# Authenticated Garbling with Tensor Gates*

Nakul Khambhati [1], Turan Vural [1], David Heath [2], and Rafail Ostrovsky [1]

[1]University of California, Los Angeles, nakul@cs.ucla.edu, turan@cs.ucla.edu, rafail@cs.ucla.edu
[2]University of Illinois Urbana-Champaign, daheath@illinois.edu

**Abstract**

Authenticated garbling (Wang et al., CCS'17) is the leading framework for achieving constant-round maliciously secure 2PC, achieving total communication comparable to protocols in the semi-honest setting. In the semi-honest setting, the one-hot garbling scheme of Heath and Kolesnikov (CCS'21) supports tensor gates $\boldsymbol{x} \otimes \boldsymbol{y}$, for $\boldsymbol{x} \in \{0,1\}^n$ and $\boldsymbol{y} \in \{0,1\}^m$, in $O(n+m)\kappa$ bits, versus $O(nm)\kappa$ for an AND-gate-based garbling. Extending this improvement to the malicious setting has remained open.

We present two authenticated garbling protocols that efficiently handle tensor gates. Our first protocol minimizes garbled circuit size, matching the size of semi-honest one-hot garbling; all additional overhead is confined to preprocessing. We also design a custom preprocessing protocol for efficiently generating random authenticated tensor triples, proved secure assuming only a circular correlation robust hash function and base OTs.

Our second protocol minimizes total communication via compressed (PCG-style) preprocessing. We construct an authenticated variant of one-hot garbling where the additional overhead in garbled-circuit size as compared to the semi-honest garbling scales with tensor-gate input length rather than output length.

As in the semi-honest case, tensor gates are restricted to inputs of length $O(\log \kappa)$ and arbitrary-length tensor products are handled by tiling multiple short input gates. This yields an $O(\log \kappa)$ factor reduction in communication for general tensor products compared to previous state-of-the-art authenticated garbling protocols, with concrete improvements in communication and runtime for practical workloads such as integer and matrix multiplication within an authenticated garbled circuit.

**Keywords:** Garbled Circuits, Malicious Security, Tensor Operations

*An extended abstract of this work appears in the proceedings of ACM CCS 2026, https://doi.org/10.1145/3830454.3846761. This is the full version.

# Contents


**1 Introduction** **3**
- 1.1 Our Contribution . . . 4

**2 Preliminaries** **5**
- 2.1 Notation . . . 5
- 2.2 Circuits with Tensor Gates . . . 5
- 2.3 Hash Functions . . . 5
- 2.4 Authenticated Shares . . . 5
- 2.5 $(m, L)$-Independent Matrix . . . 6

**3 Overview** **6**
- 3.1 One-Hot Garbling . . . 7
- 3.2 Authenticated Garbling . . . 8
- 3.3 Protocol 1: Optimizing Garbled Circuit Size . . . 9
- 3.4 Compressed Preprocessing . . . 10
- 3.5 Protocol 2: Optimizing Total Communication . . . 11
- 3.6 A Note on Selective-Failure . . . 12

**4 Protocol 1: Optimizing Garbled Circuit Size** **12**
- 4.1 Half Tensor Gate . . . 15
- 4.2 Garbling and evaluation algorithms . . . 15
- 4.3 The full 2PC protocol . . . 16
- 4.4 Communication complexity . . . 16
- 4.5 Security . . . 17

**5 Protocol 2: Optimizing total communication** **17**
- 5.1 Communication complexity . . . 18
- 5.2 Security . . . 18

**6 Applications** **18**

**7 Experiments** **19**

**A Helper Functionalities** **25**
- A.1 Equality Check . . . 25
- A.2 Correlated OT . . . 25

**B Security Proofs** **25**
- B.1 Proof of Lemma 5 . . . 25
- B.2 Proof of Theorem 6 . . . 26
- B.3 Proof of Theorem 8 . . . 29

**C Preprocessing for Protocol 1** **31**
- C.1 Generalized Half-Tensor Gate . . . 32
- C.2 Leaky Tensor Triples . . . 33
- C.3 Combining Leaky Tensor Triples . . . 35
- C.4 Randomized Bucketing . . . 36

**D Compressed Preprocessing For Protocol 2** **37**

# 1 Introduction

Yao's Garbled Circuit (GC) is a cryptographic technique allowing a *garbler* $\mathsf{G}$ to construct from some program $P$ a *privacy-preserving* encoding of that program $\tilde{P}$. $\mathsf{G}$ sends this encoding to an *evaluator* $\mathsf{E}$, along with an encoding of program input $\tilde{x}$, and then $\mathsf{E}$ can locally compute $\tilde{P}(\tilde{x}) = P(x)$. Crucially, the encoding $\tilde{P}, \tilde{x}$ *keeps $x$ hidden.*

GC is particularly relevant to the study of secure two-party and multi-party computation (2PC and MPC), where it is used to achieve protocols that complete within only a small constant number of communication rounds. In the semi-honest setting, it is relatively straightforward to achieve 2PC by combining GC with oblivious transfer; see e.g. [LP09]. The state-of-the-art concretely efficient semi-honest symmetric-key-based GC protocol for boolean circuits communicates $\approx 1.5\kappa$ bits per AND gate, where $\kappa$ is the computational security parameter, and XOR gates are free [RR21].

We remark that this work focuses on GC techniques that are concretely efficient: recent works [MORS24, ILL25, LWYY25b, LWYY25a] have demonstrated remarkable theoretical results from stronger cryptography (e.g. homomorphic secret sharing), but these remain several orders of magnitude slower in practice.

**Malicious Security:** Our focus in this work is on constructing 2PC protocols that remain secure even in the presence of a malicious adversary who deviates arbitrarily from the protocol description. Garbling techniques are often automatically secure against a malicious evaluator $\mathsf{E}$, but *insecure* against a malicious garbler $\mathsf{G}$: $\mathsf{G}$ can garble a circuit incorrectly, such that an honest $\mathsf{E}$ obtains incorrect output. This insecurity is largely inherent, as the very mechanisms that protect $\mathsf{G}$'s privacy also prevent $\mathsf{E}$ from detecting that the circuit has been incorrectly garbled.

*Cut and choose* is a classic mechanism by which to protect against malicious $\mathsf{G}$ [LP15, KNR$^+$17]. In this paradigm, $\mathsf{G}$ generates *many* garblings of the same program, $\mathsf{E}$ checks that a random subset of them is correctly generated, and the parties evaluate the remaining unchecked garblings. With some care, this basic idea can lift *any* semi-honest-secure garbling technique to the malicious setting in a black-box manner. Unfortunately, the need to generate, transmit, and evaluate many garblings incurs multiplicative overhead in a statistical security parameter $\rho$, in terms of both computation and communication.

The state-of-the-art *authenticated garbling* framework [WRK17] adopts a different approach: $\mathsf{G}$ sends only one, potentially incorrect garbling. $\mathsf{E}$ evaluates this single garbling and checks correctness of obtained values *after evaluation*. In particular, rather than checking the garbling itself, the evaluator checks (masked) intermediate wire values that the garbling reveals to her.

As we explain in Section 3, the authenticated garbling approach requires that $\mathsf{G}$ and $\mathsf{E}$ jointly preprocess appropriate *correlated randomness*. This randomness is sampled and secret-shared between the parties in such a manner that some underlying random string (consisting of AND triples) is both hidden from and authenticated to each party. $\mathsf{G}$ is then forced to garble the circuit starting from its share of the randomness. Doing so lets $\mathsf{E}$ detect when a wire value has been tampered with, and randomizes wire values that emerge during evaluation, providing privacy against the evaluator and preventing a form of *selective failure* attack by the garbler.

Recent protocols [DILO22, CWYY23] have discovered communication-efficient methods for preprocessing the required randomness, and the resulting constant-round maliciously-secure protocols incur total communication comparable to the best semi-honest protocol: [KRRW18] garbles at $2\kappa + 1$ bits per AND gate with preprocessing costing $O(\kappa)$ bits per triple, while [CWYY23] achieves total communication $2\kappa + \rho + 5$ bits, amortized per AND gate.

Unlike cut and choose, authenticated garbling is *not* a compiler for upgrading semi-honest GC to the malicious setting: an innovation in the semi-honest setting does not automatically imply state-of-the-art malicious performance, and custom techniques are needed.

*Garbling Beyond Boolean Circuits:* Classically, GC techniques operate on simple fan-in two Boolean gates. However, the semi-honest GC literature has evolved to include concretely-efficient ways to garble more expressive computational primitives. Examples include special-purpose techniques for garbling RAM programs [LO13, HKO22, PLS23, HKO23, GTRS25], programs with conditional branching [HK20, HK21a], and arithmetic circuits [BMR16].

One exciting semi-honest GC technique is the so-called *one-hot garbling* technique [HK21b]. This technique reduces the communication cost of garbling certain *tensor operations*, such as matrix multiplications,

and it offers concretely efficient solutions for certain high-fan-in gates. The one-hot garbling technique can be understood as a *generalization* of the fan-in-two "half gates" approach of [ZRE15], a technique that remains state of the art in the authenticated garbling setting [KRRW18]. The low-level nature of one-hot garbling has made it useful as a building block for other semi-honest techniques, including garbled lookup tables [HKN24, LLZ25, KN25] and garbled arithmetic circuits [Hea24]. The technique is also noteworthy in that it offers a communication/computation tradeoff, allowing end-to-end runtime improvement in settings where network bandwidth is the bottleneck. To make the improvement concrete, garbling the tensor product of two length-$n$ vectors from AND gates costs $O(n^2)$ ciphertexts, while one-hot garbling costs $O(n)$, and tensor products are the core of binary matrix multiplication and of schoolbook integer multiplication.

It was not known how to bring efficient one-hot-style techniques to the malicious setting. A natural question is whether authenticated garbling of richer circuit models already implies such a result. [HKO23] authenticate *tri-state circuits*, the basis of garbled RAM, using a WRK-style approach [WRK17]. Tri-state circuits do not suffice for one-hot garbling: the improved one-hot construction relies on the bidirectional capabilities of switch systems [Hea24], which have not been proved secure in the authenticated setting, and even granting such a proof, tensor gates consume richer preprocessed correlations than AND triples. Our work identifies how to generalize the consistency checks and preprocessing of prior work to this setting. In particular, we give a selective-failure analysis of the one-hot construction itself, enabling cost optimizations that prior authenticated garbling of non-Boolean gates did not perform.

## 1.1 Our Contribution

We upgrade the one-hot garbling technique with malicious security. Specifically, we augment the authenticated garbling technique with support for *tensor gates*. Given two strings $\boldsymbol{x} \in \{0,1\}^n, \boldsymbol{y} \in \{0,1\}^m$, a tensor gate computes the $n \times m$ matrix $\boldsymbol{x} \otimes \boldsymbol{y}$.

We give two authenticated garbling protocols for tensor gates, one in each style of state-of-the-art authenticated garbling, distinguished by how the consistency check is handled: the first optimizes the garbled circuit size, with much of the overhead pushed to preprocessing [KRRW18], and the second optimizes total communication [WRK17, DILO22, CWYY23].

Our Protocol 1 (Section 4) optimizes garbled circuit size and results in a GC of size $2(n+m-1)\kappa+2(n+m)$ bits of communication per tensor gate, matching the semi-honest one-hot tensor-gate cost [HK21b, Hea24] up to lower-order terms. We prove this protocol secure in Minicrypt assuming base OTs. Preprocessing dominates total communication, and a main challenge for Protocol 1 is preprocessing appropriate correlated randomness. Naively generating a tensor triple from $nm$ authenticated AND triples would cost asymptotically more than the garbled circuit itself, so we design a preprocessing protocol that produces tensor triples at cost $2B(n+m)(\kappa+\rho)$ bits per gate, where $B = \rho/\log|\mathcal{C}|$ is a bucketing parameter.

Our Protocol 2 (Section 5) uses so-called *compressed preprocessing* to optimize total communication. This protocol uses a pseudorandom correlation generator (PCG) for efficiently generating preprocessed correlations, and it therefore requires additionally assuming LPN [BCG$^+$19b, YWL$^+$20]. In particular, the preprocessing for each tensor gate incurs cost $4nm$ bits, independent of the security parameter $\kappa$. With careful handling, our use of compressed preprocessing enables an authenticated tensor gate with total communication cost $2(n+m-1)\kappa + (n+m)(\rho+1) + 4nm$ bits.

Due to limitations inherited from the semi-honest one-hot garbling technique, each basic tensor gate consumes *computation* that scales exponentially in its input lengths, so we handle longer tensor products by tiling them with $\log\kappa$-input gates. Setting $n = m = \log\kappa$, our protocols concretely improve over prior state of the art handling of tensor products by a factor $\approx \log\kappa/2$.

Finally, we present concrete applications of our protocol, building on the work from the semi-honest setting [HK21b], describing how our improvement speeds up integer and matrix multiplication within an authenticated GC. We remark that secure matrix multiplication is a key component of many AI applications, including neural network implementations [KNR$^+$17, WGC19, WTB$^+$21]. We provide exact costs and benchmark a Rust implementation of our protocols in Section 7, which shows a 5.8× improvement in communication and 2.9–4.0× improvement in wall clock time over prior state of the art when computing a $256 \times 256$ bit tensor product.

# 2 Preliminaries

## 2.1 Notation

We denote the computational and statistical security parameters using $\kappa$ and $\rho$, respectively. We write log to denote base-2 logarithms. For $m \in \mathbb{N}$, we write $[m] := \{0, 1, \ldots, m-1\}$, and we index vectors and matrices from 0. Given a finite set $S$, we write $x \leftarrow S$ to denote sampling $x$ uniformly at random from $S$.

We use bold symbols (e.g., $\boldsymbol{x}, \boldsymbol{X}$) to denote vectors and matrices. We write $x_i$ for the $i$-th component of vector $\boldsymbol{x}$, and $X_{i,j}$ for the $(i,j)$-th entry of matrix $\boldsymbol{X}$.

We denote garbling and evaluation using $\mathsf{Garble}$, $\mathsf{Eval}$, and refer to $\mathsf{G}$, $\mathsf{E}$ as the garbler and evaluator. We use $[\cdot]$ to denote an XOR sharing of a bit(string) between $\mathsf{G}$ and $\mathsf{E}$, with $[\cdot]^{\mathsf{G}}$ (resp. $[\cdot]^{\mathsf{E}}$) denoting the garbler's (resp. evaluator's) share.

As Section 2.4 explains, we will extensively use the notion of *dual-authenticated shares*, which we denote using $[\![x]\!] := [x(\Delta_{\mathsf{G}} \| \Delta_{\mathsf{E}} \| 1)]$ where $\Delta_{\mathsf{G}}, \Delta_{\mathsf{E}}$ are the garbler's and evaluator's global keys.

Finally, to declutter notation, we use circuit wire *indices* and *values* interchangeably; a wire $x$ carries bit $x$.

## 2.2 Circuits with Tensor Gates

We consider a class of Boolean circuits consisting of XOR gates and arbitrary-fan-in *tensor* gates. Specifically, a circuit $\mathcal{C}$ consists of a list of gates in the form $(\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c}, T)$ where $\boldsymbol{a}$, $\boldsymbol{b}$ are vectors of input wire indices of length $n$ and $m$, respectively, $\boldsymbol{c}$ is an $n \times m$ matrix of output wire indices, and $T \in \{\oplus, \otimes\}$. If $T = \oplus$, then $|\boldsymbol{a}| = |\boldsymbol{b}| = |\boldsymbol{c}|$; such a gate computes the bit-wise XOR of its input vectors. If $T = \otimes$ then $|\boldsymbol{c}| = |\boldsymbol{a}| \cdot |\boldsymbol{b}|$ and we compute the tensor of two Boolean vectors as $c_{ij} = a_i b_j$ for $(i,j) \in [n] \times [m]$.

## 2.3 Hash Functions

Our work uses a circular correlation robust under naturally derived keys ($\mathsf{ccrnd}$) hash function $\mathsf{H}$ [CKKZ12, ZRE15]. Intuitively, a $\mathsf{ccrnd}$ produces random-looking output, even when hashing strings related by some correlation $\Delta$, and even when using the output of the hash to encrypt strings that also involve the same correlation $\Delta$. This enables the Free XOR technique [KS08], which in turn is required for the half-gates [ZRE15] and one-hot garbling [HK21b] techniques which we build upon. We use the definition of [ZRE15]:

**Definition 1** (Circular Correlation Robustness Under Naturally Derived Keys)**.** *Define two oracles that each accept as input a label $K \in \{0,1\}^\kappa$, a nonce $i$, and a bit $b$: the oracle $\mathsf{circ}_\Delta(K, i, b) \triangleq \mathsf{H}(K \oplus \Delta, i) \oplus b \cdot \Delta$, where $\Delta \in \{0,1\}^{\kappa-1} || 1$, and a random function $\mathcal{R}(K, i, b)$ with $\kappa$-bit output. A sequence of oracle queries $(K, i, b)$ is* legal *when the same value $(K, i)$ is never queried with different values of $b$. $\mathsf{H}$ is* circular correlation robust under naturally derived keys ($\mathsf{ccrnd}$) *if no poly-time adversary issuing legal queries can distinguish $\mathsf{circ}_\Delta$ (over uniform $\Delta$) from $\mathcal{R}$.*

$\mathsf{H}$ is often efficiently instantiated using fixed key AES [GKWY20].

## 2.4 Authenticated Shares

We use information-theoretic message authentication codes (IT-MACs) [BDOZ11, NNOB12] to authenticate bits. Let $\Delta_{\mathsf{G}}, \Delta_{\mathsf{E}} \in \mathbb{F}_2^\kappa$ be *global keys* held by $\mathsf{G}$ and $\mathsf{E}$ respectively.

Given a bit $x$, $[x\Delta_{\mathsf{G}}]$ is an XOR share of $x\Delta_{\mathsf{G}}$, which serves as an IT-MAC on $x$ under the garbler's key; similarly $[x\Delta_{\mathsf{E}}]$ is an IT-MAC under the evaluator's key. We write $[x]$ for a plain XOR share of $x$ itself. A *dual-authenticated sharing* of $x$, denoted $[\![x]\!]$, bundles all three: $[x\Delta_{\mathsf{G}}]$, $[x\Delta_{\mathsf{E}}]$, and $[x]$. The defining feature of such sharings is that *either* party can authentically reveal $x$ to the other: the share under the receiver's key serves as a MAC that the receiver verifies against its own key.

Since dual-authenticated sharings consist of XOR secret shares, they are naturally linearly homomorphic, and we extend the notation $[\![\cdot]\!]$ to vectors and matrices in the natural element-wise manner.

**Preprocessed Correlations**

Our protocols consume two kinds of preprocessed correlations, generated in batch for efficiency: dual-authenticated sharings of uniformly random bits $\lambda$, and dual-authenticated sharings of *tensor triples*, i.e. $\boldsymbol{\lambda}_a$, $\boldsymbol{\lambda}_b$, $\boldsymbol{\lambda}_a \otimes \boldsymbol{\lambda}_b$ for uniformly random $\boldsymbol{\lambda}_a, \boldsymbol{\lambda}_b$.

**Opening and Zero-Checking**

We define two standard procedures [NNOB12, DNNR17, CWYY23], parameterized by the *prover* $p \in \{\mathsf{G}, \mathsf{E}\}$ who asserts values to the other party (the *verifier* $v$). Each procedure operates on shares authenticated under $v$'s key.

$\mathsf{CheckZero}_p([x_1\Delta_v], \ldots, [x_\ell\Delta_v])$: This procedure checks that each value $x_1, \ldots, x_\ell$ is a zero. $p$ sends $h_p := \mathcal{R}([x_1\Delta_v]^p, \ldots, [x_\ell\Delta_v]^p)$ to $v$, who computes $h_v := \mathcal{R}([x_1\Delta_v]^v, \ldots, [x_\ell\Delta_v]^v)$ and aborts if $h_p \neq h_v$. Here $\mathcal{R}$ is a hash function modeled as a random oracle. Communication is $\kappa$ bits total, independent of $\ell$. The above yields a batch-opening procedure:
$\mathsf{Open}_p([x_1], [x_1\Delta_v], \ldots, [x_\ell], [x_\ell\Delta_v])$: reveals $x_1, \ldots, x_\ell$ to $v$. For each $i$, $p$ sends its share $[x_i]^p$; $v$ computes $x_i := [x_i]^p \oplus [x_i]^v$. Parties then run $\mathsf{CheckZero}_p$ on $([x_1\Delta_v] \oplus x_1\Delta_v, \ldots, [x_\ell\Delta_v] \oplus x_\ell\Delta_v)$.

**Lemma 2** ([CWYY23, DNNR17, WYKW21])**.** *If $\mathcal{R}$ is modeled as a random oracle and $\Delta_v \leftarrow \mathbb{F}_2^\kappa$ is uniform, then the probability that $\mathsf{CheckZero}_p$ accepts when some $x_i \neq 0$ is at most $2/2^\kappa$.*

Our protocols use standard functionalities $\mathcal{F}_{\mathrm{COT}}$ and $\mathcal{F}_{\mathsf{EQ}}$ from prior works. For completeness, we provide a description of these functionalities and how they can be instantiated in Appendix A.

## 2.5 $(m, L)$-Independent Matrix

We recall the notion of an $(m, L)$-independent matrix, which originates from [DK99] and has been applied in the context of authenticated garbling in [DILO22, CWYY23].

**Definition 3** ([CWYY23])**.** *We say that a matrix $\boldsymbol{M} \in \mathbb{F}_2^{n\times L}$ is $(m, L)$-independent if any $m$ rows of $\boldsymbol{M}$ are linearly independent.*

This property implies the following: if $\boldsymbol{M} \in \mathbb{F}_2^{n\times L}$ is $(m, L)$-independent and $\boldsymbol{s} \leftarrow \mathbb{F}_2^L$ is sampled uniformly at random, then the product $\boldsymbol{u} := \boldsymbol{M} \cdot \boldsymbol{s}$ yields an $m$-wise independent distribution.

**Lemma 4** ([CWYY23])**.** *Let $L = \left\lceil \rho + m \cdot \log\left(\frac{en}{m}\right) + \frac{\log m}{2} \right\rceil$, and let $\boldsymbol{M} \leftarrow \mathbb{F}_2^{n\times L}$ be chosen uniformly at random. Then $\boldsymbol{M}$ is $(m, L)$-independent except with probability $2^{-\rho}$.*

We refer the reader to [CWYY23] for the proof. We include this definition and lemma as they are used in the compressed preprocessing functionality $\mathcal{F}_{\mathrm{cpre}}$ (Appendix D).

# 3 Overview

This section presents intuition underlying our two authenticated garbling protocols for tensor gates. For simplicity, we consider tensor gates with input vectors of length $n = m = \log\kappa$, as the one-hot tensor-gate construction has computation exponential in the input lengths.

State-of-the-art authenticated garbling protocols use the half-gate construction [ZRE15] as a subroutine within their garbling procedures. Two such approaches are relevant, distinguished by the mechanism by which the garbler proves correctness of values revealed to the evaluator.

- **Optimize garbled circuit size:** The protocol of [KRRW18] moves the overhead of malicious security to a preprocessing step: it runs the semi-honest half-gate construction as a black box, with the parties' correlated randomness supplied as preprocessed shares. The half-gate construction reveals intermediate masked bits to $\mathsf{E}$ to guide its actions. After evaluation, $\mathsf{E}$ sends the revealed bits to $\mathsf{G}$, who constructs a "proof" that each is *consistent* with a correct execution and sends it back to $\mathsf{E}$. Here, cost is dominated by preprocessing.

| | Authenticated GC (bits/gate) | Preprocessing (bits/gate) | Assumptions |
|---|---|---|---|
| Semi-honest one-hot [HK21b, Hea24] | $4\kappa \log \kappa$ | — | RO |
| [KRRW18] on $\log^2 \kappa$ AND gates | $2\kappa \log^2 \kappa$ | $2B\kappa \log^2 \kappa$ | base OTs + RO |
| [CWYY23] on $\log^2 \kappa$ AND gates | $(2\kappa + \rho) \log^2 \kappa$ | $4 \log^2 \kappa$ | base OTs + RO + LPN |
| Protocol 1 (Section 4) | $4\kappa \log \kappa$ | $4B\kappa \log \kappa$ | base OTs + RO |
| Protocol 2 (Section 5) | $(4\kappa + 2\rho) \log \kappa$ | $4 \log^2 \kappa$ | base OTs + RO + LPN |

Table 1: Amortized communication per tensor gate at $n = m = \log \kappa$, with lower-order terms dropped and preprocessing measured in the $(\mathcal{F}_{\mathrm{COT}}, \mathcal{F}_{\mathsf{EQ}})$-hybrid model. $B \approx \rho / \log |\mathcal{C}|$ is the bucketing parameter and $|\mathcal{C}|$ the number of non-XOR gates.

- **Optimize total communication:** The line of work introduced by [WRK17] and refined in [DILO22, CWYY23] optimizes *total* communication: it augments the half-gate construction in such a way that each revealed bit is authenticated by the garbled circuit itself *during evaluation*, and $\mathsf{E}$ never needs to send these revealed bits to $\mathsf{G}$. Because these bits are never revealed to $\mathsf{G}$, these protocols can use so-called *compressed preprocessing*, whereby the correlated randomness does not have full entropy from $\mathsf{G}$'s perspective, allowing more efficient preprocessing if one additionally assumes LPN for constructing PCGs.

We begin by observing that the tensor-gate construction of [HK21b, Hea24] strictly generalizes the semi-honest half-gate construction [ZRE15]. Our goal is to lift this generalization to the malicious setting.

We construct two protocols, one for each above setting, each generalizing [HK21b]'s semi-honest tensor-gate construction along a different axis. Our first protocol (Section 3.3) optimizes garbled circuit size and inherits the semi-honest cost of the tensor-gate construction, $4\kappa \log \kappa$ bits per tensor gate, compared to $2\kappa \log^2 \kappa$ using the naive approach of decomposing a $\log \kappa \times \log \kappa$ tensor gate into $\log^2 \kappa$ AND gates and using [KRRW18]. All overhead is moved to a preprocessing phase, and a main technical challenge is in efficiently generating the required authenticated shares of random *tensor correlations*. We give a preprocessing protocol tailored to tensor triples with cost $2B(\kappa+\rho) \log \kappa$ bits per gate, where $B = \rho / \log |\mathcal{C}|$ is a bucketing parameter, using a generalized one-hot garbling that accepts arbitrary XOR-sharings as one input and an optimized bucketing procedure.

Our second protocol (Section 3.5) optimizes total communication and is compatible with compressed preprocessing. Prior protocols in this style augment the GC with a mechanism to authenticate individual AND gates, which for our setting would exceed the target communication. We construct an authenticated variant of [HK21b]'s one-hot garbling that authenticates all values revealed to an evaluator, with cost scaling in inputs to the tensor gate resulting in a garbled circuit size of $(4\kappa + 2\rho) \log \kappa$ bits per tensor gate and preprocessing cost of $O(\log^2 \kappa)$ bits per tensor gate.

Table 1 summarizes the per-gate communication of both protocols alongside the baselines, dropping lower-order terms. The first row recalls the semi-honest one-hot cost, making the price of malicious security explicit. Protocol 1 matches the semi-honest garbled circuit size up to lower-order terms and pays for malicious security entirely in preprocessing. Protocol 2 pays an additive $2\rho \log \kappa$ bits of garbled circuit and $4 \log^2 \kappa$ bits of preprocessing per gate. Concrete counts for a representative workload are deferred to Section 7. Our concrete improvement over prior state of the art for *short input* tensor gates is $\log \kappa / 2$. In Section 6, we explain how we can tile arbitrary length tensor products using tensor gates so that our concrete improvement lies between $\log \kappa / 2$ and $\log \kappa$, depending on the size of the computed tensor product.

Our first protocol is the simpler of the two constructions; we present it first both as an exposition step and as a protocol of independent interest as it does not require assuming LPN. Our second protocol achieves the best total communication and is correspondingly more involved.

## 3.1 One-Hot Garbling

Naively computing a tensor product within a GC using AND gates would require $O(nm)$ ciphertexts, and [HK21b]'s one-hot garbling technique improves this to $O(m + n)$ ciphertexts, achieving a communication/-computation tradeoff that is concretely efficient.

