(x_{\mathsf{Right}_{d,i}}))$.

- If $\mathcal{L}_d$ is an AND layer, by Lemma 19, it holds after running the BatchAND$^{\mathsf{H}}$ procedure that $k_{\text{out}} + \ell_{\text{out}} = x_{\text{l}} \cdot x_{\text{r}}$ and $K_{\text{out}} - L_{\text{out}} = \Delta \cdot \ell_{\text{out}}(N) \bmod p-1$.
- If $\mathcal{L}_d$ is an AND layer, by Lemma 17, it holds after running the Unpack$^{\mathsf{H}}$ procedure that $(k[i] \oplus \ell[i])_{i \le t-1} = \Psi(x_{\text{l}} \cdot x_{\text{r}})$ and $K[i] - L[i] = \Delta \cdot \ell[i] \bmod p-1$ for $i = 0$ to $t-1$.
- By an identical reasoning, if $\mathcal{L}_d$ is a XOR layer, by Lemma 18 and Lemma 17, it holds after running the BatchXOR$^{\mathsf{H}}$ and Unpack$^{\mathsf{H}}$ procedures that $(k[i] \oplus \ell[i])_{i \le t-1} = \Phi^{-1}(x_{\text{l}} \oplus x_{\text{r}})$ and $K[i] - L[i] = \Delta \cdot \ell[i] \bmod p-1$ for $i = 0$ to $t-1$.

It follows that after each AND layer, the gates in $\mathcal{B}_{d,i}$ are labeled with the first $|\mathcal{B}_{d,i}|$ entries of $\Psi(\Phi(\mathsf{pad}_t(x_{\mathsf{Left}_{d,i}})) \cdot \Phi(\mathsf{pad}_t(x_{\mathsf{Right}_{d,i}})))$. By definition of the RMFE maps (Definition 5), this value equal to $\mathsf{pad}_t(x_{\mathsf{Left}_{d,i}}) \odot \mathsf{pad}_t(x_{\mathsf{Right}_{d,i}})$, hence its first $|\mathcal{B}_{d,i}|$ entries are exactly the products $x_{u_{\text{l}}} \cdot x_{u_{\text{r}}}$, where $u_{\text{l}}, u_{\text{r}}$ denote the left and right parents of each gate $u \in \mathcal{B}_{d,i}$ respectively. Similarly, after each XOR layer, each gate u of the layer gets labeled with $x_{u_{\text{l}}} \oplus x_{u_{\text{r}}}$. Eventually, after all layers have been computed, it holds that $k_o \oplus \ell_o = x_o$ for each output gate o, and we have $(x_o)_{o \in O(C)} = y = C(x)$.

6.3 Packing Procedures

Given $x \in \mathbb{N}$, let us write $|x| := \lceil \log_2(x) \rceil$. Let $\mathsf{toBits} : \mathbb{N} \to \mathbb{F}_2^*$ denote the function that, on input an integer $x \in \mathbb{N}$, output the bit decomposition $x[1], \cdots, x[|x|]$ of x, viewed as an element of $\mathbb{F}_2^{|x|}$.

Algorithm $\mathsf{Pack}_{\mathsf{G}}^{\mathsf{H}}(\mathsf{msk}, (k_u, K_u)_{u \in \mathcal{B}}, \mathsf{salt})$

Input. Master secret key msk. Wire keys $(k_u, K_u)_{u \in \mathcal{B}} \in (\{0,1\} \times \mathbb{Z}_{p-1})^{|\mathcal{B}|}$ for a batch of gates $\mathcal{B}$ of size $|\mathcal{B}| \le t$. Salt salt. Parse msk as $(\mathsf{mpk}, \Delta, h_0)$.

Procedure.

- $k_{\text{out}} := \Phi(\mathsf{pad}_t((k_u)_{i \in \mathcal{B}}))$
- $(K[0], \cdots, K[t-1]) := \mathsf{pad}_t((K_u)_{i \in \mathcal{B}})$
- $K \leftarrow \sum_{i=0}^{t-1} K[i] \cdot 2^i \bmod p-1$ $\quad \triangleright\ K = \langle \Delta \cdot \sum_{i=0}^{t-1} \ell[i] \cdot 2^i \rangle_{\mathsf{G}}$
- $(\mathsf{shift}, K_{\text{out}}) := [2^t]\text{-}\mathsf{BatchfAuth}_{\mathsf{G}}^{\mathsf{H}}(\mathsf{msk}, \Delta, K, \mathsf{Eval}_N \circ \Phi \circ \mathsf{toBits}, \mathsf{salt})$

Output. $(k_{\text{out}}, K_{\text{out}}, \mathsf{shift})$

Algorithm 5: Garbler packing procedure. Given ordered indices $(u_0, \cdots, u_{|\mathcal{B}|-1})$, it packs multiples shares (k_{u_i}, ℓ_{u_i}) of bits $x_i \in \mathbb{F}_2$, and $\mathbb{Z}_{p-1}$-shares $(K_{u_i}, L_{u_i})_i$ of $\Delta \cdot \ell_{u_i}$, into $\mathbb{F}_{2^m}$-shares of $\Phi((x_0, \cdots, x_{|\mathcal{B}|-1})$ and $\mathbb{Z}_{p-1}$-shares of $\Delta \cdot \mathsf{Eval}_N(\Phi(\ell_{u_0}, \cdots, \ell_{u_{|\mathcal{B}|-1}}))$.

Algorithm $\mathsf{Pack}_{\mathsf{E}}^{\mathsf{H}}(\mathsf{mpk}, (\ell_u, L_u)_{u \in \mathcal{B}}, \mathsf{salt}, \mathsf{shift})$

Input. Wire labels $(\ell_u, L_u)_{u \in \mathcal{B}} \in (\{0,1\} \times \mathbb{Z}_{p-1})^{|\mathcal{B}|}$ for a batch of gates $\mathcal{B}$ of size $|\mathcal{B}| \le t$, garbling material shift.

Procedure.

- $\ell_{\mathsf{out}} \coloneqq \Phi(\ell[0], \cdots, \ell[t-1]) = \Phi(\mathsf{pad}_t((\ell_u)_{i\in\mathcal{B}}))$
- $(L[0], \cdots, L[t-1]) \coloneqq \mathsf{pad}_t((L_u)_{i\in\mathcal{B}})$
- $\tilde{\ell}_{\mathsf{out}} \coloneqq \sum_{i=0}^{t-1} \ell[i] \cdot 2^i$
- $L \leftarrow \sum_{i=0}^{t-1} L[i] \cdot 2^i \bmod p-1$ $\quad \triangleright\ L = \langle \Delta \cdot \tilde{\ell}_{\mathsf{out}} \rangle_{\mathsf{E}}$
- $L_{\mathsf{out}} \coloneqq [2^t]\text{-}\mathsf{BatchfAuth}_{\mathsf{E}}^{\mathsf{H}}(\mathsf{mpk}, \tilde{\ell}_{\mathsf{out}}, L, \mathsf{Eval}_N \circ \Phi \circ \mathsf{toBits}, \mathsf{salt}, \mathsf{shift})$

Output. $(\ell_{\mathsf{out}}, L_{\mathsf{out}})$

Algorithm 6: Evaluator packing procedure. Given ordered indices $(u_0, \cdots, u_{|\mathcal{B}|-1})$, it packs multiples shares (k_{u_i}, ℓ_{u_i}) of bits $x_i \in \mathbb{F}_2$, and $\mathbb{Z}_{p-1}$-shares $(K_{u_i}, L_{u_i})_i$ of $\Delta \cdot \ell_{u_i}$, into $\mathbb{F}_{2^m}$-shares of $\Phi((x_0, \cdots, x_{|\mathcal{B}|-1})$ and $\mathbb{Z}_{p-1}$-shares of $\Delta \cdot \mathsf{Eval}_N(\Phi(\ell_{u_0}, \cdots, \ell_{u_{|\mathcal{B}|-1}}))$.

Lemma 16 (Correctness of packing)**.** *Fix* $(\mathsf{mpk}, \mathsf{msk}, (\ell_u, L_u, k_u, K_u)_{u\in\mathcal{B}}, \mathsf{salt})$ *where* $\mathsf{msk} \coloneqq (\mathsf{mpk}, \Delta, h_0)$. *Assume that for each $u \in \mathcal{B}$, it holds that* $L_u - K_u = \Delta \cdot \ell_u \bmod p-1$. *Then, denoting*

$$\begin{aligned}(x_u)_{u\in\mathcal{B}} &\coloneqq (k_u \oplus \ell_u)_{u\in\mathcal{B}}\\ (k_{\mathsf{out}}, K_{\mathsf{out}}, \mathsf{shift}) &\coloneqq \mathsf{Pack}_{\mathsf{G}}^{\mathsf{H}}(\mathsf{msk}, (k_u, K_u)_{u\in\mathcal{B}}, \mathsf{salt})\\ (\ell_{\mathsf{out}}, L_{\mathsf{out}}) &\coloneqq \mathsf{Pack}_{\mathsf{E}}^{\mathsf{H}}(\mathsf{mpk}, (\ell_u, L_u)_{u\in\mathcal{B}}, \mathsf{salt}, \mathsf{shift}),\end{aligned}$$

it holds that

$$\begin{aligned}k_{\mathsf{out}} + \ell_{\mathsf{out}} &= \Phi(\mathsf{pad}_t((x_u)_{u\in\mathcal{B}}))\\ L_{\mathsf{out}} - K_{\mathsf{out}} &= \Delta \cdot \mathsf{Eval}_N(\ell_{\mathsf{out}}) \bmod p-1.\end{aligned}$$

Proof. The first part of Lemma 16 follows immediately from the linearity of $\Phi \circ \mathsf{pad}_t$. As for the second part, we have

$$\begin{aligned}L - K &= \sum_{i=0}^{t-1}(L[i] - K[i]) \cdot 2^i \bmod p-1\\ &= \Delta \cdot \sum_{i=0}^{t-1} \ell[i] \cdot 2^i = \Delta \cdot \tilde{\ell}_{\mathsf{out}} \bmod p-1 \qquad \triangleright \text{ by assumption of Lemma 16}\end{aligned}$$

Hence, the conditions of Lemma 13 are satisfied. Applying Lemma 13, we get

$$L_{\mathsf{out}} - K_{\mathsf{out}} = \Delta \cdot \mathsf{Eval}_N(\Phi(\mathsf{toBits}(\tilde{\ell}_{\mathsf{out}}))) \bmod p-1,$$

and we conclude by observing that $\mathsf{pad}_t((\ell_u)_{u\in\mathcal{B}}) = \mathsf{toBits}(\tilde{\ell}_{\mathsf{out}})$ by construction, hence $\ell_{\mathsf{out}} = \Phi(\mathsf{toBits}(\tilde{\ell}_{\mathsf{out}}))$. ■

6.4 Unpacking Procedures

We also introduce unpacking procedures for the garbler and the evaluator. We let Bit_i denote the function that, given a value $v \in \mathbb{F}_2^t$, outputs the i-th co-efficient $v[i]$ of v. Given an operation $\circledast \in \{+, \cdot\}$ (bitwise-XOR or bitwise-AND), the procedures convert shares (k, ℓ) of $\Phi(x) \circledast \Phi(y) \in \mathbb{F}_{2^m}$ and $\Delta \cdot \ell(n)$ back to bitwise shares of $f(\Phi(x) \circledast \Phi(y)) \in \mathbb{F}_2^t$ and of $(\Delta \cdot f(\ell)[i])_{i\leq t}$ (that is, Δ-authenticated substractive shares of each bit of $\Phi^{-1}(\ell)$ over $\mathbb{Z}_{p-1}$). Depending on the operation $\circledast$, we use a different "inverse mapping" f:

- If $\star$ = + (when unpacking the output of a batch-XOR), we set $f = \Phi^{-1}$, since $\Phi^{-1}(\Phi(x)+\Phi(y)) = x \oplus y$ (by linearity of Φ).
- If $\star$ = $\cdot$ (when unpacking the output of a batch-AND), we set $f = \Psi$, since $\Psi(\Phi(x) \cdot \Phi(y)) = x \odot y$ (by definition of RMFEs).

We define the general procedures below.

Algorithm $\mathsf{Unpack}_{\mathsf{G}}^{\mathsf{H}}(f, \mathsf{msk}, k, K, \mathsf{salt})$

Input. Function $f \in \{\Phi^{-1}, \Psi\}$. Master secret key msk, packed keys $(k, K) \in \mathbb{F}_{2^m} \times \mathbb{Z}_{p-1}$, salt salt. Parse $\mathsf{msk} \coloneqq (\mathsf{mpk}, \Delta, h_0)$.

Procedure.

- $(k[0], \cdots, k[t-1]) \coloneqq f(k)$
- $(\mathsf{shift}, K_0, \cdots, K_{t-1}) \coloneqq \mathsf{BatchfAuth}_{\mathsf{G}}^{\mathsf{H}}(\mathsf{msk}, \Delta, K, (\mathsf{Bit}_i \circ f \circ \mathsf{toPoly}_N)_{0 \le i \le t-1})$

Output. $((k[i], K_i)_{i \le t-1}, \mathsf{shift})$

Algorithm 7: Garbler procedure for unpacking the result of a batch evaluation. The procedure toPoly_N returns an element of $\mathbb{N}[X]$, but in our context, toPoly_N will always take as input an integer embedding of a polynomial in $\mathbb{F}_2[X]/P(X)$, hence its output is guaranteed to be a polynomial in $\mathbb{F}_2[X; m]$ with coefficients in $\{0, 1\}$. We interpret this polynomial as an element of $\mathbb{F}_{2^m}$ when evaluating f.

Algorithm $\mathsf{Unpack}_{\mathsf{E}}^{\mathsf{H}}(f, \mathsf{mpk}, \ell, L, \mathsf{salt}, \mathsf{shift})$

Input. Function $f \in \{\Phi^{-1}, \Psi\}$. Master public key mpk, packed labels $(\ell, L) \in \mathbb{F}_{2^m} \times \mathbb{Z}_{p-1}$, salt salt, garbling material shift.

Procedure.

- $(\ell[0], \cdots, \ell[t-1]) \coloneqq f(\ell)$
- $(L_0, \cdots, L_{t-1}) \coloneqq \mathsf{BatchfAuth}_{\mathsf{E}}^{\mathsf{H}}(\mathsf{mpk}, \ell(N), L, (\mathsf{Bit}_i \circ f \circ \mathsf{toPoly}_N)_{0 \le i \le t-1})$

Output. $(\ell[i], L_i)_{i \le t-1}$

Algorithm 8: Evaluator procedure for unpacking the result of a batch evaluation. The procedure toPoly_N returns an element of $\mathbb{N}[X]$, but in our context, toPoly_N will always take as input an integer embedding of a polynomial in $\mathbb{F}_2[X]/P(X)$, hence its output is guaranteed to always be a polynomial in $\mathbb{F}_2[X; m]$. We interpret this polynomial as an element of $\mathbb{F}_{2^m}$ when evaluating f.

Lemma 17 (Correctness of unpacking). *Fix* $f \in \{\Phi^{-1}, \Psi\}$ *and* $(\mathsf{mpk}, \mathsf{msk}, (k, K, \ell, L), \mathsf{salt})$ *where* $\mathsf{msk} \coloneqq (\mathsf{mpk}, \Delta, h_0)$. *Assume that* $L - K = \Delta \cdot \mathsf{Eval}_N(\ell) \bmod p-1$. *Then, denoting*

$$\begin{aligned} x &\coloneqq k + \ell \qquad \triangleright \textit{ over } \mathbb{F}_{2^m} \\ ((k[i], K[i])_{i \le t-1}, \mathsf{shift}) &\coloneqq \mathsf{Unpack}_{\mathsf{G}}^{\mathsf{H}}(f, \mathsf{msk}, k, K, \mathsf{salt}) \\ (\ell[i], L[i])_{i \le t-1} &\coloneqq \mathsf{Unpack}_{\mathsf{E}}^{\mathsf{H}}(f, \mathsf{mpk}, \ell, L, \mathsf{salt}, \mathsf{shift}), \end{aligned}$$

it holds that

$$(k[i] \oplus \ell[i])_{i \le t-1} = f(x)$$
$$(K[i] - L[i])_{i \le t-1} = (\Delta \cdot \ell[i] \bmod p-1)_{i \le t-1}$$

Proof. The first part of Lemma 17 follows immediately from the linearity of $f : \mathbb{F}_{2^m} \longrightarrow \mathbb{F}_2^t$. As for the second part, since we have $L - K = \Delta \cdot \mathsf{Eval}_N(\ell) \bmod p-1$ by assumption, the conditions of Lemma 13 are satisfied. Applying Lemma 13 with $q = t$, we get for $i = 0$ to $t-1$:

$$\begin{aligned} L[i] - K[i] &= \Delta \cdot \mathsf{Bit}_i(f(\mathsf{toPoly}_N(\mathsf{Eval}_N(\ell)))) \bmod p-1 \\ &= \Delta \cdot \mathsf{Bit}_i(f(\ell)) = \Delta \cdot \ell[i] \bmod p-1 \end{aligned}$$

∎

Eventually, we define $\mathsf{UnpackXOR}^{\mathsf{H}}$ and $\mathsf{UnpackAND}^{\mathsf{H}}$ as the above general procedures with f set to either Φ^{-1} or Ψ and hardcoded in the function:

- $\mathsf{UnpackXOR}^{\mathsf{H}}_{\mathsf{G}}(\mathsf{msk}, k, K, \mathsf{salt}) \coloneqq \mathsf{Unpack}^{\mathsf{H}}_{\mathsf{G}}(\Phi^{-1}, \mathsf{msk}, k, K, \mathsf{salt})$
- $\mathsf{UnpackXOR}^{\mathsf{H}}_{\mathsf{E}}(\mathsf{mpk}, \ell, L, \mathsf{salt}, \mathsf{shift}) \coloneqq \mathsf{Unpack}^{\mathsf{H}}_{\mathsf{E}}(\Phi^{-1}, \mathsf{mpk}, \ell, L, \mathsf{salt}, \mathsf{shift})$
- $\mathsf{UnpackAND}^{\mathsf{H}}_{\mathsf{G}}(\mathsf{msk}, k, K, \mathsf{salt}) \coloneqq \mathsf{Unpack}^{\mathsf{H}}_{\mathsf{G}}(\Psi, \mathsf{msk}, k, K, \mathsf{salt})$
- $\mathsf{UnpackAND}^{\mathsf{H}}_{\mathsf{E}}(\mathsf{mpk}, \ell, L, \mathsf{salt}, \mathsf{shift}) \coloneqq \mathsf{Unpack}^{\mathsf{H}}_{\mathsf{E}}(\Psi, \mathsf{mpk}, \ell, L, \mathsf{salt}, \mathsf{shift})$

6.5 Batch-XOR Gadget

The procedures below are used by the garbler and the evaluator, who hold as inputs $\mathbb{F}_{2^m}$-shares $(k_{\mathsf{l}}, \ell_{\mathsf{l}})$ of $x_{\mathsf{l}} \in \mathbb{F}_{2^m}$, $\mathbb{F}_{2^m}$-shares $(k_{\mathsf{r}}, \ell_{\mathsf{r}})$ of $x_{\mathsf{r}} \in \mathbb{F}_{2^m}$, and $\mathbb{Z}_{p-1}$-shares $(K_{\mathsf{l}}, L_{\mathsf{l}})$ and $(K_{\mathsf{r}}, L_{\mathsf{r}})$ of $\Delta \cdot x_{\mathsf{l}}(N)$ and $\Delta \cdot x_{\mathsf{r}}(N)$ respectively. The procedure outputs $\mathbb{F}_{2^m}$-shares $(k_{\mathsf{out}}, \ell_{\mathsf{out}})$ of $x_{\mathsf{l}} + x_{\mathsf{r}}$, and $\mathbb{Z}_{p-1}$-shares $(K_{\mathsf{out}}, L_{\mathsf{out}})$ of $\Delta \cdot \mathsf{Eval}_N(x_{\mathsf{l}} \oplus x_{\mathsf{r}})$. Let $S_{N,m}$ denote the set of sums of two elements from $\mathsf{I}_{N,m}$.

