Actively Secure Half-Gates with Minimum Overhead under Duplex Networks

Hongrui Cui
Shanghai Jiao Tong University
rickfreeman@sjtu.edu.cn

Xiao Wang
Northwestern University
wangxiao@cs.northwestern.edu

Kang Yang
State Key Laboratory of Cryptology
yangk@sklc.org

Yu Yu
Shanghai Jiao Tong University
yuyu@yuyu.hk

February 24, 2023

Abstract

Actively secure two-party computation (2PC) is one of the canonical building blocks in modern cryptography. One main goal for designing actively secure 2PC protocols is to reduce the communication overhead, compared to semi-honest 2PC protocols. In this paper, we propose a new actively secure constant-round 2PC protocol with one-way communication of $2\kappa+5$ bits per AND gate (for κ-bit computational security and any statistical security), essentially matching the one-way communication of semi-honest half-gates protocol. This is achieved by two new techniques:

1. The recent compression technique by Dittmer et al. (Crypto 2022) shows that a relaxed preprocessing is sufficient for authenticated garbling that does not reveal masked wire values to the garbler. We introduce a new form of authenticated bits and propose a new technique of generating authenticated AND triples to reduce the one-way communication of preprocessing from $5\rho + 1$ bits to 2 bits per AND gate for ρ-bit statistical security.
2. Unfortunately, the above compressing technique is only compatible with a less compact authenticated garbled circuit of size $2\kappa + 3\rho$ bits per AND gate. We designed a new authenticated garbling that does not use information theoretic MACs but rather dual execution without leakage to authenticate wire values in the circuit. This allows us to use a more compact half-gates based authenticated garbled circuit of size $2\kappa + 1$ bits per AND gate, and meanwhile keep compatible with the compression technique. Our new technique can achieve one-way communication of $2\kappa + 5$ bits per AND gate.

Our technique of yielding authenticated AND triples can also be used to optimize the two-way communication (i.e., the total communication) by combining it with the authenticated garbled circuits by Dittmer et al., which results in an actively secure 2PC protocol with two-way communication of $2\kappa + 3\rho + 4$ bits per AND gate.

1 Introduction

Based on garbled circuits (GCs) [Yao86], constant-round secure two-party computation (2PC) has obtained huge practical improvements in recent years in both communication [BMR90, KS08, ZRE15, RR21] and computation [BHKR13, GKWY20, GKW+20]. However, compared to passively

2PC	Rounds		Communication per AND gate	
	Prep.	Online	one-way (bits)	two-way (bits)
Half-gates	1	2	2κ	2κ
HSS-PCG [HSS17]	8	2	$8\kappa + 11$ (4.04×)	$16\kappa + 22$ (8.09×)
KRRW-PCG [KRRW18]	4	4	$5\kappa + 7$ (2.53×)	$8\kappa + 14$ (4.05×)
DILO [DILO22a]	7	2	$2\kappa + 8\rho + 1$ (2.25×)	$2\kappa + 8\rho + 5$ (2.27×)
This work	8	3	$2\kappa + 5$ ($\approx \mathbf{1}\times$)	$4\kappa + 10$ (2.04×)
This work+DILO	8	2	$2\kappa + 3\rho + 2$ (1.48×)	$2\kappa + 3\rho + 4$ ($\approx \mathbf{1.48}\times$)

Table 1: Comparing our protocol with prior works in terms of round and communication complexity. Here κ, ρ denote the computational and statistical security parameters instantiated by 128 and 40 respectively. Round complexity is counted in the random COT/VOLE-hybrid model. One-way communication is the greater of the two parties' communication; two-way communication is the sum of all communication. For the KRRW and HSS protocol we take the bucket size as $B = 3$.

secure (a.k.a., semi-honest) 2PC protocols, their actively secure counterparts require significant overhead. Building upon the authenticated garbling framework [WRK17a, WRK17b, KRRW18, YWZ20] and, more generally, working in the BMR family [BMR90, LPSY15, LSS16, HSS20, HIV17], the most recent work by Dittmer, Ishai, Lu and Ostrovsky [DILO22a] (denoted as DILO hereafter) is able to bring down the communication cost to $2\kappa + 8\rho + O(1)$ bits per AND gate, where κ and ρ are the computational and statistical security parameters, respectively.

Although huge progress, there is still a gap between actively secure and passively secure 2PC protocols based on garbled circuits. In particular, the size of a garbled circuit has been recently reduced from 2κ bits (half-gates [ZRE15]) to 1.5κ bits (three-halves [RR21]) per AND gate, while even the latest authenticated garbling cannot reach the communication efficiency of half-gates. It is possible to close this gap between active and passive security using the GMW compiler [GMW87], and its concrete efficiency was studied in [ASH+20]. However, it requires non-black-box use of the underlying garbling scheme and thus requires prohibitive overhead.