For a string $\boldsymbol{x} \in \{0,1\}^n$, write $\mathrm{val}(\boldsymbol{x}) := \sum_{j=0}^{n-1} x_j \cdot 2^j \in [2^n]$ for its integer representation. The *one-hot encoding* of $\boldsymbol{x}$ is the string $\mathcal{H}(\boldsymbol{x}) \in \{0,1\}^{2^n}$ that is 1 at index $\mathrm{val}(\boldsymbol{x})$ and 0 at all other indices [HK21b].

Given shares $[\boldsymbol{x}\Delta_{\mathsf{G}}]$ and $[\boldsymbol{y}\Delta_{\mathsf{G}}]$, where $\boldsymbol{x} \in \{0,1\}^n, \boldsymbol{y} \in \{0,1\}^m$, and provided $\mathsf{E}$ knows $\boldsymbol{x}$ in cleartext, [HK21b] compute shares of the one-hot tensor product $[\mathcal{H}(\boldsymbol{x}) \otimes \boldsymbol{y}\ \Delta_{\mathsf{G}}]$ using $2(n-1)+m$ ciphertexts in total. [Hea24], in line with observations of [GYW+23], improved this cost to $n-1+m$ ciphertexts. [Hea24] presented this improved construction in the language of *switch-systems*; we present an alternate formulation in Section 4 and upgrade it to the authenticated setting. The technique incurs exponential scaling in computation and is therefore restricted to $n = O(\log \kappa)$.

Observe that the one-hot encoding $\mathcal{H}(\boldsymbol{x})$ is fully homomorphic in the sense that it linearizes arbitrary function evaluation. Let $f : \{0,1\}^n \to \{0,1\}^m$ be any function and let $\mathcal{T}(f) \in \{0,1\}^{2^n \times m}$ denote its truth table, represented as a matrix. Then

$$\mathcal{T}(f)^{\intercal} \cdot \mathcal{H}(\boldsymbol{x}) = f(\boldsymbol{x}).$$

To obtain $[\boldsymbol{x} \otimes \boldsymbol{y}\ \Delta_{\mathsf{G}}]$ from $[\mathcal{H}(\boldsymbol{x}) \otimes \boldsymbol{y}\ \Delta_{\mathsf{G}}]$, we apply the identity $\mathcal{T}(\mathrm{id})^{\intercal} \cdot (\mathcal{H}(\boldsymbol{x}) \otimes \boldsymbol{y}) = \boldsymbol{x} \otimes \boldsymbol{y}$ (Lemma 3, [HK21b]). This allows the parties to locally convert shares of the one-hot tensor product to shares of the standard tensor product using XOR homomorphism.

## 3.2 Authenticated Garbling

We now provide a high-level overview of the authenticated garbling framework, focusing on how it addresses the shortcomings of semi-honest GC.

**Semi-honest garbled circuits.** In the FreeXOR regime [KS08], semi-honest GC protocols maintain the invariant that parties hold XOR shares $[w\Delta_{\mathsf{G}}]$ of each circuit wire value $w$, authenticated under a global key $\Delta_{\mathsf{G}}$ held by the garbler. XOR gates are handled locally by XORing shares. For non-XOR gates, the garbler sends cryptographic material (the garbled circuit) that lets the evaluator propagate shares of gate inputs to shares of gate outputs. The constant-round requirement forces the garbler to sample all of its shares before evaluation begins, so the garbled circuit material is sent once, upfront.

Authentication under $\Delta_{\mathsf{G}}$ provides *integrity*: an evaluator cannot flip a share from $[w\Delta_{\mathsf{G}}]$ to $[(w \oplus 1)\Delta_{\mathsf{G}}]$ without guessing $\Delta_{\mathsf{G}}$. To protect the garbler's privacy, the garbler samples a random mask bit $\lambda$ for each wire $w$ and reveals to the evaluator the *masked value* $w \oplus \lambda$. The garbled circuit is structured so that the evaluator can use $w \oplus \lambda$ to propagate its shares $[w\Delta_{\mathsf{G}}]$ across circuit gates without learning $w$.

**Malicious garbler.** Observe that nothing above prevents a malicious garbler from corrupting the garbled material it sends so that an honest evaluator would obtain an incorrect output. In fact, the privacy mechanism above is precisely what makes it difficult for an evaluator to detect such a corruption.

The authenticated garbling framework addresses this by having parties propagate XOR shares not just under $\Delta_{\mathsf{G}}$, but also under a second global key $\Delta_{\mathsf{E}}$ held by the evaluator. Shares under $\Delta_{\mathsf{E}}$ function as linearly homomorphic commitments on wire values: once evaluation is complete, the garbler can use its $\Delta_{\mathsf{E}}$-shares to prove to the evaluator, in a single *consistency check*, that revealed (masked) values the evaluator obtained are consistent with a correct garbling of the circuit. In particular, this approach does *not* directly check correctness of the garbled circuit ciphertexts, but rather of all values revealed by the garbled circuit to an evaluator.

**Selective failure and jointly-sampled masks.** Irrespective of the exact consistency-check, there is a subtle issue that arises in the above framework. In standard GC protocols, when evaluating a non-XOR gate, the evaluator decrypts a subset of the received ciphertexts, determined by the masked values that were revealed to it. A malicious garbler can corrupt specific ciphertexts in the garbled material, so that an evaluator receives correct shares (and the consistency check passes) iff the evaluator did not use any of the corrupted ones: a *selective failure attack*. The abort outcome reveals whether or not the evaluator used the corrupted ciphertexts, so such an attack can be used to *guess* a revealed (masked) value $w \oplus \lambda$ used in evaluation. If the garbler had sampled the masks $\{\lambda\}$ itself, it would then recover $w = (w \oplus \lambda) \oplus \lambda$, violating privacy.

The fix (observed by [WRK17]) is for parties to instead jointly sample masks during preprocessing, as authenticated XOR shares under both keys: $[\lambda\Delta_{\mathsf{G}}], [\lambda\Delta_{\mathsf{E}}], [\lambda]$. These are the dual-authenticated bits introduced in Section 2.4. The garbler uses its $\Delta_{\mathsf{G}}$-shares of the masks to produce the cryptographic material for share propagation, exactly as in the semi-honest setting except with masks held as shares rather than in the clear. The consistency check uses the $\Delta_{\mathsf{E}}$-shares. Now, a masked value $w \oplus \lambda$ revealed via a selective failure attack is uniformly distributed from the garbler's view, because $\lambda$ is, so the attack reveals nothing about $w$ (this assumes the masks are uniform in the garbler's view, which we revisit in Section 3.6).

Authenticated garbling thus consists of three components: a preprocessing protocol generating dual-authenticated bits of (correlated) masks; a garbling/evaluation procedure to propagate input to output authenticated shares; and a post-evaluation consistency check using the $\Delta_{\mathsf{E}}$-shares to verify correctness. Protocols within this framework differ primarily in how the consistency check is structured, which in turn affects how the $\Delta_{\mathsf{E}}$-shares are propagated through the circuit. Our two protocols sit at different points within this framework, as we discuss in Sections 3.3 and 3.5.

### 3.3 Protocol 1: Optimizing Garbled Circuit Size

Recall that the half-gate construction [ZRE15] is core to all state-of-the-art authenticated garbling protocols [KRRW18, DILO22, CWYY23], and [HK21b]'s tensor-gate construction generalizes half-gates in the semi-honest setting. The hope is therefore to "slot in" the tensor-gate construction into one of these protocols to obtain authenticated garbling of tensor gates. To see whether this works, we examine how the half-gate construction garbles an AND gate, and how [KRRW18], the first protocol to use half-gates for authenticated garbling, uses it within the framework of Section 3.2.

The half-gate construction [ZRE15] is built around a privacy-free AND gate that, given $[a\Delta_{\mathsf{G}}]$ where the bit $a$ is held by the evaluator in the clear and $[b\Delta_{\mathsf{G}}]$, propagates shares $[ab\,\Delta_{\mathsf{G}}]$ using one ciphertext of communication. A private AND gate can be garbled by rewriting

$$ab \;=\; (a \oplus \lambda_a)\,\lambda_b \;\oplus\; (b \oplus \lambda_b)\,a \;\oplus\; \lambda_a\lambda_b,$$

by revealing masked values $a \oplus \lambda_a, b \oplus \lambda_b$ to an evaluator and using two privacy-free AND gates, assuming parties hold $\Delta_{\mathsf{G}}$-shares of $\lambda_b$, $a$ and $\lambda_a\lambda_b$. In the semi-honest setting, we let the garbler sample these masks $\lambda_a, \lambda_b, \lambda_a\lambda_b$ so it can simply set its share to $\lambda\Delta_{\mathsf{G}}$, and the evaluator sets its share to 0.

The *only* difference in propagating $\Delta_{\mathsf{G}}$-shares through an AND gate in the malicious setting is that parties now need to preprocess $\Delta_{\mathsf{G}}$-shares of $(\lambda_\alpha, \lambda_\beta, \lambda_\alpha\lambda_\beta)$, as the masks themselves are secret-shared between the garbler and evaluator. [KRRW18] takes the following approach for using the $\Delta_{\mathsf{E}}$-shares to check consistency. After evaluation, the evaluator sends back all revealed masked values $a \oplus \lambda_a, b \oplus \lambda_b$ to the garbler, which is safe because the masks are jointly sampled so masked values are uniformly distributed from the garbler's view. Input encoding is modified so parties also hold shares under $\Delta_{\mathsf{E}}$, and preprocessing additionally authenticates the masks under $\Delta_{\mathsf{E}}$ now. With masked values public, they can *locally* propagate these shares through the circuit. XOR gates are again free. At each AND gate, parties compute $[ab\Delta_{\mathsf{E}}] = (a \oplus \lambda_a)[\lambda_b\Delta_{\mathsf{E}}] \oplus (b \oplus \lambda_b)[a\Delta_{\mathsf{E}}] \oplus [\lambda_a\lambda_b\Delta_{\mathsf{E}}]$. Now, the garbler can use its $\Delta_{\mathsf{E}}$-share to prove consistency of each revealed value to the evaluator. These checks can be batched into a single hash: the garbler sends the hash of its contribution across all revealed values, the evaluator hashes its own, and aborts if they disagree. If the garbler reveals an incorrect masked value on some wire, passing the check requires guessing $\Delta_{\mathsf{E}}$, which succeeds with negligible probability. All the communication overhead of malicious security is thus moved to the preprocessing phase, whose task is to efficiently generate random authenticated AND triples.

**Checking inputs versus outputs.** Our consistency check deviates from [KRRW18] in one respect, which we now explain. To evaluate a non-XOR gate, the evaluator must hold the masked values of the gate's inputs in the clear. There are two ways to maintain this invariant. [KRRW18] reveals the masked value of each AND gate output and of each circuit input, which inductively places a masked bit on every wire. We instead reveal the masked values entering each non-XOR gate. For AND gates the second option is slightly worse, revealing two bits per gate rather than one, but for tensor gates, whose output length is asymptotically larger than their input length, it is an improvement. Revealing inputs also supports circuit-independent preprocessing, as each gate consumes a uniformly random triple rather than one bound to particular wire masks. The two checks verify the same relation between revealed values. Once the masked values entering

non-XOR gates are public, every gate becomes linear, so the parties can locally propagate IT-MACs from circuit inputs to circuit outputs, and a single batched check then validates all revealed bits at once.

The tensor-gate construction [HK21b, Hea24] follows the same pattern. At its core is a privacy-free protocol that, given $[\boldsymbol{a}\,\Delta_{\mathsf{G}}]$ where $\boldsymbol{a} \in \{0,1\}^n$ is held by the evaluator in the clear and $[\boldsymbol{b}\,\Delta_{\mathsf{G}}]$ for $\boldsymbol{b} \in \{0,1\}^m$, propagates shares $[\boldsymbol{a} \otimes \boldsymbol{b}\,\Delta_{\mathsf{G}}]$ using $(n+m-1)\kappa$ bits of communication. A private tensor gate is then garbled by the lifted rewrite

$$\boldsymbol{a} \otimes \boldsymbol{b} \;=\; (\boldsymbol{a} \oplus \boldsymbol{\lambda}_a) \otimes \boldsymbol{\lambda}_b \;\oplus\; ((\boldsymbol{b} \oplus \boldsymbol{\lambda}_b) \otimes \boldsymbol{a})^{\intercal} \;\oplus\; \boldsymbol{\lambda}_a \otimes \boldsymbol{\lambda}_b,$$

revealing $\boldsymbol{a} \oplus \boldsymbol{\lambda}_a$ and $\boldsymbol{b} \oplus \boldsymbol{\lambda}_b$ to an evaluator and using two privacy-free tensor gates for a total of $2(n+m-1)\kappa$ bits per tensor gate.

The structural parallel with the AND case means we can slot the tensor-gate construction into [KRRW18]: the same procedure with privacy-free tensor gates in place of privacy-free AND gates to propagate $\Delta_{\mathsf{G}}$-shares through tensor gates, provided the preprocessing supplies *authenticated tensor triples* $(\boldsymbol{\lambda}_a, \boldsymbol{\lambda}_b, \boldsymbol{\lambda}_a \otimes \boldsymbol{\lambda}_b)$ in place of authenticated AND triples. The same consistency check goes through, unchanged. We provide details in Section 4.

The remaining task is to preprocess authenticated tensor triples efficiently. Using KRRW's preprocessing on $\log^2 \kappa$ AND triples to instantiate one tensor triple (setting input vector lengths to $\log \kappa$) costs a factor $\log \kappa$ above our target, and KRRW's preprocessing protocol is not black-box in (privacy-free) AND gate garbling, so we design a custom preprocessing protocol for tensor triples.

**Generalized tensor subroutine.** We observe that the privacy-free semi-honest tensor gate of [HK21b] does not require the second input to be held as a share specifically of the form $\boldsymbol{b}\,\Delta_{\mathsf{G}}$. In fact, this input can be a vector of arbitrary XOR shares $[\boldsymbol{B}]$ held by parties, which propagates to the output as shares of $[\boldsymbol{a} \otimes \boldsymbol{B}]$; the analogous generalization of half-gates was observed in [KRRW18]. Setting $\boldsymbol{B} = \boldsymbol{b}(\Delta_{\mathsf{G}} \oplus \Delta_{\mathsf{E}})$ optimizes the preprocessing communication by reusing garbled material for simultaneously generating and authenticating tensor triples. Two calls to the generalized subroutine, with $\mathsf{G}$ and $\mathsf{E}$ swapping roles, produce a (leaky) authenticated tensor triple from jointly-sampled dual authenticated random vectors $\boldsymbol{x}, \boldsymbol{y}$.

**Leaky tensor triples.** Checking correctness of such triples is not sufficient as a malicious garbler can induce a selective abort based on bits of the evaluator's input $\boldsymbol{x}$, so the triples produced above are *leaky* in the sense of [KRRW18]. A full characterization of the leakage and its simulation is in Appendix C; the main takeaway is that the leakage is bounded and can be eliminated by bucketing.

**Bucketing with tensor permutation.** Standard bucketing randomly partitions and combines $B$ leaky triples into one correct triple such that the combined triple is leaky if and only if all $B$ triples were leaky, reducing the adversary's advantage. In the tensor setting, we can exploit another degree of freedom: before combining tensor triples, the parties randomly permute the coordinates of $\boldsymbol{x}$ and the corresponding rows of $\boldsymbol{x} \otimes \boldsymbol{y}$. The adversary's guessed bits land on random target coordinates, so a smaller bucketing parameter suffices to achieve the same security target. We formalize and prove this in Appendix C.

**Cost.** Protocol 1 generates a garbled circuit of size $4\kappa \log \kappa + 4 \log \kappa$ bits per tensor gate, almost matching the semi-honest cost, except the negligible (less than 1%) cost of masked values sent back from the evaluator to the garbler for the consistency check. The preprocessing costs $4B\kappa \log \kappa$ bits (in the $\mathcal{F}_{\mathrm{COT}}, \mathcal{F}_{\mathsf{EQ}}$-hybrid model), where $B = \rho / \log |\mathcal{C}|$ is the bucket size, determined by number of tensor gates in the circuit ($B = 4$ for a circuit computing a $256 \times 256$ bit tensor product).

Compared to [KRRW18] when tensor gates are constructed using AND gates, this is a factor $\log \kappa / 2$ saving in both the online and preprocessing communication. Note that preprocessing still dominates the total communication, motivating our second protocol.

## 3.4 Compressed Preprocessing

Recent authenticated garbling protocols [DILO22, CWYY23] reduce preprocessing to $O(1)$ bits per AND gate using pseudorandom correlation generators (PCGs) for VOLE which can be efficiently constructed assuming LPN [BCG+19b, YWL+20] and a technique known as *compressed preprocessing*. Parties preprocess a short

uniformly random vector of authenticated correlations and locally expand it by multiplying by a public matrix $\boldsymbol{M}$. Because IT-MACs are linearly homomorphic, authenticating the short seed suffices to authenticate the expanded correlations. This would reduce preprocessing to $O(\log^2 \kappa)$ bits per tensor gate, much less than Protocol 1, where preprocessing is the communication bottleneck.

The tradeoff is that the expanded correlations are not uniform: any single mask remains uniform from the garbler's view, but the full mask vector $\boldsymbol{\lambda}$ lies in a subspace determined by $\boldsymbol{M}$. Such compressed correlations are incompatible with Protocol 1's consistency check, as revealing masked values to the garbler would also reveal the corresponding linear relations among the underlying wire values.

To use compressed preprocessing, the consistency check must avoid revealing masked values to the garbler. The approach introduced by [WRK17] and refined in [DILO22, CWYY23] does this by having the garbler authenticate the masked values to the evaluator during evaluation itself: the garbled circuit is augmented so that the evaluator can use it to propagate shares under both $\Delta_{\mathsf{G}}$ and $\Delta_{\mathsf{E}}$, enabling a consistency check at the end that doesn't require revealing masked values to the garbler. Protocol 2, discussed next, designs an authenticated garbling scheme for tensor gates in this style.

## 3.5 Protocol 2: Optimizing Total Communication

We now describe Protocol 2, which uses compressed preprocessing to optimize total communication, adopting the approach of [WRK17, DILO22, CWYY23] in which the garbler authenticates masked values to the evaluator during evaluation itself.

**Propagating $\Delta_{\mathsf{E}}$ shares.** In this style of authenticated garbling, the garbled circuit is augmented so that, for each non-XOR gate output $\boldsymbol{c}$, the evaluator derives not only a share $[\boldsymbol{c}\,\Delta_{\mathsf{G}}]$ under the garbler's key, but also a share $[\boldsymbol{c}\,\Delta_{\mathsf{E}}]$ on the underlying value under the evaluator's key. Propagating these $\Delta_{\mathsf{E}}$-MACs from gate inputs to gate outputs requires the garbler to transmit additional garbled material per non-XOR gate, beyond the material needed to propagate $\Delta_{\mathsf{G}}$-shares. After evaluation, the garbler opens its $\Delta_{\mathsf{E}}$-shares of all revealed values, i.e. masked inputs to tensor gates and the final circuit output, and the evaluator checks consistency, once again batched via a single hash. Crucially, the masked values are never revealed to the garbler.

**Scaling issue of prior approaches.** Prior protocols of this style augment the garbled circuit with material that scales with *output wires* of AND gates: $4\rho$ bits per wire in [WRK17], $3\rho$ in [DILO22], and $\rho$ in [CWYY23]. For a tensor gate with $nm$ output wires, this would contribute $\rho nm$ bits of online communication per gate, which at $n = m = \log \kappa$ gives $\rho \log^2 \kappa$, exceeding our target. We need a strategy for propagating $\Delta_{\mathsf{E}}$-shares, where cost scales with input length $n + m$ rather than output length $nm$.

**Authenticated tensor gates.** We observe that the semi-honest tensor gate construction has a natural mechanism for propagating values under an additional key: the ciphertexts $G_k$ that allow an evaluator to obtain its respective share for the one-hot location of each output row are of size $\kappa$ and carry $\Delta_{\mathsf{G}}$-shares material, and can be widened to size $\kappa + \rho$ to simultaneously carry $\Delta_{\mathsf{E}}$-shares on the same output bit. The evaluator recovers both kinds of share from one decryption. The resulting privacy-free authenticated tensor subroutine has cost $(n-1)\kappa + m(\kappa + \rho)$ bits per call, and a private authenticated tensor gate costs $2(n + m - 1)\kappa + (n + m)\rho + 2(n + m)$ bits, scaling with input length as required. We give the construction in Section 5.

Specializing the authenticated tensor gate construction to $n = m = 1$ yields an authenticated half-gate protocol with online communication $\approx 2\kappa + 2\rho$ bits per AND gate, slightly worse than the $2\kappa + \rho$ of [CWYY23] but conceptually simple. We have not seen this construction formalized previously.

**Cost.** Protocol 2's online communication is $(4\kappa + 2\rho) \log \kappa$ bits per $\log \kappa$-input tensor gate from the authenticated tensor gate subroutines. Preprocessing is $4 \log^2 \kappa$ bits per tensor gate using [CWYY23]'s compressed-preprocessing protocol for AND gates, naively applied to generate tensor correlations.

### 3.6 A Note on Selective-Failure

Section 3.2 argued that jointly-sampling masks via preprocessing renders the selective-failure attack harmless: a masked value $w \oplus \lambda$ learned through an abort event reveals nothing about $w$ when $\lambda$ is uniform from the garbler's view. This argument breaks down under Protocol 2's compressed preprocessing, because individual masks remain uniform but linear combinations across wires do not.

Concretely, suppose the expansion matrix $\boldsymbol{M}$ constrains the evaluator's expanded shares on two wires $x, y$ to be equal, so the garbler knows $[\lambda_x]^{\mathsf{E}} = [\lambda_y]^{\mathsf{E}}$. A garbler who corrupts ciphertexts so that the evaluator aborts iff $x \oplus \lambda_x, y \oplus \lambda_y \neq 0$ learns $x \oplus \lambda_x = y \oplus \lambda_y = 0$ from a no-abort outcome, which combined with its own shares reveals $x \oplus y$, violating privacy. More generally, any linear dependence among the evaluator's expanded masks admits such an attack: if the protocol does not abort, the garbler learns some linear relation on the underlying values.

Preventing this reduces to ensuring that there is no "short" linear dependence in the expanded masks. Either the garbler corrupts only few ciphertexts, in which case all masked values appear uniformly random and he learns nothing, or he corrupts many ciphertexts, in which case he almost surely gets caught and the protocol aborts (allowing him to learn something useful but with negligible probability). As shown in Lemma 9, $\rho$-wise independent expanded masks suffice for selective-failure resistance, and by Lemma 4 it suffices to authenticate a length $\approx \rho \log |\mathcal{C}|$ correlation and locally expand it by a public random matrix $\boldsymbol{M}$. Selective-failure resistance has been analyzed for the half-gate construction in [DILO22, CWYY23]; ours is, to our knowledge, the first such analysis for a non-Boolean garbling construction.

## 4 Protocol 1: Optimizing Garbled Circuit Size

We present our first authenticated garbling protocol, which follows the blueprint of [KRRW18] in order to optimize the size of the garbled circuit transmitted, realizing the ideal 2PC functionality $\mathcal{F}_{\mathsf{2pc}}$ in the $(\mathcal{F}_{\mathsf{Pre}}, \mathcal{F}_{\mathsf{EQ}})$-hybrid model assuming a circular correlation robust hash function $\mathsf{H}$. The size of the transmitted GC almost matches the semi-honest cost [HK21b, Hea24] of $\approx 2(n+m)\kappa$ bits per tensor gate.

Recall from Section 3.3 the high-level structure. A preprocessing functionality $\mathcal{F}_{\mathsf{Pre}}$ (Functionality 1) samples global keys $\Delta_{\mathsf{G}} \in \{0,1\}^\kappa$ and $\Delta_{\mathsf{E}} \in \{0,1\}^\rho$ for parties and uniform masks $\lambda$ for every input of the circuit. Additionally, for each tensor gate $\boldsymbol{c} = \boldsymbol{a} \otimes \boldsymbol{b}$, it samples tensor triples $\boldsymbol{\lambda}_a, \boldsymbol{\lambda}_b, \boldsymbol{\lambda}_a \otimes \boldsymbol{\lambda}_b$ (of the appropriate dimensions). For each such mask, preprocessing supplies dual-authenticated bits $[\![\lambda]\!] = ([\lambda\Delta_{\mathsf{G}}], [\lambda\Delta_{\mathsf{E}}], [\lambda])$ to both parties, letting an adversary choose its randomness. We design a custom preprocessing protocol $\Pi_{\mathsf{Pre}}$ (Appendix C) to meet our target asymptotics, and in this section assume access to $\mathcal{F}_{\mathsf{Pre}}$.

**Functionality 1: $\mathcal{F}_{\mathsf{Pre}}$ – Preprocessing**

Parameterized by $N \in \mathbb{N}$ authenticated bits and tensor-triple dimensions $\{(n_t, m_t)\}_{t \in [T]} \subseteq \mathbb{N}^2$. Runs with parties $\mathsf{G}$, $\mathsf{E}$ and ideal-world adversary $\mathcal{S}$.

**Initialize.** Sample $\Delta_{\mathsf{G}}, \Delta_{\mathsf{E}} \leftarrow \mathbb{F}_{2^\kappa}$ and output $\Delta_{\mathsf{G}}$ to $\mathsf{G}$, $\Delta_{\mathsf{E}}$ to $\mathsf{E}$. Use $[\![\cdot]\!]$ to denote $[(\cdot)(\Delta_{\mathsf{G}} \| \Delta_{\mathsf{E}} \| 1)]$.

**Authenticated bits.** Sample $\boldsymbol{r} \leftarrow \{0,1\}^N$ uniformly and output $[\![\boldsymbol{r}]\!]$.

**Authenticated tensor triples.** For each $t \in [T]$: sample $\boldsymbol{x}^{(t)} \leftarrow \{0,1\}^{n_t}$, $\boldsymbol{y}^{(t)} \leftarrow \{0,1\}^{m_t}$ uniformly; set $\boldsymbol{Z}^{(t)} := \boldsymbol{x}^{(t)} \otimes \boldsymbol{y}^{(t)}$; output $[\![\boldsymbol{x}^{(t)}]\!], [\![\boldsymbol{y}^{(t)}]\!], [\![\boldsymbol{Z}^{(t)}]\!]$.

**Corrupted parties.** If party $P \in \{\mathsf{G}, \mathsf{E}\}$ is corrupted, $\mathcal{S}$ specifies $\Delta_P$ and, for each output $[\![v]\!]$, the corrupted party's share; this determines the honest party's share.

With the required preprocessing, the garbler constructs a garbled circuit for $\mathcal{C}$ using the tensor-gate construction of [HK21b] with $\Delta_{\mathsf{G}}$-shares of the masks in place of the cleartext masks of the semi-honest setting, and the evaluator decrypts it using its own $\Delta_{\mathsf{G}}$-shares, learning masked input values $\boldsymbol{a} \oplus \boldsymbol{\lambda}_a, \boldsymbol{b} \oplus \boldsymbol{\lambda}_b$ at each tensor gate. After evaluation, $\mathsf{E}$ reveals these masked values to $\mathsf{G}$ who, in turn, propagates its $\Delta_{\mathsf{E}}$-shares of inputs to $\Delta_{\mathsf{E}}$-shares of outputs, which are used for a consistency check and decoding the output to $\mathsf{E}$.

## Construction 1: Tensor (and Authenticated Tensor) macros

*Legend.* Pieces in magenta are present only in the authenticated variant AuthTensor; without them the construction is the unauthenticated Tensor.