Algorithm $\mathsf{BatchXOR}^{\mathsf{H}}_{\mathsf{G}}(\mathsf{msk}, (k_{\mathsf{l}}, K_{\mathsf{l}}), (k_{\mathsf{r}}, K_{\mathsf{r}}), \mathsf{salt})$

Input. Master secret key msk. Packed keys $(k_{\mathsf{l}}, K_{\mathsf{l}}), (k_{\mathsf{r}}, K_{\mathsf{r}})$. Salt salt. Parse $\mathsf{msk} \coloneqq (\mathsf{mpk}, \Delta, h_0)$.

Procedure.

- $k_{\mathsf{out}} \coloneqq k_{\mathsf{l}} + k_{\mathsf{r}}$ ▷ Over $\mathbb{F}_{2^m}$
- $\tilde{K}_{\mathsf{out}} \coloneqq K_{\mathsf{l}} + K_{\mathsf{r}} \bmod p-1$
- $(\mathsf{shift}, K_{\mathsf{out}}) \coloneqq S_{N,m}\text{-}\mathsf{BatchfAuth}^{\mathsf{H}}_{\mathsf{G}}(\mathsf{msk}, \Delta, \tilde{K}_{\mathsf{out}}, \mathsf{Mod}_N(\cdot, 2), \mathsf{salt})$

Output. $((k_{\mathsf{out}}, K_{\mathsf{out}}), \mathsf{shift})$

Algorithm 9: Garbler batch-XOR gadget

Algorithm $\mathsf{BatchXOR}^{\mathsf{H}}_{\mathsf{E}}(\mathsf{mpk}, (\ell_{\mathsf{l}}, L_{\mathsf{l}}), (\ell_{\mathsf{r}}, L_{\mathsf{r}}), \mathsf{salt}, \mathsf{shift})$

Input. Master public key mpk. Packed keys $(k_{\mathsf{l}}, K_{\mathsf{l}}), (k_{\mathsf{r}}, K_{\mathsf{r}})$. Salt salt, garbling material shift.

Procedure.

- $\tilde{\ell}_{\mathrm{out}} \coloneqq \ell_{\mathsf{l}} + \ell_{\mathsf{r}}$ $\quad \triangleright$ Over $\mathbb{Z}[X]$
- $\ell_{\mathrm{out}} \coloneqq \tilde{\ell}_{\mathrm{out}} \bmod 2$ $\quad \triangleright$ $\ell_{\mathrm{out}} \in \mathbb{F}_{2^m}$
- $\tilde{L}_{\mathrm{out}} \coloneqq L_{\mathsf{l}} + L_{\mathsf{r}} \bmod p-1$
- $L_{\mathrm{out}} \coloneqq S_{N,m}\text{-}\mathsf{BatchfAuth}^{\mathsf{H}}_{\mathsf{E}}(\mathsf{mpk}, \tilde{\ell}_{\mathrm{out}}(N), \tilde{L}_{\mathrm{out}}, \mathsf{Mod}_N(\cdot, 2), \mathsf{salt}, \mathsf{shift})$

Output. $(\ell_{\mathrm{out}}, L_{\mathrm{out}})$

Algorithm 10: Evaluator batch-XOR gadget

Lemma 18 (Correctness of batch-XOR). *Fix* $(\mathsf{mpk}, \mathsf{msk}, (k_{\mathsf{l}}, K_{\mathsf{l}}, \ell_{\mathsf{l}}, L_{\mathsf{l}}), (k_{\mathsf{r}}, K_{\mathsf{r}}, \ell_{\mathsf{r}}, L_{\mathsf{r}}), \mathsf{salt})$ *where* $\mathsf{msk} \coloneqq (\mathsf{mpk}, \Delta, h_0)$. *Assume that for* $u \in \{\mathsf{l}, \mathsf{r}\}$, $L_u - K_u = \Delta \cdot \mathsf{Eval}_N(\ell_u) \bmod p-1$. *Then, denoting*

$$
\begin{aligned}
x_u &\coloneqq k_u + \ell_u \text{ for } u \in \{\mathsf{l}, \mathsf{r}\} \qquad \triangleright \text{ over } \mathbb{F}_{2^m}\\
((k_{\mathrm{out}}, K_{\mathrm{out}}), \mathsf{shift}) &\coloneqq \mathsf{BatchXOR}^{\mathsf{H}}_{\mathsf{G}}(\mathsf{msk}, (k_{\mathsf{l}}, K_{\mathsf{l}}), (k_{\mathsf{r}}, K_{\mathsf{r}}), \mathsf{salt})\\
(\ell_{\mathrm{out}}, L_{\mathrm{out}}) &\coloneqq \mathsf{BatchXOR}^{\mathsf{H}}_{\mathsf{E}}(\mathsf{mpk}, (\ell_{\mathsf{l}}, L_{\mathsf{l}}), (\ell_{\mathsf{r}}, L_{\mathsf{r}}), \mathsf{salt}, \mathsf{shift}),
\end{aligned}
$$

it holds that

$$
\begin{aligned}
k_{\mathrm{out}} + \ell_{\mathrm{out}} &= x_{\mathsf{l}} + x_{\mathsf{r}} \qquad \triangleright \text{ over } \mathbb{F}_{2^m}\\
K_{\mathrm{out}} - L_{\mathrm{out}} &= \Delta \cdot \mathsf{Eval}_N(\ell_{\mathrm{out}})) \bmod p-1
\end{aligned}
$$

Proof. The first part of Lemma 18 follows immediately from the fact that $\ell_{\mathrm{out}} = \tilde{\ell}_{\mathrm{out}} \bmod 2$ with $\tilde{\ell}_{\mathrm{out}} = \ell_{\mathsf{l}} + \ell_{\mathsf{r}}$ over $\mathbb{Z}[X]$, hence $\ell_{\mathrm{out}} = \ell_{\mathsf{l}} + \ell_{\mathsf{r}}$ over $\mathbb{F}_{2^m}$. As for the second part, since we have

$$
\begin{aligned}
\tilde{L}_{\mathrm{out}} - \tilde{K}_{\mathrm{out}} &= (L_{\mathsf{l}} - K_{\mathsf{l}}) + (L_{\mathsf{r}} - K_{\mathsf{r}}) \bmod p-1\\
&= \Delta \cdot (\mathsf{Eval}_N(\ell_{\mathsf{l}}) + \mathsf{Eval}_N(\ell_{\mathsf{r}})) \bmod p-1 \qquad \triangleright \text{ by assumption of Lemma 18}\\
&= \Delta \cdot \mathsf{Eval}_N(\ell_{\mathsf{l}} + \ell_{\mathsf{r}}) \bmod p-1 \qquad \triangleright \text{ sum over } \mathbb{Z}[X], \text{ since } N \gg 2\\
&= \Delta \cdot \tilde{\ell}_{\mathrm{out}}(N) \bmod p-1
\end{aligned}
$$

hence the conditions of Lemma 13 are satisfied. Applying Lemma 13, we get:

$$
\begin{aligned}
L_{\mathrm{out}} - K_{\mathrm{out}} &= \Delta \cdot \mathsf{Mod}_N(\tilde{\ell}_{\mathrm{out}}(N), 2) \bmod p-1\\
&= \Delta \cdot \mathsf{Eval}_N(\ell_{\mathrm{out}}) \bmod p-1 \qquad \triangleright \text{ by definition of } \mathsf{Mod}_N(\cdot, 2).
\end{aligned}
$$

■

6.6 Batch-AND Gadget

Algorithm $\mathsf{BatchAND}_{\mathsf{G}}^{\mathsf{H}}(\mathsf{msk}, (k_{\mathsf{l}}, K_{\mathsf{l}}), (k_{\mathsf{r}}, K_{\mathsf{r}}), \mathsf{salt})$

Input. Master secret key msk. Packed keys $(k_{\mathsf{l}}, K_{\mathsf{l}}), (k_{\mathsf{r}}, K_{\mathsf{r}})$. Salt salt. Parse $\mathsf{msk} \coloneqq (\mathsf{mpk}, \Delta, h_0)$. For $i = 1$ to 6, let $\mathsf{salt}_i \coloneqq \mathsf{salt}||i$.

Procedure.

- $(k'_{\mathsf{l}}, k'_{\mathsf{r}}) \leftarrow\$ \mathsf{Pert}_c(k_{\mathsf{l}}) \times \mathsf{Pert}_c(k_{\mathsf{r}})$
- $(\mathsf{shift}_a, \tilde{a}_{\mathsf{G}}) \coloneqq \mathbb{Z}\text{-}\mathsf{VtO}_{\mathsf{G}}^{\mathsf{H}_0}(\mathsf{msk}, k'_{\mathsf{l}}(N), K_{\mathsf{r}}, \mathsf{salt}_1)$
  $\triangleright\ \tilde{a}_{\mathsf{G}} = \langle k'_{\mathsf{l}}(N)\ell_{\mathsf{r}}(N)\rangle_{\mathsf{G}}$
- $(\mathsf{shift}_b, \tilde{b}_{\mathsf{G}}) \coloneqq \mathbb{Z}\text{-}\mathsf{VtO}_{\mathsf{G}}^{\mathsf{H}_0}(\mathsf{msk}, k'_{\mathsf{r}}(N), K_{\mathsf{l}}, \mathsf{salt}_2)$
  $\triangleright\ \tilde{b}_{\mathsf{G}} = \langle k'_{\mathsf{r}}(N)\ell_{\mathsf{l}}(N)\rangle_{\mathsf{G}}$
- $a_{\mathsf{G}} \coloneqq [\mathsf{toPoly}_N(\tilde{a}_{\mathsf{G}}) \bmod 2, P]$ $\quad\triangleright\ a_{\mathsf{G}} \in \mathbb{F}_2[X]/P(X) \cong \mathbb{F}_{2^m}$
- $b_{\mathsf{G}} \coloneqq [\mathsf{toPoly}_N(\tilde{b}_{\mathsf{G}}) \bmod 2, P]$ $\quad\triangleright\ b_{\mathsf{G}} \in \mathbb{F}_2[X]/P(X) \cong \mathbb{F}_{2^m}$
- $k_{\mathsf{out}} \coloneqq k_{\mathsf{l}}k_{\mathsf{r}} + a_{\mathsf{G}} + b_{\mathsf{G}}$ $\quad\triangleright$ over $\mathbb{F}_{2^m}$
- $(\mathsf{shift}_\alpha, \alpha_{\mathsf{G}}) \coloneqq \mathsf{VtO}_{\mathsf{G}}^{\mathsf{H}}(\mathsf{msk}, \Delta k'_{\mathsf{l}}(N), K_{\mathsf{r}}, \mathsf{salt}_3)$
  $\triangleright\ \alpha_{\mathsf{G}} = \langle \Delta k'_{\mathsf{l}}(N) \cdot \ell_{\mathsf{r}}(N)\rangle_{\mathsf{G}}$
- $(\mathsf{shift}_\beta, \beta_{\mathsf{G}}) \coloneqq \mathsf{VtO}_{\mathsf{G}}^{\mathsf{H}}(\mathsf{msk}, \Delta k'_{\mathsf{r}}(N), K_{\mathsf{l}}, \mathsf{salt}_4)$
  $\triangleright\ \beta_{\mathsf{G}} = \langle \Delta k'_{\mathsf{r}}(N) \cdot \ell_{\mathsf{l}}(N)\rangle_{\mathsf{G}}$
- $(\mathsf{shift}_\gamma, \gamma_{\mathsf{G}}) \coloneqq \mathsf{VtO}_{\mathsf{G}}^{\mathsf{H}}(\mathsf{msk}, K_{\mathsf{l}}, K_{\mathsf{r}}, \mathsf{salt}_5)$ $\quad\triangleright\ \gamma_{\mathsf{G}} = \langle K_{\mathsf{l}} \cdot \ell_{\mathsf{r}}(N)\rangle_{\mathsf{G}}$
- $\tilde{K}_{\mathsf{out}} \coloneqq \alpha_{\mathsf{G}} + \beta_{\mathsf{G}} - \gamma_{\mathsf{G}} + \Delta(\tilde{a}_{\mathsf{G}} + \tilde{b}_{\mathsf{G}})$ $\quad\triangleright\ \tilde{K}_{\mathsf{out}} = \langle \Delta\tilde{\ell}_{\mathsf{out}}\rangle_{\mathsf{G}}$
- $(\mathsf{shift}_K, K_{\mathsf{out}}) \coloneqq [N^m]\text{-}\mathsf{BatchfAuth}_{\mathsf{G}}^{\mathsf{H}}(\mathsf{msk}, \Delta, \tilde{K}_{\mathsf{out}}, \mathsf{Mod}_N(\cdot, 2, P), \mathsf{salt}_6)$ $\quad\triangleright\ K_{\mathsf{out}} = \langle \Delta\ell_{\mathsf{out}}\rangle_{\mathsf{G}}$
- $S_{\mathsf{out}} \coloneqq (\mathsf{shift}_a, \mathsf{shift}_b, \mathsf{shift}_\alpha, \mathsf{shift}_\beta, \mathsf{shift}_\gamma, \mathsf{shift}_K)$

Output. $((k_{\mathsf{out}}, K_{\mathsf{out}}), S_{\mathsf{out}})$

Algorithm 11: Garbler batch-AND gadget. Unlike the other gadgets, $\mathsf{BatchAND}_{\mathsf{G}}^{\mathsf{H}}$ is a randomized procedure.

Algorithm $\mathsf{BatchAND}_{\mathsf{E}}^{\mathsf{H}}(\mathsf{msk}, (\ell_{\mathsf{l}}, L_{\mathsf{l}}), (\ell_{\mathsf{r}}, L_{\mathsf{r}}), \mathsf{salt}, S)$

Inputs. Master public key mpk. Packed labels $(\ell_{\mathsf{l}}, L_{\mathsf{l}}), (\ell_{\mathsf{r}}, L_{\mathsf{r}})$, salt salt and garbling material S. Parse $S \coloneqq (\mathsf{shift}_a, \mathsf{shift}_b, \mathsf{shift}_\alpha, \mathsf{shift}_\beta, \mathsf{shift}_\gamma, \mathsf{shift}_K)$. For $i = 1$ to 6, let $\mathsf{salt}_i \coloneqq \mathsf{salt}||i$.

Procedure.

- $\tilde{a}_{\mathsf{E}} \coloneqq \mathbb{Z}\text{-}\mathsf{VtO}_{\mathsf{E}}^{\mathsf{H}_0}(\mathsf{mpk}, \ell_{\mathsf{r}}(N), L_{\mathsf{r}}, \mathsf{salt}_1, \mathsf{shift}_a)$
  $\triangleright\ \tilde{a}_{\mathsf{E}} = \langle k'_{\mathsf{l}}(N)\ell_{\mathsf{r}}(N)\rangle_{\mathsf{E}}$
- $\tilde{b}_{\mathsf{E}} \coloneqq \mathbb{Z}\text{-}\mathsf{VtO}_{\mathsf{E}}^{\mathsf{H}_0}(\mathsf{mpk}, \ell_{\mathsf{l}}(N), L_{\mathsf{l}}, \mathsf{salt}_2, \mathsf{shift}_b)$ $\quad\triangleright\ \tilde{b}_{\mathsf{E}} = \langle k'_{\mathsf{r}}(N)\ell_{\mathsf{l}}(N)\rangle_{\mathsf{E}}$
- $\tilde{\ell}_{\mathsf{out}} \coloneqq \ell_{\mathsf{l}}(N)\ell_{\mathsf{r}}(N) + \tilde{a}_{\mathsf{E}} + \tilde{b}_{\mathsf{E}}$
- $a_{\mathsf{E}} \coloneqq [\mathsf{toPoly}_N(\tilde{a}_{\mathsf{E}}) \bmod 2, P]$ $\quad\triangleright\ a_{\mathsf{E}} \in \mathbb{F}_2[X]/P(X) \cong \mathbb{F}_{2^m}$

- $b_\mathsf{E} \coloneqq [\mathsf{toPoly}_N(\tilde{b}_\mathsf{E}) \bmod 2, P]$ $\quad \triangleright\ b_\mathsf{E} \in \mathbb{F}_2[X]/P(X) \cong \mathbb{F}_{2^m}$
- $\ell_\mathsf{out} \coloneqq \ell_\mathsf{l}\ell_\mathsf{r} + a_\mathsf{E} + b_\mathsf{E}$ $\quad \triangleright\ \ell_\mathsf{out} = \mathsf{Mod}_N(\tilde{\ell}_\mathsf{out}) \in \mathbb{F}_{2^m}$
- $\alpha_\mathsf{E} \coloneqq \mathsf{VtO}^\mathsf{H}_\mathsf{E}(\mathsf{mpk}, \ell_\mathsf{r}(N), L_\mathsf{r}, \mathsf{salt}_3, \mathsf{shift}_\alpha)$
  $\triangleright\ \alpha_\mathsf{E} = \langle \Delta k'_\mathsf{l}(N) \cdot \ell_\mathsf{r}(N) \rangle_\mathsf{E}$
- $\beta_\mathsf{E} \coloneqq \mathsf{VtO}^\mathsf{H}_\mathsf{E}(\mathsf{mpk}, \ell_\mathsf{l}(N), L_\ell, \mathsf{salt}_4, \mathsf{shift}_\beta)$
  $\triangleright\ \beta_\mathsf{E} = \langle \Delta k'_\mathsf{r}(N) \cdot \ell_\mathsf{l}(N) \rangle_\mathsf{E}$
- $\gamma_\mathsf{E} \coloneqq \mathsf{VtO}^\mathsf{H}_\mathsf{E}(\mathsf{mpk}, \ell_\mathsf{r}(N), L_\mathsf{r}, \mathsf{salt}_5, \mathsf{shift}_\gamma)$ $\quad \triangleright\ \gamma_\mathsf{E} = \langle K_\mathsf{l} \cdot \ell_\mathsf{r}(N) \rangle_\mathsf{E}$
- $\tilde{L}_\mathsf{out} \coloneqq \alpha_\mathsf{E} + \beta_\mathsf{E} - \gamma_\mathsf{E} + L_\mathsf{l}\ell_\mathsf{r}(N)$ $\quad \triangleright\ \tilde{L}_\mathsf{out} = \langle \Delta\tilde{\ell}_\mathsf{out} \rangle_\mathsf{E}$
- $L_\mathsf{out} \coloneqq [N^m]\text{-}\mathsf{BatchfAuth}^\mathsf{H}_\mathsf{E}(\mathsf{mpk}, \tilde{\ell}_\mathsf{out}, \tilde{L}_\mathsf{out}, \mathsf{Mod}_N(\cdot, 2, P), \mathsf{salt}_6, \mathsf{shift}_K)$ $\quad \triangleright\ L_\mathsf{out} = \langle \Delta\ell_\mathsf{out} \rangle_\mathsf{E}$

Output. $(\ell_\mathsf{out}, L_\mathsf{out})$

Algorithm 12: Evaluator batch-AND gadget

Lemma 19 (Correctness of batch-AND)**.** *Fix* $(\mathsf{mpk}, \mathsf{msk}, (k_\mathsf{l}, K_\mathsf{l}, \ell_\mathsf{l}, L_\mathsf{l}), (k_\mathsf{r}, K_\mathsf{r}, \ell_\mathsf{r}, L_\mathsf{r}), \mathsf{salt})$ *where* $\mathsf{msk} \coloneqq (\mathsf{mpk}, \Delta, h_0)$. *Assume that for* $u \in \{\mathsf{l}, \mathsf{r}\}$, $L_u - K_u = \Delta \cdot \mathsf{Eval}_N(\ell_u) \bmod p-1$. *Then, denoting*

$$
\begin{aligned}
x_u &\coloneqq k_u + \ell_u \text{ for } u \in \{\mathsf{l}, \mathsf{r}\} \quad \triangleright \text{ over } \mathbb{F}_{2^m} \\
((k_\mathsf{out}, K_\mathsf{out}), S) &\coloneqq \mathsf{BatchAND}^\mathsf{H}_\mathsf{G}(\mathsf{msk}, (k_\mathsf{l}, K_\mathsf{l}), (k_\mathsf{r}, K_\mathsf{r}), \mathsf{salt}) \\
(\ell_\mathsf{out}, L_\mathsf{out}) &\coloneqq \mathsf{BatchAND}^\mathsf{H}_\mathsf{E}(\mathsf{mpk}, (\ell_\mathsf{l}, L_\mathsf{l}), (\ell_\mathsf{r}, L_\mathsf{r}), \mathsf{salt}, S),
\end{aligned}
$$

it holds that

$$
\begin{aligned}
k_\mathsf{out} + \ell_\mathsf{out} &= x_\mathsf{l} \cdot x_\mathsf{r} \quad \triangleright \text{ over } \mathbb{F}_{2^m} \\
K_\mathsf{out} - L_\mathsf{out} &= \Delta \cdot \mathsf{Eval}_N(\ell_\mathsf{out})) \bmod p-1
\end{aligned}
$$