Bringing down the cost of authenticated garbling at this stage requires overcoming several challenges. First of all, we need the authenticated GC itself to be as small as the underlying GC construction. This could be achieved for half-gates as Katz et al. [KRRW18] (denoted as KRRW hereafter) proposed an authenticated half-gates construction in the two-party setting. However, when it comes to three-halves, there is no known construction. These authenticated GCs are usually generated in some preprocessing model, and thus the second challenge is to instantiate the preprocessing with only *constant additive overhead*. Together with recent works on pseudorandom correlation generators (PCGs) [BCG+19b, BCG+19a, YWL+20, CRR21, BCG+22], Katz et al. [KRRW18] can achieve $O(\kappa)$ bits per AND gate, while Dittmer et al. [DILO22a] can achieve $O(\rho)$ bits per AND gate. However, the latest advancement by Dittmer et al. [DILO22a] is not compatible with the optimal authenticated half-gates construction and requires an authenticated GC of size $2\kappa + 3\rho$ bits per AND gate.

1.1 Our Contribution

We make significant progress in closing the communication gap between passive and active GC-based 2PC protocols by proposing a new actively secure 2PC protocol with constant rounds and one-way communication essentially the same as the half-gates 2PC protocol in the semi-honest setting.

1. We manage to securely instantiate the preprocessing phase with $O(1)$ bits per AND gate. Our starting point is the compression technique by Dittmer et al. [DILO22a], who showed that in authenticated garbling, the random masks of the evaluator need not be of full entropy and can be compressed with entropy sublinear to the circuit size. This observation leads to an efficient construction from vector oblivious linear evaluation (VOLE) to the desired preprocessing functionality. This reduces the communication overhead of preprocessing to $5\rho + 1$ bits per AND gate. To further reduce their communication, we introduce a new tool called "dual-key authentication". Intuitively this form of authentication allows two parties to commit to a value that can later be checked against subsequent messages by both parties. Together with a new technique of generating authenticated AND triples from correlated oblivious transfer (COT), we avoid the ρ-time blow-up of the DILO protocol, and the one-way communication cost is reduced to 2 bits per AND gate.

2. As mentioned earlier, the above compression technique is not compatible with KRRW authenticated half-gates; this is because the compression technique requires that the garbler does not learn the masked values since the entropy of wire masks provided by the evaluator is low. We observe that the dual-execution protocol [HKE12, HsV20] can essentially be used for this purpose, and it is highly compatible with the authenticated garbling technique. In particular, the masked value of each wire is implicitly authenticated by the garbled label. Therefore we can perform two independent executions and check the actual value of each wire against each other. Since every wire is checked, we are able to eliminate the 1-bit leakage in ordinary dual-execution protocols. The overall one-way communication is $2\kappa + 5$ bits per AND gate.

We note that this is only a partial solution because dual execution requires both parties to send GCs. Under full-duplex networks (e.g., most wired communication) where communication in both directions can happen simultaneously, this effectively imposes no slow down; however, for half-duplex networks (e.g., most wireless communication), it would not be a preferable option. Nevertheless, our preprocessing protocol can be combined with the construction of authenticated garbled circuits by Dittmer et al. [DILO22a] to achieve the best two-way communication of $2\kappa + 3\rho + 4$ bits per AND gate, leading to a $1.53\times$ improvement. We provide a detailed comparison in Table 1.

We do not compare our actively secure 2PC protocol with the protocol (denoted by DILOv2) by Dittmer et al. [DILO22a] building on doubly authenticated multiplication triples. Compared to DILO, the DILOv2 protocol is less efficient, as DILOv2 requires quasi-linear computational complexity. Moreover, DILOv2 can only generate authenticated triples over $\mathbb{F}_{2^\rho}$, while authenticated garbling requires triples over $\mathbb{F}_2$. This incurs a ρ-time overhead when utilizing such triples.

2 Preliminaries

2.1 Notation

We use κ and ρ to denote the computational and statistical security parameters, respectively. We use log to denote logarithms in base 2. We write $x \leftarrow S$ to denote sampling x uniformly at random from a finite set S. We define $[a, b) = \{a, \ldots, b-1\}$ and write $[a, b] = \{a, \ldots, b\}$. We use bold lower-case letters like $\boldsymbol{a}$ for column vectors, and bold upper-case letters like $\mathbf{A}$ for matrices. We let a_i denote the i-th component of $\boldsymbol{a}$ (with a_1 the first entry). We use $\{x_i\}_{i \in S}$ to denote the set that consists of all elements with indices in set S. When the context is clear, we abuse the notation and use $\{x_i\}$ to denote such a set. For a string x, we use $\mathsf{lsb}(x)$ to denote the least significant bit (LSB) and $\mathsf{msb}(x)$ to denote the most significant bit (MSB).