$\mathsf{Tensor.Gb}(n, m, \Delta_{\mathsf{G}}, [\boldsymbol{a}\Delta_{\mathsf{G}}]^{\mathsf{G}}, [\boldsymbol{b}\Delta_{\mathsf{G}}]^{\mathsf{G}}, [\boldsymbol{b}\Delta_{\mathsf{E}}]^{\mathsf{G}})$:

1. Interpret $[\boldsymbol{a}\Delta_{\mathsf{G}}]^{\mathsf{G}} = A_0 A_1 \cdots A_{n-1}$ as $n$ labels of length $\kappa$.
2. Initialize the level-0 width-2 encoding: $S_{0,0} := A_0 \oplus \Delta_{\mathsf{G}}$, $S_{0,1} := A_0$.
3. For $i = 1, 2, \ldots, n-1$:
   (a) Compute $R_{i,j} := \mathsf{H}(S_{i-1,j}, \nu_{i,j})$ for each $j \in [2^i]$, where $\nu_{i,j}$ is a one-time nonce.
   (b) Define level-$i$ seeds: for $j \in [2^i]$,
   $$S_{i,j} := R_{i,j} \oplus S_{i-1,j}, \qquad S_{i,2^i+j} := R_{i,j}.$$
   (c) Set $G_i := \left(\bigoplus_{j\in[2^i]} R_{i,j}\right) \oplus A_i$.
4. Set $\boldsymbol{G} := \{G_i\}_{i=1}^{n-1}$.
5. Let $\mathsf{L}_\ell := S_{n-1,\ell}$ for $\ell \in [2^n]$. Use $\mathsf{H}$ with one-time nonces to expand each $\mathsf{L}_\ell$ into $m$ seeds $X_{\ell,k}$ of length $\kappa+\rho$ for $k \in [m]$.
6. Interpret $[\boldsymbol{b}\Delta_{\mathsf{G}}]^{\mathsf{G}} = B_0 B_1 \cdots B_{m-1}$ and $[\boldsymbol{b}\Delta_{\mathsf{E}}]^{\mathsf{G}} = B'_0 B'_1 \cdots B'_{m-1}$. For each $k \in [m]$, define
   $$G_k := \left(\bigoplus_{\ell=0}^{2^n-1} X_{\ell,k}\right) \oplus (B_k \,\|\, B'_k)$$
   and append $\boldsymbol{G} := \boldsymbol{G} \,\|\, \{G_k\}_{k=0}^{m-1}$.
7. Set $\boldsymbol{X}_{\mathsf{G}} := (X_{\ell,k})_{\ell\in[2^n],k\in[m]}$, let id denote the identity on $\{0,1\}^n$, and output
   $$\boldsymbol{Z}_{\mathsf{G}} \,\|\, \boldsymbol{Z}'_{\mathsf{G}} := \mathcal{T}(\mathrm{id})^{\intercal} \cdot \boldsymbol{X}_{\mathsf{G}}, \quad \boldsymbol{G},$$
   where $\boldsymbol{Z}_{\mathsf{G}}$ and $\boldsymbol{Z}'_{\mathsf{G}}$ are the $\kappa$-prefix and $\rho$-suffix of each entry.

$\mathsf{Tensor.Ev}(n, m, \boldsymbol{a}, \boldsymbol{G}, [\boldsymbol{a}\Delta_{\mathsf{G}}]^{\mathsf{E}}, [\boldsymbol{b}\Delta_{\mathsf{G}}]^{\mathsf{E}}, [\boldsymbol{b}\Delta_{\mathsf{E}}]^{\mathsf{E}})$:

1. Parse $\boldsymbol{G} = \{G_i\}_{i=1}^{n-1} \,\|\, \{G_k\}_{k=0}^{m-1}$ and $[\boldsymbol{a}\Delta_{\mathsf{G}}]^{\mathsf{E}} = A_0 \oplus a_0\Delta_{\mathsf{G}}, \ldots, A_{n-1} \oplus a_{n-1}\Delta_{\mathsf{G}}$.
2. Set $S^{\mathsf{E}}_{0,0} := S^{\mathsf{E}}_{0,1} := A_0 \oplus a_0\Delta_{\mathsf{G}}$ and $\alpha_0 := a_0$.
3. For $i = 1, 2, \ldots, n-1$:
   (a) Compute $R^{\mathsf{E}}_{i,j} := \mathsf{H}(S^{\mathsf{E}}_{i-1,j}, \nu_{i,j})$ for each $j \in [2^i] \setminus \{\alpha_{i-1}\}$.
   (b) Recover the active seed from $G_i$:
   $$R^{\mathsf{E}}_{i,\alpha_{i-1}} := G_i \oplus \left(\bigoplus_{j\neq\alpha_{i-1}} R^{\mathsf{E}}_{i,j}\right) \oplus (A_i \oplus a_i\Delta_{\mathsf{G}}).$$
   (c) Compute the level-$i$ seeds: for each $j \in [2^i]$,
   $$S^{\mathsf{E}}_{i,j} := R^{\mathsf{E}}_{i,j} \oplus S^{\mathsf{E}}_{i-1,j}, \qquad S^{\mathsf{E}}_{i,2^i+j} := R^{\mathsf{E}}_{i,j}.$$
   (d) Set $\alpha_i := \alpha_{i-1} + a_i \cdot 2^i$.
4. Let $\mathsf{L}^{\mathsf{E}}_\ell := S^{\mathsf{E}}_{n-1,\ell}$ for $\ell \in [2^n]$. Using $\mathsf{H}$ and one-time nonces expand $\mathsf{L}^{\mathsf{E}}_\ell$ for $\ell \neq \alpha_{n-1}$ into seeds $X^{\mathsf{E}}_{\ell,k}$ of length $\kappa+\rho$ for $k \in [m]$.
5. Parse $[\boldsymbol{b}\Delta_{\mathsf{G}}]^{\mathsf{E}} = B_0 \oplus b_0\Delta_{\mathsf{G}}, \ldots, B_{m-1} \oplus b_{m-1}\Delta_{\mathsf{G}}$ and $[\boldsymbol{b}\Delta_{\mathsf{E}}]^{\mathsf{E}} = B'_0 \oplus b_0\Delta_{\mathsf{E}}, \ldots, B'_{m-1} \oplus b_{m-1}\Delta_{\mathsf{E}}$. For each $k \in [m]$, set
   $$X^{\mathsf{E}}_{\alpha_{n-1},k} := \left(\bigoplus_{\ell\neq\alpha_{n-1}} X^{\mathsf{E}}_{\ell,k}\right) \oplus G_k \oplus \left((B_k \oplus b_k\Delta_{\mathsf{G}}) \,\|\, (B'_k \oplus b_k\Delta_{\mathsf{E}})\right).$$
6. Set $\boldsymbol{X}_{\mathsf{E}} := (X^{\mathsf{E}}_{\ell,k})_{\ell\in[2^n],k\in[m]}$ and output
   $$\boldsymbol{Z}_{\mathsf{E}} \,\|\, \boldsymbol{Z}'_{\mathsf{E}} := \mathcal{T}(\mathrm{id})^{\intercal} \cdot \boldsymbol{X}_{\mathsf{E}}.$$

**Construction 2: Garbling and Evaluation Algorithms for $\Pi_{\text{2pc-tensor-1}}$ (and $\Pi_{\text{2pc-tensor-2}}$)**

*Legend.* Pieces in magenta are present only for $\Pi_{\text{2pc-tensor-2}}$; without them you have $\Pi_{\text{2pc-tensor-1}}$.

- $\mathsf{Garble}(1^\kappa, \mathcal{C}, [\boldsymbol{\lambda}\Delta_\mathsf{G}], [\boldsymbol{\lambda}\Delta_\mathsf{E}])$:

  1. For circuit input $\boldsymbol{x} \in \mathcal{I}_\mathsf{G}, \boldsymbol{y} \in \mathcal{I}_\mathsf{E}$, sample the garbler's share under $\Delta_\mathsf{G}$ uniformly at random $[(\boldsymbol{x}\|\boldsymbol{y})\Delta_\mathsf{G}] \leftarrow (\{0,1\}^\kappa)^{|\mathcal{I}|}$. Set shares under the evaluator's key as $[(\boldsymbol{x}\|\boldsymbol{y})\Delta_\mathsf{E}] := [(\boldsymbol{\lambda}_x\|\boldsymbol{\lambda}_y)\Delta_\mathsf{E}]$.
  2. Following the circuit's topological order, for each gate $G = (\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c}, T)$:
     - If $T = \oplus$, compute garbler's shares
     $$[\boldsymbol{c}\Delta_\mathsf{G}] := [\boldsymbol{a}\Delta_\mathsf{G}] \oplus [\boldsymbol{b}\Delta_\mathsf{G}], \qquad [\boldsymbol{c}\Delta_\mathsf{E}] := [\boldsymbol{a}\Delta_\mathsf{E}] \oplus [\boldsymbol{b}\Delta_\mathsf{E}].$$
     - If $T = \otimes$ with $n = |\boldsymbol{a}|, m = |\boldsymbol{b}|$, compute garbler's shares $[(\boldsymbol{a} \oplus \boldsymbol{\lambda}_a)\Delta_\mathsf{G}]$ and $[(\boldsymbol{b} \oplus \boldsymbol{\lambda}_b)\Delta_\mathsf{G}]$, and reveal masked values by appending $[\boldsymbol{a} \oplus \boldsymbol{\lambda}_a], [\boldsymbol{b} \oplus \boldsymbol{\lambda}_b]$ to the garbled circuit $\mathcal{GC}$. Generate ciphertexts and garbler's share of output via two calls to the half-tensor gate:
     $$\begin{aligned}
     (\boldsymbol{Z}_{c,0}, \boldsymbol{Z}'_{c,0}, \boldsymbol{G}_{c,0}) &:= \mathsf{Tensor.Gb}(n, m, \Delta_\mathsf{G}, [(\boldsymbol{a} \oplus \boldsymbol{\lambda}_a)\Delta_\mathsf{G}], [\boldsymbol{\lambda}_b\Delta_\mathsf{G}], [\boldsymbol{\lambda}_b\Delta_\mathsf{E}]),\\
     (\boldsymbol{Z}_{c,1}, \boldsymbol{Z}'_{c,1}, \boldsymbol{G}_{c,1}) &:= \mathsf{Tensor.Gb}(m, n, \Delta_\mathsf{G}, [(\boldsymbol{b} \oplus \boldsymbol{\lambda}_b)\Delta_\mathsf{G}], [\boldsymbol{a}\Delta_\mathsf{G}], [\boldsymbol{a}\Delta_\mathsf{E}]),\\
     [\boldsymbol{c}\Delta_\mathsf{G}] &:= \boldsymbol{Z}_{c,0} \oplus \boldsymbol{Z}^\intercal_{c,1} \oplus [(\boldsymbol{\lambda}_a \otimes \boldsymbol{\lambda}_b)\Delta_\mathsf{G}],\\
     [\boldsymbol{c}\Delta_\mathsf{E}] &:= \boldsymbol{Z}'_{c,0} \oplus (\boldsymbol{Z}'_{c,1})^\intercal \oplus [(\boldsymbol{\lambda}_a \otimes \boldsymbol{\lambda}_b)\Delta_\mathsf{E}].
     \end{aligned}$$
     Append $\boldsymbol{G}_{c,0}, \boldsymbol{G}_{c,1}$ to $\mathcal{GC}$.
  3. Output
     $$\mathcal{GC}, \quad [(\boldsymbol{x}\|\boldsymbol{y})\Delta_\mathsf{G}]$$
     and $\Delta_\mathsf{E}$-shares $[\boldsymbol{a}\Delta_\mathsf{E}], [\boldsymbol{b}\Delta_\mathsf{E}]$ of all tensor gate inputs and circuit output $[\boldsymbol{z}\Delta_\mathsf{E}]$.

- $\mathsf{Eval}(1^\kappa, \mathcal{C}, \mathcal{GC}, [(\boldsymbol{x}\|\boldsymbol{y})\Delta_\mathsf{G}]^\mathsf{E}, [(\boldsymbol{x}\|\boldsymbol{y})\Delta_\mathsf{E}]^\mathsf{E}, [\boldsymbol{\lambda}\Delta_\mathsf{G}]^\mathsf{E}, [\boldsymbol{\lambda}\Delta_\mathsf{E}]^\mathsf{E})$:

  1. Following the circuit's topological order, for each gate $G = (\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c}, T)$:
     - If $T = \oplus$, compute
     $$[\boldsymbol{c}\Delta_\mathsf{G}] := [\boldsymbol{a}\Delta_\mathsf{G}] \oplus [\boldsymbol{b}\Delta_\mathsf{G}], \qquad [\boldsymbol{c}\Delta_\mathsf{E}] := [\boldsymbol{a}\Delta_\mathsf{E}] \oplus [\boldsymbol{b}\Delta_\mathsf{E}].$$
     - If $T = \otimes$ with $n = |\boldsymbol{a}|, m = |\boldsymbol{b}|$, locally compute evaluator's shares $[(\boldsymbol{a} \oplus \boldsymbol{\lambda}_a)\Delta_\mathsf{G}]$ and $[(\boldsymbol{b} \oplus \boldsymbol{\lambda}_b)\Delta_\mathsf{G}]$, and reconstruct masked values $\boldsymbol{a} \oplus \boldsymbol{\lambda}_a, \boldsymbol{b} \oplus \boldsymbol{\lambda}_b$ by XORing with $\mathsf{G}$'s shares in $\mathcal{GC}$. Compute output shares via two calls to the half-tensor gate:
     $$\begin{aligned}
     (\boldsymbol{Z}_{c,0}, \boldsymbol{Z}'_{c,0}) &:= \mathsf{Tensor.Ev}(n, m, \boldsymbol{a} \oplus \boldsymbol{\lambda}_a, \boldsymbol{G}_{c,0}, [(\boldsymbol{a} \oplus \boldsymbol{\lambda}_a)\Delta_\mathsf{G}], [\boldsymbol{\lambda}_b\Delta_\mathsf{G}], [\boldsymbol{\lambda}_b\Delta_\mathsf{E}]),\\
     (\boldsymbol{Z}_{c,1}, \boldsymbol{Z}'_{c,1}) &:= \mathsf{Tensor.Ev}(m, n, \boldsymbol{b} \oplus \boldsymbol{\lambda}_b, \boldsymbol{G}_{c,1}, [(\boldsymbol{b} \oplus \boldsymbol{\lambda}_b)\Delta_\mathsf{G}], [\boldsymbol{a}\Delta_\mathsf{G}], [\boldsymbol{a}\Delta_\mathsf{E}]),\\
     [\boldsymbol{c}\Delta_\mathsf{G}] &:= \boldsymbol{Z}_{c,0} \oplus \boldsymbol{Z}^\intercal_{c,1} \oplus [(\boldsymbol{\lambda}_a \otimes \boldsymbol{\lambda}_b)\Delta_\mathsf{G}],\\
     [\boldsymbol{c}\Delta_\mathsf{E}] &:= \boldsymbol{Z}'_{c,0} \oplus (\boldsymbol{Z}'_{c,1})^\intercal \oplus [(\boldsymbol{\lambda}_a \otimes \boldsymbol{\lambda}_b)\Delta_\mathsf{E}].
     \end{aligned}$$
  2. Output masked values $\{\boldsymbol{a} \oplus \boldsymbol{\lambda}_a, \boldsymbol{b} \oplus \boldsymbol{\lambda}_b\}$ for all inputs to tensor gates, $\Delta_\mathsf{E}$-shares $[\boldsymbol{a}\Delta_\mathsf{E}], [\boldsymbol{b}\Delta_\mathsf{E}]$ of all tensor gate inputs, and $\Delta_\mathsf{E}$-shares of the circuit output $[\boldsymbol{z}\Delta_\mathsf{E}]$.

## 4.1 Half Tensor Gate

We modify [HK21b]'s privacy-free tensor gate construction to use the improved one-hot garbling construction of [Hea24], which requires sending only 1 ciphertext per level of the GGM tree, in line with [GYW+23]'s observation in the correlated OT and distributed point function setting. Construction 1 gives our formulation, where macros $\mathsf{Tensor.Gb}$ and $\mathsf{Tensor.Ev}$ formalize the two sides. The protocol takes as input XOR shares $[\boldsymbol{a}\Delta_{\mathsf{G}}]$ where $\boldsymbol{a} \in \{0,1\}^n$ is held by $\mathsf{E}$ in the clear, and XOR shares $[\boldsymbol{b}\Delta_{\mathsf{G}}]$ for $\boldsymbol{b} \in \{0,1\}^m$. The garbler sends $\boldsymbol{G}$ comprising $(n-1)+m$ ciphertexts of length $\kappa$, and parties compute shares $[(\boldsymbol{a}\otimes\boldsymbol{b})\Delta_{\mathsf{G}}]$. The *authenticated* variant $\mathsf{AuthTensor}$, used by $\Pi_{\mathsf{2pc\text{-}tensor\text{-}2}}$ in Section 5, additionally takes $[\boldsymbol{b}\Delta_{\mathsf{E}}]$ as input and produces $[(\boldsymbol{a}\otimes\boldsymbol{b})\Delta_{\mathsf{E}}]$ alongside $[(\boldsymbol{a}\otimes\boldsymbol{b})\Delta_{\mathsf{G}}]$, at an overhead of $m\rho$ bits.

**Intuition.** We build a one-hot encoding of $\boldsymbol{x} \in \{0,1\}^n$ inductively, one bit at a time. Suppose you hold a width-$2^j$ encoding $\boldsymbol{h}_j$ of the first $j$ bits of $\boldsymbol{x}$, one-hot at position $i \in [2^j]$. The width-$2^{j+1}$ encoding

$$\boldsymbol{h}_{j+1} = (\boldsymbol{h}_j \oplus x_j\boldsymbol{h}_j) \,\|\, x_j\boldsymbol{h}_j$$

is one-hot at position $x_j \cdot 2^j + i$, a valid one-hot encoding of the first $j+1$ bits of $\boldsymbol{x}$. The improved construction maintains the invariant that the parties hold $[\boldsymbol{h}_j\Delta_{\mathsf{G}}]$ at every level. The two halves of $\boldsymbol{h}_{j+1}$ are not independent: the left half equals the right half XORed with $\boldsymbol{h}_j$, which $\mathsf{E}$ already holds in garbled form from the previous level. This reduces the task to computing $x_j\boldsymbol{h}_j$ within a GC, which can be done at the cost of only one ciphertext. Parties locally compute the other half via FreeXOR. The original construction of [HK21b] instead sends two ciphertexts at each level without exploiting the dependence between the two halves of the one-hot vector.

This intuition maps onto Construction 1 as follows. Write $\boldsymbol{h}_i \in \{0,1\}^{2^i}$ for the one-hot encoding of the first $i$ bits of $\boldsymbol{a}$. The invariant is that the level-$i$ labels form shares of $\boldsymbol{h}_{i+1}\Delta_{\mathsf{G}}$: the evaluator holds the garbler's label at every position except the one-hot location, where its label is offset by $\Delta_{\mathsf{G}}$. At iteration $i$ of step 3, the hashes $R_{i,j}$ of step 3(a) give fresh shares of $a_i\boldsymbol{h}_i\Delta_{\mathsf{G}}$, up to a correction at the active position that the single ciphertext $G_i$ of step 3(c) supplies while folding in the bit $a_i$. Step 3(b) then writes both halves of the doubled encoding: the right half $S_{i,2^i+j} := R_{i,j}$ carries $a_i\boldsymbol{h}_i$, and the left half $S_{i,j} := R_{i,j} \oplus S_{i-1,j}$ carries $\boldsymbol{h}_i \oplus a_i\boldsymbol{h}_i$ by FreeXOR.

**Lemma 5** (Correctness of tensor macros)**.** *Let $[\boldsymbol{a}\Delta_{\mathsf{G}}]$ and $[\boldsymbol{b}\Delta_{\mathsf{G}}]$ be XOR shares of $\boldsymbol{a}\Delta_{\mathsf{G}}, \boldsymbol{b}\Delta_{\mathsf{G}}$ for $\boldsymbol{a} \in \{0,1\}^n, \boldsymbol{b} \in \{0,1\}^m$, with $\boldsymbol{a}$ additionally held by $\mathsf{E}$ in the clear, and let $[\boldsymbol{b}\Delta_{\mathsf{E}}]$ be an XOR share of $\boldsymbol{b}\Delta_{\mathsf{E}}$. Let*

$$\boldsymbol{Z}_{\mathsf{G}} \,\|\, \boldsymbol{Z}'_{\mathsf{G}}, \boldsymbol{G} := \mathsf{Tensor.Gb}(n, m, \Delta_{\mathsf{G}}, [\boldsymbol{a}\Delta_{\mathsf{G}}]^{\mathsf{G}}, [\boldsymbol{b}\Delta_{\mathsf{G}}]^{\mathsf{G}}, [\boldsymbol{b}\Delta_{\mathsf{E}}]^{\mathsf{G}}),$$
$$\boldsymbol{Z}_{\mathsf{E}} \,\|\, \boldsymbol{Z}'_{\mathsf{E}} := \mathsf{Tensor.Ev}(n, m, \boldsymbol{a}, \boldsymbol{G}, [\boldsymbol{a}\Delta_{\mathsf{G}}]^{\mathsf{E}}, [\boldsymbol{b}\Delta_{\mathsf{G}}]^{\mathsf{E}}, [\boldsymbol{b}\Delta_{\mathsf{E}}]^{\mathsf{E}}).$$

*Then $\boldsymbol{Z}_{\mathsf{G}} \oplus \boldsymbol{Z}_{\mathsf{E}} = (\boldsymbol{a}\otimes\boldsymbol{b})\Delta_{\mathsf{G}}$, and additionally $\boldsymbol{Z}'_{\mathsf{G}} \oplus \boldsymbol{Z}'_{\mathsf{E}} = (\boldsymbol{a}\otimes\boldsymbol{b})\Delta_{\mathsf{E}}$ in the authenticated variant.*

*Proof.* Deferred to Appendix B. □

Specializing to $n = m = 1$ recovers the privacy-free half-gate protocol of [ZRE15] with one ciphertext communication. The communication overhead of the authenticated variant over the unauthenticated one is $m \cdot \rho$ bits per call, due to widening the $m$ leaf ciphertexts $\{G_k\}$ from $\kappa$ to $\kappa + \rho$ bits; the $n-1$ tree-level ciphertexts remain $\kappa$ bits long. Since the $\Delta_{\mathsf{E}}$-shares are never used as inputs to a hash function, we only require that they are difficult to guess, so $\rho < \kappa$ suffices.

## 4.2 Garbling and evaluation algorithms

For readability of our 2PC protocols, we separate out the garbling and evaluation algorithms and present them in a single merged box covering both $\Pi_{\mathsf{2pc\text{-}tensor\text{-}1}}$ and $\Pi_{\mathsf{2pc\text{-}tensor\text{-}2}}$ (Section 5). At a high level, these algorithms take in the required preprocessing material and proceed gate-by-gate through the circuit, either locally XORing shares or invoking the tensor gate subroutine on input shares to obtain output shares and additionally generate/consume ciphertexts. $\Pi_{\mathsf{2pc\text{-}tensor\text{-}2}}$ additionally propagates $\Delta_{\mathsf{E}}$-shares of every wire alongside the $\Delta_{\mathsf{G}}$-shares, which is achieved by enabling the highlighted additions throughout. In Construction 2, the garbling algorithm is run by $\mathsf{G}$ and evaluation by $\mathsf{E}$. To declutter notation we omit the party superscripts, as each algorithm references the running party's own shares.

**Construction 3: Protocol $\Pi_{\text{2pc-tensor-1}}$ (and $\Pi_{\text{2pc-tensor-2}}$)**

*Legend.* Pieces in magenta appear only in $\Pi_{\text{2pc-tensor-2}}$; pieces in teal appear only in $\Pi_{\text{2pc-tensor-1}}$.

Parties agree on a circuit $\mathcal{C}$ to evaluate. $\mathsf{G}$ holds $\boldsymbol{x} \in \{0,1\}^{|\mathcal{I}_\mathsf{G}|}$, $\mathsf{E}$ holds $\boldsymbol{y} \in \{0,1\}^{|\mathcal{I}_\mathsf{E}|}$, and $\mathsf{E}$ receives output $\boldsymbol{z} = \mathcal{C}(\boldsymbol{x}, \boldsymbol{y})$.

*Preprocess:*

1. Parties call $\mathcal{F}_{\mathsf{Pre}}$ (or $\mathcal{F}_{\mathsf{cpre}}$ for $\Pi_{\text{2pc-tensor-2}}$) to obtain global keys $\Delta_\mathsf{G}$, $\Delta_\mathsf{E}$, dual-authenticated shares of random bits $[\![\lambda]\!]$ for each input to the circuit and random tensor triples $([\![\boldsymbol{\lambda}_a]\!], [\![\boldsymbol{\lambda}_b]\!], [\![\boldsymbol{\lambda}_a \otimes \boldsymbol{\lambda}_b]\!])$ for each tensor gate.

*Garble:*

2. $\mathsf{G}$ runs $(\mathcal{GC}, [(\boldsymbol{x}\|\boldsymbol{y})\Delta_\mathsf{G}]^\mathsf{G}, [(\boldsymbol{x}\|\boldsymbol{y})\Delta_\mathsf{E}]^\mathsf{G}) \leftarrow \mathsf{Garble}(1^\kappa, \mathcal{C}, [\boldsymbol{\lambda}\Delta_\mathsf{G}]^\mathsf{G}, [\boldsymbol{\lambda}\Delta_\mathsf{E}]^\mathsf{G})$ and sends $\mathcal{GC}$ to $\mathsf{E}$.

*Encode inputs:*

3. For $\mathsf{G}$'s input $\boldsymbol{x}$, $\mathsf{G}$ sends $\mathsf{E}$ its share of the input under its own global key $[\boldsymbol{x}\Delta_\mathsf{G}]^\mathsf{E} := [\boldsymbol{x}\Delta_\mathsf{G}]^\mathsf{G} \oplus \boldsymbol{x}\Delta_\mathsf{G}$. Then parties run $\mathsf{Open}([\boldsymbol{\lambda}_x\Delta_\mathsf{G}])$ to reveal $\boldsymbol{\lambda}_x$ to $\mathsf{G}$, who sends a masked input $\boldsymbol{x} \oplus \boldsymbol{\lambda}_x$ to $\mathsf{E}$. $\mathsf{G}$ sets $[\boldsymbol{x}\Delta_\mathsf{E}]^\mathsf{G} = [\boldsymbol{\lambda}_x\Delta_\mathsf{E}]^\mathsf{G}$, $\mathsf{E}$ sets $[\boldsymbol{x}\Delta_\mathsf{E}]^\mathsf{E} = [\boldsymbol{\lambda}_x\Delta_\mathsf{E}]^\mathsf{E} \oplus (\boldsymbol{x} \oplus \boldsymbol{\lambda}_x)\Delta_\mathsf{E}$.
4. For $\mathsf{E}$'s input $\boldsymbol{y}$, parties run $\mathsf{Open}([\boldsymbol{\lambda}_y\Delta_\mathsf{E}])$ to reveal $\boldsymbol{\lambda}_y$ to $\mathsf{E}$. $\mathsf{G}$ sets $[\boldsymbol{y}\Delta_\mathsf{E}]^\mathsf{G} = [\boldsymbol{\lambda}_y\Delta_\mathsf{E}]^\mathsf{G}$, $\mathsf{E}$ sets $[\boldsymbol{y}\Delta_\mathsf{E}]^\mathsf{E} = [\boldsymbol{\lambda}_y\Delta_\mathsf{E}]^\mathsf{E} \oplus (\boldsymbol{y} \oplus \boldsymbol{\lambda}_y)\Delta_\mathsf{E}$. Then $\mathsf{E}$ sends masked input $\boldsymbol{y} \oplus \boldsymbol{\lambda}_y$ to $\mathsf{G}$ who sends a correction term $W = [\boldsymbol{y}\Delta_\mathsf{G}]^\mathsf{G} \oplus [\boldsymbol{\lambda}_y\Delta_\mathsf{G}]^\mathsf{G} \oplus (\boldsymbol{y} \oplus \boldsymbol{\lambda}_y)\Delta_\mathsf{G}$, letting $\mathsf{E}$ set $[\boldsymbol{y}\Delta_\mathsf{G}]^\mathsf{E} = [\boldsymbol{\lambda}_y\Delta_\mathsf{G}]^\mathsf{E} \oplus W$.