Proof. Using the assumptions of Lemma 19, the conditions of Lemma 11 are satisfied. Hence, we have by Lemma 11 that

$$
\begin{aligned}
\tilde{a}_\mathsf{E} - \tilde{a}_\mathsf{G} &= \ell_\mathsf{r}(N) \cdot k'_\mathsf{l}(N) \\
\tilde{b}_\mathsf{E} - \tilde{b}_\mathsf{G} &= \ell_\mathsf{l}(N) \cdot k'_\mathsf{r}(N).
\end{aligned}
$$

Then since $N > c \cdot m$, using Lemma 8 with $T = 1$, we get

$$
\begin{aligned}
\ell_\mathsf{r} \cdot k_\mathsf{l} &\coloneqq \mathsf{toPoly}_N(\mathsf{Mod}_N(\ell_\mathsf{r}(N) \cdot k'_\mathsf{l}(N), 2)) \\
\ell_\mathsf{l} \cdot k_\mathsf{r} &\coloneqq \mathsf{toPoly}_N(\mathsf{Mod}_N(\ell_\mathsf{l}(N) \cdot k'_\mathsf{r}(N), 2)),
\end{aligned}
$$

where the products on the left are computed over $\mathbb{F}_2[X]$. Furthermore, by definition of VtO_G, we have

$$
\begin{aligned}
\tilde{a}_\mathsf{G} &= \sum_{x \in \mathsf{I}_{N,m}} x \cdot \mathsf{F}^{\mathsf{H}_0}(\mathsf{k}, x, \mathsf{salt}_1) \\
\tilde{b}_\mathsf{G} &= \sum_{x \in \mathsf{I}_{N,m}} x \cdot \mathsf{F}^{\mathsf{H}_0}(\mathsf{k}, x, \mathsf{salt}_2),
\end{aligned}
$$

where each term $\mathsf{F}^{\mathsf{H}_0}(\mathsf{k}, x, \mathsf{salt}_i)$ is a term of the form $\mathsf{map}\left(\mathsf{H}\left(h_x^{\mathsf{sk}}, \mathsf{salt}_i\right)\right)$. By definition of map, each such term is of the form $r(N)$ where $r \in \mathbb{Z}[X; m]$ is a polynomial with coefficients bounded by $\|r\|_\infty \le c$. Then, applying Lemma 8 using $T = |\mathsf{I}_{N,m}|+1 = 2^m+1$ (which is possible because $N = 2c\cdot m\cdot(2^m+1)+1 > c\cdot m\cdot T$), we get

$$\begin{aligned}\mathsf{toPoly}_N(\tilde{a}_\mathsf{E}) &= \mathsf{toPoly}_N(\ell_\mathsf{r}(N)\cdot k'_\mathsf{l}(N) + \tilde{a}_\mathsf{G}) = \ell_\mathsf{r}\cdot k'_\mathsf{l} + \mathsf{toPoly}_N(\tilde{a}_\mathsf{G})\\ \mathsf{toPoly}_N(\tilde{b}_\mathsf{E}) &= \mathsf{toPoly}_N(\ell_\mathsf{l}(N)\cdot k'_\mathsf{r}(N) + \tilde{b}_\mathsf{G}) = \ell_\mathsf{l}\cdot k'_\mathsf{r} + \mathsf{toPoly}_N(\tilde{b}_\mathsf{G}),\end{aligned}$$

and since the map $[\cdot, 2, P]$ is linear, we obtain

$$\begin{aligned}a_\mathsf{E} &= \ell_\mathsf{r}\cdot k_\mathsf{l} + a_\mathsf{G} && \triangleright \text{ over } \mathbb{F}_{2^m} = \mathbb{F}_2[X]/P(X)\\ b_\mathsf{E} &= \ell_\mathsf{l}\cdot k_\mathsf{r} + b_\mathsf{G} && \triangleright \text{ over } \mathbb{F}_{2^m} = \mathbb{F}_2[X]/P(X),\end{aligned}$$

and therefore (note that $+$ and $-$ coincide over $\mathbb{F}_{2^m}$)

$$\begin{aligned}\ell_\mathsf{out} + k_\mathsf{out} &= (k_\mathsf{r}k_\mathsf{l} + a_\mathsf{G} + b_\mathsf{G}) + (k_\mathsf{l}k_\mathsf{r} + a_\mathsf{E} + b_\mathsf{E})\\ &= k_\mathsf{r}k_\mathsf{l} + k_\mathsf{r}\ell_\mathsf{l} + k_\mathsf{l}\ell_\mathsf{r} + \ell_\mathsf{r}\ell_\mathsf{l}\\ &= (k_\mathsf{r}+\ell_\mathsf{r})\cdot(k_\mathsf{l}+\ell_\mathsf{l}) = x_\mathsf{l}\cdot x_\mathsf{r}. && \triangleright \text{ over } \mathbb{F}_{2^m} = \mathbb{F}_2[X]/P(X)\end{aligned}$$

Moving on to the last part of Lemma 19, we start by observing that

$$\begin{aligned}\ell_\mathsf{out} &= \ell_\mathsf{l}\ell_\mathsf{r} + a_\mathsf{E} + b_\mathsf{E}\\ &= [\mathsf{toPoly}_N(\ell_\mathsf{r}(N)\cdot k'_\mathsf{l}(N)) + \mathsf{toPoly}_N(\tilde{a}_\mathsf{E}) + \mathsf{toPoly}_N(\tilde{b}_\mathsf{E}) \bmod 2, P]\\ &= [\mathsf{toPoly}_N(\tilde{\ell}_\mathsf{out}) \bmod 2, P],\end{aligned}$$

where the second equality is by linearity of the map $[\cdot, 2, P]$, and the last is by applying Lemma 8 with $T = 2^{m+1}+1$, using this time the fact that $N = 2c\cdot m\cdot(2^m+1)+1 > c\cdot m\cdot T$. We therefore have $\ell_\mathsf{out}(N) = \mathsf{Mod}_N(\tilde{\ell}_\mathsf{out}, 2, P)$.

Then, using the assumptions of Lemma 19, the conditions of Lemma 11 are satisfied. Hence, we have by Lemma 11 that

$$\begin{aligned}\alpha_\mathsf{E} - \alpha_\mathsf{G} &= \ell_\mathsf{r}(N)\cdot(\Delta k'_\mathsf{l}(N))\\ \beta_\mathsf{E} - \beta_\mathsf{G} &= \ell_\mathsf{l}(N)\cdot(\Delta k'_\mathsf{r}(N))\\ \gamma_\mathsf{E} - \gamma_\mathsf{G} &= \ell_\mathsf{r}(N)\cdot K_\mathsf{l}.\end{aligned}$$

Then,

$$\begin{aligned}\tilde{L}_\mathsf{out} - \tilde{K}_\mathsf{out} &= \alpha_\mathsf{E} + \beta_\mathsf{E} - \gamma_\mathsf{E} + L_\mathsf{l}\ell_\mathsf{r}(N) - (\alpha_\mathsf{G} + \beta_\mathsf{G} - \gamma_\mathsf{G} + \Delta(\tilde{a}_\mathsf{G} + \tilde{b}_\mathsf{G}))\\ &= (\alpha_\mathsf{E}-\alpha_\mathsf{G}) + (\beta_\mathsf{E}-\beta_\mathsf{G}) - (\gamma_\mathsf{E}-\gamma_\mathsf{G}) + \Delta(\tilde{a}_\mathsf{G} + \tilde{b}_\mathsf{G}) + L_\mathsf{l}\ell_\mathsf{r}(N)\\ &= \Delta\cdot(\ell_\mathsf{r}(N)\cdot k'_\mathsf{l}(N) + \ell_\mathsf{l}(N)\cdot k'_\mathsf{r}(N)) - \ell_\mathsf{r}(N)\cdot K_\mathsf{l}\\ &\qquad + \Delta(\tilde{a}_\mathsf{E} - \ell_\mathsf{r}(N)\cdot k'_\mathsf{l}(N) + \tilde{b}_\mathsf{E} - \ell_\mathsf{l}(N)\cdot k'_\mathsf{r}(N)) + L_\mathsf{l}\ell_\mathsf{r}(N)\\ &= \Delta\cdot(\tilde{a}_\mathsf{E} + \tilde{b}_\mathsf{E}) + \ell_\mathsf{r}(N)\cdot(L_\mathsf{l} - K_\mathsf{l})\\ &= \Delta\cdot(\tilde{a}_\mathsf{E} + \tilde{b}_\mathsf{E} + \ell_\mathsf{r}(N)\cdot\ell_\mathsf{l}(N)) && \triangleright\ L_\mathsf{l} - K_\mathsf{l} = \Delta\cdot\ell_\mathsf{l}(N)\\ &= \Delta\cdot\tilde{\ell}_\mathsf{out},\end{aligned}$$

and the conditions of Lemma 13 are satisfied. Applying Lemma 13, we get:

$$\begin{aligned} L_{\text{out}} - K_{\text{out}} &= \Delta \cdot \mathsf{Mod}_N(\tilde{\ell}_{\text{out}}, 2, P) \bmod p-1 \\ &= \Delta \cdot \ell_{\text{out}}(N) \bmod p-1, \end{aligned}$$

which concludes the proof. ∎

7 Security Analysis

In this section, we prove the following theorem:

Theorem 20. *The garbling scheme of Section 6 is private.*

Fix a security parameter λ, a circuit C, and an input x. We prove privacy through a sequence of games. The initial game samples $(\hat{C}, \mathsf{e}, \mathsf{d}) \leftarrow\$ \mathsf{GC.Garble}(1^\lambda, C)$, $\hat{x} \leftarrow\$ \mathsf{GC.Enc}(\mathsf{e}, x_\lambda)$, and outputs $(\hat{C}, \hat{x}, \mathsf{d})$.

7.1 The Hybrids

Hybrid$_1$. In the first hybrid, we describe a simulator $\mathsf{SimGC}_1(1^\lambda, C, x)$. The simulator is obtained by making a few simple changes to GC.Garble:

- During the initialization phase of $\mathsf{GC.Garble}(1^\lambda, C)$, instead of sampling $(k_i, K_i) \leftarrow\$ \{0,1\} \times \mathbb{Z}_{p-1}$ for each input wire $i \le n$, SimGC_1 samples labels $(\ell_i, L_i) \leftarrow\$ \{0,1\} \times \mathbb{Z}_{p-1}$ for $i = 1$ to n and sets $(k, K_i) \coloneqq (\ell_i \oplus x_i, L_i - \Delta \cdot \ell_i \bmod p-1)$. This is the *only place* where SimGC_1 uses the input x. Observe that $(k_i, K_i)_{i \le n}$ is distributed exactly as in the initial game, and furthermore $\mathsf{GC.Enc}(\mathsf{e}, x) = (\ell_i, L_i)_{i \le n}$.
- SimGC_1 runs the rest of the simulation identically to GC.Garble to compute $(\hat{C}, \mathsf{d})$, and sets $\hat{x} \coloneqq (\ell_i, L_i)_{i \le n}$. It outputs $(\hat{C}, \hat{x}, \mathsf{d})$.

The change between Hybrid$_1$ and the initial game is purely cosmetic, and the games are distributed identically.

Hybrid$_2$. In this game, we replace $\mathsf{SimGC}_1(1^\lambda, C, x)$ with $\mathsf{SimGC}_2(1^\lambda, C, x)$. At a high level, SimGC_2 computes $(\hat{C}, \hat{x}, \mathsf{d})$ without using msk anymore. Instead, SimGC_2 receives mpk from the challenger for the security game of the TCCR hash for exponential correlations with $2N^m - 1$ auxiliary powers over $\mathbb{G}$ (Definition 7). Whenever msk is required in a procedure to compute a value, SimGC_2 computes the same value using instead a call to the oracle $O \coloneqq O_{\mathsf{H},\Delta,h_0}$ defined in Definition 7. While the full description of the game hop is involved (as it requires adapting all the subprocedures of GC.Garble), the change is purely syntactical: when executed with the same random tape R, $\mathsf{SimGC}_1(1^\lambda, C, x)$ and $\mathsf{SimGC}_2(1^\lambda, C, x)$ compute exactly the same functionality and produce the same output:

Lemma 21. *For every Boolean circuit C and input $x \in \{0,1\}^n$, for every random tape R, it holds that*

$$\mathsf{SimGC}_1(1^\lambda, C, x; R) = \mathsf{SimGC}_2(1^\lambda, C, x; R).$$

We defer the full description of SimGC_2 and the proof of Lemma 21 to Section 7.2.

Hybrid$_3$. In this game, the oracle O is replaced with a random oracle $\mathcal{R} : \mathbb{Z}_{p-1} \times \{0,1\}^* \times \mathbb{Z}_{p-1} \to \mathbb{Z}_{p-1}$. We denote SimGC_3 the resulting algorithm. We have the following immediate lemma:

$\mathsf{mSimVtO}^{\mathsf{H},\cancel{\mathcal{R}}}(\mathsf{mpk}, v_\mathsf{E}, \langle \Delta v_\mathsf{E} \rangle_\mathsf{E}, \cancel{a, b}, \mathsf{salt})$

1 : $\mathsf{psk} \coloneqq \mathsf{F.Punct}(\mathsf{mpk}, \langle \Delta v_\mathsf{E} \rangle_\mathsf{E})$
2 : $\mathsf{k}^* \coloneqq (\mathsf{mpk}, \mathsf{psk})$
3 : **for** $x \in \mathsf{I}_{N,m} \setminus \{v_\mathsf{E}\}$:
4 : $\quad y_x \coloneqq \mathsf{pF}^\mathsf{H}(\mathsf{k}^*, v_\mathsf{E}, x, \mathsf{salt})$
5 : $\mathsf{shift} \leftarrow\$ \mathbb{Z}_{p-1}$
6 : $z_\mathsf{E} \coloneqq \mathsf{VtO}^\mathsf{H}_\mathsf{E}(\mathsf{mpk}, v_\mathsf{E}, \langle \Delta v_\mathsf{E} \rangle_\mathsf{E}, \mathsf{salt}, \mathsf{shift})$
7 : **return** $(\mathsf{shift}, z_\mathsf{E})$

$\mathsf{mSim'VtO}^{\mathsf{H},\cancel{\mathcal{R}}}(\mathsf{mpk}, v_\mathsf{E}, \langle \Delta v_\mathsf{E} \rangle_\mathsf{E}, \mathsf{map}, \cancel{b,}\mathsf{salt})$

1 : $\mathsf{psk} \coloneqq \mathsf{F.Punct}(\mathsf{mpk}, \langle \Delta v_\mathsf{E} \rangle_\mathsf{E})$
2 : $\mathsf{k}^* \coloneqq (\mathsf{mpk}, \mathsf{psk})$
3 : **for** $x \in \mathsf{I}_{N,m} \setminus \{v_\mathsf{E}\}$:
4 : $\quad y_x \coloneqq \mathsf{map}\left(\mathsf{pF}^\mathsf{H}(\mathsf{k}^*, v_\mathsf{E}, x, \mathsf{salt})\right)$
5 : $s \leftarrow\$ \mathcal{D}_{m,c}$ $\triangleright$ defined in Lemma 8
6 : $\mathsf{shift} \coloneqq \sum_{x \in \mathsf{I}_{N,m} \setminus \{v_\mathsf{E}\}} y_x + s$
7 : $z_\mathsf{E} \coloneqq \mathsf{VtO}^{\mathsf{H}_0}_\mathsf{E}(\mathsf{mpk}, v_\mathsf{E}, \langle \Delta v_\mathsf{E} \rangle_\mathsf{E}, \mathsf{salt}, \mathsf{shift})$
8 : **return** $(\mathsf{shift}, z_\mathsf{E})$

$\mathsf{mSimBatchfAuth}^{\mathsf{H},\cancel{\mathcal{R}}}(\mathsf{mpk}, v_\mathsf{E}, \langle \Delta v_\mathsf{E} \rangle_\mathsf{E}, \mathsf{salt})$

1 : $\mathsf{psk} \coloneqq \mathsf{F.Punct}(\mathsf{mpk}, \langle \Delta v_\mathsf{E} \rangle_\mathsf{E})$
2 : $\mathsf{k}^* \coloneqq (\mathsf{mpk}, \mathsf{psk})$
3 : **for** $x \in \mathsf{I}_{N,m} \setminus \{v_\mathsf{E}\}$:
4 : $\quad y_x \coloneqq \mathsf{pF}^\mathsf{H}(\mathsf{k}^*, v_\mathsf{E}, x, \mathsf{salt})$
5 : $\mathsf{shift} \leftarrow\$ \mathbb{Z}_{p-1}$
6 : $z_\mathsf{E} \coloneqq \mathsf{BatchfAuth}^\mathsf{H}_\mathsf{E}(\mathsf{mpk}, v_\mathsf{E}, \langle \Delta v_\mathsf{E} \rangle_\mathsf{E}, \mathsf{salt}, \mathsf{shift})$
7 : **return** $(\mathsf{shift}, z_\mathsf{E})$

Figure 1: Modifications of SimVtO, Sim′VtO, and SimBatchfAuth used by SimGC_4, with changes compared to the original procedures highlighted in red.

Lemma 22. *Assume that* H *is a TCCR hash for exponential correlations with* $2N^m - 1$ *auxiliary powers over* $\mathbb{G}$ *(Definition 7). Then* Hybrid_2 *is computationally indistinguishable from* Hybrid_1.

The straightforward reduction is given access to an oracle $\mathcal{O}_?$ that is either $\mathcal{O}$ or $\mathcal{R}$, and given a distinguisher for the TCCR game with advantage ε, runs $\mathsf{SimGC}_2^{\mathsf{H},\mathcal{O}_?}$ and distinguishes between Hybrid_2 and Hybrid_3 with advantage exactly ε. Note that the condition that all queries (x, y, z) to $\mathcal{O}$ have distinct (x, y) is enforced by the fact that all calls to $\mathcal{O}$ in all procedures used by SimGC_2 use a unique and distinct salt.

Hybrid_4. In this game, we define the algorithm SimGC_4 identically to SimGC_3, except that we modify the procedures SimVtO, Sim′VtO, and SimBatchfAuth that it uses internally. We replace them with *modified procedures*, where the changes compared to the original procedures are highlighted in red, represented on Figure 1

- On line 5 of $\mathsf{mSimVtO}^\mathsf{H}$, shift is sampled uniformly at random. This is distributed exactly as in $\mathsf{SimVtO}^{\mathsf{H},\mathcal{R}}$, where shift is masked by the output of $\mathcal{R}$ on a query that involves a unique salt (hence is never repeated).
- On line 6 of $\mathsf{mSim'VtO}^\mathsf{H}$, shift is computed as $\sum_{x \in \mathsf{I}_{N,m} \setminus \{v_\mathsf{E}\}} \mathsf{map}(\mathsf{pF}^\mathsf{H}(\mathsf{k}^*, v_\mathsf{E}, x, \mathsf{salt})) + s$, where $s \leftarrow\$ \mathcal{D}_{t,c}$. In $\mathsf{Sim'VtO}^{\mathsf{H},\mathcal{R}}$, the only difference is that s is computed as $\mathsf{map}\left(\mathcal{O}(\mathsf{psk}, \mathsf{salt}, 0)\right) + b$. As salt is a unique

salt, map $(O(\mathsf{psk}, \mathsf{salt}, 0)) + b$ is distributed as a random sample from $\mathcal{D}_{t,c}^{(b)}$. By Lemma 8, the statistical distance between $\mathcal{D}_{t,c}^{(b)}$ and $\mathcal{D}_{t,c}$ is at most $t/2^c$.

- Eventually, on line 5 of $\mathsf{mSimBatchfAuth}^{\mathsf{H}}$, shift is sampled uniformly at random. This is distributed exactly as in $\mathsf{SimBatchfAuth}^{\mathsf{H},\mathcal{R}}$, where shift is masked by the output of $\mathcal{R}$ on a query that involves a unique salt.