For an extension field $\mathbb{F}_{2^\kappa}$ of a binary field $\mathbb{F}_2$, we fix some monic, irreducible polynomial $f(X)$ of degree κ and then write $\mathbb{F}_{2^\kappa} \cong \mathbb{F}_2[X]/f(X)$. Thus, every element $x \in \mathbb{F}_{2^\kappa}$ can be denoted uniquely as $x = \sum_{i\in[0,\kappa)} x_i \cdot X^i$ with $x_i \in \mathbb{F}_2$ for all $i \in [0, \kappa)$. We could view elements over $\mathbb{F}_{2^\kappa}$ equivalently as vectors in $\mathbb{F}_2^\kappa$ or strings in $\{0,1\}^\kappa$, and consider a bit $x \in \mathbb{F}_2$ as an element in $\mathbb{F}_{2^\kappa}$. Depending on the context, we use $\{0,1\}^\kappa$, $\mathbb{F}_2^\kappa$ and $\mathbb{F}_{2^\kappa}$ interchangeably, and thus addition in $\mathbb{F}_2^\kappa$ and $\mathbb{F}_{2^\kappa}$ corresponds to XOR in $\{0,1\}^\kappa$. We also define two macros to convert between $\mathbb{F}_{2^\kappa}$ and $\mathbb{F}_2^\kappa$.

- $x \leftarrow \mathsf{B2F}(\boldsymbol{x})$: Given $\boldsymbol{x} = (x_0, ..., x_{\kappa-1}) \in \mathbb{F}_2^\kappa$, output $x := \sum_{i\in[0,\kappa)} x_i \cdot X^i \in \mathbb{F}_{2^\kappa}$.
- $\boldsymbol{x} \leftarrow \mathsf{F2B}(x)$: Given $x = \sum_{i\in[0,\kappa)} x_i \cdot X^i \in \mathbb{F}_{2^\kappa}$, output $\boldsymbol{x} = (x_0, ..., x_{\kappa-1}) \in \mathbb{F}_2^\kappa$.

A Boolean circuit $\mathcal{C}$ consists of a list of gates in the form of (i, j, k, T), where i, j are the indices of input wires, k is the index of output wire and $T \in \{\oplus, \wedge\}$ is the type of the gate. In the 2PC setting, we use $\mathcal{I}_\mathsf{A}$ (resp., $\mathcal{I}_\mathsf{B}$) to denote the set of circuit-input wire indices corresponding to the input of $\mathsf{P_A}$ (resp., $\mathsf{P_B}$). We also use $\mathcal{W}$ to denote the set of output-wire indices of all AND gates, and $\mathcal{O}$ to denote the set of circuit-output wire indices in the circuit $\mathcal{C}$. We denote by $\mathcal{C}_\mathsf{and}$ the set of all AND gates in the form of (i, j, k, T).

Our protocol in the two-party setting is proven secure against static and malicious adversaries in the standard simulation-based security model [Can00, Gol04]. We recall the security model, a relaxed equality-check functionality $\mathcal{F}_\mathsf{EQ}$ and the coin-tossing functionality $\mathcal{F}_\mathsf{Rand}$ as well as the summary of the notations and macros used in our protocols at Appendix A.

2.2 Information-Theoretic Message Authentication Codes

We use information-theoretic message authentication codes (IT-MACs) [BDOZ11, NNOB12] to authenticate bits or field elements in $\mathbb{F}_{2^\kappa}$. Specifically, let $\Delta \in \mathbb{F}_{2^\kappa}$ be a *global key*. We adopt $[x] = (\mathsf{K}[x], \mathsf{M}[x], x)$ to denote that an element $x \in \mathbb{F}$ (where $\mathbb{F} \in \{\mathbb{F}_2, \mathbb{F}_{2^\kappa}\}$) known by one party can be authenticated by the other party who holds $\Delta \in \mathbb{F}_{2^\kappa}$ and a *local key* $\mathsf{K}[x] \in \mathbb{F}_{2^\kappa}$, where an MAC tag $\mathsf{M}[x] = \mathsf{K}[x] + x \cdot \Delta \in \mathbb{F}_{2^\kappa}$ is given to the party holding x. For a vector $\boldsymbol{x} \in \mathbb{F}^\ell$, we denote by $[\boldsymbol{x}] = ([x_1], ..., [x_\ell])$ a vector of authenticated values. We refer to $([x], [y], [z])$ with $z = x \cdot y$ as an authenticated multiplication triple. If $x, y, z \in \{0, 1\}$, this tuple is also called authenticated AND triple. For a constant value $c \in \mathbb{F}_{2^\kappa}$, it is easy to define $[c] = (c \cdot \Delta, 0^\kappa, c)$. It is well-known that IT-MACs are additively homomorphic. That is, given public coefficients $c_0, c_1, \ldots, c_\ell \in \mathbb{F}_{2^\kappa}$, two parties can *locally* compute $[y] := c_0 + \sum_{i=1}^{\ell} c_i \cdot [x_i]$.