*Evaluate:*

5. $\mathsf{E}$ runs $\mathsf{Eval}(1^\kappa, \mathcal{C}, \mathcal{GC}, [(\boldsymbol{x}\|\boldsymbol{y})\Delta_\mathsf{G}]^\mathsf{E}, [(\boldsymbol{x}\|\boldsymbol{y})\Delta_\mathsf{E}]^\mathsf{E}, [\boldsymbol{\lambda}\Delta_\mathsf{G}]^\mathsf{E}, [\boldsymbol{\lambda}\Delta_\mathsf{E}]^\mathsf{E})$ and outputs revealed masked values $\{\boldsymbol{a} \oplus \boldsymbol{\lambda}_a, \boldsymbol{b} \oplus \boldsymbol{\lambda}_b\}$, $\Delta_\mathsf{E}$-shares $[\boldsymbol{a}\Delta_\mathsf{E}], [\boldsymbol{b}\Delta_\mathsf{E}]$ of all tensor gate inputs, and $\Delta_\mathsf{E}$-shares of the circuit output $[\boldsymbol{z}\Delta_\mathsf{E}]$.

*Consistency check:* In $\Pi_{\text{2pc-tensor-1}}$, $\mathsf{E}$ does not yet hold $\Delta_\mathsf{E}$-shares of intermediate wires; the first two steps below recover them via $\mathsf{G}$. $\Pi_{\text{2pc-tensor-2}}$ skips these (the $\mathsf{Eval}$ call already produced the $\Delta_\mathsf{E}$-shares).

6. $\mathsf{E}$ sends all revealed masked values $\boldsymbol{a} \oplus \boldsymbol{\lambda}_a, \boldsymbol{b} \oplus \boldsymbol{\lambda}_b$ to $\mathsf{G}$.
7. From input encoding, parties hold shares $[(\boldsymbol{x}\|\boldsymbol{y})\Delta_\mathsf{E}]$ of circuit inputs. Following the circuit's topological order, for each gate $G = (\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c}, T)$:
   - If $T = \oplus$, compute $[\boldsymbol{c}\Delta_\mathsf{E}] := [\boldsymbol{a}\Delta_\mathsf{E}] \oplus [\boldsymbol{b}\Delta_\mathsf{E}]$ locally.
   - If $T = \otimes$ with $n = |\boldsymbol{a}|, m = |\boldsymbol{b}|$, compute
   $$[\boldsymbol{c}\Delta_\mathsf{E}] := \big((\boldsymbol{a} \oplus \boldsymbol{\lambda}_a) \otimes [\boldsymbol{b}\Delta_\mathsf{E}]\big) \oplus \big((\boldsymbol{b} \oplus \boldsymbol{\lambda}_b) \otimes [\boldsymbol{\lambda}_a\Delta_\mathsf{E}]\big)^\intercal \oplus [(\boldsymbol{\lambda}_a \otimes \boldsymbol{\lambda}_b)\Delta_\mathsf{E}]$$
   locally.
8. For each $(\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c}, \otimes) \in \mathcal{C}$, parties locally compute
$$[\boldsymbol{e}_a\Delta_\mathsf{E}] := [\boldsymbol{a}\Delta_\mathsf{E}] \oplus [\boldsymbol{\lambda}_a\Delta_\mathsf{E}] \oplus (\boldsymbol{a} \oplus \boldsymbol{\lambda}_a)\Delta_\mathsf{E},$$
$$[\boldsymbol{e}_b\Delta_\mathsf{E}] := [\boldsymbol{b}\Delta_\mathsf{E}] \oplus [\boldsymbol{\lambda}_b\Delta_\mathsf{E}] \oplus (\boldsymbol{b} \oplus \boldsymbol{\lambda}_b)\Delta_\mathsf{E}.$$
9. Parties run $\mathsf{CheckZero}(\{[\boldsymbol{e}_a\Delta_\mathsf{E}], [\boldsymbol{e}_b\Delta_\mathsf{E}]\}_{(\boldsymbol{a},\boldsymbol{b},\boldsymbol{c})\in\mathcal{W}})$, and $\mathsf{E}$ aborts if the check fails, i.e. any $\boldsymbol{e}_a, \boldsymbol{e}_b \neq \boldsymbol{0}$.

*Decode outputs:*

10. Parties run $\mathsf{Open}([\boldsymbol{z}\Delta_\mathsf{E}])$ revealing the circuit output $\boldsymbol{z}$ to $\mathsf{E}$.

## 4.3 The full 2PC protocol

We now present the full 2PC protocol $\Pi_{\text{2pc-tensor-1}}$, optimized to minimize the garbled circuit size. Construction 3 also presents the compressed-preprocessing variant $\Pi_{\text{2pc-tensor-2}}$ (Section 5), and input encoding and output decoding are done in a standard manner, similar to prior works.

## 4.4 Communication complexity

Excluding preprocessing and input/output processing (whose size is independent of circuit size up to constants), $\Pi_{\text{2pc-tensor-1}}$ has online communication

$$\sum_{(\boldsymbol{a},\boldsymbol{b},\boldsymbol{c},\otimes)\in\mathcal{C}} 2(n+m-1)\kappa + 2(n+m) \quad \text{bits},$$

where $n = |\boldsymbol{a}|$ and $m = |\boldsymbol{b}|$ per gate. The $2(n+m-1)\kappa$ term is from two $\mathsf{Tensor.Gb}$ invocations per tensor gate inside $\mathcal{GC}$; the $2(n+m)$ term is from $\mathsf{G}$ revealing masked tensor gate inputs to $\mathsf{E}$ as part of the garbled circuit and $\mathsf{E}$ revealing these masked values to $\mathsf{G}$ for the consistency check and output decoding.

In Appendix C we show that the preprocessing cost per random tensor triple $(\boldsymbol{\lambda}_a, \boldsymbol{\lambda}_b, \boldsymbol{\lambda}_a \otimes \boldsymbol{\lambda}_b)$ is

$$B \cdot \big[2(n+m-1)\kappa + 2nm\big] + (B-1)m \quad \text{bits},$$

in the $(\mathcal{F}_{\mathrm{COT}}, \mathcal{F}_{\mathsf{EQ}})$-hybrid model, where $B = \rho/\log(n|\mathcal{C}|)$ is the bucketing parameter. Instantiating $\mathcal{F}_{\mathrm{COT}}$ with SoftSpokenOT [Roy22] gives us total communication $O(B \cdot \kappa \log \kappa)$ for generating tensor triples of length $\log \kappa$, assuming only base OTs and the existence of a circular correlation robust hash function. This is the bottleneck of $\Pi_{\mathsf{2pc\text{-}tensor\text{-}1}}$'s total communication, and Section 5 presents a second protocol that avoids it.

### 4.5 Security

**Theorem 6.** *$\Pi_{\mathsf{2pc\text{-}tensor\text{-}1}}$ realizes $\mathcal{F}_{\mathsf{2pc}}$ against a static malicious adversary in the $(\mathcal{F}_{\mathsf{Pre}}, \mathcal{F}_{\mathsf{EQ}})$-hybrid model, assuming the existence of a circular correlation robust hash function.*

The key step in the analysis is showing that a malicious garbler gains nothing by corrupting ciphertexts:

**Lemma 7** (Selective-failure resistance)**.** *For any malicious $\mathsf{G}$ strategy and any $\boldsymbol{y}, \boldsymbol{y}' \in \{0,1\}^{|\mathcal{I}_{\mathsf{E}}|}$,*

$$\Pr[\mathsf{Abort}_{\boldsymbol{y}}] = \Pr[\mathsf{Abort}_{\boldsymbol{y}'}],$$

*where $\mathsf{Abort}_{\boldsymbol{y}}$ is the event that honest $\mathsf{E}$ aborts in $\Pi_{\mathsf{2pc\text{-}tensor\text{-}1}}$ on input $\boldsymbol{y}$.*

*Proof Sketch.* Correctness against a malicious $\mathsf{G}$ reduces to the consistency check, since any deviation in a revealed masked value leaves a nonzero residual $\boldsymbol{\delta}\Delta_{\mathsf{E}}$ that passes $\mathsf{CheckZero}$ only by guessing $\Delta_{\mathsf{E}}$, and privacy against $\mathsf{G}$ reduces to Lemma 7, which follows from the leaky-tensor analysis of Appendix C. Security against a malicious $\mathsf{E}$ uses the semi-honest obliviousness simulator, replacing active-path ciphertexts with uniform strings by circular correlation robustness of $\mathsf{H}$. The full proof appears in Appendix B. □

## 5 Protocol 2: Optimizing total communication

Next, we present our second authenticated garbling protocol, following the blueprint of [WRK17, DILO22, CWYY23] and realizing the ideal 2PC functionality $\mathcal{F}_{\mathsf{2pc}}$ in the $(\mathcal{F}_{\mathrm{cpre}}, \mathcal{F}_{\mathsf{EQ}})$-hybrid model. $\Pi_{\mathsf{2pc\text{-}tensor\text{-}1}}$'s preprocessing dominates its total communication, so to improve on it we use compressed preprocessing in which the correlations are derived from a short authenticated seed expanded locally by a public matrix. As discussed in Section 3.4, such correlations are incompatible with $\Pi_{\mathsf{2pc\text{-}tensor\text{-}1}}$'s strategy of revealing masked values to $\mathsf{G}$.

Recall from Section 3.5 the high-level structure of our approach, using the garbled circuit to propagate $\Delta_{\mathsf{E}}$-shares as well as $\Delta_{\mathsf{G}}$-shares. The compressed preprocessing functionality $\mathcal{F}_{\mathrm{cpre}}$ (Appendix D) outputs the same correlations as $\mathcal{F}_{\mathsf{Pre}}$ from the previous section, but the evaluator's shares of masks have lower entropy. Instantiating $\mathcal{F}_{\mathrm{cpre}}$ with our desired asymptotics requires additionally assuming LPN.

With the required preprocessing, the garbler constructs an augmented garbled circuit that, alongside the usual $\Delta_{\mathsf{G}}$-shares, propagates $\Delta_{\mathsf{E}}$-shares $[w\Delta_{\mathsf{E}}]$ of each wire value through tensor gates, and the evaluator decrypts both kinds of share from the garbled material. The masked values revealed to the evaluator are never sent to the garbler, and once $\Delta_{\mathsf{E}}$-shares are obtained, the consistency check and output decoding follow the same shared steps as $\Pi_{\mathsf{2pc\text{-}tensor\text{-}1}}$.

We instantiate the half tensor gate using the authenticated variant of the tensor macros from Construction 1 (the magenta-highlighted additions): the macros take an extra input $[\boldsymbol{b}\Delta_{\mathsf{E}}]$ and additionally produce $[(\boldsymbol{a} \otimes \boldsymbol{b})\Delta_{\mathsf{E}}]$ alongside $[(\boldsymbol{a} \otimes \boldsymbol{b})\Delta_{\mathsf{G}}]$. The communication cost is $(n+m-1)\kappa + m \cdot \rho$ bits per call, an overhead of $m \cdot \rho$ bits over the unauthenticated variant.

The garbling and evaluation algorithms for $\Pi_{\mathsf{2pc\text{-}tensor\text{-}2}}$ are obtained by enabling the magenta-highlighted additions in Construction 2: an extra $\Delta_{\mathsf{E}}$-share is propagated alongside each $\Delta_{\mathsf{G}}$-share through XOR and tensor gates.

$\Pi_{\mathsf{2pc\text{-}tensor\text{-}2}}$ is presented as the magenta-highlighted variant of Construction 3: $\mathcal{F}_{\mathsf{Pre}}$ is replaced by $\mathcal{F}_{\mathrm{cpre}}$, the garble/eval calls take additional $\Delta_{\mathsf{E}}$-share arguments, and the teal reveal-and-propagate steps of $\Pi_{\mathsf{2pc\text{-}tensor\text{-}1}}$'s consistency check are skipped since the $\mathsf{Eval}$ call already produces $\Delta_{\mathsf{E}}$-shares.

### 5.1 Communication complexity

Excluding preprocessing and input/output processing, $\Pi_{\mathsf{2pc\text{-}tensor\text{-}2}}$ has online communication

$$\sum_{(\boldsymbol{a},\boldsymbol{b},\boldsymbol{c},\otimes)\in\mathcal{C}} 2(n+m-1)\kappa + (n+m)\rho + (n+m) \quad \text{bits},$$

where $n = |\boldsymbol{a}|$ and $m = |\boldsymbol{b}|$ per gate, with the terms arising as in Section 4 plus the $(n+m)\rho$ widening of the leaf ciphertexts.

Preprocessing is handled by $\mathcal{F}_{\mathrm{cpre}}$, where using [CWYY23]'s compressed preprocessing protocol, the amortized cost of generating tensor triples is $4nm$ bits. Total communication is much less than $\Pi_{\mathsf{2pc\text{-}tensor\text{-}1}}$.

### 5.2 Security

**Theorem 8.** *$\Pi_{\mathsf{2pc\text{-}tensor\text{-}2}}$ realizes $\mathcal{F}_{\mathsf{2pc}}$ against a static malicious adversary in the $(\mathcal{F}_{\mathrm{cpre}}, \mathcal{F}_{\mathsf{EQ}})$-hybrid model.*

**Lemma 9** (Selective-failure resistance)**.** *For any malicious $\mathsf{G}$ strategy and any $\boldsymbol{y}, \boldsymbol{y}' \in \{0,1\}^{|\mathcal{I}_{\mathsf{E}}|}$,*

$$\left|\Pr[\mathsf{Abort}_{\boldsymbol{y}}] - \Pr[\mathsf{Abort}_{\boldsymbol{y}'}]\right| \le 2^{-\rho},$$

*where $\mathsf{Abort}_{\boldsymbol{y}}$ is the event that honest $\mathsf{E}$ aborts in $\Pi_{\mathsf{2pc\text{-}tensor\text{-}2}}$ on input $\boldsymbol{y}$.*

*Proof Sketch.* The argument follows Theorem 6, except that Lemma 9 argues from $\rho$-wise independence of the expanded masks (Lemma 4). The full proof appears in Appendix B. □

## 6 Applications

We now describe how tensor gates improve concrete workloads, following the semi-honest treatment of [HK21b]: we use our short input tensor gates to compute arbitrary length tensor products, and apply that to matrix and integer multiplication. Throughout, improvements are stated against the corresponding AND-gate-based authenticated garbling.

**Generalized Outer Product.** Tensor gates in our protocols are limited to one input (the one revealed to the evaluator) having length $O(\log \kappa)$ due to its exponential scaling in computation, so to support longer vectors we use a rectangular tiling approach. Partition $\boldsymbol{x} \oplus \boldsymbol{\lambda}_x$ into $N := \lceil n/t \rceil$ chunks of length $t = \log \kappa$ and compute $(\boldsymbol{x} \oplus \boldsymbol{\lambda}_x) \otimes \boldsymbol{\lambda}_y$ via $N$ tensor gate calls of size $t \times m$; symmetrically, $\boldsymbol{x} \otimes (\boldsymbol{y} \oplus \boldsymbol{\lambda}_y)$ uses $M := \lceil m/t \rceil$ calls of size $n \times t$.

A half tensor gate call of size $x \times y$ with $\min(x, y) \le t$ has communication $(x + y - 1)$ ciphertexts, so the total communication cost is

$$N(t+m-1) + M(n+t-1) \;=\; \frac{2nm + (n+m)(t-1)}{t} \text{ ciphertexts},$$

dominated by $2nm/\log \kappa$. The saving factor over the $2nm$-ciphertext naive AND-gate baseline is $2nmt/(2nm + (n+m)(t-1))$, which is monotone in $\min(n, m)$: it equals $t^2/(2t-1) \approx t/2$ at $n = m = t$ and approaches $t$ as $\min(n, m) \to \infty$, so the improvement is better for longer tensor products. The number of hashes computed for a single short tensor gate call of size $t \times N$ is $2^t \cdot N = O(N\kappa)$, so the $N + M$ calls compute a total of $O((N+M)m\kappa + (N+M)n\kappa) = O(nm\kappa/\log \kappa)$ hashes compared to $O(nm)$ hashes for the naive approach. The total computation therefore increases by a factor of $\kappa/\log \kappa$. However, since communication is typically the bottleneck in GC protocols, this leads to improvement in run time (see Section 7). The (optimal) tiling size of 6 shows up as a 3–6× improvement over prior state of the art, based on the length of the tensor product being computed, which is consistent with the above analysis.

We remark on the comparison methodology used in Section 7. Our baselines [KRRW18, CWYY23] are instantiated by setting the tile size to $t = 1$. At $t = 1$ a tensor gate is a single AND gate, the per-gate cost $2(n+m-1)\kappa + 2(n+m)$ of Section 4 collapses to the $2\kappa$ bits of half gates plus lower-order terms, and the cryptographic operations performed are identical to those of the corresponding protocol. The tiled construction therefore improves over the AND-gate baseline by the factor derived above, between $t/2$ and $t$ depending on the dimensions of the computed tensor product. Figure 1 plots this improvement against the tensor input length $N$.

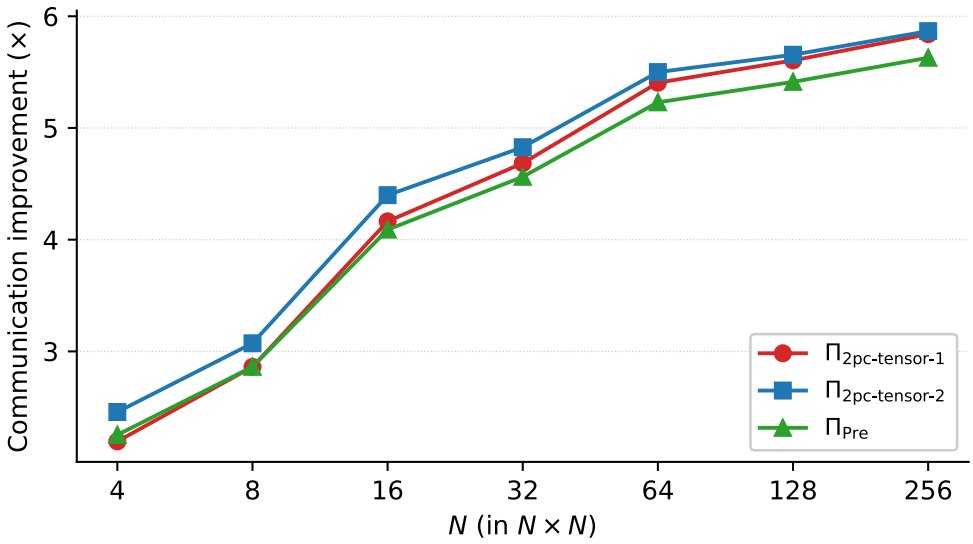



Figure 1: Communication improvement of our protocols ($\Pi_{\text{2pc-tensor-1}}$, $\Pi_{\text{2pc-tensor-2}}$, and $\Pi_{\text{Pre}}$) over baseline with varying tensor input length $N$. All three protocols approach an improvement proportional to the tiling size as $N$ grows, consistent with the analysis above.

**Binary Matrix Multiplication.** Given binary input matrices $\boldsymbol{X} \in \mathbb{F}_2^{n \times m}$ and $\boldsymbol{Y} \in \mathbb{F}_2^{m \times \ell}$, the product $\boldsymbol{C} = \boldsymbol{X} \cdot \boldsymbol{Y}$ can be expressed as a sum of $m$ outer products: for each $i \in [m]$, compute the outer product of column $i$ of $\boldsymbol{X}$ and row $i$ of $\boldsymbol{Y}$, and XOR the resulting matrices together.

Our protocol for generalized tensor products, like that of [HK21b], can be applied directly as a black box to each pair of vectors. The reduction in cost per tensor product (via the tiling approach) reduces the overall communication from $O(nm\ell)$ to $O(nm\ell/\log\kappa)$ ciphertexts.

**Integer Multiplication.** Tensor gates also enable more efficient garbling of integer schoolbook multiplication. In the schoolbook method, the product $\boldsymbol{x} \cdot \boldsymbol{y}$ is computed as a sum of terms of the form $x_i \cdot \boldsymbol{y} \ll i$, where $i \in [n]$ and $\boldsymbol{y} \ll i$ denotes $\boldsymbol{y}$ left-shifted by $i$ bits. This corresponds to computing all pairwise products $x_i \cdot y_j$ for $i, j \in [n]$, which are precisely the entries of the binary tensor product $\boldsymbol{x} \otimes \boldsymbol{y}$. Using the tiling technique and tensor gates, these pairwise products can be computed efficiently in an authenticated garbled circuit. While the computation of $\boldsymbol{x} \otimes \boldsymbol{y}$ is improved, the summation of the resulting partial products is still performed inside GC using traditional methods, and becomes the new bottleneck of performance. This routine can also serve as a base case in recursive schemes like Karatsuba multiplication.

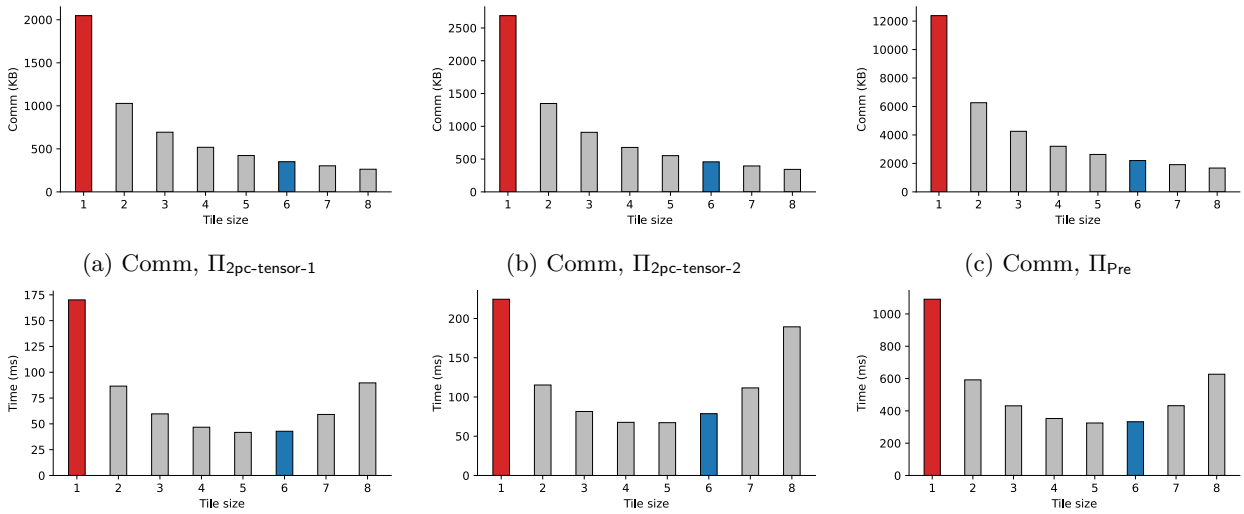
# 7 Experiments

In this section, we benchmark the online phases of $\Pi_{\text{2pc-tensor-1}}$ and $\Pi_{\text{2pc-tensor-2}}$ on boolean tensor products, and preprocessing protocol $\Pi_{\text{Pre}}$ (Appendix C) that generates the correlations consumed by $\Pi_{\text{2pc-tensor-1}}$. Our two protocols generalize [KRRW18] and [CWYY23] respectively, and in fact, when handling arbitrary length tensor products, if we set the *tile size* (Section 6) to 1, we recover these as baselines and compare against them while presenting our results. For a $256 \times 256$ bit tensor product with $\kappa = 127$ and $\rho = 40$, and the optimal tile size of 6 (Figure 2 and Figure 3), we observe a 5.84× improvement in communication and 3.97× improvement in wall clock time for $\Pi_{\text{2pc-tensor-1}}$ over [KRRW18]'s transmitted garbled circuit, 5.87/2.85× for $\Pi_{\text{2pc-tensor-2}}$ over [CWYY23]'s transmitted garbled circuit, and 5.63/3.28× over [KRRW18]'s preprocessing protocol. Our online benchmarks represent a round of garbling, evaluation and consistency checking, omitting input encoding and output decoding, and our preprocessing benchmarks cover leaky triple generation and bucketing.

**Setup.** We implemented the online phases of $\Pi_{\mathsf{2pc\text{-}tensor\text{-}1}}$ and $\Pi_{\mathsf{2pc\text{-}tensor\text{-}2}}$ in Rust, alongside our preprocessing protocol $\Pi_{\mathsf{Pre}}$ that instantiates $\mathcal{F}_{\mathsf{Pre}}$ used in $\Pi_{\mathsf{2pc\text{-}tensor\text{-}1}}$. To simulate the online phase of $\Pi_{\mathsf{2pc\text{-}tensor\text{-}2}}$, we implemented an ideal compressed preprocessing functionality. We use [CWYY23] to instantiate $\mathcal{F}_{\mathrm{cpre}}$ and consider implementing it beyond our scope. All experiments ran on a MacBook Pro laptop with an M4 Pro 2.6 GHz processor and 24 GB of RAM, using the Criterion crate [AH25] to run benchmarks. Communication was measured as the size of the garbled circuits transmitted between parties. We simulated a 100 Mb per second network; no jitter or delay was modeled as our protocol is constant round and latency has no measurable impact on wall clock time. We report time in milliseconds and communication in kilobytes.

**Results.** Figure 2 reports both metrics across tile sizes 1 through 8. As expected, communication decreases monotonically with tile size, while wall clock time follows a U-shape with optimum at tile size 6 (in blue).

The near-uniform $\approx 6\times$ communication factor across all three protocols matches the expected improvement between $\log \kappa/2$ and $\log \kappa$ derived in Section 6: the tiling reuses each one-hot encoding across a rectangle, so for large tensor products the saving approaches $t$ rather than $t/2$.




(a) Comm, $\Pi_{\text{2pc-tensor-1}}$ (b) Comm, $\Pi_{\text{2pc-tensor-2}}$ (c) Comm, $\Pi_{\text{Pre}}$

(d) Time, $\Pi_{\text{2pc-tensor-1}}$ (e) Time, $\Pi_{\text{2pc-tensor-2}}$ (f) Time, $\Pi_{\text{Pre}}$

Figure 2: Wall clock time (ms) and communication (KB) at tile sizes 1 through 8 for a $256 \times 256$-bit input. Baseline (tile size 1) in red, optimal (tile size 6) in blue.

| | | Comm. (KB) | Time (ms) |
|---|---|---|---|
| $\Pi_{\text{2pc-tensor-1}}$ | Baseline | 2,048.1 | 170.1 |
| | This work | 350.7 | 42.8 |
| | *Improvement* | 5.84× | 3.97× |
| $\Pi_{\text{2pc-tensor-2}}$ | Baseline | 2,688.0 | 224.5 |
| | This work | 458.2 | 78.7 |
| | *Improvement* | 5.87× | 2.85× |
| $\Pi_{\text{Pre}}$ | Baseline | 12,384.2 | 1,091.2 |
| | This work | 2,200.1 | 332.7 |
| | *Improvement* | 5.63× | 3.28× |

Figure 3: Baseline (tile size 1) vs. this work (optimal tile size 6) comparison for $\Pi_{\text{2pc-tensor-1}}$, $\Pi_{\text{2pc-tensor-2}}$, and $\Pi_{\text{Pre}}$. All three comparisons show communication improvement almost as much as the tile size, with a corresponding wall-clock speedup.

## Acknowledgements

This research was supported in part by NSF grants OAC-2531010, CNS-2246355 and CNS-2246353, US-Israel BSF grant 2022370, Sunday Group, and an Amazon Research Award (AWS Cryptography 2025).

# References

[AH25] Jorge Aparicio and Brook Heisler. criterion.rs: Statistics-driven micro-benchmarking library, 2014–2025.

[BCG+19a] Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, Lisa Kohl, Peter Rindal, and Peter Scholl. Efficient two-round OT extension and silent non-interactive secure computation. In Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz, editors, *ACM CCS 2019*, pages 291–308. ACM Press, November 2019.

[BCG+19b] Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, Lisa Kohl, and Peter Scholl. Efficient pseudorandom correlation generators: Silent OT extension and more. In Alexandra Boldyreva and Daniele Micciancio, editors, *CRYPTO 2019, Part III*, volume 11694 of *LNCS*, pages 489–518. Springer, Cham, August 2019.