To conclude, we observe that the procedures SimVtO, Sim′VtO, and SimBatchfAuth are called at most 9 times in total for a given batch, of which there are at most $|C|/t$. We therefore have the following lemma:

Lemma 23. *No (possibly unbounded) distinguisher can distinguish between* Hybrid_3 *and* Hybrid_4 *with advantage better than* $t \cdot |C|/(t \cdot 2^c)$.

Hybrid_5. In this game, we observe that from Hybrid_4, the values k_u are not required anymore by SimGC_4, because the modified procedures mSimVtO and mSim′VtO do not expects inputs (a, b) anymore. Formally, we define SimGC_5 as SimGC_4 by removing the parts denoted in blue from all procedures described in Section 7.2. Then, to compute d, it sets

$$\mathsf{d} \coloneqq (\ell_o \oplus y_o)_{o \in O(C)}),$$

where $y = C(x)$. Using the invariant lemmas for each of the subprocedures (Lemma 26, Lemma 27, Lemma 28, and Lemma 29), it follows from an analysis identical to the correctness analysis in Section 6.2 that $(\ell_o \oplus y_o)_{o \in O(C)})$ is identical to the values $(k_o)_{o \in O(C)}$ computed by SimGC_4, hence Hybrid_5 is perfectly indistinguishable from Hybrid_4. Notice that SimGC_5 does not use the input x anymore; this concludes the proof of Theorem 20.

7.2 Lemmas and Proofs for Hybrid_2

As for GC.Garble, we first describe the high-level structure of SimGC_2, and the main lemma, before introducing and analyzing the subprocedures it uses.

Algorithm $\mathsf{SimGC}_2(1^\lambda, C, x)$

Input. A Boolean circuit C with $|C| = s$ gates and depth $\mathsf{depth}(C) = D$ represented as described in Section 6.1; an input $x \in \{0,1\}^{|I(C)|}$.

Initialization. The parts in blue are omitted in Hybrid_5.

- Receive $(h_i)_{i \in [\pm N^m] \setminus \{0\}}$ from a challenger for the security game of the TCCR hash for exponential correlations with $2N^m - 1$ auxiliary powers over $\mathbb{G}$ (Definition 7).
- Set $\mathsf{mpk} \coloneqq \left(\mathbb{G}, p, G, (h_i)_{i \in [\pm N^m] \setminus \{0\}}\right)$
- Let $O \coloneqq O_{\mathsf{H}, h_0, \Delta}$ denote the oracle of the TCCR security game.
- For each input wire i receiving an input bit x_i, sample $(\ell_i, L_i) \leftarrow\$ \{0,1\} \times \mathbb{Z}_{p-1}$ and set $k_i \coloneqq \ell_i \oplus x_i$.

Procedure. The simulation proceeds in a layer-by-layer fashion, from $\mathcal{L}_1$ to $\mathcal{L}_D$. After evaluating a layer $\mathcal{L}_d$, it labels each gate u in the layer with a triple (ℓ_u, L_u, k_u) and stores a garbling $\hat{\mathcal{L}}_d$ of $\mathcal{L}_d$. The parts in blue are omitted in Hybrid_5.

On layer $\mathcal{L}_d$. For $i = 1$ to n_d,

- Let $\mathsf{Left}_{d,i}$ (resp. $\mathsf{Right}_{d,i}$) denote the multisets of gates that are the left parent (resp. right parent) of a gate in $\mathcal{B}_{i,d}$. Retrieve the triples (ℓ_u, L_u, k_u) labeling each $u \in \mathsf{Left}_{d,i} \cup \mathsf{Right}_{d,i}$ and compute

$$(\ell_\mathsf{l}, L_\mathsf{l}, k_\mathsf{l}, \mathsf{shift}_{\mathsf{l},d,i}) \coloneqq \mathsf{SimPack}^{\mathsf{H},\mathcal{O}}(\mathsf{mpk}, (\ell_u, L_u, k_u)_{u \in \mathsf{Left}_{i,d}}, \mathsf{salt}_{i,d,0})$$
$$(\ell_\mathsf{r}, L_\mathsf{r}, k_\mathsf{r}, \mathsf{shift}_{\mathsf{r},d,i}) \coloneqq \mathsf{SimPack}^{\mathsf{H},\mathcal{O}}(\mathsf{mpk}, (\ell_u, L_u, k_u)_{u \in \mathsf{Right}_{i,d}}, \mathsf{salt}_{i,d,1}).$$

- If $\mathcal{L}_d$ is an AND layer:
  - $(\ell_\mathsf{out}, L_\mathsf{out}, k_\mathsf{out}, S_{d,i}) \coloneqq \mathsf{SimBatchAnd}^{\mathsf{H},\mathcal{O}}(\mathsf{mpk}, (\ell_\mathsf{l}, L_\mathsf{l}, k_\mathsf{l}), (\ell_\mathsf{r}, L_\mathsf{r}, k_\mathsf{r}), \mathsf{salt}_{d,i,2})$
  - $((\ell[j], L_j, k[j])_{0 \le j \le t-1}, \mathsf{shift}_{\mathsf{out},d,i}) \coloneqq \mathsf{SimUnpack}^{\mathsf{H},\mathcal{O}}(\Psi, \mathsf{mpk}, \ell_\mathsf{out}, L_\mathsf{out}, k_\mathsf{out}, \mathsf{salt}_{d,i,3})$
  - $(\ell_u, L_u, k_u)_{u \in \mathcal{B}_{d,i}} \coloneqq (\ell[j], L_j, k[j])_{0 \le j \le |\mathcal{B}_{d,i}|-1}$
- If $\mathcal{L}_d$ is a XOR layer:
  - $(\ell_\mathsf{out}, L_\mathsf{out}, k_\mathsf{out}, S_{d,i}) \coloneqq \mathsf{SimBatchXor}^{\mathsf{H},\mathcal{O}}(\mathsf{mpk}, (\ell_\mathsf{l}, L_\mathsf{l}, k_\mathsf{l}), (\ell_\mathsf{r}, L_\mathsf{r}, k_\mathsf{r}), \mathsf{salt}_{d,i,2})$
  - $((\ell[j], L_j, k_j)_{0 \le j \le t-1}, \mathsf{shift}_{\mathsf{out},d,i}) \coloneqq \mathsf{SimUnpack}^{\mathsf{H},\mathcal{O}}(\Phi^{-1}, \mathsf{mpk}, \ell_\mathsf{out}, L_\mathsf{out}, \mathsf{salt}_{d,i,3})$
  - $(\ell_u, L_u, k_u)_{u \in \mathcal{B}_{d,i}} \coloneqq (\ell[j], L_j, k[j])_{0 \le j \le |\mathcal{B}_{d,i}|-1}$
- Label each $u \in \mathcal{B}_{d,i}$ with (ℓ_u, L_u, k_u).

Set $\hat{\mathcal{L}}_d \coloneqq (\mathsf{shift}_{\mathsf{l},d,i}, \mathsf{shift}_{\mathsf{r},d,i}, S_{d,i}, \mathsf{shift}_{\mathsf{out},d,i})_{i \le n_d}$.

Output. Return $\hat{C} = (C, \mathsf{mpk}, (\hat{\mathcal{L}}_d)_{d \le D})$, $\hat{x} \coloneqq (\ell_i, L_i)_{i \le n}$, and $\mathsf{d} \coloneqq (k_o)_{o \in O(C)}$.

Algorithm 13: Garbling procedure of the Boolean garbling scheme

The lemma below shows that Hybrid_1 and Hybrid_2 are perfectly indistinguishable (in fact, it shows an even stronger result):

Lemma 24. *For any Boolean circuit C with $|C| = s$ gates and depth $\mathsf{depth}(C) = D$ represented as described in Section 6.1, any input $x \in \{0,1\}^{|I(C)|}$, and any random tape R, it holds that*

$$\mathsf{SimGC}_1(1^\lambda, C, x; R) = \mathsf{SimGC}_2(1^\lambda, C, x; R).$$

Proof. Fix a Boolean circuit C, an input $x \in \{0,1\}^{|I(C)|}$, and a random tape R. We consider a parallel run of $\mathsf{SimGC}_1(1^\lambda, C, x; R)$ and $\mathsf{SimGC}_2(1^\lambda, C, x; R)$. Both algorithms share a similar structure, up to the following distinction: whenever SimGC_1 invokes a procedure $\mathsf{Proc}(\mathsf{msk}, (\mathbf{k}, \mathbf{K}), \mathsf{salt})$, SimGC_2 invokes a corresponding simulated procedure $\mathsf{SimProc}(\mathsf{mpk}, (\ell, \mathbf{L}, \mathbf{k}), \mathsf{salt})$. We state a meta-lemma that captures the invariant that these procedures jointly maintain:

Lemma 25. *(Meta invariant lemma) Let* $\mathsf{Proc} \in \{\mathsf{Pack}^\mathsf{H}_\mathsf{G}, \mathsf{BatchAND}^\mathsf{H}_\mathsf{G}, \mathsf{UnpackAND}^\mathsf{H}_\mathsf{G}, \mathsf{BatchXOR}^\mathsf{H}_\mathsf{G}, \mathsf{UnpackXOR}^\mathsf{H}_\mathsf{G}\}$, *and let* $\mathsf{SimProc} \in \{\mathsf{SimPack}^{\mathsf{H},\mathcal{O}}, \mathsf{SimBatchAnd}^{\mathsf{H},\mathcal{O}}, \mathsf{SimUnpack}^{\mathsf{H},\mathcal{O}}(\Psi, \cdot), \mathsf{SimBatchXor}^{\mathsf{H},\mathcal{O}}, \mathsf{SimUnpack}^{\mathsf{H},\mathcal{O}}(\Phi^{-1}, \cdot)\}$ *denote the corresponding simulated procedure. Let* $(\mathsf{msk}, (\mathbf{k}, \mathbf{K}), \mathsf{salt})$ *denote the input to* Proc *and* $(\mathsf{mpk}, (\ell, \mathbf{L}, \mathbf{k}), \mathsf{salt})$ *denote the input to* $\mathsf{SimProc}$ *(note that we require that both algorithms receive the same* $\mathbf{k}$ *as input). Fix a* *random tape R. Let* Emb, Emb′ *denote the input and output embeddings defined by* Proc. *Denote*

$$(\mathbf{k}_1, \mathbf{K}_1, S_1) \coloneqq \mathsf{Proc}(\mathsf{msk}, (\mathbf{k}, \mathbf{K}), \mathsf{salt}; R)$$
$$(\ell_2, \mathbf{L}_2, \mathbf{k}_2, S_2) \coloneqq \mathsf{SimProc}(\mathsf{mpk}, (\ell, \mathbf{L}, \mathbf{k}), \mathsf{salt}; R).$$

Then, if it holds that $\mathbf{L} - \mathbf{K} = \Delta \cdot \mathsf{Emb}(\ell)$*, the following holds:*

$$\mathbf{k}_1 = \mathbf{k}_2, \qquad S_1 = S_2, \qquad \mathbf{L}_2 - \mathbf{K}_1 = \Delta \cdot \mathsf{Emb}'(\ell_2).$$

The above lemma is a template "meta-lemma": we will prove a corresponding formal lemma for each of the procedures. Then, we observe that the invariant is guaranteed at the input level: SimGC_1 and SimGC_2 first sample identical input labels $(\ell_i, L_i)_{i \le n}$ (as they use the same random tape) and define $k_i \coloneqq \ell_i \oplus x_i$; SimGC_1 additionally sets $K_i \coloneqq L_i - \Delta \cdot \ell_i$ (via the canonical embedding of $\mathbb{F}_2$ into $\mathbb{Z}_{p-1}$). It follows immediately that SimGC_1 and SimGC_2 output the same garbled input $\hat{x} \coloneqq (\ell_i, L_i)_{i \le n}$.

From there, the invariant lemma guarantees that the invariant propagates throughout the entire evaluation of SimGC_1 and SimGC_2, maintaining the same wire key k_u on each wire u and producing the same sets of shifts $S_{d,i}$ for each batch $\mathcal{B}_{d,i}$. Therefore, SimGC_1 and SimGC_2 output identical $\hat{C}$ and e. ∎

To finish the proof, we introduce each of the simulated procedures, and formally state and prove the corresponding invariant lemma.

Simulator for the packing procedure. The parts in blue are omitted in Hybrid_5.

Algorithm $\mathsf{SimPack}^{\mathsf{H},\mathcal{O}}(\mathsf{mpk}, (\ell_u, L_u, k_u)_{u \in \mathcal{B}}, \mathsf{salt})$

Input. Master public key mpk. Wire labels and keys $(\ell_u, L_u, k_u)_{u \in \mathcal{B}} \in (\{0,1\} \times \mathbb{Z}_{p-1} \times \{0,1\})^{|\mathcal{B}|}$ for a batch of gates $\mathcal{B}$ of size $|\mathcal{B}| \le t$. Salt salt.

Procedure.

- Order the elements of $\mathcal{B}$ lexicographically as $(u_0, \cdots, u_{|\mathcal{B}|-1})$, and set $(\ell[i], L[i]) \coloneqq (\ell_{u_i}, L_{u_i})$ for $i = 0$ to $|\mathcal{B}|-1$. For $i = |\mathcal{B}|$ to $t-1$, set $(\ell[i], L[i]) \coloneqq (0, 0)$.
- $k_{\mathsf{out}} \coloneqq \Phi(k[0], \cdots, k[t-1])$
- $\ell_{\mathsf{out}} \coloneqq \Phi(\ell[0], \cdots, \ell[t-1])$
- $\tilde{\ell}_{\mathsf{out}} \coloneqq \sum_{i=0}^{t-1} \ell[i] \cdot 2^i$
- $L \leftarrow \sum_{i=0}^{t-1} L[i] \cdot 2^i \bmod p-1 \qquad \triangleright\ L = \langle \Delta \cdot \tilde{\ell}_{\mathsf{out}} \rangle_{\mathsf{E}}$
- $(\mathsf{shift}, L_{\mathsf{out}}) \coloneqq [2^t]\text{-}\mathsf{SimBatchfAuth}^{\mathsf{H},\mathcal{O}}(\mathsf{mpk}, \tilde{\ell}_{\mathsf{out}}, L, \mathsf{Eval}_N \circ \Phi \circ \mathsf{toBits}, \mathsf{salt}, \mathsf{shift})$

Output. $(\ell_{\mathsf{out}}, L_{\mathsf{out}}, k_{\mathsf{out}}, \mathsf{shift})$

Algorithm 14: Simulator procedure for packing wire labels before a batch evaluation.

Lemma 26 (Invariant lemma for packing). *Fix* $(\mathsf{mpk}, \mathsf{msk}, (\ell_u, L_u, k_u, K_u)_{u \in \mathcal{B}}, \mathsf{salt})$ *where* $\mathsf{msk} \coloneqq (\mathsf{mpk}, \Delta, h_0)$. *Assume that for each* $u \in \mathcal{B}$*, it holds that*

$$L_u - K_u = \Delta \cdot \ell_u \bmod p-1.$$

Then, denoting

$$(\ell_{\mathrm{out}}, L_{\mathrm{out}}, k_{\mathrm{out}}, \mathsf{shift}) \coloneqq \mathsf{SimPack}^{\mathsf{H},\mathcal{O}}(\mathsf{mpk}, (\ell_u, L_u, k_u)_{u\in\mathcal{B}}, \mathsf{salt})$$
$$(k'_{\mathrm{out}}, K'_{\mathrm{out}}, \mathsf{shift}') \coloneqq \mathsf{Pack}^{\mathsf{H}}_{\mathsf{G}}(\mathsf{msk}, (k_u, K_u)_{u\in\mathcal{B}}, \mathsf{salt})$$

It holds that $k_{\mathrm{out}} = k'_{\mathrm{out}}$, $\mathsf{shift} = \mathsf{shift}'$, *and* $L_{\mathrm{out}} - K_{\mathrm{out}} = \Delta \cdot \mathsf{Eval}_N(\ell_{\mathrm{out}}) \bmod p-1$.

Proof. Both $\mathsf{SimPack}^{\mathsf{H},\mathcal{O}}(\mathsf{mpk}, (\ell_u, L_u, k_u)_{u\in\mathcal{B}}, \mathsf{salt})$ and $\mathsf{Pack}^{\mathsf{H}}_{\mathsf{G}}(\mathsf{msk}, (k_u, K_u)_{u\in\mathcal{B}}, \mathsf{salt})$ compute k_{out} identically as $k_{\mathrm{out}} \coloneqq \Phi(k[0], \cdots, k[t-1])$. By construction, we also have

$$L - K = \sum_{i=0}^{t-1}(L[i] - K[i]) \cdot 2^i = \Delta \cdot \sum_{i=0}^{t-1} \ell[i] \cdot 2^i = \Delta \cdot \tilde{\ell}_{\mathrm{out}} \qquad \triangleright \text{ by assumption.}$$

Then, by Lemma 14, denoting $f \coloneqq \mathsf{Eval}_N \circ \Phi \circ \mathsf{toBits}$, given $(\mathsf{shift}, L_{\mathrm{out}}) \coloneqq \mathsf{SimBatchfAuth}^{\mathsf{H},\mathcal{O}}(\mathsf{mpk}, \tilde{\ell}_{\mathrm{out}}, L, f, \mathsf{salt}, \mathsf{shift})$, we have

$$\mathsf{shift} = \mathsf{BatchfAuth}^{\mathsf{H}}_{\mathsf{G}}(\mathsf{msk}, \Delta, K, f, \mathsf{salt}), \text{ and}$$
$$L_{\mathrm{out}} = \mathsf{BatchfAuth}^{\mathsf{H}}_{\mathsf{E}}(\mathsf{mpk}, \tilde{\ell}_{\mathrm{out}}, L, f, \mathsf{salt}, \mathsf{shift}).$$

Eventually, by correctness of BatchfAuth (Lemma 13), it holds that $L_{\mathrm{out}} - K_{\mathrm{out}} = \Delta \cdot f(\tilde{\ell}_{\mathrm{out}}) \bmod p-1$, hence $L_{\mathrm{out}} - K_{\mathrm{out}} = \Delta \cdot \mathsf{Eval}_N(\ell_{\mathrm{out}}) \bmod p-1$ (by definition of f and ℓ_{out}). This concludes the proof. ∎

Simulator for the unpacking procedure. The parts in blue are omitted in Hybrid_5.

Algorithm $\mathsf{SimUnpack}^{\mathsf{H},\mathcal{O}}(f, \mathsf{mpk}, \ell, L, k, \mathsf{salt})$

Input. Function $f \in \{\Phi^{-1}, \Psi\}$. Master public key msk, packed labels $(\ell, L) \in \mathbb{F}_{2^m} \times \mathbb{Z}_{p-1}$, packed key k, salt salt.

Procedure.

- $(k[0], \cdots, k[t-1]) \coloneqq f(k)$
- $(\ell[0], \cdots, \ell[t-1]) \coloneqq f(\ell)$
- $(\mathsf{shift}, L_0, \cdots, L_{t-1}) \coloneqq \mathsf{SimBatchfAuth}^{\mathsf{H},\mathcal{O}}(\mathsf{mpk}, \ell(N), L, (\mathsf{Bit}_i \circ f \circ \mathsf{toPoly}_N)_{0\leq i\leq t-1})$

Output. $((\ell[i], L_i, k[i])_{i\leq t-1}, \mathsf{shift})$

Algorithm 15: Simulator procedure for unpacking the result of a batch evaluation.

Lemma 27 (Invariant lemma for unpacking). *Fix* $(f, \mathsf{mpk}, \mathsf{msk}, \ell, L, k, K, \mathsf{salt})$ *where* $f \in \{\Phi^{-1}, \Psi\}$ *and* $\mathsf{msk} \coloneqq (\mathsf{mpk}, \Delta, h_0)$. *Assume that it holds that*

$$L - K = \Delta \cdot \mathsf{Eval}_N(\ell_u) \bmod p-1.$$

Then, denoting

$$((\ell[i], L_i, k[i])_{i\le t-1}, \mathsf{shift}) \coloneqq \mathsf{SimUnpack}^{\mathsf{H},\mathcal{O}}(f, \mathsf{mpk}, \ell, L, k, \mathsf{salt})$$
$$((k'[i], K'_i)_{i\le t-1}, \mathsf{shift}') \coloneqq \mathsf{Unpack}^{\mathsf{H}}_{\mathsf{G}}(f, \mathsf{msk}, k, K, \mathsf{salt})$$

It holds that $k[i] = k'[i]$ *for all* $i \le t-1$, $\mathsf{shift} = \mathsf{shift}'$*, and* $L_i - K_i = \Delta \cdot (\ell[i]) \bmod p-1$ *for all* $i \le t-1$.