When applying IT-MACs into 2PC, secret values are authenticated by either $\mathsf{P_A}$ or $\mathsf{P_B}$. We use subscripts A and B in authenticated values to distinguish which party ($\mathsf{P_A}$ or $\mathsf{P_B}$) holds the secret values. For example, $[x]_\mathsf{A} = (\mathsf{K_B}[x], \mathsf{M_A}[x], x)$ denotes that $\mathsf{P_A}$ holds $(x, \mathsf{M_A}[x])$ and $\mathsf{P_B}$ holds $(\Delta_\mathsf{B}, \mathsf{K_B}[x])$. In the case that other global keys are used, we explicitly add a subscript to keys and MAC tags. For example, when $G \in \mathbb{F}_{2^\kappa}$ is used and held by $\mathsf{P_B}$, we write $[x]_{\mathsf{A},G} = (\mathsf{K_B}[x]_G, \mathsf{M_A}[x]_G, x)$ and $\mathsf{M_A}[x]_G = \mathsf{K_B}[x]_G + x \cdot G$. When the context is clear, we will omit the subscripts A and B for the sake of simplicity.

Batch opening of authenticated values. In the following, we describe the known procedure [NNOB12, DNNR17] to open authenticated values in a batch. Here we always assume that $\mathsf{P_A}$ holds the values and MAC tags, and $\mathsf{P_B}$ holds the global and local keys. In this case, we write $[x]$ instead of $[x]_\mathsf{A}$. For the case that $\mathsf{P_B}$ holds the values authenticated by $\mathsf{P_A}$, these procedures can be defined similarly. We first define the following procedure (denoted by $\mathsf{CheckZero}$) to check that all values are zero in constant small communication.

- $\mathsf{CheckZero}([x_1], \ldots, [x_\ell])$: On input authenticated values $[x_1], \ldots, [x_\ell]$, $\mathsf{P_A}$ convinces $\mathsf{P_B}$ that $x_i = 0$ for all $i \in [1, \ell]$ as follows:

Functionality $\mathcal{F}^L_{\mathsf{bCOT}}$

This functionality is parameterized by an integer $L \geq 1$. Running with a sender $\mathsf{P_A}$, a receiver $\mathsf{P_B}$ and an ideal adversary, it operates as follows.

Initialize. Upon receiving $(\mathsf{init}, sid, \Delta_1, ..., \Delta_L)$ from $\mathsf{P_A}$ and (init, sid) from $\mathsf{P_B}$ where $\Delta_i \in \mathbb{F}_{2^\kappa}$ for all $i \in [1, L]$, store $(sid, \Delta_1, ..., \Delta_L)$ and then ignore all subsequent (init, sid) commands.

Extend. Upon receiving $(\mathsf{extend}, sid, \ell)$ from $\mathsf{P_A}$ and $\mathsf{P_B}$, do the following:

- For $i \in [1, L]$, if $\mathsf{P_A}$ is honest, sample $\mathsf{K_A}[\boldsymbol{u}]_{\Delta_i} \leftarrow \mathbb{F}^\ell_{2^\kappa}$; otherwise, receive $\mathsf{K_A}[\boldsymbol{u}]_{\Delta_i} \in \mathbb{F}^\ell_{2^\kappa}$ from the adversary.
- If $\mathsf{P_B}$ is honest, sample $\boldsymbol{u} \leftarrow \mathbb{F}^\ell_2$ and compute $\mathsf{M_B}[\boldsymbol{u}]_{\Delta_i} := \mathsf{K_A}[\boldsymbol{u}]_{\Delta_i} + \boldsymbol{u} \cdot \Delta_i \in \mathbb{F}^\ell_{2^\kappa}$ for $i \in [1, L]$. Otherwise, receive $\boldsymbol{u} \in \mathbb{F}^\ell_2$ and $\mathsf{M_B}[\boldsymbol{u}]_{\Delta_i} \in \mathbb{F}^\ell_{2^\kappa}$ for $i \in [1, L]$ from the adversary, and recomputes $\mathsf{K_A}[\boldsymbol{u}]_{\Delta_i} := \mathsf{M_B}[\boldsymbol{u}]_{\Delta_i} + \boldsymbol{u} \cdot \Delta_i \in \mathbb{F}^\ell_{2^\kappa}$ for $i \in [1, L]$.
- For $i \in [1, L]$, output $(sid, \mathsf{K_A}[\boldsymbol{u}]_{\Delta_i})$ to $\mathsf{P_A}$ and $(sid, \boldsymbol{u}, \mathsf{M_B}[\boldsymbol{u}]_{\Delta_i})$ to $\mathsf{P_B}$.