[BDOZ11] Rikke Bendlin, Ivan Damgård, Claudio Orlandi, and Sarah Zakarias. Semi-homomorphic encryption and multiparty computation. In Kenneth G. Paterson, editor, *EUROCRYPT 2011*, volume 6632 of *LNCS*, pages 169–188. Springer, Berlin, Heidelberg, May 2011.

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

[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*, volume 7194 of *LNCS*, pages 39–53. Springer, Berlin, Heidelberg, March 2012.

[CWYY23] Hongrui Cui, Xiao Wang, Kang Yang, and Yu Yu. Actively secure half-gates with minimum overhead under duplex networks. In Carmit Hazay and Martijn Stam, editors, *EUROCRYPT 2023, Part II*, volume 14005 of *LNCS*, pages 35–67. Springer, Cham, April 2023.

[DILO22] Samuel Dittmer, Yuval Ishai, Steve Lu, and Rafail Ostrovsky. Authenticated garbling from simple correlations. In Yevgeniy Dodis and Thomas Shrimpton, editors, *CRYPTO 2022, Part IV*, volume 13510 of *LNCS*, pages 57–87. Springer, Cham, August 2022.

[DK99] Yevgeniy Dodis and Sanjeev Khanna. Space-time tradeoffs for graph properties. In Jirí Wiedermann, Peter van Emde Boas, and Mogens Nielsen, editors, *Automata, Languages and Programming*, pages 291–300, Berlin, Heidelberg, 1999. Springer Berlin Heidelberg.

[DNNR17] Ivan Damgård, Jesper Buus Nielsen, Michael Nielsen, and Samuel Ranellucci. The TinyTable protocol for 2-party secure computation, or: Gate-scrambling revisited. In Jonathan Katz and Hovav Shacham, editors, *CRYPTO 2017, Part I*, volume 10401 of *LNCS*, pages 167–187. Springer, Cham, August 2017.

[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. IEEE Computer Society Press, May 2020.

[GTRS25] Tianyao Gu, Afonso Tinoco, Sri Harish G. Rajan, and Elaine Shi. PicoGRAM: Practical garbled RAM from decisional Diffie-Hellman. In Yael Tauman Kalai and Seny F. Kamara, editors, *CRYPTO 2025, Part VIII*, volume 16007 of *LNCS*, pages 247–281. Springer, Cham, August 2025.

[GYW+23] Xiaojie Guo, Kang Yang, Xiao Wang, Wenhao Zhang, Xiang Xie, Jiang Zhang, and Zheli Liu. Half-tree: Halving the cost of tree expansion in COT and DPF. In Carmit Hazay and Martijn Stam, editors, *EUROCRYPT 2023, Part I*, volume 14004 of *LNCS*, pages 330–362. Springer, Cham, April 2023.

[Hea24] David Heath. Efficient arithmetic in garbled circuits. In Marc Joye and Gregor Leander, editors, *EUROCRYPT 2024, Part V*, volume 14655 of *LNCS*, pages 3–31. Springer, Cham, May 2024.

[HK20] David Heath and Vladimir Kolesnikov. Stacked garbling - garbled circuit proportional to longest execution path. In Daniele Micciancio and Thomas Ristenpart, editors, *CRYPTO 2020, Part II*, volume 12171 of *LNCS*, pages 763–792. Springer, Cham, August 2020.

[HK21a] David Heath and Vladimir Kolesnikov. LogStack: Stacked garbling with $O(b \log b)$ computation. In Anne Canteaut and François-Xavier Standaert, editors, *EUROCRYPT 2021, Part III*, volume 12698 of *LNCS*, pages 3–32. Springer, Cham, October 2021.

[HK21b] David Heath and Vladimir Kolesnikov. One hot garbling. In Giovanni Vigna and Elaine Shi, editors, *ACM CCS 2021*, pages 574–593. ACM Press, November 2021.

[HKN24] David Heath, Vladimir Kolesnikov, and Lucien K. L. Ng. Garbled circuit lookup tables with logarithmic number of ciphertexts. In Marc Joye and Gregor Leander, editors, *EUROCRYPT 2024, Part V*, volume 14655 of *LNCS*, pages 185–215. Springer, Cham, May 2024.

[HKO22] David Heath, Vladimir Kolesnikov, and Rafail Ostrovsky. EpiGRAM: Practical garbled RAM. In Orr Dunkelman and Stefan Dziembowski, editors, *EUROCRYPT 2022, Part I*, volume 13275 of *LNCS*, pages 3–33. Springer, Cham, May / June 2022.

[HKO23] David Heath, Vladimir Kolesnikov, and Rafail Ostrovsky. Tri-state circuits - A circuit model that captures RAM. In Helena Handschuh and Anna Lysyanskaya, editors, *CRYPTO 2023, Part IV*, volume 14084 of *LNCS*, pages 128–160. Springer, Cham, August 2023.

[ILL25] Yuval Ishai, Hanjun Li, and Huijia Lin. A unified framework for succinct garbling from homomorphic secret sharing. In Yael Tauman Kalai and Seny F. Kamara, editors, *CRYPTO 2025, Part IV*, volume 16003 of *LNCS*, pages 390–425. Springer, Cham, August 2025.

[KN25] Vladimir Kolesnikov and Lucien Ng. Toss: Garbled PIR from table-only stacking. In *ACM CCS*, 2025.

[KNR+17] Vladimir Kolesnikov, Jesper Buus Nielsen, Mike Rosulek, Ni Trieu, and Roberto Trifiletti. DUPLO: Unifying cut-and-choose for garbled circuits. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *ACM CCS 2017*, pages 3–20. ACM Press, October / November 2017.

[KRRW18] Jonathan Katz, Samuel Ranellucci, Mike Rosulek, and Xiao Wang. Optimizing authenticated garbling for faster secure two-party computation. In Hovav Shacham and Alexandra Boldyreva, editors, *CRYPTO 2018, Part III*, volume 10993 of *LNCS*, pages 365–391. Springer, Cham, August 2018.

[KS08] Vladimir Kolesnikov and Thomas Schneider. Improved garbled circuit: Free XOR gates and applications. In Luca Aceto, Ivan Damgård, Leslie Ann Goldberg, Magnús M. Halldórsson, Anna Ingólfsdóttir, and Igor Walukiewicz, editors, *ICALP 2008, Part II*, volume 5126 of *LNCS*, pages 486–498. Springer, Berlin, Heidelberg, July 2008.

[LLZ25] Wei-Kai Lin, Zhenghao Lu, and Hong-Sheng Zhou. Efficient garbled pseudorandom functions and lookup tables from minimal assumption. In Benny Applebaum and Huijia (Rachel) Lin, editors, *TCC 2025, Part I*, volume 16268 of *LNCS*, pages 64–96. Springer, Cham, December 2025.

[LO13] Steve Lu and Rafail Ostrovsky. How to garble RAM programs. In Thomas Johansson and Phong Q. Nguyen, editors, *EUROCRYPT 2013*, volume 7881 of *LNCS*, pages 719–734. Springer, Berlin, Heidelberg, May 2013.

[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.

[LP15] Yehuda Lindell and Benny Pinkas. An efficient protocol for secure two-party computation in the presence of malicious adversaries. *Journal of Cryptology*, 28(2):312–350, April 2015.

[LWYY25a] Hanlin Liu, Xiao Wang, Kang Yang, and Yu Yu. Authenticated BitGC for actively secure rate-one 2PC. In Yael Tauman Kalai and Seny F. Kamara, editors, *CRYPTO 2025, Part IV*, volume 16003 of *LNCS*, pages 652–687. Springer, Cham, August 2025.

[LWYY25b] Hanlin Liu, Xiao Wang, Kang Yang, and Yu Yu. BitGC: Garbled circuits with 1 bit per gate. In Serge Fehr and Pierre-Alain Fouque, editors, *EUROCRYPT 2025, Part VII*, volume 15607 of *LNCS*, pages 437–466. Springer, Cham, May 2025.

[MORS24] Pierre Meyer, Claudio Orlandi, Lawrence Roy, and Peter Scholl. Rate-1 arithmetic garbling from homomorphic secret sharing. In Elette Boyle and Mohammad Mahmoody, editors, *TCC 2024, Part IV*, volume 15367 of *LNCS*, pages 71–97. Springer, Cham, December 2024.

[NNOB12] Jesper Buus Nielsen, Peter Sebastian Nordholt, Claudio Orlandi, and Sai Sheshank Burra. A new approach to practical active-secure two-party computation. In Reihaneh Safavi-Naini and Ran Canetti, editors, *CRYPTO 2012*, volume 7417 of *LNCS*, pages 681–700. Springer, Berlin, Heidelberg, August 2012.

[PLS23] Andrew Park, Wei-Kai Lin, and Elaine Shi. NanoGRAM: Garbled RAM with $\widetilde{O}(\log N)$ overhead. In Carmit Hazay and Martijn Stam, editors, *EUROCRYPT 2023, Part I*, volume 14004 of *LNCS*, pages 456–486. Springer, Cham, April 2023.

[Roy22] Lawrence Roy. SoftSpokenOT: Quieter OT extension from small-field silent VOLE in the minicrypt model. In Yevgeniy Dodis and Thomas Shrimpton, editors, *CRYPTO 2022, Part I*, volume 13507 of *LNCS*, pages 657–687. Springer, Cham, August 2022.

[RR21] Mike Rosulek and Lawrence Roy. Three halves make a whole? Beating the half-gates lower bound for garbled circuits. In Tal Malkin and Chris Peikert, editors, *CRYPTO 2021, Part I*, volume 12825 of *LNCS*, pages 94–124, Virtual Event, August 2021. Springer, Cham.

[WGC19] Sameer Wagh, Divya Gupta, and Nishanth Chandran. SecureNN: 3-party secure computation for neural network training. *PoPETs*, 2019(3):26–49, July 2019.

[WRK17] Xiao Wang, Samuel Ranellucci, and Jonathan Katz. Authenticated garbling and efficient maliciously secure two-party computation. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *ACM CCS 2017*, pages 21–37. ACM Press, October / November 2017.

[WTB+21] Sameer Wagh, Shruti Tople, Fabrice Benhamouda, Eyal Kushilevitz, Prateek Mittal, and Tal Rabin. Falcon: Honest-majority maliciously secure framework for private deep learning. *PoPETs*, 2021(1):188–208, January 2021.

[WYKW21] Chenkai Weng, Kang Yang, Jonathan Katz, and Xiao Wang. Wolverine: Fast, scalable, and communication-efficient zero-knowledge proofs for boolean and arithmetic circuits. In *2021 IEEE Symposium on Security and Privacy*, pages 1074–1091. IEEE Computer Society Press, May 2021.

[YWL+20] Kang Yang, Chenkai Weng, Xiao Lan, Jiang Zhang, and Xiao Wang. Ferret: Fast extension for correlated OT with small communication. In Jay Ligatti, Xinming Ou, Jonathan Katz, and Giovanni Vigna, editors, *ACM CCS 2020*, pages 1607–1626. ACM Press, November 2020.

[ZRE15] Samee Zahur, Mike Rosulek, and David Evans. Two halves make a whole - reducing data transfer in garbled circuits using half gates. In Elisabeth Oswald and Marc Fischlin, editors, *EUROCRYPT 2015, Part II*, volume 9057 of *LNCS*, pages 220–250. Springer, Berlin, Heidelberg, April 2015.

# A Helper Functionalities

## A.1 Equality Check

Our preprocessing protocol and consistency checks use a relaxed equality-checking functionality $\mathcal{F}_{\mathsf{EQ}}$ [NNOB12, CWYY23] provided below.

**Functionality 2: $\mathcal{F}_{\mathsf{EQ}}$**

On input $(\mathsf{eq}, sid, \ell, x)$ from $\mathsf{G}$ and $(\mathsf{eq}, sid, \ell, y)$ from $\mathsf{E}$, where $x, y \in \{0,1\}^\ell$:

- If $x = y$, send $(sid, \mathsf{true})$ to both parties.
- Otherwise, send $(sid, \mathsf{false})$ to both parties, and send the honest party's input to the adversary.

$\mathcal{F}_{\mathsf{EQ}}$ can be securely realized by parties exchanging hashes of their inputs in the random oracle model.

## A.2 Correlated OT

Our protocols adopt the standard functionality [BCG+19a, Roy22] of correlated oblivious transfer (COT) to generate random authenticated bits. This functionality $\mathcal{F}_{\mathrm{COT}}$ is shown below.

**Functionality 3: $\mathcal{F}_{\mathrm{COT}}$**

Running with sender $\mathsf{G}$, receiver $\mathsf{E}$, and an ideal adversary:

**Initialize.** On input $(\mathsf{init}, sid, \Delta)$ from $\mathsf{G}$ with $\Delta \in \{0,1\}^\kappa$, and $(\mathsf{init}, sid)$ from $\mathsf{E}$, store $(sid, \Delta)$ and ignore subsequent $(\mathsf{init}, sid)$ commands.

**Extend.** On input $(\mathsf{extend}, sid, \ell)$ from both parties:

- Sample $[\boldsymbol{u}\Delta]^{\mathsf{G}} \leftarrow (\{0,1\}^\kappa)^\ell$.
- Sample $\boldsymbol{u} \leftarrow (\{0,1\})^\ell$ and set $[\boldsymbol{u}\Delta]^{\mathsf{E}} = [\boldsymbol{u}\Delta]^{\mathsf{G}} \oplus \boldsymbol{u} \cdot \Delta \in (\{0,1\}^\kappa)^\ell$.
- Output $(sid, [\boldsymbol{u}\Delta]^{\mathsf{G}})$ to $\mathsf{G}$ and $(sid, \boldsymbol{u}, [\boldsymbol{u}\Delta]^{\mathsf{E}})$ to $\mathsf{E}$.

**Corrupted parties.** Allow the corrupted party to choose its own output, compute the honest party's output accordingly.

It is straightforward to use $\mathcal{F}_{\mathrm{COT}}$ to generate *dual authenticated shares* (Section 2.4) of uniformly random bits [NNOB12, WRK17].

# B Security Proofs

In this section we prove security of our two authenticated garbling protocols. We begin by proving correctness of the tensor macros (Construction 1), a building block used by both protocols' security proofs.

## B.1 Proof of Lemma 5

We argue by induction on $i$ that $S^{\mathsf{E}}_{i,j} = S_{i,j} \oplus [j = \alpha_i]\Delta_{\mathsf{G}}$ for all $j \in [2^{i+1}]$.

*Base case ($i = 0$).* $\alpha_0 = a_0$ and $S^{\mathsf{E}}_{0,j} = A_0 \oplus a_0\Delta_{\mathsf{G}}$ for $j \in \{0,1\}$, while $S_{0,0} = A_0 \oplus \Delta_{\mathsf{G}}$ and $S_{0,1} = A_0$. For $j = a_0$: $S^{\mathsf{E}}_{0,j} = S_{0,j} \oplus \Delta_{\mathsf{G}}$. For $j = \bar{a_0}$: $S^{\mathsf{E}}_{0,j} = S_{0,j}$.

*Inductive step.* Assume the invariant at level $i-1$. For $j \neq \alpha_{i-1}$, $\mathsf{E}$'s label at $j$ equals the garbler's level-$(i-1)$ zero-label, so $R^{\mathsf{E}}_{i,j} = R_{i,j}$. The active-seed recovery gives

$$R^{\mathsf{E}}_{i,\alpha_{i-1}} = G_i \oplus \bigoplus_{j \neq \alpha_{i-1}} R_{i,j} \oplus (A_i \oplus a_i \Delta_{\mathsf{G}}) = R_{i,\alpha_{i-1}} \oplus a_i \Delta_{\mathsf{G}},$$

using $G_i = \bigoplus_j R_{i,j} \oplus A_i$. Combined: $R^{\mathsf{E}}_{i,j} = R_{i,j} \oplus [j = \alpha_{i-1}]\, a_i \Delta_{\mathsf{G}}$.

For each $j \in [2^i]$,

$$S^{\mathsf{E}}_{i,j} = S_{i,j} \oplus [j = \alpha_{i-1}](1 \oplus a_i)\Delta_{\mathsf{G}}, \quad S^{\mathsf{E}}_{i,2^i+j} = S_{i,2^i+j} \oplus [j = \alpha_{i-1}]\, a_i \Delta_{\mathsf{G}}.$$

Since $\alpha_i = \alpha_{i-1} + a_i \cdot 2^i$: when $a_i = 0$, the $\Delta_{\mathsf{G}}$ shift is applied at position $\alpha_{i-1} = \alpha_i$ in the left half; when $a_i = 1$, it's applied at position $2^i + \alpha_{i-1} = \alpha_i$ in the right half. Either way, the invariant holds at level $i$.

At level $n-1$, $\mathsf{E}$ has $\mathsf{L}^{\mathsf{E}}_\ell = \mathsf{L}_\ell$ for $\ell \neq \alpha_{n-1}$, so $X^{\mathsf{E}}_{\ell,k} = X_{\ell,k}$ for $\ell \neq \alpha_{n-1}$. The leaf step recovers

$$\begin{aligned} X^{\mathsf{E}}_{\alpha_{n-1},k} &= \bigoplus_{\ell \neq \alpha_{n-1}} X_{\ell,k} \oplus G_k \oplus \big((B_k \oplus b_k \Delta_{\mathsf{G}}) \,\|\, (B'_k \oplus b_k \Delta_{\mathsf{E}})\big) \\ &= X_{\alpha_{n-1},k} \oplus \big(b_k \Delta_{\mathsf{G}} \,\|\, b_k \Delta_{\mathsf{E}}\big). \end{aligned}$$

Letting $\boldsymbol{u}_{\alpha_{n-1}}$ be the indicator of position $\alpha_{n-1} = \mathrm{val}(\boldsymbol{a})$, $\boldsymbol{X}_{\mathsf{E}} = \boldsymbol{X}_{\mathsf{G}} \oplus \boldsymbol{u}_{\alpha_{n-1}} \otimes (\boldsymbol{b}\Delta_{\mathsf{G}} \,\|\, \boldsymbol{b}\Delta_{\mathsf{E}})$. Multiplying by $\mathcal{T}(\mathrm{id})^{\intercal}$ (which maps $\boldsymbol{u}_{\alpha_{n-1}}$ to $\boldsymbol{a}$) gives $\boldsymbol{Z}_{\mathsf{E}} = \boldsymbol{Z}_{\mathsf{G}} \oplus (\boldsymbol{a} \otimes \boldsymbol{b})\Delta_{\mathsf{G}}$and $\boldsymbol{Z}'_{\mathsf{E}} = \boldsymbol{Z}'_{\mathsf{G}} \oplus (\boldsymbol{a} \otimes \boldsymbol{b})\Delta_{\mathsf{E}}$. □

## B.2 Proof of Theorem 6

We start by proving security of $\Pi_{\text{2pc-tensor-1}}$ against a static malicious adversary that corrupts either the garbler $\mathsf{G}$ or the evaluator $\mathsf{E}$.

*Malicious* $\mathsf{G}$. Before describing the simulator, we prove two lemmas that establish properties of $\Pi_{\text{2pc-tensor-1}}$'s execution under malicious $\mathsf{G}$ and honest $\mathsf{E}$. With these lemmas, the simulation and indistinguishability arguments will be straightforward.

**Lemma 10** (Correctness)**.** *If both parties are honest, at the end of* $\Pi_{\text{2pc-tensor-1}}$ *the consistency check passes and* $\mathsf{E}$ *outputs* $\boldsymbol{z} = \mathcal{C}(\boldsymbol{x}, \boldsymbol{y})$*. For any malicious* $\mathsf{G}$ *strategy, with probability at least* $1 - 2^{-\rho}$*, either* $\mathsf{E}$ *aborts before output decoding or outputs* $\mathcal{C}(\boldsymbol{x}, \boldsymbol{y})$*.*

*Proof.* For each tensor gate $(\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c}, \otimes)$ in $\mathcal{C}$, let $\boldsymbol{\delta}_a$ denote $\mathsf{G}$'s deviation from the true masked value at input wire $\boldsymbol{a}$ under inputs $(\boldsymbol{x}, \boldsymbol{y})$: $\mathsf{E}$ reconstructs $(\boldsymbol{a} \oplus \boldsymbol{\lambda}_a) \oplus \boldsymbol{\delta}_a$ from $\mathcal{GC}$ during evaluation. Define $\boldsymbol{\delta}_b$ analogously. Honest $\mathsf{G}$ corresponds to all $\boldsymbol{\delta} = \boldsymbol{0}$, in which case we show by induction on the circuit's topological order that parties hold XOR shares of $w\Delta_{\mathsf{G}}$ and $w\Delta_{\mathsf{E}}$ on every wire $w$, where $w$ is the value at the wire under inputs $(\boldsymbol{x}, \boldsymbol{y})$.

*Input wires.* For $\mathsf{G}$'s input $\boldsymbol{x}$, defined by the input-encoding masked message $\boldsymbol{x} \oplus \boldsymbol{\lambda}_x$: $\mathsf{G}$ samples $[\boldsymbol{x}\Delta_{\mathsf{G}}]^{\mathsf{G}}$ and sets $[\boldsymbol{x}\Delta_{\mathsf{G}}]^{\mathsf{E}} := [\boldsymbol{x}\Delta_{\mathsf{G}}]^{\mathsf{G}} \oplus \boldsymbol{x}\Delta_{\mathsf{G}}$, summing to $\boldsymbol{x}\Delta_{\mathsf{G}}$. For $\Delta_{\mathsf{E}}$, $\mathsf{G}$ sets $[\boldsymbol{x}\Delta_{\mathsf{E}}]^{\mathsf{G}} := [\boldsymbol{\lambda}_x\Delta_{\mathsf{E}}]^{\mathsf{G}}$ and $\mathsf{E}$ sets $[\boldsymbol{x}\Delta_{\mathsf{E}}]^{\mathsf{E}} := [\boldsymbol{\lambda}_x\Delta_{\mathsf{E}}]^{\mathsf{E}} \oplus (\boldsymbol{x} \oplus \boldsymbol{\lambda}_x)\Delta_{\mathsf{E}}$, summing to $\boldsymbol{x}\Delta_{\mathsf{E}}$. For $\mathsf{E}$'s input $\boldsymbol{y}$, the $\Delta_{\mathsf{E}}$-shares are set analogously. For $\Delta_{\mathsf{G}}$, $\mathsf{G}$ sends $W = [\boldsymbol{y}\Delta_{\mathsf{G}}]^{\mathsf{G}} \oplus [\boldsymbol{\lambda}_y\Delta_{\mathsf{G}}]^{\mathsf{G}} \oplus (\boldsymbol{y} \oplus \boldsymbol{\lambda}_y)\Delta_{\mathsf{G}}$ and $\mathsf{E}$ sets $[\boldsymbol{y}\Delta_{\mathsf{G}}]^{\mathsf{E}} := [\boldsymbol{\lambda}_y\Delta_{\mathsf{G}}]^{\mathsf{E}} \oplus W$, summing to $\boldsymbol{y}\Delta_{\mathsf{G}}$.

*XOR gates.* Both invariants are preserved by linearity.

*Tensor gates,* $\Delta_{\mathsf{G}}$*-shares.* By induction on $\boldsymbol{a}, \boldsymbol{b}$ and the preprocessed shares of $\boldsymbol{\lambda}_a\Delta_{\mathsf{G}}, \boldsymbol{\lambda}_b\Delta_{\mathsf{G}}$,

$$[(\boldsymbol{a} \oplus \boldsymbol{\lambda}_a)\Delta_{\mathsf{G}}]^{\mathsf{G}} \oplus [(\boldsymbol{a} \oplus \boldsymbol{\lambda}_a)\Delta_{\mathsf{G}}]^{\mathsf{E}} = (\boldsymbol{a} \oplus \boldsymbol{\lambda}_a)\Delta_{\mathsf{G}},$$

and symmetrically for $\boldsymbol{b}$. Combined with the cleartext shares $[\boldsymbol{a} \oplus \boldsymbol{\lambda}_a]^{\mathsf{G}}, [\boldsymbol{b} \oplus \boldsymbol{\lambda}_b]^{\mathsf{G}}$ in $\mathcal{GC}$, $\mathsf{E}$ reconstructs $\boldsymbol{a} \oplus \boldsymbol{\lambda}_a$ and $\boldsymbol{b} \oplus \boldsymbol{\lambda}_b$.

Lemma 5 gives

$$\boldsymbol{Z}^{\mathsf{G}}_{c,0} \oplus \boldsymbol{Z}^{\mathsf{E}}_{c,0} = \big((\boldsymbol{a} \oplus \boldsymbol{\lambda}_a) \otimes \boldsymbol{\lambda}_b\big)\Delta_{\mathsf{G}}, \qquad \boldsymbol{Z}^{\mathsf{G}}_{c,1} \oplus \boldsymbol{Z}^{\mathsf{E}}_{c,1} = \big((\boldsymbol{b} \oplus \boldsymbol{\lambda}_b) \otimes \boldsymbol{a}\big)\Delta_{\mathsf{G}}.$$

Summing the $\Delta_{\mathsf{G}}$-shares from $\mathsf{Garble}$ and $\mathsf{Eval}$,

$$\begin{aligned} [\boldsymbol{c}\Delta_{\mathsf{G}}]^{\mathsf{G}} \oplus [\boldsymbol{c}\Delta_{\mathsf{G}}]^{\mathsf{E}} &= \big((\boldsymbol{a} \oplus \boldsymbol{\lambda}_a) \otimes \boldsymbol{\lambda}_b \oplus ((\boldsymbol{b} \oplus \boldsymbol{\lambda}_b) \otimes \boldsymbol{a})^{\intercal} \oplus \boldsymbol{\lambda}_a \otimes \boldsymbol{\lambda}_b\big)\Delta_{\mathsf{G}} \\ &= (\boldsymbol{a} \otimes \boldsymbol{b})\Delta_{\mathsf{G}} = \boldsymbol{c}\Delta_{\mathsf{G}}. \end{aligned}$$

*Tensor gates, $\Delta_{\mathsf{E}}$-shares.* By induction on $\boldsymbol{a}, \boldsymbol{b}$, the masked values $\boldsymbol{a} \oplus \boldsymbol{\lambda}_a, \boldsymbol{b} \oplus \boldsymbol{\lambda}_b$ used in propagation, and the preprocessed shares $[\boldsymbol{\lambda}_a \Delta_{\mathsf{E}}], [(\boldsymbol{\lambda}_a \otimes \boldsymbol{\lambda}_b)\Delta_{\mathsf{E}}]$, the $\Delta_{\mathsf{E}}$ computation in the consistency check sums to

$$\big((\boldsymbol{a} \oplus \boldsymbol{\lambda}_a) \otimes \boldsymbol{b} \oplus ((\boldsymbol{b} \oplus \boldsymbol{\lambda}_b) \otimes \boldsymbol{\lambda}_a)^{\intercal} \oplus \boldsymbol{\lambda}_a \otimes \boldsymbol{\lambda}_b\big)\Delta_{\mathsf{E}} = (\boldsymbol{a} \otimes \boldsymbol{b})\Delta_{\mathsf{E}} = \boldsymbol{c}\Delta_{\mathsf{E}}.$$

*Consistency check.* By the $\Delta_{\mathsf{E}}$-invariant on $\boldsymbol{a}, \boldsymbol{b}$ and the preprocessed shares of $\boldsymbol{\lambda}_a, \boldsymbol{\lambda}_b$, $[\boldsymbol{e}_a \Delta_{\mathsf{E}}]$ and $[\boldsymbol{e}_b \Delta_{\mathsf{E}}]$ each sum to $\mathbf{0}$, and $\mathsf{CheckZero}$ accepts.

*Output decoding.* The $\Delta_{\mathsf{E}}$-invariant holds on every wire including circuit outputs so $\mathsf{Open}([\boldsymbol{z}\Delta_{\mathsf{E}}])$ reveals $\boldsymbol{z}$ and $\mathsf{E}$ outputs $\boldsymbol{z} = \mathcal{C}(\boldsymbol{x}, \boldsymbol{y})$.