Proof. The first equality follows immediately, as $k[i], k'[i]$ are computed identically in $\mathsf{SimUnpack}^{\mathsf{H},\mathcal{O}}$ and $\mathsf{Unpack}^{\mathsf{H}}$. Then, using the assumptions of Lemma 27, we can invoke the perfect simulation of SimBatchfAuth to get

$$(\mathsf{shift}, \_) = \mathsf{BatchfAuth}^{\mathsf{H}}_{\mathsf{G}}(\mathsf{msk}, \Delta, K, (\mathsf{Bit}_i \circ f \circ \mathsf{toPoly}_N)_{0\le i\le t-1})$$
$$(L_0, \cdots, L_{t-1}) = \mathsf{BatchfAuth}^{\mathsf{H}}_{\mathsf{E}}(\mathsf{mpk}, \ell(N), L, (\mathsf{Bit}_i \circ f \circ \mathsf{toPoly}_N)_{0\le i\le t-1}),$$

and we conclude using the perfect correctness of $\mathsf{Unpack}^{\mathsf{H}}$ (Lemma 17). ∎

Simulator for the batch-XOR gadget. The parts in blue are omitted in Hybrid_5. $S_{N,m}$ denote the set of sums of two elements from $\mathsf{I}_{N,m}$.

Algorithm $\mathsf{SimBatchXor}^{\mathsf{H},\mathcal{O}}(\mathsf{mpk}, (\ell_\mathsf{l}, L_\mathsf{l}), (\ell_\mathsf{r}, L_\mathsf{r}), k_\mathsf{l}, k_\mathsf{r}, \mathsf{salt})$

Input. Master public key mpk. Packed labels $(\ell_\mathsf{l}, L_\mathsf{l}), (\ell_\mathsf{r}, L_\mathsf{r})$, keys $(k_\mathsf{l}, k_\mathsf{r})$, and salt salt.

Procedure.

- $\tilde{\ell}_{\mathsf{out}} \coloneqq \ell_\mathsf{l} + \ell_\mathsf{r}$ ▷ Over $\mathbb{Z}[X]$
- $k_{\mathsf{out}} \coloneqq k_\mathsf{l} + k_\mathsf{r}$ ▷ Over $\mathbb{F}_{2^m}$
- $\ell_{\mathsf{out}} = \tilde{\ell}_{\mathsf{out}} \bmod 2$ ▷ $\ell_{\mathsf{out}} \in \mathbb{F}_{2^m}$
- $\tilde{L}_{\mathsf{out}} \coloneqq L_\mathsf{l} + L_\mathsf{r} \bmod p-1$
- $(\mathsf{shift}, L_{\mathsf{out}}) \coloneqq S_{N,m}\text{-}\mathsf{SimBatchfAuth}^{\mathsf{H},\mathcal{O}}(\mathsf{mpk}, \tilde{\ell}_{\mathsf{out}}, \tilde{L}_{\mathsf{out}}, \mathsf{Mod}_N(\cdot, 2), \mathsf{salt})$

Output. $(\ell_{\mathsf{out}}, L_{\mathsf{out}}, k_{\mathsf{out}}, \mathsf{shift})$

Algorithm 16: Simulator for the batch-XOR gadget

Lemma 28 (Invariant lemma for batch-XOR)**.** *Fix* $(\mathsf{mpk}, \mathsf{msk}, \ell_\mathsf{l}, L_\mathsf{l}, \ell_\mathsf{r}, L_\mathsf{r}, k_\mathsf{l}, K_\mathsf{l}, k_\mathsf{r}, K_\mathsf{r}, \mathsf{salt})$ *where* $\mathsf{msk} \coloneqq (\mathsf{mpk}, \Delta, h_0)$*. Assume that for* $u \in \{\mathsf{l}, \mathsf{r}\}$*, it holds that*

$$L_u - K_u = \Delta \cdot \mathsf{Eval}_N(\ell_u) \bmod p-1.$$

Then, denoting

$$(\ell_{\mathsf{out}}, L_{\mathsf{out}}, k_{\mathsf{out}}, \mathsf{shift}) \coloneqq \mathsf{SimBatchXor}^{\mathsf{H},\mathcal{O}}(\mathsf{mpk}, (\ell_\mathsf{l}, L_\mathsf{l}, k_\mathsf{l}), (\ell_\mathsf{r}, L_\mathsf{r}, k_\mathsf{r}), \mathsf{salt})$$
$$(k'_{\mathsf{out}}, K'_{\mathsf{out}}, \mathsf{shift}') \coloneqq \mathsf{BatchXOR}^{\mathsf{H}}_{\mathsf{G}}(\mathsf{msk}, (k_\mathsf{l}, K_\mathsf{l}), (k_\mathsf{r}, K_\mathsf{r}), \mathsf{salt}),$$

it holds that $k_{\mathsf{out}} = k'_{\mathsf{out}}$, $\mathsf{shift} = \mathsf{shift}'$*, and* $L_{\mathsf{out}} - K_{\mathsf{out}} = \Delta \cdot \mathsf{Eval}_N(\ell_{\mathsf{out}}) \bmod p-1$.

Proof. $\tilde{\ell}_{\text{out}}$, ℓ_{out}, and $\tilde{L}_{\text{out}}$ are computed identically in $\text{BatchXOR}^{\text{H}}_{\text{E}}$. Denoting $\tilde{K}'_{\text{out}}$ the value computed in $\text{BatchXOR}^{\text{H}}_{\text{G}}$ as $K_{\text{l}} + K_{\text{r}} \bmod p-1$, we established in the proof of Lemma 18 that

$$\tilde{L}'_{\text{out}} - \tilde{K}'_{\text{out}} = \Delta \cdot \tilde{\ell}'_{\text{out}}(N) \bmod p-1$$

and we can therefore invoke the perfect simulation of SimBatchfAuth to get

$$\begin{aligned}(\text{shift}, \_) &= \text{BatchfAuth}^{\text{H}}_{\text{G}}(\text{msk}, \Delta, \tilde{K}_{\text{out}}, \text{Mod}_N(\cdot, 2), \text{salt})\\ L_{\text{out}} &= \text{BatchfAuth}^{\text{H}}_{\text{E}}(\text{mpk}, \tilde{\ell}_{\text{out}}(N), \tilde{L}_{\text{out}}, \text{Mod}_N(\cdot, 2), \text{salt}, \text{shift}),\end{aligned}$$

and we conclude using the perfect correctness of $\text{BatchXOR}^{\text{H}}$ (Lemma 18). ∎

Simulation for the batch-AND gadget. The parts in blue are omitted in Hybrid_5; because $\mathbb{Z}$-Sim′VtO and SimVtO expect an input there, but this input is not used anymore by the modified subprocedures mSim′VtO and mSimVtO, an arbitrary dummy input (e.g. 0) can be passed as input instead.

Algorithm $\text{SimBatchAnd}^{\text{H},O}(\text{mpk}, (\ell_{\text{l}}, L_{\text{l}}, k_{\text{l}}), (\ell_{\text{r}}, L_{\text{r}}, k_{\text{r}}), \text{salt})$

Input. Master public key mpk. Packed labels $(\ell_{\text{l}}, L_{\text{l}}), (\ell_{\text{r}}, L_{\text{r}})$, keys $(k_{\text{l}}, k_{\text{r}})$, and salt salt. For $i = 1$ to 6, we let $\text{salt}_i \coloneqq \text{salt}||i$.

Procedure.

- $(k'_{\text{l}}, k'_{\text{r}}) \leftarrow\$ \text{Pert}_c(k_{\text{l}}) \times \text{Pert}_c(k_{\text{r}})$
- $(\text{shift}_a, \tilde{a}_{\text{E}}) \coloneqq \mathbb{Z}\text{-Sim}'\text{VtO}^{\text{H},O}(\text{mpk}, \ell_{\text{r}}(N), L_{\text{r}}, \text{map}, k'_{\text{l}}(N), \text{salt}_1)$
- $(\text{shift}_b, \tilde{b}_{\text{E}}) \coloneqq \mathbb{Z}\text{-Sim}'\text{VtO}^{\text{H},O}(\text{mpk}, \ell_{\text{l}}(N), L_{\text{l}}, \text{map}, k'_{\text{r}}(N), \text{salt}_2)$
- $\tilde{\ell}_{\text{out}} \coloneqq \ell_{\text{l}}(N)\ell_{\text{r}}(N) + \tilde{a}_{\text{E}} + \tilde{b}_{\text{E}}$
- $a_{\text{E}} \coloneqq [\text{toPoly}_N(\tilde{a}_{\text{E}}) \bmod 2, P]$ ▷ $a_{\text{E}} \in \mathbb{F}_2[X]/P(X) \cong \mathbb{F}_{2^m}$
- $b_{\text{E}} \coloneqq [\text{toPoly}_N(\tilde{b}_{\text{E}}) \bmod 2, P]$ ▷ $b_{\text{E}} \in \mathbb{F}_2[X]/P(X) \cong \mathbb{F}_{2^m}$
- $\ell_{\text{out}} \coloneqq \ell_{\text{l}}\ell_{\text{r}} + a_{\text{E}} + b_{\text{E}}$ ▷ $\ell_{\text{out}} = \text{Mod}_N(\tilde{\ell}_{\text{out}}) \in \mathbb{F}_{2^m}$
- $k_{\text{out}} \coloneqq \ell_{\text{out}} + (\ell_{\text{l}} + k_{\text{l}}) \cdot (\ell_{\text{r}} + k_{\text{r}})$ ▷ over $\mathbb{F}_{2^m}$
- $(\text{shift}_\alpha, \alpha_{\text{E}}) \coloneqq \text{SimVtO}^{\text{H},O}(\text{mpk}, \ell_{\text{r}}(N), L_{\text{r}}, k'_{\text{l}}(N), 0, \text{salt}_3)$
- $(\text{shift}_\beta, \beta_{\text{E}}) \coloneqq \text{SimVtO}^{\text{H},O}(\text{mpk}, \ell_{\text{l}}(N), L_{\text{l}}, k'_{\text{r}}(N), 0, \text{salt}_4)$
- $(\text{shift}_\gamma, \gamma_{\text{E}}) \coloneqq \text{SimVtO}^{\text{H},O}(\text{mpk}, \ell_{\text{r}}(N), L_{\text{r}}, -k_{\text{l}}(N), L_{\text{l}}, \text{salt}_5)$
- $\tilde{L}_{\text{out}} \coloneqq \alpha_{\text{E}} + \beta_{\text{E}} - \gamma_{\text{E}} + L_{\text{l}}\ell_{\text{r}}(N)$ ▷ $\tilde{L}_{\text{out}} = \langle \Delta \tilde{\ell}_{\text{out}} \rangle_{\text{E}}$
- $(\text{shift}_K, L_{\text{out}}) \coloneqq [N^m]\text{- SimBatchfAuth}^{\text{H},O}(\text{mpk}, \tilde{\ell}_{\text{out}}, \tilde{L}_{\text{out}}, \text{Mod}_N(\cdot, 2, P), \text{salt}_6)$
- $S_{\text{out}} \coloneqq (\text{shift}_a, \text{shift}_b, \text{shift}_\alpha, \text{shift}_\beta, \text{shift}_\gamma, \text{shift}_K)$

Output. $(\ell_{\text{out}}, L_{\text{out}}, k_{\text{out}}, S_{\text{out}})$

Algorithm 17: Simulation for the batch-AND gadget

Lemma 29 (Invariant lemma for batch-AND). *Fix* $(\mathsf{mpk}, \mathsf{msk}, \ell_\mathsf{l}, L_\mathsf{l}, \ell_\mathsf{r}, L_\mathsf{r}, k_\mathsf{l}, K_\mathsf{l}, k_\mathsf{r}, K_\mathsf{r}, \mathsf{salt})$ *where* $\mathsf{msk} := (\mathsf{mpk}, \Delta, h_0)$. *Assume that for* $u \in \{\mathsf{l}, \mathsf{r}\}$*, it holds that*

$$L_u - K_u = \Delta \cdot \mathsf{Eval}_N(\ell_u) \bmod p - 1.$$

Then, denoting

$$(\ell_{\mathrm{out}}, L_{\mathrm{out}}, k_{\mathrm{out}}, \mathsf{shift}) := \mathsf{SimBatchAnd}^{\mathsf{H},\mathcal{O}}(\mathsf{mpk}, (\ell_\mathsf{l}, L_\mathsf{l}, k_\mathsf{l}), (\ell_\mathsf{r}, L_\mathsf{r}, k_\mathsf{r}), \mathsf{salt})$$
$$(k'_{\mathrm{out}}, K'_{\mathrm{out}}, \mathsf{shift}') := \mathsf{BatchAND}^{\mathsf{H}}_{\mathsf{G}}(\mathsf{msk}, (k_\mathsf{l}, K_\mathsf{l}), (k_\mathsf{r}, K_\mathsf{r}), \mathsf{salt}),$$

it holds that $k_{\mathrm{out}} = k'_{\mathrm{out}}$, $\mathsf{shift} = \mathsf{shift}'$, *and* $L_{\mathrm{out}} - K_{\mathrm{out}} = \Delta \cdot \mathsf{Eval}_N(\ell_{\mathrm{out}}) \bmod p - 1$.

Proof. The second and third equality can be tracked down by going through the procedures $\mathsf{BatchAND}^{\mathsf{H}}_{\mathsf{G}}$, $\mathsf{BatchAND}^{\mathsf{H}}_{\mathsf{E}}$, and $\mathsf{SimBatchAnd}^{\mathsf{H},\mathcal{O}}$, and relying on the perfect simulation lemmas for VtO and BatchfAuth. Concretely, fix $(\mathsf{mpk}, \mathsf{msk}, \ell_\mathsf{l}, L_\mathsf{l}, \ell_\mathsf{r}, L_\mathsf{r}, k_\mathsf{l}, K_\mathsf{l}, k_\mathsf{r}, K_\mathsf{r}, \mathsf{salt})$ and consider a run of $\mathsf{SimBatchAnd}^{\mathsf{H},\mathcal{O}}(\mathsf{mpk}, (\ell_\mathsf{l}, L_\mathsf{l}, k_\mathsf{l}), (\ell_\mathsf{r}, L_\mathsf{r}, k_\mathsf{r}), \mathsf{salt})$. Using perfect simulation of Sim′VtO (the second part of Lemma 12) together with the assumptions of Lemma 29 yields

$$\begin{aligned}
(\mathsf{shift}_a, _) &= \mathbb{Z}\text{-}\mathsf{VtO}^{\mathsf{H}_0}_{\mathsf{G}}(\mathsf{msk}, k'_\mathsf{l}(N), K_\mathsf{r}, \mathsf{salt}_1)\\
\tilde{a}_\mathsf{E} &= \mathbb{Z}\text{-}\mathsf{VtO}^{\mathsf{H}_0}_{\mathsf{E}}(\mathsf{mpk}, \ell_\mathsf{r}(N), L_\mathsf{r}, \mathsf{salt}_1, \mathsf{shift}_a)\\
(\mathsf{shift}_b, _) &= \mathbb{Z}\text{-}\mathsf{VtO}^{\mathsf{H}_0}_{\mathsf{G}}(\mathsf{msk}, k'_\mathsf{r}(N), K_\mathsf{l}, \mathsf{salt}_2)\\
\tilde{b}_\mathsf{E} &= \mathbb{Z}\text{-}\mathsf{VtO}^{\mathsf{H}_0}_{\mathsf{E}}(\mathsf{mpk}, \ell_\mathsf{l}(N), L_\mathsf{l}, \mathsf{salt}_2, \mathsf{shift}_b).
\end{aligned}$$

From there, the computation of $(a_\mathsf{E}, b_\mathsf{E}, \tilde{\ell}_{\mathrm{out}}, \ell_{\mathrm{out}})$ proceeds identically to $\mathsf{BatchAND}^{\mathsf{H}}_{\mathsf{E}}$. Hence, denoting $(\ell'_{\mathrm{out}}, L'_{\mathrm{out}}) := \mathsf{BatchAND}^{\mathsf{H}}_{\mathsf{E}}(\mathsf{msk}, (\ell_\mathsf{l}, L_\mathsf{l}), (\ell_\mathsf{r}, L_\mathsf{r}), \mathsf{salt}, S)$, we get $\ell'_{\mathrm{out}} = \ell_{\mathrm{out}}$. By correctness of $\mathsf{BatchAND}^{\mathsf{H}}$ (Lemma 19), $k'_{\mathrm{out}} + \ell_{\mathrm{out}} = (\ell_\mathsf{l} + k_\mathsf{l}) \cdot (\ell_\mathsf{r} + k_\mathsf{r})$, and it follows that $k_{\mathrm{out}} = k'_{\mathrm{out}}$.

Using now perfect simulation of SimVtO (the first part of Lemma 12) together with the assumptions of Lemma 29 yields

$$\begin{aligned}
(\mathsf{shift}_\alpha, _) &= \mathsf{VtO}^{\mathsf{H}}_{\mathsf{G}}(\mathsf{msk}, \Delta k'_\mathsf{l}(N), K_\mathsf{r}, \mathsf{salt}_3)\\
\alpha_\mathsf{E} &= \mathsf{VtO}^{\mathsf{H}}_{\mathsf{E}}(\mathsf{mpk}, \ell_\mathsf{r}(N), L_\mathsf{r}, \mathsf{salt}_3, \mathsf{shift}_\alpha)\\
(\mathsf{shift}_\beta, _) &= \mathsf{VtO}^{\mathsf{H}}_{\mathsf{G}}(\mathsf{msk}, \Delta k'_\mathsf{r}(N), K_\mathsf{l}, \mathsf{salt}_4)\\
\beta_\mathsf{E} &= \mathsf{VtO}^{\mathsf{H}}_{\mathsf{E}}(\mathsf{mpk}, \ell_\mathsf{l}(N), L_\ell, \mathsf{salt}_4, \mathsf{shift}_\beta)\\
(\mathsf{shift}_\gamma, _) &= \mathsf{VtO}^{\mathsf{H}}_{\mathsf{G}}(\mathsf{msk}, K_\mathsf{l}, K_\mathsf{r}, \mathsf{salt}_5)\\
\gamma_\mathsf{E} &= \mathsf{VtO}^{\mathsf{H}}_{\mathsf{E}}(\mathsf{mpk}, \ell_\mathsf{r}(N), L_\mathsf{r}, \mathsf{salt}_5, \mathsf{shift}_\gamma).
\end{aligned}$$

Then, the computation of $\tilde{L}_{\mathrm{out}}$ proceeds identically to $\mathsf{BatchAND}^{\mathsf{H}}_{\mathsf{E}}$. In particular, denoting $\tilde{K}'_{\mathrm{out}}, \tilde{L}'_{\mathrm{out}}$ the values computed in $\mathsf{BatchAND}^{\mathsf{H}}_{\mathsf{G}}$ and $\mathsf{BatchAND}^{\mathsf{H}}_{\mathsf{E}}$ respectively, we get $\tilde{L}'_{\mathrm{out}} = \tilde{L}_{\mathrm{out}}$. Furthermore, in the proof of Lemma 19, we established

$$\tilde{L}'_{\mathrm{out}} - \tilde{K}'_{\mathrm{out}} = \Delta \cdot \tilde{\ell}_{\mathrm{out}} \bmod p - 1.$$

Therefore, the assumptions of Lemma 14 are satisfied and we can invoke perfect simulation of $\mathsf{SimBatchfAuth}^{\mathsf{H},\mathcal{O}}$ (Lemma 14) to get

$$(\mathsf{shift}_K, \_) = \mathsf{BatchfAuth}^{\mathsf{H}}_{\mathsf{G}}(\mathsf{msk}, \Delta, \tilde{K}_{\mathrm{out}}, \mathsf{Mod}_N(\cdot, 2, P), \mathsf{salt}_6)$$
$$L_{\mathrm{out}} = \mathsf{BatchfAuth}^{\mathsf{H}}_{\mathsf{E}}(\mathsf{mpk}, \tilde{\ell}_{\mathrm{out}}, \tilde{L}_{\mathrm{out}}, \mathsf{Mod}_N(\cdot, 2, P), \mathsf{salt}_6, \mathsf{shift}_K),$$

which concludes the proof. ∎

8 $\omega(1/\lambda)$-Rate Boolean Garbling for Layered Circuits in the Standard Model

In this section, we describe a boolean garbling scheme that achieves a rate of $\lambda/\sqrt{\log\lambda}$, with security reducing to the power-DDH assumption (Definition 4) and a TCR hash function for exponential correlations (Definition 6). Thus, compared to the garbling scheme in Section 6, the security of this scheme no longer requires the GGM. However, this comes at the cost of supporting only layered circuits and incurring an overhead in the concrete size of the garbled circuit.