Figure 1: Functionality for block correlated oblivious transfer.

1. $\mathsf{P_A}$ sends $h := \mathsf{H}(\mathsf{M_A}[x_1], \dots, \mathsf{M_A}[x_\ell])$ to $\mathsf{P_B}$, where $\mathsf{H} : \{0,1\}^* \to \{0,1\}^\kappa$ is a random oracle.
2. $\mathsf{P_B}$ computes $h' := \mathsf{H}(\mathsf{K_B}[x_1], \dots, \mathsf{K_B}[x_\ell])$ and checks that $h = h'$. If the check fails, $\mathsf{P_B}$ aborts.

Following previous works [DNNR17, WYKW21], we have the following lemma.

Lemma 1. *If $\Delta \in \mathbb{F}_{2^\kappa}$ is sampled uniformly at random, then the probability that there exists some $i \in [1, \ell]$ such that $x_i \neq 0$ and $\mathsf{P_B}$ accepts in the* $\mathsf{CheckZero}$ *procedure is bounded by $\frac{2}{2^\kappa}$.*

The above lemma can be relaxed by allowing that Δ is sampled uniformly from a set $\mathcal{R} \subset \mathbb{F}_{2^\kappa}$. In this case, the success probability for a cheating party $\mathsf{P_A}$ is at most $\frac{1}{|\mathcal{R}|} + \frac{1}{2^\kappa}$. Based on the $\mathsf{CheckZero}$ procedure, we define the following batch-opening procedure (denoted by Open):

- $\mathsf{Open}([x_1], \dots, [x_\ell])$: On input authenticated values $[x_1], \dots, [x_\ell]$ defined over field $\mathbb{F}_{2^\kappa}$, $\mathsf{P_A}$ opens these values as follows:

  1. $\mathsf{P_A}$ sends $(x_1, \dots, x_\ell)$ to $\mathsf{P_B}$, and then both parties set $[y_i] := [x_i] + x_i$ for each $i \in [1, \ell]$.
  2. $\mathsf{P_A}$ runs $\mathsf{CheckZero}([y_1], \dots, [y_\ell])$ with $\mathsf{P_B}$. If $\mathsf{P_B}$ does not abort, it outputs $(x_1, \dots, x_\ell)$.

2.3 Correlated Oblivious Transfer

Our 2PC protocol will adopt the standard functionality [BCG+19a, YWL+20] of correlated oblivious transfer (COT) to generate random authenticated bits. This functionality (denoted by $\mathcal{F}_{\mathsf{COT}}$) is shown in Figure 1 by setting a parameter $L = 1$, where the extension phase can be executed multiple times for the same session identifier sid. Based on Learning Parity with Noise (LPN) [BFKL94], the recent protocols [BCG+19a, YWL+20, CRR21, BCG+22] with *sublinear* communication and *linear* computation can securely realize the COT functionality in the presence of malicious adversaries. In particular, these protocols can generate a COT correlation with amortized communication cost of about $0.1 \sim 0.4$ bits.

We also generalize the COT functionality into block COT (bCOT) [DILO22a], which allows to generate authenticated bits with the same choice bits and different global keys. Functionality $\mathcal{F}^L_{\mathsf{bCOT}}$ shown in Figure 1 is the same as the standard COT functionality, except that L vectors (rather than a single vector) of authenticated bits $[\boldsymbol{u}]_{\mathsf{B},\Delta_1}, \dots, [\boldsymbol{u}]_{\mathsf{B},\Delta_L}$ are generated. Here the vector of choice bits $\boldsymbol{u}$ is required to be identical in different vectors of authenticated bits. It is easy to see that $\mathcal{F}_{\mathsf{COT}}$ is a special case of $\mathcal{F}^L_{\mathsf{bCOT}}$ with $L = 1$. The protocol that securely realizes functionality