Now consider a malicious $\mathsf{G}$, and assume there exists some revealed value with $\boldsymbol{\delta} \neq \mathbf{0}$. Consider the first such deviation. The corresponding residual error in the consistency check equals $[\boldsymbol{e}\Delta_{\mathsf{E}}]^{\mathsf{G}} \oplus [\boldsymbol{e}\Delta_{\mathsf{E}}]^{\mathsf{E}} = \boldsymbol{\delta}\Delta_{\mathsf{E}} \neq \mathbf{0}$ and by IT-MAC unforgeability of $\Delta_{\mathsf{E}}$ (uniform in $\mathcal{A}$'s view), $\mathsf{CheckZero}$ rejects except with probability $2^{-\rho}$. If there are no errors in revealed masked values but a malicious $\mathsf{G}$ attempts to flip the circuit output, it again needs to guess $\Delta_{\mathsf{E}}$, else $\mathsf{Open}$ will abort. □

*Proof of Lemma 7.* A malicious $\mathsf{G}$'s strategy amounts to corrupting the ciphertexts in $\mathcal{GC}$. The leaky-tensor analysis (Section C.2) shows that, up to a $2^{-\rho}$-probability event in which $\mathsf{G}$ guesses $\Delta_{\mathsf{E}}$, any such corruption either causes $\mathsf{E}$ to abort regardless of its input, or passes iff $\mathsf{G}$ correctly guesses the masked value of some prefix of some tensor-gate input. The first case contributes to the abort probability identically across $\boldsymbol{y}, \boldsymbol{y}'$, so it suffices to analyze what happens when $\mathsf{G}$ guesses some masked inputs.

We argue against a strictly stronger adversary that, for any subset of masked-input coordinates of its choosing, submits a guess for each and aborts iff any guess is wrong. This subsumes prefix-equality, which is per-bit guessing on consecutive coordinates. We show that even this stronger adversary's abort distribution is $2^{-\rho}$-close across $\boldsymbol{y}, \boldsymbol{y}'$.

Each masked coordinate is the XOR of a wire value (which depends on $\boldsymbol{y}$) and a wire mask. Recall that masks supplied by $\mathcal{F}_{\mathsf{Pre}}$ are uniformly random from $\mathsf{G}$'s view. Therefore, each guess of a masked bit is correct with probability $1/2$ regardless of $\boldsymbol{y}$, and the probability of no abort is the same on $\boldsymbol{y}$ and $\boldsymbol{y}'$. □

We now define the simulator $\mathcal{S}_{\mathsf{G}}$ in the $\mathcal{F}_{\mathsf{Pre}}$-hybrid model that interacts with an adversary $\mathcal{A}$ controlling the garbler $\mathsf{G}$.

1. $\mathcal{S}_{\mathsf{G}}$ plays the role of $\mathcal{F}_{\mathsf{Pre}}$ and receives $\mathcal{A}$'s shares of preprocessing and samples an honest $\mathsf{E}$'s shares accordingly.

2. $\mathcal{S}_{\mathsf{G}}$ receives $\mathcal{GC}$ from $\mathcal{A}$. During input encoding for $\mathsf{G}$'s input, $\mathcal{S}_{\mathsf{G}}$ acts as honest $\mathsf{E}$, receives $\boldsymbol{x} \oplus \boldsymbol{\lambda}_x$ from $\mathcal{A}$, and extracts $\boldsymbol{x}$ using its share of $\boldsymbol{\lambda}_x$. During input encoding for $\mathsf{E}$'s input, $\mathcal{S}_{\mathsf{G}}$ runs the protocol on input $\boldsymbol{y} = \mathbf{0}$.

3. $\mathcal{S}_{\mathsf{G}}$ evaluates the garbled circuit with its received encoded input as an honest $\mathsf{E}$ would, sends masked values to $\mathcal{A}$ and computes its shares under $\Delta_{\mathsf{E}}$ for the consistency check. It uses these to perform $\mathsf{CheckZero}$ as an honest $\mathsf{E}$.

4. $\mathcal{S}_{\mathsf{G}}$ receives authenticated output values $[\boldsymbol{z}\Delta_{\mathsf{E}}]^{\mathsf{G}}$ from $\mathcal{A}$ and checks against its shares obtained in the previous step, aborting if an honest $\mathsf{E}$ would. If $\mathcal{S}_{\mathsf{G}}$ has not yet aborted, it sends $x$ to $\mathcal{F}_{\mathsf{2pc}}$.

We show that the joint distribution of the outputs of $\mathcal{A}$ and an honest $\mathsf{E}$ in the real world is indistinguishable from the joint distribution of the outputs of $\mathcal{S}_{\mathsf{G}}$ and $\mathsf{E}$ in the ideal world, by a hybrid argument.

Hybrid 1. This is the hybrid-world protocol, where we imagine $\mathcal{S}_{\mathsf{G}}$ playing the role of an honest $\mathsf{E}$ using $\mathsf{E}$'s actual input $y$, while also playing the role of $\mathcal{F}_{\mathsf{Pre}}$.

Hybrid 2. Same as the previous hybrid, except that $\mathcal{S}_{\mathsf{G}}$ extracts $\mathcal{A}$'s input $x$ during input encoding. If an honest $\mathsf{E}$ would abort in any later step, $\mathcal{S}_{\mathsf{G}}$ sends `abort` to $\mathcal{F}_{\mathsf{2pc}}$; otherwise it sends $x$ to $\mathcal{F}_{\mathsf{2pc}}$. The view of $\mathcal{A}$ in both hybrids is identical. The output $\mathsf{E}$ gets is the same due to Lemma 10.

Hybrid 3. Same as the previous hybrid, except that $\mathcal{S}_{\mathsf{G}}$ uses $y = 0$ as input. The view of $\mathcal{A}$ in both hybrids is again identical (the masked inputs it sees during preprocessing are uniformly distributed). $\mathsf{E}$'s abort distribution is identical to the previous hybrid by Lemma 7.

Note that the last hybrid corresponds to the ideal-world execution described earlier. This completes the proof for a malicious $\mathsf{G}$.

*Malicious* $\mathsf{E}$. Next we define a simulator $\mathcal{S}_\mathsf{E}$ in the $\mathcal{F}_{\mathsf{Pre}}$-hybrid model interacting with an adversary $\mathcal{A}$ controlling the evaluator $\mathsf{E}$.

1. $\mathcal{S}_\mathsf{E}$ plays the role of $\mathcal{F}_{\mathsf{Pre}}$ and receives $\mathcal{A}$'s shares of preprocessing and samples an honest $\mathsf{G}$'s shares accordingly.

2. $\mathcal{S}_\mathsf{E}$ simulates the generation of $\mathcal{GC}$ as follows. This is identical to the semi-honest obliviousness simulator.

   (a) For each tensor gate $\boldsymbol{c} = \boldsymbol{a} \otimes \boldsymbol{b}$ in the circuit sample masked inputs uniformly $\boldsymbol{a} \oplus \boldsymbol{\lambda}_a \leftarrow \{0,1\}^n, \boldsymbol{b} \oplus \boldsymbol{\lambda}_b \leftarrow \{0,1\}^m$ where $n = |\boldsymbol{a}|, m = |b|$.

   (b) Sample the evaluator's shares $[(\boldsymbol{x}\|\boldsymbol{y})\Delta_\mathsf{G}] \leftarrow (\{0,1\}^\kappa)^{|\mathcal{I}|}$, $[(\boldsymbol{x}\|\boldsymbol{y})\Delta_\mathsf{E}] \leftarrow (\{0,1\}^\rho)^{|\mathcal{I}|}$ of inputs.

   (c) Proceed through the circuit in topological order to simulate $\mathsf{E}$'s shares and received $\mathcal{GC}$ consistent with these shares. For XOR gates compute $[\boldsymbol{c}\Delta_\mathsf{G}] = [\boldsymbol{a}\Delta_\mathsf{G}] \oplus [\boldsymbol{b}\Delta_\mathsf{G}]$. For tensor gates, simulate revealed masked inputs by defining $\mathsf{G}$'s share so that it is consistent with $\mathsf{E}$'s share and previously sampled masks. Simulate each call of the $\mathsf{Tensor}$ macro as follows. For the base case, use its simulated share to define seeds at both positions of the one-hot vector, as an honest $\mathsf{E}$ would, calling the one-hot position *active*. Inductive case:

      i. Using inactive seeds at level $i$, honestly generate all but two seeds (those derived from the active seed) at level $i+1$.

      ii. Sample ciphertext $G_i \leftarrow \{0,1\}^\kappa$ uniformly at random, use an honest evaluator's strategy to derive remaining seeds, tracking the active position.

      $\mathcal{S}_\mathsf{E}$ simulates the leaf seeds as follows. For all inactive positions, derive the leaf seeds as an honest evaluator would. Sample ciphertexts $G_j \leftarrow \{0,1\}^\kappa$ uniformly at random and compute the leaf seeds at the active position as an honest evaluator would. Compute the output shares $[\boldsymbol{c}\Delta_\mathsf{G}]$ as an honest evaluator would.

3. During input encoding, use input $\boldsymbol{x} = 0$ and extract $\mathcal{A}$'s input $\boldsymbol{y}$ from the masked value and preprocessing. Also, using preprocessing open $\mathsf{E}$'s simulated input shares sampled in Step 2(b).

4. Receive from $\mathcal{A}$ masked values for inputs to tensor gates and use these to propagate simulated evaluator's shares of inputs $[(\boldsymbol{x}\|\boldsymbol{y})\Delta_\mathsf{E}]$ to shares of all intermediate values, including circuit outputs $[\boldsymbol{z}\Delta_\mathsf{E}]$. In the consistency check, $\mathcal{S}_\mathsf{E}$ simulates the garbler's share of revealed values as $[\boldsymbol{a}\Delta_\mathsf{E}]^\mathsf{E} \oplus [\boldsymbol{\lambda}_a\Delta_\mathsf{E}]^\mathsf{E} \oplus (\boldsymbol{a} \oplus \boldsymbol{\lambda}_a)\Delta_\mathsf{E}$ (and similarly for $\boldsymbol{b}$) to convince $\mathcal{A}$ that the revealed value $\boldsymbol{a} \oplus \boldsymbol{\lambda}_a$ (resp. $\boldsymbol{b} \oplus \boldsymbol{\lambda}_b$) is correct.

5. If the protocol has not yet aborted, $\mathcal{S}_\mathsf{E}$ sends $\boldsymbol{y}$ and receives $\boldsymbol{z} = f(\boldsymbol{x}, \boldsymbol{y})$ from $\mathcal{F}_{\mathsf{2pc}}$. It opens $\boldsymbol{z}$ to $\mathcal{A}$ in output decoding by sending $[\boldsymbol{z}\Delta_\mathsf{E}]^\mathsf{E} \oplus \boldsymbol{z}\Delta_\mathsf{E}$.

We argue that $\mathcal{S}_\mathsf{E}$'s simulation is computationally indistinguishable from the real execution.

The view of $\mathcal{A}$ in the real execution consists of revealed masked inputs and ciphertexts for each tensor gate, received input shares during input encoding and the decoded output.

The revealed masked inputs are uniformly random in $\mathcal{A}$'s view in both the real and ideal world. We argue they remain indistinguishable in the context of the protocol.

For the $\mathsf{AuthTensor}$ ciphertexts, we argue indistinguishability level by level. At level $i+1$ the real ciphertext is

$$G_{i+1} = \bigoplus_{j \in [2^{i+1}]} R_{i+1,j} \oplus A_{i+1}$$

where $R_{i+1,j} = H(S_{i,j})$. At the active position $\alpha_i$, $\mathsf{G}$'s zero-label $S_{i,\alpha_i}$ differs from $\mathcal{A}$'s view $S^\mathsf{E}_{i,\alpha_i}$ by $\Delta_\mathsf{G}$. By circular-correlation robustness of $H$, $H(S_{i,\alpha_i})$ is random in $\mathcal{A}$'s view, and can be replaced with a uniform ciphertext. The same argument applies to leaf ciphertexts.

The input-encoding messages are computed by $\mathcal{S}_\mathsf{E}$ such that $\mathcal{A}$'s received shares are exactly the ones used to simulate $\mathcal{GC}$.

The output-decoding messages are identically distributed in both the real and ideal world and open $\boldsymbol{z} = \mathcal{C}(\boldsymbol{x}, \boldsymbol{y})$ to an $\mathsf{E}$ that correctly evaluated $\mathcal{GC}$ and revealed the correct masked values to $\mathsf{G}$ to obtain its shares $[\boldsymbol{z}\Delta_{\mathsf{E}}]$.

## B.3 Proof of Theorem 8

We prove security against a static malicious adversary that corrupts either the garbler $\mathsf{G}$ or the evaluator $\mathsf{E}$.

*Malicious* $\mathsf{G}$. Before describing the simulator, we prove two lemmas that establish properties of $\Pi_{\mathsf{2pc\text{-}tensor\text{-}2}}$'s execution under malicious $\mathsf{G}$ and honest $\mathsf{E}$. With these lemmas, the simulation and indistinguishability arguments will be straightforward.

**Lemma 11** (Correctness)**.** *If both parties are honest, at the end of* $\Pi_{\mathsf{2pc\text{-}tensor\text{-}2}}$ *the consistency check passes and* $\mathsf{E}$ *outputs* $\boldsymbol{z} = \mathcal{C}(\boldsymbol{x}, \boldsymbol{y})$*. For any malicious* $\mathsf{G}$ *strategy, with probability at least* $1 - 2^{-\rho}$*, either* $\mathsf{E}$ *aborts before output decoding or outputs* $\mathcal{C}(\boldsymbol{x}, \boldsymbol{y})$*.*

*Proof.* For each tensor gate $(\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c}, \otimes)$ in $\mathcal{C}$, let $\boldsymbol{\delta}_a$ denote $\mathsf{G}$'s deviation from the true masked value at input wire $\boldsymbol{a}$ under inputs $(\boldsymbol{x}, \boldsymbol{y})$: $\mathsf{E}$ reconstructs $(\boldsymbol{a} \oplus \boldsymbol{\lambda}_a) \oplus \boldsymbol{\delta}_a$ from $\mathcal{GC}$ during evaluation. Define $\boldsymbol{\delta}_b$ analogously. Honest $\mathsf{G}$ corresponds to all $\boldsymbol{\delta} = \mathbf{0}$, in which case we show by induction on the circuit's topological order that parties hold XOR shares of $w\Delta_{\mathsf{G}}$ and $w\Delta_{\mathsf{E}}$ on every wire $w$, where $w$ is the value at the wire under inputs $(\boldsymbol{x}, \boldsymbol{y})$.

*Input wires.* For $\mathsf{G}$'s input $\boldsymbol{x}$, $\mathsf{G}$ samples $[\boldsymbol{x}\Delta_{\mathsf{G}}]^{\mathsf{G}}$ and sets $[\boldsymbol{x}\Delta_{\mathsf{G}}]^{\mathsf{E}} := [\boldsymbol{x}\Delta_{\mathsf{G}}]^{\mathsf{G}} \oplus \boldsymbol{x}\Delta_{\mathsf{G}}$, summing to $\boldsymbol{x}\Delta_{\mathsf{G}}$. For $\Delta_{\mathsf{E}}$, $\mathsf{G}$ sets $[\boldsymbol{x}\Delta_{\mathsf{E}}]^{\mathsf{G}} := [\boldsymbol{\lambda}_x\Delta_{\mathsf{E}}]^{\mathsf{G}}$ inside $\mathsf{Garble}$ and $\mathsf{E}$ sets $[\boldsymbol{x}\Delta_{\mathsf{E}}]^{\mathsf{E}} := [\boldsymbol{\lambda}_x\Delta_{\mathsf{E}}]^{\mathsf{E}} \oplus (\boldsymbol{x} \oplus \boldsymbol{\lambda}_x)\Delta_{\mathsf{E}}$ during input encoding, summing to $\boldsymbol{x}\Delta_{\mathsf{E}}$. For $\mathsf{E}$'s input $\boldsymbol{y}$, the $\Delta_{\mathsf{E}}$-shares are set analogously. For $\Delta_{\mathsf{G}}$, $\mathsf{G}$ sends $W = [\boldsymbol{y}\Delta_{\mathsf{G}}]^{\mathsf{G}} \oplus [\boldsymbol{\lambda}_y\Delta_{\mathsf{G}}]^{\mathsf{G}} \oplus (\boldsymbol{y} \oplus \boldsymbol{\lambda}_y)\Delta_{\mathsf{G}}$ and $\mathsf{E}$ sets $[\boldsymbol{y}\Delta_{\mathsf{G}}]^{\mathsf{E}} := [\boldsymbol{\lambda}_y\Delta_{\mathsf{G}}]^{\mathsf{E}} \oplus W$, again summing to $\boldsymbol{y}\Delta_{\mathsf{G}}$.

*XOR gates.* Both invariants are preserved by linearity.

*Tensor gates.* By induction on $\boldsymbol{a}, \boldsymbol{b}$ and the preprocessed shares of $\boldsymbol{\lambda}_a\Delta_{\mathsf{G}}, \boldsymbol{\lambda}_b\Delta_{\mathsf{G}}$,

$$[(\boldsymbol{a} \oplus \boldsymbol{\lambda}_a)\Delta_{\mathsf{G}}]^{\mathsf{G}} \oplus [(\boldsymbol{a} \oplus \boldsymbol{\lambda}_a)\Delta_{\mathsf{G}}]^{\mathsf{E}} = (\boldsymbol{a} \oplus \boldsymbol{\lambda}_a)\Delta_{\mathsf{G}},$$

and symmetrically for $\boldsymbol{b}$. Combined with the cleartext shares $[\boldsymbol{a}\oplus\boldsymbol{\lambda}_a]^{\mathsf{G}}, [\boldsymbol{b}\oplus\boldsymbol{\lambda}_b]^{\mathsf{G}}$ in $\mathcal{GC}$, $\mathsf{E}$ recovers $\boldsymbol{a}\oplus\boldsymbol{\lambda}_a, \boldsymbol{b}\oplus\boldsymbol{\lambda}_b$, establishing the masked-value invariant on tensor-gate inputs.

Lemma 5 gives

$$\begin{aligned} \boldsymbol{Z}^{\mathsf{G}}_{c,0} \oplus \boldsymbol{Z}^{\mathsf{E}}_{c,0} &= ((\boldsymbol{a} \oplus \boldsymbol{\lambda}_a) \otimes \boldsymbol{\lambda}_b)\Delta_{\mathsf{G}}, \quad (\boldsymbol{Z}'_{c,0})^{\mathsf{G}} \oplus (\boldsymbol{Z}'_{c,0})^{\mathsf{E}} = ((\boldsymbol{a} \oplus \boldsymbol{\lambda}_a) \otimes \boldsymbol{\lambda}_b)\Delta_{\mathsf{E}}, \\ \boldsymbol{Z}^{\mathsf{G}}_{c,1} \oplus \boldsymbol{Z}^{\mathsf{E}}_{c,1} &= ((\boldsymbol{b} \oplus \boldsymbol{\lambda}_b) \otimes \boldsymbol{a})\Delta_{\mathsf{G}}, \quad (\boldsymbol{Z}'_{c,1})^{\mathsf{G}} \oplus (\boldsymbol{Z}'_{c,1})^{\mathsf{E}} = ((\boldsymbol{b} \oplus \boldsymbol{\lambda}_b) \otimes \boldsymbol{a})\Delta_{\mathsf{E}}. \end{aligned}$$

Summing the $\Delta_{\mathsf{G}}$-shares from $\mathsf{Garble}$ and $\mathsf{Eval}$ and applying the expansion from Lemma 10,

$$[\boldsymbol{c}\Delta_{\mathsf{G}}]^{\mathsf{G}} \oplus [\boldsymbol{c}\Delta_{\mathsf{G}}]^{\mathsf{E}} = (\boldsymbol{a} \otimes \boldsymbol{b})\Delta_{\mathsf{G}} = \boldsymbol{c}\Delta_{\mathsf{G}},$$

and the same expansion applied to the $\Delta_{\mathsf{E}}$-halves gives $[\boldsymbol{c}\Delta_{\mathsf{E}}]^{\mathsf{G}} \oplus [\boldsymbol{c}\Delta_{\mathsf{E}}]^{\mathsf{E}} = \boldsymbol{c}\Delta_{\mathsf{E}}$.

*Consistency check.* By the $\Delta_{\mathsf{E}}$-invariant on $\boldsymbol{a}, \boldsymbol{b}$ and the preprocessed shares of $\boldsymbol{\lambda}_a, \boldsymbol{\lambda}_b$, $[\boldsymbol{e}_a\Delta_{\mathsf{E}}]$ and $[\boldsymbol{e}_b\Delta_{\mathsf{E}}]$ each sum to $\mathbf{0}$, and $\mathsf{CheckZero}$ accepts.

*Output decoding.* The $\Delta_{\mathsf{E}}$-invariant holds on every wire including output wires so $\mathsf{Open}([\boldsymbol{z}\Delta_{\mathsf{E}}])$ reveals $\boldsymbol{z}$ and $\mathsf{E}$ outputs $\boldsymbol{z} = \mathcal{C}(\boldsymbol{x}, \boldsymbol{y})$.

Now consider a malicious $\mathsf{G}$, and assume there exists some revealed value with $\boldsymbol{\delta} \neq \mathbf{0}$. Consider the first such deviation. The corresponding residual error in the consistency check equals $[\boldsymbol{e}\Delta_{\mathsf{E}}]^{\mathsf{G}} \oplus [\boldsymbol{e}\Delta_{\mathsf{E}}]^{\mathsf{E}} = \boldsymbol{\delta}\Delta_{\mathsf{E}} \neq \mathbf{0}$ and by IT-MAC unforgeability of $\Delta_{\mathsf{E}}$ (uniform in $\mathcal{A}$'s view), $\mathsf{CheckZero}$ rejects except with probability $2^{-\rho}$. If there are no errors in revealed masked values but a malicious $\mathsf{G}$ attempts to flip the circuit output, it again needs to guess $\Delta_{\mathsf{E}}$, else $\mathsf{Open}$ will abort. □

*Proof of Lemma 9.* A malicious $\mathsf{G}$'s strategy amounts to corrupting the ciphertexts in $\mathcal{GC}$. The leaky-tensor analysis (Section C.2) shows that, up to a $2^{-\rho}$-probability event in which $\mathsf{G}$ guesses $\Delta_{\mathsf{E}}$, any such corruption either causes $\mathsf{E}$ to abort regardless of its input, or passes iff $\mathsf{G}$ correctly guesses the masked value of some prefix of some tensor-gate input. The first case contributes to the abort probability identically across $\boldsymbol{y}, \boldsymbol{y}'$, so it suffices to analyze what happens when $\mathsf{G}$ guesses some masked inputs.

We argue against a strictly stronger adversary that, for any subset of masked-input coordinates of its choosing, submits a guess for each and aborts iff any guess is wrong. This subsumes prefix-equality, which is per-bit guessing on consecutive coordinates. We show that even this stronger adversary's abort distribution is $2^{-\rho}$-close across $\boldsymbol{y}, \boldsymbol{y}'$.

Each masked coordinate is the XOR of a wire value (which depends on $\boldsymbol{y}$) and a wire mask. By Lemma 4, the compressed preprocessing's expanded masks are $\rho$-wise independent, so any $\rho$-subset of mask coordinates is jointly uniform from $\mathsf{G}$'s view.

If the adversary makes fewer than $\rho$ guesses, the corresponding masks are jointly uniform, each guess matches with probability $1/2$ regardless of $\boldsymbol{y}$, and the probability of no abort is the same on $\boldsymbol{y}$ and $\boldsymbol{y}'$.

If the adversary makes at least $\rho$ guesses, restrict to any $\rho$ of them. By $\rho$-wise independence, those $\rho$ masks are jointly uniform, so all $\rho$ guesses pass with probability $2^{-\rho}$. Thus $\Pr[\neg \mathsf{Abort}_{\boldsymbol{y}}] \leq 2^{-\rho}$ for every $\boldsymbol{y}$, and the difference is $\leq 2^{-\rho}$. Indeed, if it successfully guesses sufficiently many ($\geq \rho$) bits of masked inputs, $\mathcal{A}$ does learn some linear relation on the underlying values. However, this happens with very low probability since we ensure the compressed masks still have enough entropy. □

We now define the simulator $\mathcal{S}_{\mathsf{G}}$ in the $\mathcal{F}_{\text{cpre}}$-hybrid model that interacts with an adversary $\mathcal{A}$ controlling the garbler $\mathsf{G}$. The simulator is almost identical to the one for the previous protocol. The main difference is in arguing selective failure which is handled by the previous lemma.

1. $\mathcal{S}_{\mathsf{G}}$ plays the role of $\mathcal{F}_{\text{cpre}}$ and receives $\mathcal{A}$'s shares of preprocessing and samples an honest $\mathsf{E}$'s shares accordingly.
2. $\mathcal{S}_{\mathsf{G}}$ receives $\mathcal{GC}$ from $\mathcal{A}$. During input encoding for $\mathsf{G}$'s input, $\mathcal{S}_{\mathsf{G}}$ acts as honest $\mathsf{E}$, receives $\boldsymbol{x} \oplus \boldsymbol{\lambda}_x$ from $\mathcal{A}$, and extracts $\boldsymbol{x}$ using its share of $\boldsymbol{\lambda}_x$. During input encoding for $\mathsf{E}$'s input, $\mathcal{S}_{\mathsf{G}}$ runs the protocol on input $\boldsymbol{y} = \mathbf{0}$.
3. $\mathcal{S}_{\mathsf{G}}$ evaluates the garbled circuit with its received encoded input as an honest $\mathsf{E}$ would, obtaining shares under $\Delta_{\mathsf{E}}$. It uses these to perform $\mathsf{CheckZero}$ as an honest $\mathsf{E}$.
4. $\mathcal{S}_{\mathsf{G}}$ receives authenticated output values $[\boldsymbol{z}\Delta_{\mathsf{E}}]^{\mathsf{G}}$ from $\mathcal{A}$ and checks against its shares obtained in the previous step, aborting if an honest $\mathsf{E}$ would. If $\mathcal{S}_{\mathsf{G}}$ has not yet aborted, it sends $x$ to $\mathcal{F}_{\mathsf{2pc}}$.

We show that the joint distribution of the outputs of $\mathcal{A}$ and an honest $\mathsf{E}$ in the real world is indistinguishable from the joint distribution of the outputs of $\mathcal{S}_{\mathsf{G}}$ and $\mathsf{E}$ in the ideal world, by a hybrid argument.

Hybrid 1. This is the hybrid-world protocol, where we imagine $\mathcal{S}_{\mathsf{G}}$ playing the role of an honest $\mathsf{E}$ using $\mathsf{E}$'s actual input $y$, while also playing the role of $\mathcal{F}_{\text{cpre}}$.

Hybrid 2. Same as the previous hybrid, except that $\mathcal{S}_{\mathsf{G}}$ extracts $\mathcal{A}$'s input $x$ during input encoding. If an honest $\mathsf{E}$ would abort in any later step, $\mathcal{S}_{\mathsf{G}}$ sends $\texttt{abort}$ to $\mathcal{F}_{\mathsf{2pc}}$; otherwise it sends $x$ to $\mathcal{F}_{\mathsf{2pc}}$. The view of $\mathcal{A}$ in both hybrids is identical. The output $\mathsf{E}$ gets is the same due to Lemma 11.

Hybrid 3. Same as the previous hybrid, except that $\mathcal{S}_{\mathsf{G}}$ uses $y = 0$ as input. The view of $\mathcal{A}$ in both hybrids is again identical (the masked inputs it sees during preprocessing are uniformly distributed). $\mathsf{E}$'s abort distribution is $2^{-\rho}$-indistinguishable from the previous hybrid by Lemma 9.

Note that the last hybrid corresponds to the ideal-world execution described earlier. This completes the proof for a malicious $\mathsf{G}$.

*Malicious* $\mathsf{E}$. Next we define a simulator $\mathcal{S}_{\mathsf{E}}$ in the $\mathcal{F}_{\text{cpre}}$-hybrid model interacting with an adversary $\mathcal{A}$ controlling the evaluator $\mathsf{E}$.