In more detail, the garbling scheme follows the same template as the one presented in Section 6, except that it uses the leveled TCCR hash function from Definition 8. As a result, unlike the GGM-secure TCCR hash (Definition 7), we can no longer "encrypt" linear functions of the PPRF secret key using PPRF evaluations. This, in turn, implies that we cannot directly use the PPRF secret key as input to the BatchfAuth procedure in our garbling gadgets. To circumvent this issue, we adopt a standard key-switching technique: the garbler samples multiple PPRF keys and the evaluator's share for the output of gates at each level are authenticated using a fresh key. It is easy to see that security then follows immediately from our definition of the leveled TCCR hash. However, since our garbling scheme packs multiple boolean values together when evaluating each gate, the key-switching technique requires that all of the packed values are authenticated under the same key. Consequently, we can only support layered circuits (see Section 3), where every wire in the circuit is between gates at consecutive layers. Moreover, the garbler must now send $4D$ PPRF public keys, where D is the depth of the circuit, adding an overhead to the size of the garbled circuit. However, for circuits that are sufficiently wide, this does not impact the rate. We note that layered circuits are expressive enough to capture a variety of useful computations, including FFT circuits, symmetric cryptographic primitives like block ciphers, and dynamic programming algorithms like longest common subsequence. We refer the reader to [Cou19] for a more detailed discussion.

Next, we describe the parameters and algorithms used in the garbling scheme and then proceed to present the complete description of the scheme.

Parameters. We use the same parameters as the garbling scheme in Section 6. Specifically, let $(\mathbb{G}, p, g, G) \coloneqq \mathsf{GrpGen}^*(1^\lambda)$. Let $t = \sqrt{\log\lambda}$ denote the batch parameter and (Φ, Ψ) be a $(t, m)_2$-reverse multiplication friendly embedding with $m = O(t)$. Let $c = \omega(1)$ denote a statistical security parameter with $c \leq 2^{\sqrt{\log\lambda}}$ and set $N = 2c\cdot m\cdot(2^m+1)+1$ (note that $N^m = \mathsf{poly}(\lambda)$). Let $\mathsf{H} = \{\mathsf{H}_\lambda : \mathbb{G}\times\{0,1\}^* \to \mathbb{Z}_{p-1}\}_{\lambda\in\mathbb{N}}$ denote a TCR hash family for exponential correlation with auxiliary powers over $\mathbb{G}$ (Definition 6). Let $\mathsf{H}_0 \coloneqq \mathsf{map} \circ \mathsf{H}$, where $\mathsf{map} = \mathsf{map}_{N,m,c}$ is the mapping from Definition 9.

Algorithms. As discussed earlier, employing the leveled TCCR hash necessitates the use of multiple PPRF keys. However, this does not require any modification to $(\mathsf{Pack}^{\mathsf{H}}_{\mathsf{E}}, \mathsf{UnpackAND}^{\mathsf{H}}_{\mathsf{E}}, \mathsf{BatchXOR}^{\mathsf{H}}_{\mathsf{E}}, \mathsf{UnpackXOR}^{\mathsf{H}}_{\mathsf{E}})$ from Section 6, as these procedures take the PPRF public key as input and only use it to authenticate the evaluator's share in a call to BatchfAuth. During evaluation, we simply invoke them with different keys as needed. On the other hand, we modify the garbler's algorithms $(\mathsf{Pack}^{\mathsf{H}}_{\mathsf{G}}, \mathsf{UnpackAND}^{\mathsf{H}}_{\mathsf{G}}, \mathsf{BatchXOR}^{\mathsf{H}}_{\mathsf{G}}, \mathsf{UnpackXOR}^{\mathsf{H}}_{\mathsf{G}})$ to additionally take as input Δ_{out}, which is the PPRF secret key for the next layer. These algorithms then invoke $\mathsf{BatchfAuth}(\mathsf{msk}, \Delta_{\mathsf{out}}, \cdot, \cdot, \cdot)$ in the last step when authenticating the evaluator's share. Finally, since the gadget used for garbling and evaluating AND gates makes multiple use of the PPRF key, we modify them to use a different PPRF key for intermediate computations. The modified variants are described in Algorithms 18 and 19.

Algorithm $\mathsf{BatchAND}_{\mathsf{G}}^{\mathsf{H}}(\mathsf{msk}, \mathsf{msk}', \Delta_{\mathrm{out}}, (k_{\mathsf{l}}, K_{\mathsf{l}}), (k_{\mathsf{r}}, K_{\mathsf{r}}), \mathsf{salt})$

Input. Master secret keys msk and msk′, and $\Delta_{\mathrm{out}} \in \mathbb{Z}_{p-1}$. Packed keys $(k_{\mathsf{l}}, K_{\mathsf{l}}), (k_{\mathsf{r}}, K_{\mathsf{r}})$. Salt salt. Parse $\mathsf{msk} \coloneqq (\mathsf{mpk}, \Delta, h_0)$ and $\mathsf{msk}' \coloneqq (\mathsf{mpk}', \Delta', h_0')$. For $i = 1$ to 7, let $\mathsf{salt}_i \coloneqq \mathsf{salt}||7$.

Procedure.

- $(k_{\mathsf{l}}', k_{\mathsf{r}}') \leftarrow\$ \mathsf{Pert}_c(k_{\mathsf{l}}) \times \mathsf{Pert}_c(k_{\mathsf{r}})$
- $(\mathsf{shift}_a, \tilde{a}_{\mathsf{G}}) \coloneqq \mathbb{Z}\text{-}\mathsf{VtO}_{\mathsf{G}}^{\mathsf{H}_0}(\mathsf{msk}, k_{\mathsf{l}}'(N), K_{\mathsf{r}}, \mathsf{salt}_1)$
  $\triangleright\ \tilde{a}_{\mathsf{G}} = \langle k_{\mathsf{l}}'(N)\ell_{\mathsf{r}}(N)\rangle_{\mathsf{G}}$
- $(\mathsf{shift}_b, \tilde{b}_{\mathsf{G}}) \coloneqq \mathbb{Z}\text{-}\mathsf{VtO}_{\mathsf{G}}^{\mathsf{H}_0}(\mathsf{msk}, k_{\mathsf{r}}'(N), K_{\mathsf{l}}, \mathsf{salt}_2)$
  $\triangleright\ \tilde{b}_{\mathsf{G}} = \langle k_{\mathsf{r}}'(N)\ell_{\mathsf{l}}(N)\rangle_{\mathsf{G}}$
- $a_{\mathsf{G}} \coloneqq [\mathsf{toPoly}_N(\tilde{a}_{\mathsf{G}}) \bmod 2, P]$ $\triangleright\ a_{\mathsf{G}} \in \mathbb{F}_2[X]/P(X) \cong \mathbb{F}_{2^m}$
- $b_{\mathsf{G}} \coloneqq [\mathsf{toPoly}_N(\tilde{b}_{\mathsf{G}}) \bmod 2, P]$ $\triangleright\ b_{\mathsf{G}} \in \mathbb{F}_2[X]/P(X) \cong \mathbb{F}_{2^m}$
- $k_{\mathrm{out}} \coloneqq k_{\mathsf{l}}k_{\mathsf{r}} + a_{\mathsf{G}} + b_{\mathsf{G}}$ $\triangleright$ over $\mathbb{F}_{2^m}$
- $(\mathsf{shift}_\alpha, \alpha_{\mathsf{G}}) \coloneqq \mathsf{VtO}_{\mathsf{G}}^{\mathsf{H}}(\mathsf{msk}, \Delta' k_{\mathsf{l}}'(N), K_{\mathsf{r}}, \mathsf{salt}_3)$
  $\triangleright\ \alpha_{\mathsf{G}} = \langle \Delta' k_{\mathsf{l}}'(N) \cdot \ell_{\mathsf{r}}(N)\rangle_{\mathsf{G}}$
- $(\mathsf{shift}_\beta, \beta_{\mathsf{G}}) \coloneqq \mathsf{VtO}_{\mathsf{G}}^{\mathsf{H}}(\mathsf{msk}, \Delta' k_{\mathsf{r}}'(N), K_{\mathsf{l}}, \mathsf{salt}_4)$
  $\triangleright\ \beta_{\mathsf{G}} = \langle \Delta' k_{\mathsf{r}}'(N) \cdot \ell_{\mathsf{l}}(N)\rangle_{\mathsf{G}}$
- $(\mathsf{shift}_{K'}, K_{\mathsf{l}}') \coloneqq \mathsf{VtO}_{\mathsf{G}}^{\mathsf{H}}(\mathsf{msk}, \Delta', K_{\mathsf{l}}, \mathsf{salt}_5)$
  $\triangleright\ K_{\mathsf{l}}' = \langle \Delta' \tilde{\ell}_{\mathsf{l}}\rangle_{\mathsf{G}}$
- $(\mathsf{shift}_\gamma, \gamma_{\mathsf{G}}) \coloneqq \mathsf{VtO}_{\mathsf{G}}^{\mathsf{H}}(\mathsf{msk}, K_{\mathsf{l}}', K_{\mathsf{r}}, \mathsf{salt}_6)$ $\triangleright\ \gamma_{\mathsf{G}} = \langle K_{\mathsf{l}}' \cdot \ell_{\mathsf{r}}(N)\rangle_{\mathsf{G}}$
- $\tilde{K}_{\mathrm{out}} \coloneqq \alpha_{\mathsf{G}} + \beta_{\mathsf{G}} - \gamma_{\mathsf{G}} + \Delta'(\tilde{a}_{\mathsf{G}} + \tilde{b}_{\mathsf{G}})$ $\triangleright\ \tilde{K}_{\mathrm{out}} = \langle \Delta' \tilde{\ell}_{\mathrm{out}}\rangle_{\mathsf{G}}$
- $(\mathsf{shift}_K, K_{\mathrm{out}}) \coloneqq [N^m]\text{-}\mathsf{BatchfAuth}_{\mathsf{G}}^{\mathsf{H}}(\mathsf{msk}', \Delta_{\mathrm{out}}, \tilde{K}_{\mathrm{out}}, \mathsf{Mod}_N(\cdot, 2, P), \mathsf{salt}_7)$
  $\triangleright\ K_{\mathrm{out}} = \langle \Delta_{\mathrm{out}}\ell_{\mathrm{out}}\rangle_{\mathsf{G}}$
- $S_{\mathrm{out}} \coloneqq (\mathsf{shift}_a, \mathsf{shift}_b, \mathsf{shift}_\alpha, \mathsf{shift}_\beta, \mathsf{shift}_{K'}, \mathsf{shift}_\gamma, \mathsf{shift}_K)$

Output. $((k_{\mathrm{out}}, K_{\mathrm{out}}), S_{\mathrm{out}})$

Algorithm 18: Garbler batch-AND gadget using the leveled TCCR hash. We highlight the changes from Algorithm 11 in red.

Algorithm $\mathsf{BatchAND}_{\mathsf{E}}^{\mathsf{H}}(\mathsf{mpk}, \mathsf{mpk}', (\ell_{\mathsf{l}}, L_{\mathsf{l}}), (\ell_{\mathsf{r}}, L_{\mathsf{r}}), \mathsf{salt}, S)$

Inputs. Master public keys mpk and mpk′. Packed labels $(\ell_{\mathsf{l}}, L_{\mathsf{l}}), (\ell_{\mathsf{r}}, L_{\mathsf{r}})$, salt salt and garbling material S. Parse $S \coloneqq (\mathsf{shift}_a, \mathsf{shift}_b, \mathsf{shift}_\alpha, \mathsf{shift}_\beta, \mathsf{shift}_{K'}, \mathsf{shift}_\gamma, \mathsf{shift}_K)$. For $i = 1$ to 7, let $\mathsf{salt}_i \coloneqq \mathsf{salt}||i$.

Procedure.

- $\tilde{a}_{\mathsf{E}} \coloneqq \mathbb{Z}\text{-}\mathsf{VtO}_{\mathsf{E}}^{\mathsf{H}_0}(\mathsf{mpk}, \ell_{\mathsf{r}}(N), L_{\mathsf{r}}, \mathsf{salt}_1, \mathsf{shift}_a)$
  $\triangleright\ \tilde{a}_{\mathsf{E}} = \langle k_{\mathsf{l}}'(N)\ell_{\mathsf{r}}(N)\rangle_{\mathsf{E}}$

- $\tilde{b}_\mathsf{E} \coloneqq \mathbb{Z}\text{-}\mathsf{VtO}_\mathsf{E}^{\mathsf{H}_0}(\mathsf{mpk}, \ell_\mathsf{l}(N), L_\mathsf{l}, \mathsf{salt}_2, \mathsf{shift}_b)$ $\quad \triangleright\ \tilde{b}_\mathsf{E} = \langle k'_\mathsf{r}(N)\ell_\mathsf{l}(N)\rangle_\mathsf{E}$
- $\tilde{\ell}_\mathsf{out} \coloneqq \ell_\mathsf{l}(N)\ell_\mathsf{r}(N) + \tilde{a}_\mathsf{E} + \tilde{b}_\mathsf{E}$
- $a_\mathsf{E} \coloneqq [\mathsf{toPoly}_N(\tilde{a}_\mathsf{E}) \bmod 2, P]$ $\quad \triangleright\ a_\mathsf{E} \in \mathbb{F}_2[X]/P(X) \cong \mathbb{F}_{2^m}$
- $b_\mathsf{E} \coloneqq [\mathsf{toPoly}_N(\tilde{b}_\mathsf{E}) \bmod 2, P]$ $\quad \triangleright\ b_\mathsf{E} \in \mathbb{F}_2[X]/P(X) \cong \mathbb{F}_{2^m}$
- $\ell_\mathsf{out} \coloneqq \ell_\mathsf{l}\ell_\mathsf{r} + a_\mathsf{E} + b_\mathsf{E}$ $\quad \triangleright\ \ell_\mathsf{out} = \mathsf{Mod}_N(\tilde{\ell}_\mathsf{out}) \in \mathbb{F}_{2^m}$
- $\alpha_\mathsf{E} \coloneqq \mathsf{VtO}_\mathsf{E}^\mathsf{H}(\mathsf{mpk}, \ell_\mathsf{r}(N), L_\mathsf{r}, \mathsf{salt}_3, \mathsf{shift}_\alpha)$
  $\triangleright\ \alpha_\mathsf{E} = \langle \Delta' k'_\mathsf{l}(N) \cdot \ell_\mathsf{r}(N)\rangle_\mathsf{E}$
- $\beta_\mathsf{E} \coloneqq \mathsf{VtO}_\mathsf{E}^\mathsf{H}(\mathsf{mpk}, \ell_\mathsf{l}(N), L_\mathsf{l}, \mathsf{salt}_4, \mathsf{shift}_\beta)$
  $\triangleright\ \beta_\mathsf{E} = \langle \Delta' k'_\mathsf{r}(N) \cdot \ell_\mathsf{l}(N)\rangle_\mathsf{E}$
- $L'_\mathsf{l} \coloneqq \mathsf{VtO}_\mathsf{E}^\mathsf{H}(\mathsf{mpk}, \ell_\mathsf{l}(N), L_\mathsf{l}, \mathsf{salt}_5, \mathsf{shift}_{K'})$ $\quad \triangleright\ L'_\mathsf{l} = \langle \Delta' \tilde{\ell}_\mathsf{l}\rangle_\mathsf{E}$
- $\gamma_\mathsf{E} \coloneqq \mathsf{VtO}_\mathsf{E}^\mathsf{H}(\mathsf{mpk}, \ell_\mathsf{r}(N), L_\mathsf{r}, \mathsf{salt}_6, \mathsf{shift}_\gamma)$ $\quad \triangleright\ \gamma_\mathsf{E} = \langle K_\mathsf{l} \cdot \ell_\mathsf{r}(N)\rangle_\mathsf{E}$
- $\tilde{L}_\mathsf{out} \coloneqq \alpha_\mathsf{E} + \beta_\mathsf{E} - \gamma_\mathsf{E} + L'_\mathsf{l}\ell_\mathsf{r}(N)$ $\quad \triangleright\ \tilde{L}_\mathsf{out} = \langle \Delta' \tilde{\ell}_\mathsf{out}\rangle_\mathsf{E}$
- $L_\mathsf{out} \coloneqq [N^m]\text{-}\mathsf{BatchfAuth}_\mathsf{E}^\mathsf{H}(\mathsf{mpk}', \tilde{\ell}_\mathsf{out}, \tilde{L}_\mathsf{out}, \mathsf{Mod}_N(\cdot, 2, P), \mathsf{salt}_7, \mathsf{shift}_K)$
  $\triangleright\ L_\mathsf{out} = \langle \Delta_\mathsf{out}\ell_\mathsf{out}\rangle_\mathsf{E}$

Output. $(\ell_\mathsf{out}, L_\mathsf{out})$

Algorithm 19: Evaluator batch-AND gadget using the leveled TCCR hash. We highlight the changes from Algorithm 12 in red.

Input. The input to GC.Garble is a layered boolean circuit C with $|C| = s$ gates and $n = |I(C)|$ inputs such that the gates can be partitioned into layers $(\mathcal{L}_1, \ldots, \mathcal{L}_D)$, where every wire only connects adjacent layers. Let $\mathcal{L}_d^\mathsf{and}$ and $\mathcal{L}_d^\mathsf{xor}$ denote the set of AND gates and XOR gates in the d-th layer. In each layer $\mathcal{L}_d$, we separately partition the set of AND gates and XOR gates into $n_d \coloneqq \lceil|\mathcal{L}_d^\mathsf{and}|/t\rceil + \lceil|\mathcal{L}_d^\mathsf{xor}|/t\rceil$ batches $(\mathcal{B}_{d,1}, \cdots, \mathcal{B}_{d,n_d})$ containing at most t gates each. For $d = 1$ to D, let $\mathsf{salt}_{d,1}, \cdots, \mathsf{salt}_{d,n_d}$ denote unique identifiers for each batch of gate, and write $\mathsf{salt}_{d,i,j} \coloneqq \mathsf{salt}_{d,i}||j$ for $j \in \{0, 1, 2, 3\}$. We assume that all these data can be parsed from the description C of the circuit.

Garbling scheme. We next proceed to describe the leveled garbling scheme. The encoding and decoding algorithms GC.Enc and GC.Dec are identical to those described in Section 6, namely Algorithms 2 and 4. We present GC.Garble and GC.Eval in Algorithms 3 and 13, respectively.

Algorithm $\mathsf{GC.Garble}(1^\lambda, C)$

Input. A layered boolean circuit C with $|C| = s$ gates and depth $\mathsf{depth}(C) = D$. The input gates are indexed from 1 to n.

Initialization.

- Sample $(\mathsf{mpk}_0, \mathsf{msk}_0), \ldots, (\mathsf{mpk}_{4D}, \mathsf{msk}_{4D}) \leftarrow\$\ \mathsf{F.Setup}(1^\lambda, N^m)$. Parse each $\mathsf{msk}_i \coloneqq (\mathsf{mpk}_i, \Delta_i, g_0)$.
- For each input wire i, sample $(k_i, K_i) \leftarrow\$\ \{0, 1\} \times \mathbb{Z}_{p-1}$.

Procedure. The garbling proceeds in a layer-by-layer fashion, from $\mathcal{L}_1$ to $\mathcal{L}_D$. After evaluating a layer $\mathcal{L}_d$, it labels each gate u in the layer with a pair (k_u, K_u) and stores a garbling $\hat{\mathcal{L}}_d$ of $\mathcal{L}_d$.