Functionality $\mathcal{F}_{\mathsf{DVZK}}$

This functionality runs with a prover $\mathcal{P}$ and a verifier $\mathcal{V}$, and operates as follows:

- Upon receiving $(\mathsf{dvzk}, sid, \ell, \{[x_i], [y_i], [z_i]\}_{i \in [1,\ell]})$ from $\mathcal{P}$ and $\mathcal{V}$ where $x_i, y_i, z_i \in \mathbb{F}_{2^\kappa}$ for all $i \in [1, \ell]$, if there exists some $i \in [1, \ell]$ such that one of $[x_i], [y_i], [z_i]$ is not valid, output (sid, false) to $\mathcal{V}$ and abort.
- Check that $z_i = x_i \cdot y_i \in \mathbb{F}_{2^\kappa}$ for all $i \in [1, \ell]$. If the check passes, then output (sid, true) to $\mathcal{V}$, else output (sid, false) to $\mathcal{V}$.

Figure 2: Functionality for DVZK proofs of authenticated multiplication triples.

$\mathcal{F}^L_{\mathsf{bCOT}}$ is easy to be constructed by extending the LPN-based COT protocol as described above. Specifically, we set $\Delta = (\Delta_1, \ldots, \Delta_L) \in \mathbb{F}^L_{2^\kappa} \cong \mathbb{F}_{2^{\kappa L}}$ as the global key in the LPN-based COT protocol, and the resulting choice-bits are authenticated over extension field $\mathbb{F}_{2^{\kappa L}}$. Note that the protocol to generate block COTs still has *sublinear* communication, if L is sublinear to the number of the resulting COT correlations.

While the COT functionality outputs random authenticated bits, we can convert them into chosen authenticated bits via the following procedure (denoted by Fix), which is also used in the recent DVZK protocol [BMRS21].

- $([\boldsymbol{x}]_{\mathsf{B},\Delta_1}, \ldots, [\boldsymbol{x}]_{\mathsf{B},\Delta_L}) \leftarrow \mathsf{Fix}(sid, \boldsymbol{x})$: On input a session identifier sid of $\mathcal{F}_{\mathsf{bCOT}}$, and a vector $\boldsymbol{x} \in \mathbb{F}^\ell_2$ from $\mathsf{P_B}$, two parties $\mathsf{P_A}$ and $\mathsf{P_B}$ execute the following:
  1. Both parties call $\mathcal{F}^L_{\mathsf{bCOT}}$ on input $(\mathsf{extend}, sid, \ell)$ to obtain $([\boldsymbol{r}]_{\mathsf{B},\Delta_1}, \ldots, [\boldsymbol{r}]_{\mathsf{B},\Delta_L})$ with a random vector $\boldsymbol{r} \in \mathbb{F}^\ell_2$ held by $\mathsf{P_B}$, where $\mathcal{F}^L_{\mathsf{bCOT}}$ has been initialized by sid and $(\Delta_1, \ldots, \Delta_L)$.
  2. $\mathsf{P_B}$ sends $\boldsymbol{d} := \boldsymbol{x} \oplus \boldsymbol{r}$ to $\mathsf{P_A}$.
  3. For each $i \in [1, L]$, both parties set $[\boldsymbol{x}]_{\mathsf{B},\Delta_i} := [\boldsymbol{r}]_{\mathsf{B},\Delta_i} \oplus \boldsymbol{d}$.