1. $\mathcal{S}_{\mathsf{E}}$ plays the role of $\mathcal{F}_{\text{cpre}}$ and receives $\mathcal{A}$'s shares of preprocessing and samples an honest $\mathsf{G}$'s shares accordingly.
2. $\mathcal{S}_{\mathsf{E}}$ simulates the generation of $\mathcal{GC}$ as follows. This is identical to the semi-honest obliviousness simulator.
   (a) For each tensor gate $\boldsymbol{c} = \boldsymbol{a} \otimes \boldsymbol{b}$ in the circuit sample masked inputs uniformly $\boldsymbol{a} \oplus \boldsymbol{\lambda}_a \leftarrow \{0,1\}^n, \boldsymbol{b} \oplus \boldsymbol{\lambda}_b \leftarrow \{0,1\}^m$ where $n = |\boldsymbol{a}|, m = |b|$.

(b) Sample the evaluator's shares $[(\boldsymbol{x}\|\boldsymbol{y})\Delta_{\mathsf{G}}] \leftarrow (\{0,1\}^{\kappa})^{|\mathcal{I}|}$, $[(\boldsymbol{x}\|\boldsymbol{y})\Delta_{\mathsf{E}}] \leftarrow (\{0,1\}^{\rho})^{|\mathcal{I}|}$ of inputs.

(c) Proceed through the circuit in topological order to simulate $\mathsf{E}$'s shares and received $\mathcal{GC}$ consistent with these shares. For XOR gates compute $[\boldsymbol{c}\Delta_{\mathsf{G}}] = [\boldsymbol{a}\Delta_{\mathsf{G}}] \oplus [\boldsymbol{b}\Delta_{\mathsf{G}}]$ and $[\boldsymbol{c}\Delta_{\mathsf{E}}] = [\boldsymbol{a}\Delta_{\mathsf{E}}] \oplus [\boldsymbol{b}\Delta_{\mathsf{E}}]$. For tensor gates, simulate revealed masked inputs by defining $\mathsf{G}$'s share so that it is consistent with $\mathsf{E}$'s share and previously sampled masks. Simulate each call of the $\mathsf{AuthTensor}$ macro as follows. For the base case, use its simulated share to define seeds at both positions of the one-hot vector, as an honest $\mathsf{E}$ would, calling the one-hot position *active*. Inductive case:

i. Using inactive seeds at level $i$, honestly generate all but two seeds (those derived from the active seed) at level $i+1$.

ii. Sample ciphertext $G_i \leftarrow \{0,1\}^{\kappa}$ uniformly at random, use an honest evaluator's strategy to derive remaining seeds, tracking the active position.

$\mathcal{S}_{\mathsf{E}}$ simulates the leaf seeds as follows. For all inactive positions, derive the leaf seeds as an honest evaluator would. Sample ciphertexts $G_j \leftarrow \{0,1\}^{\kappa+\rho}$ uniformly at random and compute the leaf seeds at the active position as an honest evaluator would. Compute the output shares $[\boldsymbol{c}\Delta_{\mathsf{G}}], [\boldsymbol{c}\Delta_{\mathsf{E}}]$ as an honest evaluator would.

3. During input encoding, use input $\boldsymbol{x} = 0$ and extract $\mathcal{A}$'s input $\boldsymbol{y}$ from the masked value and preprocessing. Also, using preprocessing open $\mathsf{E}$'s simulated input shares sampled in Step 2(b).

4. In the consistency check, $\mathcal{S}_{\mathsf{E}}$ simulates the garbler's share of revealed values as $[\boldsymbol{a}\Delta_{\mathsf{E}}]^{\mathsf{E}} \oplus [\boldsymbol{\lambda}_a\Delta_{\mathsf{E}}]^{\mathsf{E}} \oplus (\boldsymbol{a} \oplus \boldsymbol{\lambda}_a)\Delta_{\mathsf{E}}$ (and similarly for $\boldsymbol{b}$) to convince $\mathcal{A}$ that the revealed value $\boldsymbol{a} \oplus \boldsymbol{\lambda}_a$ (resp. $\boldsymbol{b} \oplus \boldsymbol{\lambda}_b$) is correct.

5. If the protocol has not yet aborted, $\mathcal{S}_{\mathsf{E}}$ sends $\boldsymbol{y}$ and receives $\boldsymbol{z} = f(\boldsymbol{x}, \boldsymbol{y})$ from $\mathcal{F}_{\mathsf{2pc}}$. It opens $\boldsymbol{z}$ to $\mathcal{A}$ in output decoding by sending $[\boldsymbol{z}\Delta_{\mathsf{E}}]^{\mathsf{E}} \oplus \boldsymbol{z}\Delta_{\mathsf{E}}$.

We argue that $\mathcal{S}_{\mathsf{E}}$'s simulation is computationally indistinguishable from the real execution.

The view of $\mathcal{A}$ in the real execution consists of revealed masked inputs and ciphertexts for each tensor gate, received input shares during input encoding and the decoded output.

The revealed masked inputs are uniformly random in $\mathcal{A}$'s view in both the real and ideal world. We argue they remain indistinguishable in the context of the protocol.

For the $\mathsf{AuthTensor}$ ciphertexts, we argue indistinguishability level by level. At level $i+1$ the real ciphertext is

$$G_{i+1} = \bigoplus_{j \in [2^{i+1}]} R_{i+1,j} \oplus A_{i+1}$$

where $R_{i+1,j} = H(S_{i,j})$. At the active position $\alpha_i$, $\mathsf{G}$'s zero-label $S_{i,\alpha_i}$ differs from $\mathcal{A}$'s view $S^{\mathsf{E}}_{i,\alpha_i}$ by $\Delta_{\mathsf{G}}$. By circular-correlation robustness of $H$, $H(S_{i,\alpha_i})$ is random in $\mathcal{A}$'s view, and can be replaced with a uniform ciphertext. The same argument applies to leaf ciphertexts.

The input-encoding messages are computed by $\mathcal{S}_{\mathsf{E}}$ such that $\mathcal{A}$'s received shares are exactly the ones used to simulate $\mathcal{GC}$.

The output-decoding messages are identically distributed in both the real and ideal world and open $\boldsymbol{z} = \mathcal{C}(\boldsymbol{x}, \boldsymbol{y})$ to an $\mathsf{E}$ that correctly evaluated $\mathcal{GC}$ to obtain its shares $[\boldsymbol{z}\Delta_{\mathsf{E}}]$.

# C Preprocessing for Protocol 1

Our first protocol (Section 4) consumes two kinds of correlations: authenticated bits $[\![r]\!]$ for uniform $r \in \{0,1\}$, and authenticated tensor triples $\big([\![\boldsymbol{x}]\!], [\![\boldsymbol{y}]\!], [\![\boldsymbol{x} \otimes \boldsymbol{y}]\!]\big)$, where the notation $[\![x]\!]$ denotes XOR-sharing $[x(\Delta_{\mathsf{G}}\|\Delta_{\mathsf{E}}\|1)]$ for global keys $\Delta_{\mathsf{G}}, \Delta_{\mathsf{E}}$ held by the garbler and evaluator. We bundle both into a single preprocessing functionality $\mathcal{F}_{\mathsf{Pre}}$, stated in Section 4 (Functionality 1), and instantiate it in the $(\mathcal{F}_{\mathrm{COT}}, \mathcal{F}_{\mathsf{EQ}})$-hybrid model.

Authenticated bits (in particular, dual-authenticated shares of uniformly random bits) can be generated from $\mathcal{F}_{\mathrm{COT}}$ using standard techniques [NNOB12, WRK17], at amortized cost $\kappa/\log\kappa$ bits per party per bit, using only base OTs and a circular correlation robust hash function [Roy22]. The rest of this section realizes the tensor-triple component in two stages: a generalization of the half tensor gate from Section 4 yields a

leaky variant $\mathcal{F}_{\mathsf{LeakyTensor}}$ that allows a corrupt party a bounded selective-failure attack on an honest party's input bits (Section C.2). Then, randomized bucketing in the style of [NNOB12, WRK17] (with optimizations catered to the tensor setting for smaller bucket size) eliminates the leakage (Sections C.3 and C.4).

## C.1 Generalized Half-Tensor Gate

The generalized macros are identical to Construction 1 except that an arbitrary XOR-shared vector $[\boldsymbol{B}]$ for $\boldsymbol{B} \in (\{0,1\}^\kappa)^m$ replaces the structured sharing $[\boldsymbol{b}\Delta_\mathsf{G}]$ as the second input. Step 6 of $\mathsf{Tensor.Gb}$ XORs $B_k$ into the leaf ciphertext $G_k$, step 5 of $\mathsf{Tensor.Ev}$ XORs the evaluator's share of $[\boldsymbol{B}]$ into the recovered active leaf, and no other step changes. Correctness follows as in Lemma 5: the parties obtain XOR shares of $\boldsymbol{a} \otimes \boldsymbol{B}$, at no change in cost. This generalizes the extended half-gate used in [KRRW18]'s preprocessing.

**Construction 4: Generalized Tensor macros**

$\mathsf{Tensor.Gb}(n, m, \Delta_\mathsf{G}, [\boldsymbol{a}\Delta_\mathsf{G}]^\mathsf{G}, [\boldsymbol{B}]^\mathsf{G})$:

1. Interpret $[\boldsymbol{a}\Delta_\mathsf{G}]^\mathsf{G} = A_0 A_1 \cdots A_{n-1}$ as $n$ labels of length $\kappa$.
2. Initialize the level-0 width-2 encoding: $S_{0,0} := A_0 \oplus \Delta_\mathsf{G}$, $S_{0,1} := A_0$.
3. For $i = 1, 2, \ldots, n-1$:
   (a) Compute $R_{i,j} := \mathsf{H}(S_{i-1,j}, \nu_{i,j})$ for each $j \in [2^i]$, where $\nu_{i,j}$ is a one-time nonce.
   (b) Define level-$i$ seeds: for $j \in [2^i]$,
   $$S_{i,j} := R_{i,j} \oplus S_{i-1,j}, \qquad S_{i,2^i+j} := R_{i,j}.$$
   (c) Set $G_i := \left(\bigoplus_{j\in[2^i]} R_{i,j}\right) \oplus A_i$.
4. Set $\boldsymbol{G} := \{G_i\}_{i=1}^{n-1}$.
5. Let $\mathsf{L}_\ell := S_{n-1,\ell}$ for $\ell \in [2^n]$. Use $\mathsf{H}$ with one-time nonces to expand each $\mathsf{L}_\ell$ into $m$ $\kappa$-bit seeds $X_{\ell,k}$ for $k \in [m]$.
6. Interpret $[\boldsymbol{B}]^\mathsf{G} = B_0 B_1 \cdots B_{m-1}$. For each $k \in [m]$, define
   $$G_k := \left(\bigoplus_{\ell=0}^{2^n-1} X_{\ell,k}\right) \oplus B_k$$
   and append $\boldsymbol{G} := \boldsymbol{G} \,\|\, \{G_k\}_{k=0}^{m-1}$.
7. Set $\boldsymbol{X}_\mathsf{G} := (X_{\ell,k})_{\ell\in[2^n],k\in[m]}$, let id denote the identity on $\{0,1\}^n$, and output
   $$\boldsymbol{Z}_\mathsf{G} := \mathcal{T}(\mathrm{id})^\intercal \cdot \boldsymbol{X}_\mathsf{G}, \quad \boldsymbol{G}.$$

$\mathsf{Tensor.Ev}(n, m, \boldsymbol{a}, \boldsymbol{G}, [\boldsymbol{a}\Delta_\mathsf{G}]^\mathsf{E}, [\boldsymbol{B}]^\mathsf{E})$:

1. Parse $\boldsymbol{G} = \{G_i\}_{i=1}^{n-1} \,\|\, \{G_k\}_{k=0}^{m-1}$ and $[\boldsymbol{a}\Delta_\mathsf{G}]^\mathsf{E} = A_0 \oplus a_0\Delta_\mathsf{G}, \ldots, A_{n-1} \oplus a_{n-1}\Delta_\mathsf{G}$.
2. Set $S^\mathsf{E}_{0,0} := S^\mathsf{E}_{0,1} := A_0 \oplus a_0\Delta_\mathsf{G}$ and $\alpha_0 := a_0$.
3. For $i = 1, 2, \ldots, n-1$:
   3.1. Compute $R^\mathsf{E}_{i,j} := \mathsf{H}(S^\mathsf{E}_{i-1,j}, \nu_{i,j})$ for each $j \in [2^i] \setminus \{\alpha_{i-1}\}$.
   3.2. Recover the active seed from $G_i$:
   $$R^\mathsf{E}_{i,\alpha_{i-1}} := G_i \oplus \Big(\bigoplus_{j\neq\alpha_{i-1}} R^\mathsf{E}_{i,j}\Big) \oplus (A_i \oplus a_i\Delta_\mathsf{G}).$$
   3.3. Compute the level-$i$ seeds: for each $j \in [2^i]$,
   $$S^\mathsf{E}_{i,j} := R^\mathsf{E}_{i,j} \oplus S^\mathsf{E}_{i-1,j}, \qquad S^\mathsf{E}_{i,2^i+j} := R^\mathsf{E}_{i,j}.$$
   3.4. Set $\alpha_i := \alpha_{i-1} + a_i \cdot 2^i$.
4. Let $\mathsf{L}^\mathsf{E}_\ell := S^\mathsf{E}_{n-1,\ell}$ for $\ell \in [2^n]$. Using $\mathsf{H}$ and one-time nonces expand $\mathsf{L}^\mathsf{E}_\ell$ for $\ell \neq \alpha_{n-1}$ into $X^\mathsf{E}_{\ell,k}$ for $k \in [m]$.

5. Parse $[\boldsymbol{B}]^{\mathsf{E}} = B_0 \oplus b_0\Delta, \ldots, B_{m-1} \oplus b_{m-1}\Delta$. For each $k \in [m]$, set

$$X^{\mathsf{E}}_{\alpha_{n-1},k} := \Big( \bigoplus_{\ell \neq \alpha_{n-1}} X^{\mathsf{E}}_{\ell,k} \Big) \oplus G_k \oplus (B_k \oplus b_k\Delta).$$

6. Set $\boldsymbol{X}_{\mathsf{E}} := (X^{\mathsf{E}}_{\ell,k})_{\ell \in [2^n], k \in [m]}$ and output

$$\boldsymbol{Z}_{\mathsf{E}} := \mathcal{T}(\mathrm{id})^{\intercal} \cdot \boldsymbol{X}_{\mathsf{E}}.$$

## C.2 Leaky Tensor Triples

Following [KRRW18], we use the generalized tensor macro to instantiate a *leaky* tensor functionality $\mathcal{F}_{\mathsf{LeakyTensor}}$ that permits a selective-failure attack on an honest party's input bits. Interestingly, the improved one-hot construction leaks less than an instantiation from half-gates [KRRW18] or from [HK21b]'s original construction: instead of one independent bit guess per level, the adversary gets a single *prefix-equality query* across all levels.

**Functionality 4:** $\mathcal{F}_{\mathsf{LeakyTensor}}$

Parameterized by $n, m \in \mathbb{N}$. Runs with parties $\mathsf{G}, \mathsf{E}$ and ideal-world adversary $\mathcal{S}$.

**Initialize.** As in $\mathcal{F}_{\mathsf{Pre}}$ (Functionality 1).

**Triple generation.** Sample $\boldsymbol{x} \leftarrow \{0,1\}^n$, $\boldsymbol{y} \leftarrow \{0,1\}^m$ uniformly and set $\boldsymbol{Z} := \boldsymbol{x} \otimes \boldsymbol{y}$. Output $[\![\boldsymbol{x}]\!], [\![\boldsymbol{y}]\!], [\![\boldsymbol{Z}]\!]$ to $\mathsf{G}$ and $\mathsf{E}$.

**Tree query.** $\mathsf{G}$ chooses a level $j \in \{1, \ldots, n-1\}$ and a guess $i^* \in \{0,1\}^j$ for the first $j$ bits of $[\boldsymbol{x}]^{\mathsf{E}}$. If $([x_1]^{\mathsf{E}}, \ldots, [x_j]^{\mathsf{E}}) \neq i^*$, send **fail** and abort. The query may be omitted.

**Leaf query.** $\mathsf{G}$ sends $\boldsymbol{P}_1 \in (\{0,1\}^\kappa)^m$, $\boldsymbol{p}_2 \in \{0,1\}^{n \times m}$, $\boldsymbol{P}_3 \in (\{0,1\}^\kappa)^{n \times m}$. If

$$\boldsymbol{P}_3 \oplus [\boldsymbol{x}]^{\mathsf{E}} \otimes \boldsymbol{P}_1 \;\neq\; \big(\boldsymbol{p}_2 \oplus [\boldsymbol{x}]^{\mathsf{E}} \otimes \mathsf{lsb}(\boldsymbol{P}_1)\big)\Delta_{\mathsf{E}},$$

send **fail** and abort.

($\mathsf{E}$ makes the same queries symmetrically on $[\boldsymbol{x}]^{\mathsf{G}}$.)

**Corrupted parties.** A corrupted party specifies the randomness used on its behalf.

The two queries capture every selective-failure attack a malicious $\mathsf{G}$ can launch in $\Pi_{\mathsf{LeakyTensor}}$. A tree corruption at level $j$ replaces the hash of the zero-label at some position $i^*$ with the hash of $X_{i^*,j} \oplus \Delta_{\mathsf{G}}$. Observe that this is exactly the ciphertext an honest garbler would produce after relabeling the tree so that the zero-label at position $i^*$ is $X_{i^*,j} \oplus \Delta_{\mathsf{G}}$. Recall that $\mathsf{E}$ holds the garbler's zero-labels at all inactive positions and $X_{a_j,j} \oplus \Delta_{\mathsf{G}}$ at the active position $a_j$, and that $\mathsf{E}$ never hashes its active label, recovering the active hash from the ciphertext instead. If $a_j = i^*$, then $\mathsf{E}$'s label at $i^*$ is precisely the relabeled zero-label, so $\mathsf{E}$'s view from level $j$ onward is its view of an honest garbling of the relabeled tree. The injected shift is absorbed by the relabeling, every seed $\mathsf{E}$ derives is the one this honest garbling would produce, and evaluation proceeds correctly. In effect the adversary has swapped the zero-label and one-label at the one-hot location and garbled honestly from there, so the execution is retroactively consistent and nothing detects the corruption. If $a_j \neq i^*$, then $\mathsf{E}$ hashes the unshifted label $X_{i^*,j}$, its recovered active hash is offset by $\mathsf{H}(X_{i^*,j} \oplus \Delta_{\mathsf{G}}) \oplus \mathsf{H}(X_{i^*,j})$, and every label derived below level $j$ carries an error that is unpredictable to $\mathsf{G}$. The equality check then fails unless $\mathsf{G}$ guesses $\Delta_{\mathsf{E}}$, which succeeds with probability $2^{-\rho}$. The corruption is therefore undetected iff $a_j = i^*$, equivalently iff the first $j$ bits of $[\boldsymbol{x}]^{\mathsf{E}}$ equal $i^*$, which is the tree query. Any ciphertext error not of the form $\mathsf{H}(X_{i^*,j} \oplus \Delta_{\mathsf{G}}) \oplus \mathsf{H}(X_{i^*,j})$ produces an unpredictable error on every input and aborts except with negligible probability. It does not help an adversary to make multiple prefix guesses as they either collapse to the longest prefix or are jointly unsatisfiable and force an abort.

The leaf query is the tensor generalization of the leaky AND query of [KRRW18], capturing the remaining selective failure on $[x_0]^{\mathsf{E}}$: by choosing $\boldsymbol{P}_1, \boldsymbol{p}_2, \boldsymbol{P}_3$ appropriately, $\mathsf{G}$ can introduce an error that passes the check iff its implicit guess for $[\boldsymbol{x}]^{\mathsf{E}}$ (or $\Delta_{\mathsf{E}}$) is correct.

Note that an honest $\mathsf{E}$'s share $[\boldsymbol{x}]^{\mathsf{E}}$ is uniformly random from $\mathsf{G}$'s perspective, so a length-$j$ prefix-equality query passes with probability $2^{-j}$. For our bucketing analysis we bound error against a strictly stronger functionality $\mathcal{F}'_{\mathsf{LeakyTensor}}$ permitting independent probability-$\frac{1}{2}$ guesses per coordinate, since it is simpler to

analyze; security against $\mathcal{F}_{\mathsf{LeakyTensor}}$ follows.

We now describe a protocol for generating leaky tensor triples, following the blueprint of [KRRW18]. Throughout, we abbreviate $\boldsymbol{x}_{\mathsf{G}} := [\boldsymbol{x}]^{\mathsf{G}}$, $\boldsymbol{x}_{\mathsf{E}} := [\boldsymbol{x}]^{\mathsf{E}}$, and similarly for $\boldsymbol{y}, \boldsymbol{R}$; individual bit coordinates of $\mathsf{E}$'s share are written $[x_i]^{\mathsf{E}}$.

**Blueprint.** Parties obtain from $\mathcal{F}_{\mathrm{COT}}$ uniform correlations $[\boldsymbol{x}_{\mathsf{G}}\Delta_{\mathsf{E}}]$, $[\boldsymbol{x}_{\mathsf{E}}\Delta_{\mathsf{G}}]$, $[\boldsymbol{y}_{\mathsf{G}}\Delta_{\mathsf{E}}]$, $[\boldsymbol{y}_{\mathsf{E}}\Delta_{\mathsf{G}}]$, $[\![\boldsymbol{R}]\!]$, where $\boldsymbol{R} \in \{0,1\}^{n\times m}$ is an authenticated mask. Parties can use their own shares and global keys to locally obtain correlations $[\![\boldsymbol{x}]\!]$ and $[\![\boldsymbol{y}]\!]$ as well as an XOR-sharing $[\boldsymbol{C}]$ of $\boldsymbol{C} := \boldsymbol{y}(\Delta_{\mathsf{G}} \oplus \Delta_{\mathsf{E}}) \in (\{0,1\}^{\kappa})^m$.

If we reveal $\boldsymbol{D} := \boldsymbol{x} \otimes \boldsymbol{y} \oplus \boldsymbol{R}$, parties can correct shares of the random mask $\boldsymbol{R}$ to the tensor output $\boldsymbol{Z}$: $[\![\boldsymbol{Z}]\!] = [\![\boldsymbol{R}]\!] \oplus [\![\boldsymbol{D}]\!]$. Revealing $\boldsymbol{D}$ costs $nm$ bits. As an optimization, we sample $\Delta_{\mathsf{G}}, \Delta_{\mathsf{E}}$ conditioned on $\mathsf{lsb}(\Delta_{\mathsf{G}} \oplus \Delta_{\mathsf{E}}) = 1$, so that parties extract $\boldsymbol{D}$ from an XOR-sharing of $(\boldsymbol{x}\otimes\boldsymbol{y}\oplus\boldsymbol{R})(\Delta_{\mathsf{G}}\oplus\Delta_{\mathsf{E}})$ by coordinate-wise $\mathsf{lsb}$.

The core task is therefore to obtain $[\boldsymbol{x} \otimes \boldsymbol{y}(\Delta_{\mathsf{G}} \oplus \Delta_{\mathsf{E}})]$. The generalized tensor macro with $\mathsf{G}$ as garbler, on $[\boldsymbol{x}_{\mathsf{E}}\Delta_{\mathsf{G}}]$ and $[\boldsymbol{C}]$, yields $[\boldsymbol{x}_{\mathsf{E}} \otimes \boldsymbol{y}(\Delta_{\mathsf{G}} \oplus \Delta_{\mathsf{E}})]$, and the same macro with $\mathsf{E}$ as garbler, on $[\boldsymbol{x}_{\mathsf{G}}\Delta_{\mathsf{E}}]$ and $[\boldsymbol{C}]$, yields $[\boldsymbol{x}_{\mathsf{G}} \otimes \boldsymbol{y}(\Delta_{\mathsf{G}} \oplus \Delta_{\mathsf{E}})]$. XORing the two gives $[\boldsymbol{x} \otimes \boldsymbol{y}(\Delta_{\mathsf{G}} \oplus \Delta_{\mathsf{E}})]$. We formalize this below.

**Construction 5: Protocol $\Pi_{\mathsf{LeakyTensor}}$**

**Correlated randomness.** $\mathsf{G}, \mathsf{E}$ sample $\Delta_{\mathsf{G}}, \Delta_{\mathsf{E}}$ with $\mathsf{lsb}(\Delta_{\mathsf{G}} \oplus \Delta_{\mathsf{E}}) = 1$ and obtain $[\boldsymbol{x}_{\mathsf{G}}\Delta_{\mathsf{E}}], [\boldsymbol{x}_{\mathsf{E}}\Delta_{\mathsf{G}}], [\boldsymbol{y}_{\mathsf{G}}\Delta_{\mathsf{E}}], [\boldsymbol{y}_{\mathsf{E}}\Delta_{\mathsf{G}}], [\![\boldsymbol{R}]\!]$ from $\mathcal{F}_{\mathrm{COT}}$. $\mathsf{G}$ computes

$$[\boldsymbol{C}]^{\mathsf{G}} := \boldsymbol{y}_{\mathsf{G}}\Delta_{\mathsf{G}} \oplus [\boldsymbol{y}_{\mathsf{E}}\Delta_{\mathsf{G}}]^{\mathsf{G}} \oplus [\boldsymbol{y}_{\mathsf{G}}\Delta_{\mathsf{E}}]^{\mathsf{G}},$$
$$[\boldsymbol{C}^{(\boldsymbol{R})}]^{\mathsf{G}} := \boldsymbol{R}_{\mathsf{G}}\Delta_{\mathsf{G}} \oplus [\boldsymbol{R}_{\mathsf{E}}\Delta_{\mathsf{G}}]^{\mathsf{G}} \oplus [\boldsymbol{R}_{\mathsf{G}}\Delta_{\mathsf{E}}]^{\mathsf{G}},$$

and $\mathsf{E}$ analogously computes $[\boldsymbol{C}]^{\mathsf{E}}, [\boldsymbol{C}^{(\boldsymbol{R})}]^{\mathsf{E}}$.

**Tensor macro.**

1. $\mathsf{G}$ computes $(\boldsymbol{Z}_{\mathsf{G},1}, \boldsymbol{G}_1) := \mathsf{Tensor.Gb}(n, m, \Delta_{\mathsf{G}}, [\boldsymbol{x}_{\mathsf{E}}\Delta_{\mathsf{G}}]^{\mathsf{G}}, [\boldsymbol{C}]^{\mathsf{G}})$ and sends $\boldsymbol{G}_1$ to $\mathsf{E}$, who computes $\boldsymbol{E}_1 := \mathsf{Tensor.Ev}(n, m, \boldsymbol{x}_{\mathsf{E}}, \boldsymbol{G}_1, [\boldsymbol{x}_{\mathsf{E}}\Delta_{\mathsf{G}}]^{\mathsf{E}}, [\boldsymbol{C}]^{\mathsf{E}})$.
2. $\mathsf{E}$ computes $(\boldsymbol{Z}_{\mathsf{G},2}, \boldsymbol{G}_2) := \mathsf{Tensor.Gb}(n, m, \Delta_{\mathsf{E}}, [\boldsymbol{x}_{\mathsf{G}}\Delta_{\mathsf{E}}]^{\mathsf{E}}, [\boldsymbol{C}]^{\mathsf{E}})$ and sends $\boldsymbol{G}_2$ to $\mathsf{G}$, who computes $\boldsymbol{E}_2 := \mathsf{Tensor.Ev}(n, m, \boldsymbol{x}_{\mathsf{G}}, \boldsymbol{G}_2, [\boldsymbol{x}_{\mathsf{G}}\Delta_{\mathsf{E}}]^{\mathsf{G}}, [\boldsymbol{C}]^{\mathsf{G}})$.

**Reveal.** $\mathsf{G}$ computes $\boldsymbol{S}_1 := \boldsymbol{Z}_{\mathsf{G},1} \oplus \boldsymbol{E}_2 \oplus [\boldsymbol{C}^{(\boldsymbol{R})}]^{\mathsf{G}}$ and sends $\mathsf{lsb}(\boldsymbol{S}_1)$ to $\mathsf{E}$; $\mathsf{E}$ computes $\boldsymbol{S}_2 := \boldsymbol{Z}_{\mathsf{G},2} \oplus \boldsymbol{E}_1 \oplus [\boldsymbol{C}^{(\boldsymbol{R})}]^{\mathsf{E}}$ and sends $\mathsf{lsb}(\boldsymbol{S}_2)$ to $\mathsf{G}$. Both compute $\boldsymbol{D} := \mathsf{lsb}(\boldsymbol{S}_1) \oplus \mathsf{lsb}(\boldsymbol{S}_2) \in \{0,1\}^{n\times m}$.