On layer $\mathcal{L}_d$. For $i = 1$ to n_d,

- Let $\mathsf{Left}_{d,i}$ (resp. $\mathsf{Right}_{d,i}$) denote the multisets of gates that are the left parent (resp. right parent) of a gate in $\mathcal{B}_{d,i}$. Retrieve the pairs (k_u, K_u) labeling each $u \in \mathsf{Left}_{d,i} \cup \mathsf{Right}_{d,i}$ and compute

$$(k_\mathsf{l}, K_\mathsf{l}, \mathsf{shift}_{\mathsf{l},d,i}) \coloneqq \mathsf{Pack}^{\mathsf{H}}_{\mathsf{G}}(\mathsf{msk}_{4d-4}, \Delta_{4d-3}, (k_u, K_u)_{u \in \mathsf{Left}_{d,i}}, \mathsf{salt}_{d,i,0})$$
$$(k_\mathsf{r}, K_\mathsf{r}, \mathsf{shift}_{\mathsf{r},d,i}) \coloneqq \mathsf{Pack}^{\mathsf{H}}_{\mathsf{G}}(\mathsf{msk}_{4d-4}, \Delta_{4d-3}, (k_u, K_u)_{u \in \mathsf{Right}_{d,i}}, \mathsf{salt}_{d,i,1}).$$

- If $\mathcal{B}_{d,i}$ is a batch of AND gates:
  - $(k_\mathsf{out}, K_\mathsf{out}, S_{d,i}) \leftarrow\$ \mathsf{BatchAND}^{\mathsf{H}}_{\mathsf{G}}(\mathsf{msk}_{4d-3}, \mathsf{msk}_{4d-2}, \Delta_{4d-1}, (k_\mathsf{l}, K_\mathsf{l}), (k_\mathsf{r}, K_\mathsf{r}), \Delta, \mathsf{salt}_{d,i,2})$
  - $((k[j], K_j)_{0 \le i \le t-1}, \mathsf{shift}_{\mathsf{out},d,i}) \coloneqq \mathsf{UnpackAND}^{\mathsf{H}}_{\mathsf{G}}(\mathsf{msk}_{4d-1}, \Delta_{4d}, k_\mathsf{out}, K_\mathsf{out}, \mathsf{salt}_{d,i,3})$
  - $(k_u, K_u)_{u \in \mathcal{B}_{d,i}} \coloneqq (k[j], K_j)_{0 \le j \le |\mathcal{B}_{d,i}|-1}$
- If $\mathcal{B}_{d,i}$ is a batch of XOR gates:
  - $(k_\mathsf{out}, K_\mathsf{out}, S_{d,i}) \leftarrow\$ \mathsf{BatchXOR}^{\mathsf{H}}_{\mathsf{G}}(\mathsf{msk}_{4d-3}, \Delta_{4d-1}, (k_\mathsf{l}, K_\mathsf{l}), (k_\mathsf{r}, K_\mathsf{r}), \Delta, \mathsf{salt}_{d,i,2})$
  - $((k[j], K_j)_{0 \le j \le t-1}, \mathsf{shift}_{\mathsf{out},d,i}) \coloneqq \mathsf{UnpackXOR}^{\mathsf{H}}_{\mathsf{G}}(\mathsf{msk}_{4d-1}, \Delta_{4d}, k_\mathsf{out}, K_\mathsf{out}, \mathsf{salt}_{d,i,3})$
  - $(k_u, K_u)_{u \in \mathcal{B}_{d,i}} \coloneqq (k[j], K_j)_{0 \le j \le |\mathcal{B}_{d,i}|-1}$
- Label each $u \in \mathcal{B}_{d,i}$ with the key pair (k_u, K_u).

Set $\hat{\mathcal{L}}_d \coloneqq (\mathsf{shift}_{\mathsf{l},d,i}, \mathsf{shift}_{\mathsf{r},d,i}, S_{d,i}, \mathsf{shift}_{\mathsf{out},d,i})_{i \le n_d}$.

Output. Return $\mathsf{e} \coloneqq ((k_i, K_i)_{i \le n}, \Delta_0)$, $\hat{C} \coloneqq (C, \{\mathsf{mpk}_i\}_{i=0}^{4D}, (\hat{\mathcal{L}}_d)_{d \le D})$, and $\mathsf{d} \coloneqq (k_o)_{o \in O(C)}$.

Algorithm 20: Garbling procedure of the leveled Boolean garbling scheme. We highlight the key changes to Algorithm 13 in red.

Algorithm $\mathsf{GC.Eval}(\hat{C}, \hat{x})$

Inputs. Parse $\hat{C}$ as $(C, \{\mathsf{mpk}_i\}_{i=0}^{4D}, (\hat{\mathcal{L}}_d)_{d \le D})$ and $\hat{x}$ as $(\ell_i, L_i)_{i \le n} \in (\mathbb{F}_2 \times \mathbb{Z}_{p-1})^n$.

Procedure. The evaluation proceeds in a layer-by-layer fashion, from $\mathcal{L}_1$ to $\mathcal{L}_D$. After evaluating a layer $\mathcal{L}_d$, it labels each gate u in the layer with a pair (ℓ_u, L_u).

On layer $\mathcal{L}_d$. For $i = 1$ to n_d,

- Parse $\hat{\mathcal{L}}_d$ as $\hat{\mathcal{L}}_d \coloneqq (\mathsf{shift}_{\mathsf{l},d,i}, \mathsf{shift}_{\mathsf{r},d,i}, S_{d,i})_{i \le n_d}, \mathsf{shift}_{\mathsf{out},d,i})$.
- Let $\mathsf{Left}_{d,i}$ (resp. $\mathsf{Right}_{d,i}$) denote the multisets of gates that are the left parent (resp. right parent) of a gate in $\mathcal{B}_{d,i}$. Retrieve the pairs (ℓ_u, L_u) labeling each $u \in \mathsf{Left}_{d,i} \cup \mathsf{Right}_{d,i}$ and

compute

$$(\ell_{\mathsf{l}}, L_{\mathsf{l}}) \coloneqq \mathsf{Pack}_{\mathsf{E}}^{\mathsf{H}}(\mathsf{mpk}_{4d-4}, (\ell_u, L_u)_{u \in \mathsf{Left}_{d,i}}, \mathsf{salt}_{i,d,0}, \mathsf{shift}_{\mathsf{l},d,i})$$
$$(\ell_{\mathsf{r}}, L_{\mathsf{r}}) \coloneqq \mathsf{Pack}_{\mathsf{E}}^{\mathsf{H}}(\mathsf{mpk}_{4d-4}, (\ell_u, L_u)_{u \in \mathsf{Right}_{d,i}}, \mathsf{salt}_{i,d,1}, \mathsf{shift}_{\mathsf{r},d,i}).$$

- If $\mathcal{B}_{d,i}$ is a batch of AND gates:
  - $(\ell_{\mathsf{out}}, L_{\mathsf{out}}) \coloneqq \mathsf{BatchAND}_{\mathsf{E}}^{\mathsf{H}}(\mathsf{mpk}_{4d-3}, \mathsf{mpk}_{4d-2}, (\ell_{\mathsf{l}}, L_{\mathsf{l}}), (\ell_{\mathsf{r}}, L_{\mathsf{r}}), \mathsf{salt}_{d,i,2}, S_{d,i})$
  - $(\ell[j], L_j)_{0 \le j \le t-1} \coloneqq \mathsf{UnpackAND}_{\mathsf{E}}^{\mathsf{H}}(\mathsf{mpk}_{4d-1}, \ell_{\mathsf{out}}, L_{\mathsf{out}}, \mathsf{salt}_{d,i,3}, \mathsf{shift}_{\mathsf{out},d,i})$
  - $(\ell_u, L_u)_{u \in \mathcal{B}_{d,i}} \coloneqq (\ell[j], L_j)_{0 \le j \le |\mathcal{B}_{d,i}|-1}$
- If $\mathcal{L}_d$ is a XOR layer:
  - $(\ell_{\mathsf{out}}, L_{\mathsf{out}}) \coloneqq \mathsf{BatchXOR}^{\mathsf{H}}(\mathsf{mpk}_{4d-3}, (\ell_{\mathsf{l}}, L_{\mathsf{l}}), (\ell_{\mathsf{r}}, L_{\mathsf{r}}), \mathsf{salt}_{d,i,2}, S_{d,i})$
  - $(\ell[j], L_j)_{0 \le i \le t-1} \coloneqq \mathsf{UnpackXOR}_{\mathsf{E}}^{\mathsf{H}}(\mathsf{mpk}_{4d-1}, \ell_{\mathsf{out}}, L_{\mathsf{out}}, \mathsf{salt}_{d,i,3}, \mathsf{shift}_{\mathsf{out},d,i})$
  - $(\ell_u, L_u)_{u \in \mathcal{B}_{d,i}} \coloneqq (\ell[j], L_j)_{0 \le i \le |\mathcal{B}_{d,i}|-1}$
- Label each $u \in \mathcal{B}_{d,i}$ with (ℓ_u, L_u).

Output. Return $\hat{y} \coloneqq (\ell_o)_{o \in O(C)}$.

Algorithm 21: Evaluator algorithm of the leveled Boolean garbling scheme. We highlight the key changes to Algorithm 3 in red.

Theorem 30. *Let λ be the security parameter and $N \coloneqq N(\lambda)$ and $m \coloneqq m(\lambda)$ be integer valued functions as described above. If the N^m-power DDH assumption (Definition 4) holds with respect to* GrpGen *and if* H *is a TCR hash for the exponential correlation with respect groups generated by* GrpGen *(Definition 6) then* GC *is a boolean garbling scheme for polynomial size layered circuits. Moreover, there exists a polynomial* poly(·) *such that for any layered circuit C of depth D, the garbled circuit $\hat{C} \leftarrow \mathsf{GC.Garble}(1^\lambda, C)$ satisfies*

$$|\hat{C}| \in O\left(\lambda \cdot \frac{|C|}{\sqrt{\log \lambda}} + \mathsf{poly}(\lambda) \cdot D\right).$$

Proof sketch. We first discuss efficiency, then correctness, and finally argue security of the construction.

Efficiency. Let $\mathbb{G}$ be the group output by GrpGen on input 1^λ. From the description of $\hat{C}$, we have

$$|\hat{C}| = |C| + 4D \cdot |\mathsf{mpk}| + \sum_{d=1}^{D} |\hat{\mathcal{L}}_d|,$$

where $|\mathsf{mpk}|$ is the size of a PPRF public key and $\hat{\mathcal{L}}_d$ is the shifts sent for the d-th layer. The size of $|\mathsf{mpk}|$ is $2N^m + 1$ elements of $\mathbb{G}$, which is polynomial in λ since $m = O\left(\sqrt{\log \lambda}\right)$, $c = 2^{\sqrt{\lambda}}$ and $N = O(2c \cdot m \cdot (2^m + 1))$. The size of each $\hat{\mathcal{L}}_d$ is $O(\lambda \cdot n_d)$ since it contains n_d constant-length tuples of shifts, and each shift is $O(\lambda)$. Thus, there exists a polynomial $\mathsf{poly}'(\cdot)$ such that

$$\begin{aligned} |\hat{C}| &= |C| + 4D \cdot \mathsf{poly}'(\lambda) + O(\lambda) \frac{|C|}{t} + O(\lambda) \cdot D \cdot t \\ &\le |C| + D \cdot \mathsf{poly}(\lambda) + O(\lambda) \frac{|C|}{\sqrt{\log \lambda}} \end{aligned}$$

where poly $\in \omega(\lambda \cdot \log \lambda)$.

Correctness. The proof of correctness closely follows the one discussed in Section 6.2 since most of the subprocedures remain largely similar. The primary difference is that the garbling maintains a slightly modified invariant where a different PPRF key is used for authenticating the output of gates in each layer. We first discuss the invariant in more detail and then focus on proving the correctness of the $\mathsf{BatchAND}^{\mathsf{H}}$ gadget. The proof of correctness for the rest of the gadgets follows almost immediately from the proofs of the corresponding lemmas in Section 6.

Let x denote an input C. For each gate u, let x_u denote the bit output by this gate in computation of $C(x)$. Given a batch $\mathcal{B}$ of gates, let $x_{\mathcal{B}} := (x_u)_{u \in \mathcal{B}}$. The garbling scheme then maintains the following invariant for every layer $\mathcal{L}_d$: for each $i \leq n_d$, $x_u = k_u \oplus \ell_u$ and $L_u - K_u = \Delta_{4d} \cdot \ell_u \bmod p-1$, for all $u \in \mathsf{Left}_{d,i} \cup \mathsf{Right}_{d,i}$. As the base case, for each input gate i, we have $k_i \oplus \ell_i = x_i$ and $L_i - K_i = \Delta_0 \ell_i \bmod p-1$ from the description of e and GC.Enc. Now, assuming the invariant holds for a layer $d-1$, we have the following.

- It follows directly from the proof of Lemma 16 that after running the $\mathsf{Pack}^{\mathsf{H}}$ procedures, $k_{\mathsf{l}} + \ell_{\mathsf{l}} = \Phi(\mathsf{pad}_t(x_{\mathsf{Left}_{d,i}}))$, $k_{\mathsf{r}} + \ell_{\mathsf{r}} = \Phi(\mathsf{pad}_t(x_{\mathsf{Right}_{d,i}}))$, $L_{\mathsf{l}} - K_{\mathsf{l}} = \Delta_{4d-3} \cdot \ell_{\mathsf{l}}(N) \bmod p-1$, and $L_{\mathsf{r}} - K_{\mathsf{r}} = \Delta_{4d-3} \cdot \ell_{\mathsf{r}}(N) \bmod p-1$. Let $x_{\mathsf{l}} := \Phi(\mathsf{pad}_t(x_{\mathsf{Left}_{d,i}}))$ and $x_{\mathsf{r}} := \Phi(\mathsf{pad}_t(x_{\mathsf{Right}_{d,i}}))$.
- For each batch of AND gates in $\mathcal{L}_d$, it follows from the claim below, that after running the $\mathsf{BatchAND}^{\mathsf{H}}$ procedure $k_{\mathsf{out}} + \ell_{\mathsf{out}} = x_{\mathsf{l}} \cdot x_{\mathsf{r}}$ and $L_{\mathsf{out}} - K_{\mathsf{out}} = \Delta_{4d-1} \cdot \ell_{\mathsf{out}}(N) \bmod p-1$.
- For each batch of AND gates in $\mathcal{L}_d$, it follows from the proof of Lemma 17 that running the $\mathsf{Unpack}^{\mathsf{H}}$ procedure outputs $(k[i] \oplus \ell[i])_{i \leq t-1} = \Psi(x_{\mathsf{l}} \cdot x_{\mathsf{r}})$ and $L[i] - K[i] = \Delta_{4d} \cdot \ell[i] \bmod p-1$ for $i = 0$ to $t-1$.
- Similarly, for each batch of XOR gates in $\mathcal{L}_d$, by the proof of Lemma 18 and Lemma 17, it that after running the $\mathsf{BatchXOR}^{\mathsf{H}}$ and $\mathsf{Unpack}^{\mathsf{H}}$ procedures that $(k[i] \oplus \ell[i])_{i \leq t-1} = \Phi^{-1}(x_{\mathsf{l}} \oplus x_{\mathsf{r}})$ and $L[i] - K[i] = \Delta_{4d} \cdot \ell[i] \bmod p-1$ for $i = 0$ to $t-1$.

It follows that after each AND layer, the gates in $\mathcal{B}_{d,i}$ are labeled with the first $|\mathcal{B}_{d,i}|$ entries of $\Psi(\Phi(\mathsf{pad}_t(x_{\mathsf{Left}_{d,i}})) \cdot \Phi(\mathsf{pad}_t(x_{\mathsf{Right}_{d,i}})))$. By definition of the RMFE maps (Definition 5), this is value equal to $\mathsf{pad}_t(x_{\mathsf{Left}_{d,i}}) \odot \mathsf{pad}_t(x_{\mathsf{Right}_{d,i}})$, hence its first $|\mathcal{B}_{d,i}|$ entries are exactly the products $x_{u_{\mathsf{l}}} \cdot x_{u_{\mathsf{r}}}$, where $u_{\mathsf{l}}, u_{\mathsf{r}}$ denote the left and right parents of each gate $u \in \mathcal{B}_{d,i}$ respectively. Similarly, after each XOR layer, each gate u of the layer gets labeled with $x_{u_{\mathsf{l}}} \oplus x_{u_{\mathsf{r}}}$. Eventually, after all layers have been computed, it holds that $k_o \oplus \ell_o = x_o$ for each output gate o, and we have $(x_o)_{o \in O(C)} = y = C(x)$.

We are left to prove that the invariant indeed holds for the output of $\mathsf{BatchAND}^{\mathsf{H}}$.

Claim. *Fix* $(\mathsf{mpk}, \mathsf{msk}, \mathsf{mpk}', \mathsf{msk}', \Delta_{\mathsf{out}}, (k_{\mathsf{l}}, K_{\mathsf{l}}, \ell_{\mathsf{l}}, L_{\mathsf{l}}), (k_{\mathsf{r}}, K_{\mathsf{r}}, \ell_{\mathsf{r}}, L_{\mathsf{r}}), \mathsf{salt})$ *where* $\mathsf{msk} := (\mathsf{mpk}, \Delta, h_0)$ *and* $\mathsf{msk}' := (\mathsf{mpk}', \Delta', h_0')$. *Assume that for* $u \in \{\mathsf{l}, \mathsf{r}\}$, $L_u - K_u = \Delta \cdot \mathsf{Eval}_N(\ell_u) \bmod p-1$. *Then, denoting*

$$x_u := k_u + \ell_u \text{ for } u \in \{\mathsf{l}, \mathsf{r}\} \qquad \triangleright \text{ over } \mathbb{F}_{2^m}$$
$$((k_{\mathsf{out}}, K_{\mathsf{out}}), S) := \mathsf{BatchAND}^{\mathsf{H}}_{\mathsf{G}}(\mathsf{msk}, \mathsf{msk}', \Delta_{\mathsf{out}}, (k_{\mathsf{l}}, K_{\mathsf{l}}), (k_{\mathsf{r}}, K_{\mathsf{r}}), \mathsf{salt})$$
$$(\ell_{\mathsf{out}}, L_{\mathsf{out}}) := \mathsf{BatchAND}^{\mathsf{H}}_{\mathsf{E}}(\mathsf{mpk}, \mathsf{mpk}', (\ell_{\mathsf{l}}, L_{\mathsf{l}}), (\ell_{\mathsf{r}}, L_{\mathsf{r}}), \mathsf{salt}, S),$$

it holds that

$$k_{\mathsf{out}} + \ell_{\mathsf{out}} = x_{\mathsf{l}} \cdot x_{\mathsf{r}} \qquad \triangleright \text{ over } \mathbb{F}_{2^m}$$
$$L_{\mathsf{out}} - K_{\mathsf{out}} = \Delta_{\mathsf{out}} \cdot \mathsf{Eval}_N(\ell_{\mathsf{out}})) \bmod p-1$$

Proof. Since the computation of ℓ_{out} and k_{out} in Algorithms 18 and 19 remains identical to that in Algorithms 11 and 12, it follows from the proof of Lemma 19 that

$$\ell_{\text{out}} + k_{\text{out}} = x_{\mathsf{l}} \cdot x_{\mathsf{r}}. \qquad \triangleright \text{ over } \mathbb{F}_{2^m} = \mathbb{F}_2[X]/P(X)$$

Similarly, we have

$$\begin{aligned} \alpha_{\mathsf{E}} - \alpha_{\mathsf{G}} &= \ell_{\mathsf{r}}(N) \cdot (\Delta' k'_{\mathsf{l}}(N)) \\ \beta_{\mathsf{E}} - \beta_{\mathsf{G}} &= \ell_{\mathsf{l}}(N) \cdot (\Delta' k'_{\mathsf{r}}(N)). \end{aligned}$$

Now, since the conditions of Lemma 11 are satisfied, we have

$$L'_{\mathsf{l}} - K'_{\mathsf{l}} = \Delta' \cdot \ell_{\mathsf{l}}(N),$$

which in turn implies that

$$\gamma_{\mathsf{E}} - \gamma_{\mathsf{G}} = K'_{\mathsf{l}} \cdot \ell_{\mathsf{r}}(N).$$