For a field element $x \in \mathbb{F}_{2^\kappa}$, $\mathsf{P_A}$ and $\mathsf{P_B}$ can run $\boldsymbol{x} \leftarrow \mathsf{F2B}(x)$, $([\boldsymbol{x}]_{\mathsf{B},\Delta_1}, \ldots, [\boldsymbol{x}]_{\mathsf{B},\Delta_L}) \leftarrow \mathsf{Fix}(sid, \boldsymbol{x})$ and $([x]_{\mathsf{B},\Delta_1}, \ldots, [x]_{\mathsf{B},\Delta_L}) \leftarrow \mathsf{B2F}([\boldsymbol{x}]_{\mathsf{B},\Delta_1}, \ldots, [\boldsymbol{x}]_{\mathsf{B},\Delta_L})$ to obtain the corresponding authenticated values. Note that $\mathsf{B2F}$ only involves the operations multiplied by public elements $X, \ldots, X^{\kappa-1} \in \mathbb{F}_{2^\kappa}$, and thus $([x]_{\mathsf{B},\Delta_1}, \ldots, [x]_{\mathsf{B},\Delta_L})$ can be computed locally by running $\mathsf{B2F}$. For simplicity, we abuse the Fix notation, and use $([x]_{\mathsf{B},\Delta_1}, \ldots, [x]_{\mathsf{B},\Delta_L}) \leftarrow \mathsf{Fix}(sid, x)$ to denote the conversion procedure. The Fix procedure is easy to be generalized to support that the values are defined over any field $\mathbb{F}$ such as $\mathbb{F} = \mathbb{F}_{2^\rho}$. The Fix procedure is totally similar for generating authenticated bits $[\boldsymbol{x}]_{\mathsf{A},\Delta_1}, \ldots, [\boldsymbol{x}]_{\mathsf{A},\Delta_L}$ from random authenticated bits, where here $\mathsf{P_B}$ holds $(\Delta_1, \ldots, \Delta_L)$. When the context is clear, we just write $([\boldsymbol{x}]_{\Delta_1}, \ldots, [\boldsymbol{x}]_{\Delta_L}) \leftarrow \mathsf{Fix}(sid, \boldsymbol{x})$ for simplicity. We further extend Fix to additionally allow to input vectors of random authenticated bits instead of calling $\mathcal{F}^L_{\mathsf{bCOT}}$, which is denoted by $[\boldsymbol{x}] \leftarrow \mathsf{Fix}(\boldsymbol{x}, [\boldsymbol{r}])$ for the case of $L = 1$.

2.4 Designated-Verifier Zero-Knowledge Proofs

Based on IT-MACs, a family of streamable designated-verifier zero-knowledge (DVZK) proofs with fast prover time and a small memory footprint has been proposed [WYKW21, DIO21, BMRS21, YSWW21, WYX+21, BBMH+21, DILO22b, WYY+22, BBMHS22]. While these DVZK proofs can prove arbitrary circuits, we only need them to prove a simple multiplication relation. Specifically, given a set of authenticated triples $\{([x_i], [y_i], [z_i])\}_{i \in [1,\ell]}$ over $\mathbb{F}_{2^\kappa}$, these DVZK protocols can enable a prover $\mathcal{P}$ to convince a verifier $\mathcal{V}$ that $z_i = x_i \cdot y_i$ for all $i \in [1, \ell]$. This is modeled by an ideal functionality shown in Figure 2. In this functionality, an authenticated value $[x]$ is input by two parties $\mathcal{P}$ and $\mathcal{V}$, meaning that $\mathcal{P}$ inputs (x, M) and $\mathcal{V}$ inputs (K, Δ). We say that $[x]$ is valid, if

$\mathsf{M} = \mathsf{K} + x \cdot \Delta$. Using the recent DVZK proofs, this functionality can be *non-interactively* realized in the random-oracle model using constant small communication (e.g., 2κ bits in total [YSWW21]).

3 Technical Overview

In this section, we give an overview of our techniques. The detailed protocols and their formal proofs are described in later sections. Firstly, we recall the basic approach in the state-of-the-art solution [DILO22a].

3.1 Overview of the State-of-the-Art Solution

Recently, Dittmer, Ishai, Lu and Ostrovsky [DILO22a] constructed the state-of-the-art 2PC protocol with malicious security (denoted by DILO) from simple VOLE correlations. [1] For one-way communication, this protocol takes $5\rho + 1$ bits to generate a single authenticated AND triple and $2\kappa + 3\rho$ bits per AND gate to produce one distributed garbled circuit. Their approach is outlined as follows.

In the framework of authenticated garbling [WRK17a], for each AND gate $(i, j, k, \wedge)$, the garbler $\mathsf{P_A}$ and evaluator $\mathsf{P_B}$ need to generate one authenticated triple $([a_i], [b_i], [a_j], [b_j], [\hat{a}_k], [\hat{b}_k])$ such that $\hat{a}_k \oplus \hat{b}_k = (a_i \oplus b_i) \wedge (a_j \oplus b_j)$. Let $\boldsymbol{b} \in \mathbb{F}_2^n$ (resp., $\boldsymbol{b}_{\mathcal{I}} \in \mathbb{F}_2^m$) be the vector of random masks $\{b_i\}$ held by $\mathsf{P_B}$ on the output wires of all AND gates (resp., on all circuit-input wires associated with the $\mathsf{P_B}$'s input), where n is the number of all AND gates and m is the number of all circuit-input gates. The key observation by Dittmer et al. [DILO22a] is that only evaluator $\mathsf{P_B}$ can compute masked wire values (i.e., the XOR of actual wire values and random masks), and thus $\boldsymbol{b}$ is unnecessary to be uniformly random if the masked wire values are *not* revealed to $\mathsf{P_A}$. In particular, when these masked wire values are not revealed by $\mathsf{P_B}$, a malicious garbler $\mathsf{P_A}$ can only guess some masked wire values by performing a selective-failure attack. This means that for each masked wire value, $\mathsf{P_A}$ can guess correctly with probability $1/2$, and the protocol execution will abort for an incorrect guess. In this case, $\mathsf{P_A}$ can guess at most $\rho - 1$ masked wire values, and otherwise the protocol will abort with probability at least $1 - 1/2^\rho$. The core idea of DILO is to compress vector $\boldsymbol{b}$ by defining $\boldsymbol{b} = \mathbf{M} \cdot \boldsymbol{b}^*$, where $\mathbf{M} \in \mathbb{F}_2^{n \times L}$ is a public matrix such that any ρ rows of $\mathbf{M}$ are linearly independent, $\boldsymbol{b}^* \in \mathbb{F}_2^L$ is a uniform vector and $L = O(\rho \log(n/\rho))$. Since IT-MACs are additively homomorphic, two parties only need to generate $[\boldsymbol{b}^*]$ (instead of $[\boldsymbol{b}]$) for a much shorter vector $\boldsymbol{b}^*$, and then compute $[\boldsymbol{b}] := \mathbf{M} \cdot [\boldsymbol{b}^*]$.