**Consistency check.** $\mathsf{G}$ sets $\boldsymbol{L}_1 := \boldsymbol{S}_1 \oplus \boldsymbol{D}\Delta_{\mathsf{G}}$, $\mathsf{E}$ sets $\boldsymbol{L}_2 := \boldsymbol{S}_2 \oplus \boldsymbol{D}\Delta_{\mathsf{E}}$, and they perform an equality check sending $\boldsymbol{L}_1$ and $\boldsymbol{L}_2$ to $\mathcal{F}_{\mathsf{EQ}}$ respectively. If it fails, abort; otherwise, both output $[\![\boldsymbol{Z}]\!] := [\![\boldsymbol{R}]\!] \oplus [\![\boldsymbol{D}]\!]$.

**Theorem 12.** $\Pi_{\mathsf{LeakyTensor}}$ *correctly realizes* $\mathcal{F}_{\mathsf{LeakyTensor}}$ *when both parties are honest.*

*Proof.* Observe that

$$[\boldsymbol{C}]^{\mathsf{G}} \oplus [\boldsymbol{C}]^{\mathsf{E}} = \boldsymbol{y}_{\mathsf{G}}\Delta_{\mathsf{G}} \oplus \boldsymbol{y}_{\mathsf{E}}\Delta_{\mathsf{E}} \oplus \boldsymbol{y}_{\mathsf{E}}\Delta_{\mathsf{G}} \oplus \boldsymbol{y}_{\mathsf{G}}\Delta_{\mathsf{E}} = \boldsymbol{y}(\Delta_{\mathsf{G}} \oplus \Delta_{\mathsf{E}}).$$

The same argument gives $[\boldsymbol{C}^{(\boldsymbol{R})}]^{\mathsf{G}} \oplus [\boldsymbol{C}^{(\boldsymbol{R})}]^{\mathsf{E}} = \boldsymbol{R}(\Delta_{\mathsf{G}} \oplus \Delta_{\mathsf{E}})$.

By correctness of the generalized tensor macro,

$$\boldsymbol{Z}_{\mathsf{G},1} \oplus \boldsymbol{E}_1 = \boldsymbol{x}_{\mathsf{E}} \otimes \boldsymbol{y}(\Delta_{\mathsf{G}} \oplus \Delta_{\mathsf{E}}), \qquad \boldsymbol{Z}_{\mathsf{G},2} \oplus \boldsymbol{E}_2 = \boldsymbol{x}_{\mathsf{G}} \otimes \boldsymbol{y}(\Delta_{\mathsf{G}} \oplus \Delta_{\mathsf{E}}).$$

Then

$$\begin{aligned}\boldsymbol{S}_1 \oplus \boldsymbol{S}_2 &= (\boldsymbol{Z}_{\mathsf{G},1} \oplus \boldsymbol{E}_1) \oplus (\boldsymbol{Z}_{\mathsf{G},2} \oplus \boldsymbol{E}_2) \oplus ([\boldsymbol{C}^{(\boldsymbol{R})}]^{\mathsf{G}} \oplus [\boldsymbol{C}^{(\boldsymbol{R})}]^{\mathsf{E}}) \\ &= (\boldsymbol{x}_{\mathsf{G}} \oplus \boldsymbol{x}_{\mathsf{E}}) \otimes \boldsymbol{y}(\Delta_{\mathsf{G}} \oplus \Delta_{\mathsf{E}}) \oplus \boldsymbol{R}(\Delta_{\mathsf{G}} \oplus \Delta_{\mathsf{E}}) \\ &= (\boldsymbol{x} \otimes \boldsymbol{y} \oplus \boldsymbol{R})(\Delta_{\mathsf{G}} \oplus \Delta_{\mathsf{E}}).\end{aligned}$$

Since $\mathsf{lsb}(\Delta_{\mathsf{G}} \oplus \Delta_{\mathsf{E}}) = 1$, $\boldsymbol{D} = \mathsf{lsb}(\boldsymbol{S}_1) \oplus \mathsf{lsb}(\boldsymbol{S}_2) = \mathsf{lsb}(\boldsymbol{S}_1 \oplus \boldsymbol{S}_2) = \boldsymbol{x} \otimes \boldsymbol{y} \oplus \boldsymbol{R}$, so $[\![\boldsymbol{Z}]\!] = [\![\boldsymbol{R}]\!] \oplus [\![\boldsymbol{D}]\!]$ satisfies $\boldsymbol{Z} = \boldsymbol{x} \otimes \boldsymbol{y}$. □

**Theorem 13.** $\Pi_{\mathsf{LeakyTensor}}$ *securely realizes* $\mathcal{F}_{\mathsf{LeakyTensor}}$ *in the* $(\mathcal{F}_{\mathrm{COT}}, \mathcal{F}_{\mathsf{EQ}})$*-hybrid model against malicious adversaries assuming the existence of a circular correlation robust hash function.*

*Proof.* The protocol is symmetric; we handle corrupted $\mathsf{G}$. All inputs come from $\mathcal{F}_{\mathrm{COT}}$ and local computations are deterministic, so $\mathcal{S}$ can emulate every honest message after preprocessing.

**Simulator $\mathcal{S}$ for malicious $\mathsf{G}$.**
*Setup.* $\mathcal{S}$ queries $\mathcal{F}_{\mathsf{LeakyTensor}}$ to obtain $\mathsf{G}$'s shares, including $\Delta_{\mathsf{G}}$, and uses them to emulate an honest $\mathsf{G}$'s behavior. $\mathcal{S}$ samples $\mathsf{E}$'s preprocessing $(\Delta_{\mathsf{E}}, \boldsymbol{x}_{\mathsf{E}}, \boldsymbol{y}_{\mathsf{E}}, \boldsymbol{R}_{\mathsf{E}})$ uniformly conditioned on $\mathsf{lsb}(\Delta_{\mathsf{G}} \oplus \Delta_{\mathsf{E}}) = 1$, and computes the honest messages $(\boldsymbol{G}_1, \boldsymbol{G}_2, \boldsymbol{S}_1, \boldsymbol{S}_2, \boldsymbol{L}_1, \boldsymbol{L}_2)$.
*Step 1.* $\mathcal{S}$ plays $\mathcal{F}_{\mathrm{COT}}$ and distributes preprocessing.
*Step 2.* $\mathcal{S}$ acts as honest $\mathsf{E}$ and receives $\boldsymbol{G}'_1$ from $\mathcal{A}$. For each level $i \in \{1, \ldots, n-1\}$, $\mathcal{S}$ compares the level-$i$ ciphertext $G'_{i-1}$ with the honest value $G_{i-1}$. Let $\delta_i := G'_{i-1} \oplus G_{i-1}$.

If every $\delta_i = 0$, no tree query is submitted. Otherwise let $j$ be the smallest level with $\delta_j \neq 0$. $\mathcal{S}$ checks whether there exists $i^* \in \{0,1\}^j$ such that $\delta_j = \mathsf{H}(X_{i^*,j} \oplus \Delta_{\mathsf{G}}) \oplus \mathsf{H}(X_{i^*,j})$, where $\{X_{i,j}\}_{i \in [2^j]}$ are the level-$j$ zero-labels derived from $\mathsf{G}$'s preprocessing. If no such $i^*$ exists, or if any $\delta_\ell$ for $\ell > j$ is inconsistent with the same guess $i^*$ propagated forward, $\mathcal{S}$ sends **abort** to $\mathcal{F}_{\mathsf{LeakyTensor}}$ and stops. Otherwise $\mathcal{S}$ submits $(j, i^*)$ as the tree query. If $\mathcal{F}_{\mathsf{LeakyTensor}}$ returns **fail**, $\mathcal{S}$ outputs 0 and stops.
*Step 3.* $\mathcal{S}$ sends $\boldsymbol{G}_2$ to $\mathcal{A}$.
*Step 4.* $\mathcal{S}$ sends $\mathsf{lsb}(\boldsymbol{S}_2)$ to $\mathcal{A}$ and receives $\boldsymbol{d}'_1$.
*Step 5.* $\mathcal{S}$ simulates $\mathcal{F}_{\mathsf{EQ}}$: on receiving $\boldsymbol{L}'_1$ from an adversary, computes $\boldsymbol{P}_1 := \boldsymbol{G}'_1 \oplus \boldsymbol{G}_1$, $\boldsymbol{p}_2 := \boldsymbol{d}'_1 \oplus \boldsymbol{d}_1$, $\boldsymbol{P}_3 := \boldsymbol{L}'_1 \oplus \boldsymbol{L}_1$, and submits $(\boldsymbol{P}_1, \boldsymbol{p}_2, \boldsymbol{P}_3)$ as the leaf query. $\mathcal{S}$ outputs $[\boldsymbol{L}'_1 = \boldsymbol{L}'_2]$ to the adversary, where $\boldsymbol{L}'_2 := \boldsymbol{S}_2 \oplus (\boldsymbol{d}'_1 \oplus \mathsf{lsb}(\boldsymbol{S}_2))\Delta_{\mathsf{E}}$.

**Correctness of simulation.**
*Tree selective failure.* At level $j$, $\mathsf{E}$ recovers the active label by hashing every position $i \neq a_j$ where $a_j$ is the active position. If $\mathsf{G}$'s ciphertext is honest at every level, $\mathsf{E}$ obtains correct output shares. If $\mathsf{G}$'s level-$j$ ciphertext shifts the zero-label at position $i^*$ by $\Delta_{\mathsf{G}}$, two cases: (i) $a_j = i^*$, in which case $\mathsf{E}$'s view at $i^*$ is $X_{i^*,j} \oplus \Delta_{\mathsf{G}}$, the shift coincides with the relabeling, and evaluation proceeds correctly; (ii) $a_j \neq i^*$, in which case $\mathsf{E}$ hashes the unshifted $X_{i^*,j}$, the level-$j$ active label shifts by $\mathsf{H}(X_{i^*,j} \oplus \Delta_{\mathsf{G}}) \oplus \mathsf{H}(X_{i^*,j})$ and the equality check fails with probability $1 - 2^{-\rho}$ as the only way for this check to pass is to implicitly guess $\mathsf{E}$'s global key $\Delta_{\mathsf{E}}$. This matches $\mathcal{F}_{\mathsf{LeakyTensor}}$'s tree-query failure on $a_j \neq i^*$, equivalently on $([x_1]^{\mathsf{E}}, \ldots, [x_j]^{\mathsf{E}}) \neq i^*$. Any $\delta_j$ not of the form $\mathsf{H}(X_{i^*,j} \oplus \Delta_{\mathsf{G}}) \oplus \mathsf{H}(X_{i^*,j})$ for some $i^*$ independently causes an abort with probability $1 - 2^{-\rho}$ by the same argument as above; $\mathcal{S}$ sends **abort** unconditionally and the equality check fails with negligible probability.
*Leaf query.* By construction of $\mathsf{Tensor.Ev}$, shifting $G_k$ by $P_{1,k}$ shifts $E_{1,(i,k)}$ by $[x_i]^{\mathsf{E}} \cdot P_{1,k}$, so $\boldsymbol{S}'_2 = \boldsymbol{S}_2 \oplus [\boldsymbol{x}]^{\mathsf{E}} \otimes \boldsymbol{P}_1$. The condition $\boldsymbol{L}'_1 = \boldsymbol{L}'_2$ expands to

$$\boldsymbol{P}_3 \oplus [\boldsymbol{x}]^{\mathsf{E}} \otimes \boldsymbol{P}_1 = \big(\boldsymbol{p}_2 \oplus [\boldsymbol{x}]^{\mathsf{E}} \otimes \mathsf{lsb}(\boldsymbol{P}_1)\big)\Delta_{\mathsf{E}},$$

which is the leaf-query check in $\mathcal{F}_{\mathsf{LeakyTensor}}$. □

## C.3 Combining Leaky Tensor Triples

We combine two leaky tensor triples into a less leaky one, following [WRK17] for leaky ANDs. Guessing a bit of the combined triple requires guessing both constituent bits, succeeding with probability $\leq 1/4$. Given $(\llbracket \boldsymbol{x}' \rrbracket, \llbracket \boldsymbol{y}' \rrbracket, \llbracket \boldsymbol{Z}' \rrbracket)$ and $(\llbracket \boldsymbol{x}'' \rrbracket, \llbracket \boldsymbol{y}'' \rrbracket, \llbracket \boldsymbol{Z}'' \rrbracket)$, set $\llbracket \boldsymbol{x} \rrbracket := \llbracket \boldsymbol{x}' \rrbracket \oplus \llbracket \boldsymbol{x}'' \rrbracket$, $\llbracket \boldsymbol{y} \rrbracket := \llbracket \boldsymbol{y}' \rrbracket$. Parties publicly reveal $\boldsymbol{d} := \boldsymbol{y}' \oplus \boldsymbol{y}'' \in \{0,1\}^m$ (simulatable as it is uniform and independent of $\boldsymbol{y} = \boldsymbol{y}'$). Then

$$\boldsymbol{x} \otimes \boldsymbol{y} = (\boldsymbol{x}' \oplus \boldsymbol{x}'') \otimes \boldsymbol{y}' = \boldsymbol{Z}' \oplus \boldsymbol{Z}'' \oplus \boldsymbol{x}'' \otimes \boldsymbol{d},$$

and since $\boldsymbol{d}$ is public, $\llbracket \boldsymbol{x}'' \rrbracket \otimes \boldsymbol{d}$ is local (scale each row of $\llbracket \boldsymbol{x}'' \rrbracket$ by the corresponding $d_k$). Parties set

$$\llbracket \boldsymbol{Z} \rrbracket := \llbracket \boldsymbol{Z}' \rrbracket \oplus \llbracket \boldsymbol{Z}'' \rrbracket \oplus \llbracket \boldsymbol{x}'' \rrbracket \otimes \boldsymbol{d}.$$

Iterating over $\rho$ triples drives guessing to $2^{-\rho}$, at a $\rho$-fold blowup, which the next section amortizes via random bucketing and permutations.

## C.4 Randomized Bucketing

We now amortize this overhead (following [NNOB12, WRK17]) by generating $\ell B$ leaky triples at once and randomly assigning them to $\ell$ buckets of size $B$, chosen below. The adversary does not know which bucket a leaky triple lands in, so it must guess many triples or get lucky in the assignment. We also apply an independent uniform row-permutation $\pi_j \in S_n$ to $[\boldsymbol{x}^{(j)}]^{\mathsf{G}}, [\boldsymbol{x}^{(j)}]^{\mathsf{E}}, [\![\boldsymbol{Z}^{(j)}]\!]$ (with $[\![\boldsymbol{y}^{(j)}]\!]$ unchanged) before combining triples within a bucket, so the adversary no longer controls which coordinate of the combined triple its corruption lands on.

**Construction 6: Protocol $\Pi_{\mathsf{Pre}}$**

**Inputs.** Authenticated-bit count $N \in \mathbb{N}$ and tensor-triple dimensions $\{(n_t, m_t)\}_{t\in[T]}$. Set $n^* := \max_t n_t$, $m^* := \max_t m_t$, $\ell := T$, and $B := 1 + \lceil \rho / \log(n^*\ell) \rceil$.

1. Parties sample $\Delta_{\mathsf{G}}, \Delta_{\mathsf{E}} \leftarrow \mathbb{F}_{2^\kappa}$.
2. Parties produce $[\![\boldsymbol{r}]\!]$ for $\boldsymbol{r} \leftarrow \{0,1\}^N$ by standard OT-based authenticated-bit generation [NNOB12, WRK17].
3. Parties invoke $\mathcal{F}_{\mathsf{LeakyTensor}}$ of dimension $(n^*, m^*)$ a total of $\ell B$ times, obtaining $\{([\![\boldsymbol{x}^{(j)}]\!], [\![\boldsymbol{y}^{(j)}]\!], [\![\boldsymbol{Z}^{(j)}]\!])\}_{j=1}^{\ell B}$.
4. Via shared randomness, parties sample a uniform partition of $[\ell B]$ into $\ell$ buckets $\mathcal{B}_1, \ldots, \mathcal{B}_\ell$ of size $B$, and independent uniform permutations $\pi_j \in S_{n^*}$. Each party applies $\pi_j$ locally to $[\![\boldsymbol{x}^{(j)}]\!]$ and the rows of $[\![\boldsymbol{Z}^{(j)}]\!]$.
5. For each $\mathcal{B}_k$, parties iterate the combining procedure of Section C.3 across the $B$ permuted triples to produce $([\![\boldsymbol{x}_k]\!], [\![\boldsymbol{y}_k]\!], [\![\boldsymbol{Z}_k]\!])$. For each $t \in [T]$, truncate the $t$-th combined triple to $(n_t, m_t)$ by discarding unused coordinates.

**Lemma 14.** *Fix $n, \ell \in \mathbb{N}$ and $B := 1 + \lceil \rho / \log(n\ell) \rceil$. Given $\ell B$ triples from $\mathcal{F}_{\mathsf{LeakyTensor}}$ of dimension $(n, m)$, the bucketing-and-permutation procedure outputs $\ell$ authenticated triples whose joint distribution is $2^{-\rho}$-close to $\ell$ independent uniform authenticated tensor triples of the same dimension.*

*Proof.* We work in the stronger $\mathcal{F}'_{\mathsf{LeakyTensor}}$, where the adversary guesses each coordinate $[x_i^{(j)}]^{\mathsf{E}}$ independently with detection probability $\frac{1}{2}$; security against $\mathcal{F}_{\mathsf{LeakyTensor}}$ follows. The adversary commits to its corruption set $T_j \subseteq [n]$ per triple before seeing the partition or permutations; write $c_j := |T_j|$ and $\gamma := \sum_j c_j$. All guesses go undetected with probability $2^{-\gamma}$.

**Corruption budget.** For each triple $j \in [\ell B]$, the adversary chooses a set $T_j \subseteq [n]$ of coordinates to guess, with $|T_j| = c_j$. Since each guess is independently caught with probability $\frac{1}{2}$, the probability that all guesses go undetected is $2^{-\gamma}$ where $\gamma = \sum_j c_j$.

**Leakiness under random permutation.** For a fixed coordinate $i^* \in [n]$, call the $j$-th triple *$i^*$-leaky* if $\pi_j^{-1}(i^*) \in T_j$, i.e. some guessed coordinate maps onto $i^*$ under the random permutation. Since $\pi_j$ is uniform and independent of $T_j$, this occurs with probability exactly $c_j/n$. Since the $\pi_j$'s are independent across triples, $i^*$-leakiness is independent across $j$ for any fixed $i^*$. The adversary learns coordinate $i^*$ of the combined triple in bucket $\mathcal{B}_k$ if and only if every triple $j \in \mathcal{B}_k$ is $i^*$-leaky.

**Bounding the success probability.** Fix any partition $\mathcal{B}_1, \ldots, \mathcal{B}_\ell$. The adversary wins if they learn any coordinate of any combined triple. Since events across different coordinates $i^*$ are no longer independent (the same permutations $\pi_j$ shuffle all coordinates simultaneously) we apply a union bound over all $n$ coordinates and all $\ell$ buckets. For fixed $\mathcal{B}_k$ and $i^*$, the probability over the permutations that every triple in $\mathcal{B}_k$ is $i^*$-leaky is $\prod_{j\in\mathcal{B}_k} c_j/n$ by independence of the $\pi_j$'s across triples. Therefore, for any fixed partition:

$$\Pr_{\{\pi_j\}}[\text{win} \mid \text{partition}] \;\le\; 2^{-\gamma} \cdot \sum_{i^*=1}^{n} \sum_{k=1}^{\ell} \prod_{j\in\mathcal{B}_k} \frac{c_j}{n}.$$

Since this holds for every fixed partition, it holds in expectation over the random partition:

$$\Pr[\text{win}] \;\le\; 2^{-\gamma} \cdot \mathbb{E}_{\text{partition}} \left[ \sum_{i^*=1}^{n} \sum_{k=1}^{\ell} \prod_{j\in\mathcal{B}_k} \frac{c_j}{n} \right].$$

First note we can also pull out the $n^{-B}$ factor. Also, since the summand does not depend on $i^*$, the sum over coordinates contributes a factor of $n$. Using symmetry of the random partition, each bucket contributes equally in expectation so $\mathbb{E}[\sum_k \prod_{j\in\mathcal{B}_k} c_j] = \ell \cdot \mathbb{E}[\prod_{j\in\mathcal{B}_1} c_j]$ which gives:

$$\Pr[\text{win}] \;\le\; \frac{n\ell \cdot 2^{-\gamma}}{n^B} \cdot \mathbb{E}_{\text{partition}}\left[\prod_{j\in\mathcal{B}_1} c_j\right].$$

**Bounding the expected product.** Since the random partition assigns a uniformly random size-$B$ subset of $\{c_j\}_{j=1}^{\ell B}$ to bucket $\mathcal{B}_1$, the expectation $\mathbb{E}_{\text{partition}}[\prod_{j\in\mathcal{B}_1} c_j]$ is exactly the average of $\prod_{j\in S} c_j$ over all size-$B$ subsets $S \subseteq [\ell B]$:

$$\mathbb{E}_{\text{partition}}\left[\prod_{j\in\mathcal{B}_1} c_j\right] \;=\; \frac{\displaystyle\sum_{S\subseteq[\ell B],\,|S|=B}\;\prod_{j\in S} c_j}{\binom{\ell B}{B}}.$$

By Maclaurin's inequality, this average of products of $B$ distinct elements is at most the $B$-th power of the arithmetic mean of all elements:

$$\frac{\displaystyle\sum_{S\subseteq[\ell B],\,|S|=B}\;\prod_{j\in S} c_j}{\binom{\ell B}{B}} \;\le\; \left(\frac{\sum_j c_j}{\ell B}\right)^B \;=\; \left(\frac{\gamma}{\ell B}\right)^B.$$

Substituting back:

$$\Pr[\text{win}] \;\le\; n\ell \cdot \left(\frac{\gamma}{n\ell B}\right)^B \cdot 2^{-\gamma}.$$

**Optimizing over $\gamma$.** The right hand side is maximized over $\gamma > 0$ at $\gamma^* = B/\ln 2$, obtained by differentiating the logarithm $B\ln\gamma - B\ln(n\ell B) - \gamma\ln 2$ and setting to zero. Substituting $\gamma^*$ and using $2^{-1/\ln 2} = e^{-1}$:

$$\Pr[\text{win}] \;\le\; n\ell \cdot \left(\frac{1}{n\ell\ln 2}\right)^B \cdot e^{-B} \;=\; \frac{n\ell}{(en\ell\ln 2)^B} \;\le\; (n\ell)^{1-B},$$

where the last step drops the constant $e\ln 2 > 1$ for simplicity. Setting $(n\ell)^{1-B} \le 2^{-\rho}$ rearranges to $B \ge 1 + \rho/\log(n\ell)$. Taking $B = 1 + \lceil \rho/\log(n\ell)\rceil$ therefore suffices. □

Without the permutations a larger bucket size $B \ge 1 + \rho/\log\ell$ is needed.

**Theorem 15.** *Assuming a circular-correlation-robust hash, $\Pi_{\mathsf{Pre}}$ securely realizes $\mathcal{F}_{\mathsf{Pre}}$ in the $(\mathcal{F}_{\text{COT}}, \mathcal{F}_{\mathsf{EQ}})$-hybrid model with statistical error $2^{-\rho}$, using the communication stated in Section 4.*

*Proof.* Follows straightforwardly by composition of Theorem 13 (leaky-triple generation) and Lemma 14 (bucketing removes leakage). □

# D Compressed Preprocessing For Protocol 2

We now describe the compressed preprocessing functionality $\mathcal{F}_{\text{cpre}}$ used by our second protocol (Section 5). [CWYY23] reduces the communication cost of generating AND triples to 4 bits per triple (amortized) by compressing one party's shares into a short seed, authenticating this seed, and then linearly expanding using a public random matrix. We instantiate $\mathcal{F}_{\text{cpre}}$ using $\Pi_{\text{cpre}}$ [CWYY23], whose security proof additionally assumes LPN for the VOLE PCG.

**Functionality 5: $\mathcal{F}_{\text{cpre}}$ – Compressed Preprocessing**

This functionality is parameterized by a count $N \in \mathbb{N}$ of authenticated bits, a list of tensor-triple dimensions $\{(n_t, m_t)\}_{t\in[T]} \subseteq \mathbb{N}^2$, and a seed length $L \in \mathbb{N}$. It runs with parties $\mathsf{G}$, $\mathsf{E}$ and the ideal-world adversary $\mathcal{S}$.

**Initialize.** Sample $\Delta_{\mathsf{G}}, \Delta_{\mathsf{E}} \leftarrow \mathbb{F}_{2^\kappa}$ and output $\Delta_{\mathsf{G}}$ to $\mathsf{G}$ and $\Delta_{\mathsf{E}}$ to $\mathsf{E}$. Use $[\![\cdot]\!]$ to denote $[(\cdot)(\Delta_{\mathsf{G}} \| \Delta_{\mathsf{E}} \| 1)]$.

**Authenticated bits.** Sample $\boldsymbol{r} \leftarrow \{0,1\}^N$ uniformly and output $[\![\boldsymbol{r}]\!]$ to $\mathsf{G}$ and $\mathsf{E}$.

**Authenticated tensor triples.** Let $\mathcal{L} := \sum_{t\in[T]}(n_t + m_t)$. Sample $\boldsymbol{M} \leftarrow \{0,1\}^{\mathcal{L}\times L}$ and output $\boldsymbol{M}$ to $\mathsf{G}$ and $\mathsf{E}$.

- If $\mathsf{G}$ is corrupted: sample $\boldsymbol{s} \leftarrow \{0,1\}^L$ and compute $\boldsymbol{u} := \boldsymbol{M}\boldsymbol{s} \in \{0,1\}^{\mathcal{L}}$.
- Otherwise: sample $\boldsymbol{u} \leftarrow \{0,1\}^{\mathcal{L}}$.

Parse $\boldsymbol{u}$ as $(\boldsymbol{x}^{(t)}, \boldsymbol{y}^{(t)})_{t\in[T]}$ with $\boldsymbol{x}^{(t)} \in \{0,1\}^{n_t}$ and $\boldsymbol{y}^{(t)} \in \{0,1\}^{m_t}$. For each $t \in [T]$: set $\boldsymbol{Z}^{(t)} := \boldsymbol{x}^{(t)} \otimes \boldsymbol{y}^{(t)}$; output $[\![\boldsymbol{x}^{(t)}]\!]$, $[\![\boldsymbol{y}^{(t)}]\!]$, $[\![\boldsymbol{Z}^{(t)}]\!]$ to $\mathsf{G}$ and $\mathsf{E}$.

**Corrupted parties.** If party $P \in \{\mathsf{G}, \mathsf{E}\}$ is corrupted, $\mathcal{S}$ specifies $\Delta_P$ and, for each output $[\![v]\!]$, the corrupted share; this determines the honest party's share.

Our presentation of the $\mathcal{F}_{\text{cpre}}$ functionality differs from [CWYY23] in an attempt to make it easier to understand. The original work parameterized this functionality with a Boolean circuit and generated circuit-dependent preprocessed correlations. Since our online phase consumes random authenticated bits and tensor triples directly, we can simplify our functionality so that it only requires an upper bound on each of these quantities. Here, we choose to generate tensor triples by decomposing a tensor gate into AND gates, leading to a communication cost of $4nm$ bits per tensor triple. We also express the functionality's outputs using the notion of dual-authenticated shares $[\![\cdot]\!]$ (Section 2.4), which we think significantly simplifies notation. Finally, the compression asymmetry, i.e., $\mathsf{G}$'s shares of masks are uniformly random whereas $\mathsf{E}$'s are compressed, is reflected in how the functionality samples correlations conditioned on which party is corrupted. It is straightforward to verify that, modulo these changes, $\mathcal{F}_{\text{cpre}}$ is equivalent to the compressed preprocessing functionality presented in [CWYY23].