Thus, we have

$$\begin{aligned} \tilde{L}_{\text{out}} - \tilde{K}_{\text{out}} &= \alpha_{\mathsf{E}} + \beta_{\mathsf{E}} - \gamma_{\mathsf{E}} + L'_{\mathsf{l}}\ell_{\mathsf{r}}(N) - (\alpha_{\mathsf{G}} + \beta_{\mathsf{G}} - \gamma_{\mathsf{G}} + \Delta'(\tilde{a}_{\mathsf{G}} + \tilde{b}_{\mathsf{G}})) \\ &= (\alpha_{\mathsf{E}} - \alpha_{\mathsf{G}}) + (\beta_{\mathsf{E}} - \beta_{\mathsf{G}}) - (\gamma_{\mathsf{E}} - \gamma_{\mathsf{G}}) + \Delta(\tilde{a}_{\mathsf{G}} + \tilde{b}_{\mathsf{G}}) + L'_{\mathsf{l}}\ell_{\mathsf{r}}(N) \\ &= \Delta' \cdot (\ell_{\mathsf{r}}(N) \cdot k'_{\mathsf{l}}(N) + \ell_{\mathsf{l}}(N) \cdot k'_{\mathsf{r}}(N)) - \ell_{\mathsf{r}}(N) \cdot K'_{\mathsf{l}} \\ &\qquad + \Delta(\tilde{a}_{\mathsf{E}} - \ell_{\mathsf{r}}(N) \cdot k'_{\mathsf{l}}(N) + \tilde{b}_{\mathsf{E}} - \ell_{\mathsf{l}}(N) \cdot k'_{\mathsf{r}}(N)) + L'_{\mathsf{l}}\ell_{\mathsf{r}}(N) \\ &= \Delta' \cdot (\tilde{a}_{\mathsf{E}} + \tilde{b}_{\mathsf{E}}) + \ell_{\mathsf{r}}(N) \cdot (L'_{\mathsf{l}} - K'_{\mathsf{l}}) \\ &= \Delta' \cdot (\tilde{a}_{\mathsf{E}} + \tilde{b}_{\mathsf{E}} + \ell_{\mathsf{r}}(N) \cdot \ell_{\mathsf{l}}(N)) \qquad \triangleright L_{\mathsf{l}} - K_{\mathsf{l}} = \Delta' \cdot \ell_{\mathsf{l}}(N) \\ &= \Delta' \cdot \tilde{\ell}_{\text{out}}, \end{aligned}$$

Consequently, apply Lemma 13, we have

$$\begin{aligned} L_{\text{out}} - K_{\text{out}} &= \Delta_{\text{out}} \cdot \mathsf{Mod}_N(\tilde{\ell}_{\text{out}}, 2, P) \bmod p - 1 \\ &= \Delta_{\text{out}} \cdot \tilde{\ell}_{\text{out}}(N) \bmod p - 1. \end{aligned}$$

□

Security. The proof of security closely follows the one discussed in Section 7 and we only highlight the differences. In more detail, Hybrid_1 remains identical, while in Hybrid_2 we modify the simulator $\mathsf{SimBatchAnd}^{\mathsf{H},\mathcal{O}}$ to simulate L'_{l} using SimVtO and compute $\tilde{L}_{\text{out}} \coloneqq \alpha_{\mathsf{E}} + \beta_{\mathsf{E}} - \gamma_{\mathsf{E}} + L'_{\mathsf{l}}\ell_{\mathsf{r}}(N)$. In the simulator for each gadget, we modify use the modified variant of $\mathsf{SimBatchfAuth}^{\mathsf{H},\mathcal{O}}$ as discussed in Remark 2. Using a similar argument as the one used in Section 7, it follows that Hybrid_1 is identical to Hybrid_2. In Hybrid_3, we replace the oracle $\mathcal{O}$ with a random oracle $\mathcal{R}$. Observe that calls by $\mathsf{SimBatchfAuth}^{\mathsf{H},\mathcal{O}}$ to $\mathcal{O}$, under the same PPRF public key mpk, can be batched and can thus be computed using the leveled TCCR hash of Definition 8. This is because simulating the shift and evaluator's output share for $\mathsf{Pack}^{\mathsf{H}}$ only requires the evaluator's shares computed as the output of the previous layer. Similarly, simulating $\mathsf{BatchXOR}^{\mathsf{H}}$ and $\mathsf{Unpack}^{\mathsf{H}}$ only requires the evaluator's share computing using $\mathsf{Pack}^{\mathsf{H}}$, and $\mathsf{BatchAND}^{\mathsf{H}}$ or $\mathsf{BatchXOR}^{\mathsf{H}}$ in the current layer. Finally, it is easy to see that this is true for $\mathsf{BatchAND}^{\mathsf{H}}$ too since shift_α, shift_β, $\mathsf{shift}_{K'}$

and α_{E}, β_{E}, and L'_{I} can be simulated in parallel. It then follows from Theorem 7 that Hybrid_3 is indistinguishable from Hybrid_2. The proof then proceeds similarly to that in Section 7.1 where in Hybrid_4, SimVtO and $\mathsf{SimBatchfAuth}^{\mathsf{H},\mathcal{O}}$ are modified to not require the garbler's shares and subsequently, in Hybrid_5, we rely on the correctness of the scheme to set $\mathsf{d} \coloneqq (\ell_o \oplus y_o)_{o \in O(C)})$. It follows that the garbling scheme is secure. ∎

The following corollary immediately follows from Theorem 30 and the definition of rate of a boolean garbling scheme.

Corollary 31. *Let λ be the security parameter. If for every integer-valued polynomial $\mathsf{B} \coloneqq \mathsf{B}(\lambda)$ the B-power DDH holds with respect to a group generator and there exists a TCR hash for the exponential correlation with respect to this group generator, then there exists a boolean garbling scheme for polynomial size layered circuits with rate $\frac{\lambda}{\sqrt{\log \lambda}}$.*

References

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

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

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

[BGI16] Elette Boyle, Niv Gilboa, and Yuval Ishai. Breaking the circuit size barrier for secure computation under DDH. In Matthew Robshaw and Jonathan Katz, editors, *Advances in Cryptology – CRYPTO 2016, Part I*, volume 9814 of *Lecture Notes in Computer Science*, pages 509–539, Santa Barbara, CA, USA, August 14–18, 2016.

[BGL+15] Nir Bitansky, Sanjam Garg, Huijia Lin, Rafael Pass, and Sidharth Telang. Succinct randomized encodings and their applications. In Rocco A. Servedio and Ronitt Rubinfeld, editors, *47th Annual ACM Symposium on Theory of Computing*, pages 439–448, Portland, OR, USA, June 14–17, 2015. ACM Press.

[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: 19th Conference on Computer and Communications Security*, pages 784–796, Raleigh, NC, USA, October 16–18, 2012. ACM Press.

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

[CCXY18] Ignacio Cascudo, Ronald Cramer, Chaoping Xing, and Chen Yuan. Amortized complexity of information-theoretically secure MPC revisited. In Hovav Shacham and Alexandra Boldyreva, editors, *Advances in Cryptology – CRYPTO 2018, Part III*, volume 10993 of *Lecture Notes in Computer Science*, pages 395–426, Santa Barbara, CA, USA, August 19–23, 2018.

[CG20] Ignacio Cascudo and Jaron Skovsted Gundersen. A secret-sharing based MPC protocol for boolean circuits with good amortized complexity. In Rafael Pass and Krzysztof Pietrzak, editors, *TCC 2020: 18th Theory of Cryptography Conference, Part II*, volume 12551 of *Lecture Notes in Computer Science*, pages 652–682, Durham, NC, USA, November 16–19, 2020.

[CHHK25] Geoffroy Couteau, Carmit Hazay, Aditya Hegde, and Naman Kumar. Breaking the $1/\lambda$-rate barrier for arithmetic garbling. In *Advances in Cryptology – EUROCRYPT 2025*, 2025.

[CKKZ12] Seung Geol Choi, Jonathan Katz, Ranjit Kumaresan, and Hong-Sheng Zhou. On the security of the "free-XOR" technique. In Ronald Cramer, editor, *TCC 2012: 9th Theory of Cryptography Conference*, volume 7194 of *Lecture Notes in Computer Science*, pages 39–53, Taormina, Sicily, Italy, March 19–21, 2012.

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

[Cou19] Geoffroy Couteau. A note on the communication complexity of multiparty computation in the correlated randomness model. In Yuval Ishai and Vincent Rijmen, editors, *Advances in Cryptology – EUROCRYPT 2019, Part II*, volume 11477 of *Lecture Notes in Computer Science*, pages 473–503, Darmstadt, Germany, May 19–23, 2019.

[DGM23] Nico Döttling, Phillip Gajland, and Giulio Malavolta. Laconic function evaluation for Turing machines. In Alexandra Boldyreva and Vladimir Kolesnikov, editors, *PKC 2023: 26th International Conference on Theory and Practice of Public Key Cryptography, Part II*, volume 13941 of *Lecture Notes in Computer Science*, pages 606–634, Atlanta, GA, USA, May 7–10, 2023.

[DI05] Ivan Damgård and Yuval Ishai. Constant-round multiparty computation using a black-box pseudorandom generator. In Victor Shoup, editor, *Advances in Cryptology – CRYPTO 2005*, volume 3621 of *Lecture Notes in Computer Science*, pages 378–394, Santa Barbara, CA, USA, August 14–18, 2005.

[DIK10] Ivan Damgård, Yuval Ishai, and Mikkel Krøigaard. Perfectly secure multiparty computation and the computational overhead of cryptography. In Henri Gilbert, editor, *Advances in Cryptology – EUROCRYPT 2010*, volume 6110 of *Lecture Notes in Computer Science*, pages 445–465, French Riviera, May 30 – June 3, 2010.

[EHL+23] Daniel Escudero, Cheng Hong, Hongqing Liu, Chaoping Xing, and Chen Yuan. Degree-D reverse multiplication-friendly embeddings: Constructions and applications. In Jian Guo and Ron Steinfeld, editors, *Advances in Cryptology – ASIACRYPT 2023, Part I*, volume 14438 of *Lecture Notes in Computer Science*, pages 106–138, Guangzhou, China, December 4–8, 2023.

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

[GKWY20] Chun Guo, Jonathan Katz, Xiao Wang, and Yu Yu. Efficient and secure multiparty computation from fixed-key block ciphers. In *2020 IEEE Symposium on Security and Privacy*, pages 825–841, San Francisco, CA, USA, May 18–21, 2020. IEEE Computer Society Press.

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

[HIMV19] Carmit Hazay, Yuval Ishai, Antonio Marcedone, and Muthuramakrishnan Venkitasubramaniam. LevioSA: Lightweight secure arithmetic computation. In Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz, editors, *ACM CCS 2019: 26th Conference on Computer and Communications Security*, pages 327–344, London, UK, November 11–15, 2019. ACM Press.

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

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

[IKSS22] Yuval Ishai, Dakshita Khurana, Amit Sahai, and Akshayaram Srinivasan. Round-optimal black-box protocol compilers. In Orr Dunkelman and Stefan Dziembowski, editors, *Advances in Cryptology – EUROCRYPT 2022, Part I*, volume 13275 of *Lecture Notes in Computer Science*, pages 210–240, Trondheim, Norway, May 30 – June 3, 2022.

[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, *Advances in Cryptology – CRYPTO 2014, Part II*, volume 8617 of *Lecture Notes in Computer Science*, pages 440–457, Santa Barbara, CA, USA, August 17–21, 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: 35th International Colloquium on Automata, Languages and Programming, Part II*, volume 5126 of *Lecture Notes in Computer Science*, pages 486–498, Reykjavik, Iceland, July 7–11, 2008. Springer, Heidelberg, Germany.

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

[LWYY24] Hanlin Liu, Xiao Wang, Kang Yang, and Yu Yu. Garbled circuits with 1 bit per gate. Cryptology ePrint Archive, Report 2024/712, 2024.

[MORS24] Pierre Meyer, Claudio Orlandi, Lawrence Roy, and Peter Scholl. Rate-1 arithmetic garbling from homomorphic secret sharing. Lecture Notes in Computer Science, pages 71–97, November 2024.

[PS21] Antigoni Polychroniadou and Yifan Song. Constant-overhead unconditionally secure multi-party computation over binary fields. In Anne Canteaut and François-Xavier Standaert, editors, *Advances in Cryptology – EUROCRYPT 2021, Part II*, volume 12697 of *Lecture Notes in Computer Science*, pages 812–841, Zagreb, Croatia, October 17–21, 2021.

[PSSW09] Benny Pinkas, Thomas Schneider, Nigel P. Smart, and Stephen C. Williams. Secure two-party computation is practical. In Mitsuru Matsui, editor, *Advances in Cryptology – ASIACRYPT 2009*, volume 5912 of *Lecture Notes in Computer Science*, pages 250–267, Tokyo, Japan, December 6–10, 2009.

[PW09] Rafael Pass and Hoeteck Wee. Black-box constructions of two-party protocols from one-way functions. In Omer Reingold, editor, *TCC 2009: 6th Theory of Cryptography Conference*, volume 5444 of *Lecture Notes in Computer Science*, pages 403–418, March 15–17, 2009.

[QWW18] Willy Quach, Hoeteck Wee, and Daniel Wichs. Laconic function evaluation and applications. In Mikkel Thorup, editor, *59th Annual Symposium on Foundations of Computer Science*, pages 859–870, Paris, France, October 7–9, 2018. IEEE Computer Society Press.

[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, *Advances in Cryptology – CRYPTO 2021, Part I*, volume 12825 of *Lecture Notes in Computer Science*, pages 94–124, Virtual Event, August 16–20, 2021.

[Sho97] Victor Shoup. Lower bounds for discrete logarithms and related problems. In Walter Fumy, editor, *Advances in Cryptology – EUROCRYPT'97*, volume 1233 of *Lecture Notes in Computer Science*, pages 256–266, Konstanz, Germany, May 11–15, 1997.

[SS24] Sacha Servan-Schreiber. Constrained pseudorandom functions for inner-product predicates from weaker assumptions. Cryptology ePrint Archive, Report 2024/058, 2024.

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

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

A Security Proofs for the Building Blocks

A.1 Proof of Lemma 8

Proof. We first consider the case of constant polynomials, that is, when $m = 0$, and then generalize to an arbitrary $m \in \mathbb{N}$. It is easy to see that the support of $\mathsf{Pert}_c(a)$ and $\mathsf{RandSum}_{0,c}$ is contained in $[c]$ and consequently, the support of $\mathcal{D}_{0,c}^{(a)}$ and $\mathcal{D}_{0,c}$ is contained in $[2c]$. Moreover, the output of $\mathsf{RandSum}_{0,c}$ is the binomial distribution $\mathsf{Binomial}(c, 1/2)$. Next, observe that the output of $\mathsf{Pert}_c(a)$ is distributed identically to that of $\mathsf{RandSum}_{0,c}$ conditioned on the latter outputting a value congruent to a mod 2 i.e., for any $u \in [c]$,

$$\Pr_{a' \leftarrow\$ \mathsf{Pert}_c(a)}[a' = u] = \Pr_{a' \leftarrow\$ \mathsf{RandSum}_{0,c}}\left[a' = u \;\middle|\; a' \equiv a \bmod 2\right].$$

Therefore, for any $u \in [2c]$, we have

$$\begin{aligned}
&\left|\Pr_{a'+r\sim\mathcal{D}_{0,c}^{(a)}}[a' + r = u] - \Pr_{a'+r\sim\mathcal{D}_{0,c}}[a' + r = u]\right| \\
= \;&\left|\Pr_{a'+r\sim\mathcal{D}_{0,c}}\left[a' + r = u \;\middle|\; a' \equiv a \bmod 2\right] - \Pr_{a'+r\sim\mathcal{D}_{0,c}}[a' + r = u]\right| \\
= \;&\frac{1}{2}\left|\Pr[a' + r = u \mid a' \equiv a \bmod 2] - \Pr[a' + r = u \mid a' \not\equiv a \bmod 2]\right|
\end{aligned} \tag{1}$$

where the second equality follows from the fact that $\Pr[a' \equiv a \bmod 2] = 1/2$ when $a' \leftarrow\$ \mathsf{RandSum}_{0,c}$. When u is odd, we have

$$\begin{aligned}
&\Pr[a' + r = u \mid a' \equiv a \bmod 2] - \Pr[a' + r = u \mid a' \not\equiv a \bmod 2] \\
= \;&\sum_{\substack{v\in[u] \\ v\equiv a \bmod 2}} \frac{1}{2^{2c}} \cdot \binom{c}{v} \cdot \binom{c}{u-v} - \sum_{\substack{v\in[u] \\ v\not\equiv a \bmod 2}} \frac{1}{2^{2c}} \cdot \binom{c}{v} \cdot \binom{c}{u-v} \\
= \;&0,
\end{aligned}$$

where the first equality follows from the fact that a' and r are distributed as $\mathsf{Binomial}(c, 1/2)$ and the second equality follows from recognizing that the summation is the co-efficient of X^u in $(1 - X)^c \cdot (1 + X)^c = (1 - X^2)^c$.

Similarly, when u is even, we have

$$\left|\Pr[a' + r = u \mid a' \equiv a \bmod 2] - \Pr[a' + r = u \mid a' \not\equiv a \bmod 2]\right| = \frac{1}{2^{2c}} \cdot \binom{c}{u/2}.$$

In summary, Equation (1) simplifies to

$$\left|\Pr_{a'+r\sim\mathcal{D}_{0,c}^{(a)}}[a' + r = u] - \Pr_{a'+r\sim\mathcal{D}_{0,c}}[a' + r = u]\right| = \begin{cases} 0 & \text{if } u \text{ is odd,} \\ \frac{1}{2^{2c}} \cdot \binom{c}{u/2} & \text{otherwise} \end{cases}.$$

Consequently,

$$\begin{aligned}
\mathsf{SD}\left(\mathcal{D}_{0,c}^{(a)}, \mathcal{D}_{0,c}\right) &= \frac{1}{2} \cdot \sum_{u=0}^{2c}\left|\Pr_{a'+r\sim\mathcal{D}_{0,c}^{(a)}}[a' + r = u] - \Pr_{a'+r\sim\mathcal{D}_{0,c}}[a' + r = u]\right| \\
&= \frac{1}{2}\sum_{u=0}^{c} \frac{1}{2^{2c}} \cdot \binom{c}{u/2} \\
&= \frac{1}{2^{c+1}}
\end{aligned}$$

To conclude the proof, observe that for all $m \in \mathbb{N}$, every coefficient in the output of $\mathcal{D}^{(a)}_{m,c}$ and $\mathcal{D}_{m,c}$ is independently and identically distributed to $\mathcal{D}^{(a)}_{0,c}$ and $\mathcal{D}_{0,c}$ respectively. It follows that

$$\mathrm{SD}\left(\mathcal{D}^{(a)}_{m,c}, \mathcal{D}_{m,c}\right) = 1 - \left(1 - \frac{1}{2^{c+1}}\right)^m \leq \frac{m}{2^c}.$$

■

A.2 Proof of Lemma 9

Proof. Intuitively, the proof follows from the fact that when $N > T \cdot c \cdot m$, the sum used to compute $\tilde{v}$ does not produce any carries when viewed as a base-N integer.

More formally, let $u = \sum_{i=1}^{T} a_i \cdot b'_i$ where the sum is computed by interpreting each a_i and b'_i as elements of $\mathbb{N}[X]$. Observe that for every $i \in [2m]$, the co-efficient $u[i]$ of X^i in u, is of the form

$$u[i] = \sum_{\ell=1}^{T} \sum_{j=\max(0,i-m)}^{\min(i,m)} a_\ell[j] \cdot b'_\ell[i-j],$$

where $a_\ell[j]$ and $b'_\ell[i-j]$ denote the co-efficients of X^j and X^{i-j} in a_ℓ and b'_ℓ respectively. Since each $a_\ell[j]$ and $b'_\ell[i-j]$ are non-negative integers such that $a_\ell[j] \leq 1$ and $b'_\ell[i-j] \leq c$, it follows that $u[i]$ is a non-negative integer with $u[i] \leq T \cdot c \cdot m < N$. In particular, this means that $(u[i])_{i \in [2m]}$ represents the unique N-ary decomposition of $u(N)$. It then immediately follows that $v = \mathsf{toPoly}_N\left(\mathsf{Mod}_N\left(\tilde{v}, 2\right)\right)$ since, by definition, $u \equiv v \bmod 2$ and $u(N) = \tilde{v}$. ■