Dittmer et al. [DILO22a] assume that $\boldsymbol{b}_{\mathcal{I}}$ is uniform and authenticated AND triples related to $\boldsymbol{b}_{\mathcal{I}}$ are generated using the previous approach such as [KRRW18]. Therefore, we only show how to generate compressed authenticated AND triples, where random masks held by $\mathsf{P_B}$ are compressed. Two parties can first generate compressed authenticated AND triple $([a_i], [b_i], [a_j], [b_j], [\hat{a}_k], [\hat{b}_k])$ for each AND gate with $\Delta_{\mathsf{A}} \leftarrow \mathbb{F}_{2^\rho}$, and then convert them into that with $\Delta'_{\mathsf{A}} \leftarrow \mathbb{F}_{2^\kappa}$ using extra 2 bits of communication per AND gate, where a ρ-bit global key can guarantee that communication only depends on ρ rather than κ and $\Delta'_{\mathsf{A}} \in \mathbb{F}_{2^\kappa}$ is required for garbled circuits. In the following, we give an overview of Dittmer et al.'s approach on how to generate circuit-dependent compressed authenticated AND triples $\{([a_i], [b_i], [a_j], [b_j], [\hat{a}_k], [\hat{b}_k])\}$ with $\Delta_{\mathsf{A}}, \Delta_{\mathsf{B}} \in \mathbb{F}_{2^\rho}$.

1. $\mathsf{P_A}$ and $\mathsf{P_B}$ generates a vector of authenticated bits $[\boldsymbol{b}^*]$ with a uniform $\boldsymbol{b}^* \in \mathbb{F}_2^L$ by calling $\mathcal{F}_{\mathsf{COT}}$. Then, both parties define $[\boldsymbol{b}] := \mathbf{M} \cdot [\boldsymbol{b}^*]$.

[1] VOLE is an arithmetic generalization of COT, and enables $\mathsf{P_A}$ to obtain $(\Delta, \mathsf{K}[\boldsymbol{u}]) \in \mathbb{F} \times \mathbb{F}^\ell$ and $\mathsf{P_B}$ to get $(\boldsymbol{u}, \mathsf{M}[\boldsymbol{u}]) \in \mathbb{F}^\ell \times \mathbb{F}^\ell$ such that $\mathsf{M}[\boldsymbol{u}] = \mathsf{K}[\boldsymbol{u}] + \boldsymbol{u} \cdot \Delta$, where $\mathbb{F}$ is a large field such as $\mathbb{F} = \mathbb{F}_{2^\rho}$.

2. Both parties compute authenticated bit $[b_{i,j}]$ for each AND gate $(i,j,k,\wedge)$ via running the Fix procedure with input $\{b_{i,j}\}$ where $b_{i,j} := b_i \cdot b_j$.

3. $\mathsf{P_B}$ samples $\Delta_\mathsf{B}, \gamma \leftarrow \mathbb{F}_{2^\rho}$. Then, both parties initializes two functionalities $\mathcal{F}_{\mathsf{bCOT}}^{L+2}$ and $\mathcal{F}_{\mathsf{bVOLE}}^{L+2}$ with the same global keys $(b_1^* \cdot \Delta_\mathsf{B} + \gamma, \ldots, b_L^* \cdot \Delta_\mathsf{B} + \gamma, \Delta_\mathsf{B} + \gamma, \gamma)$, where $\mathcal{F}_{\mathsf{bVOLE}}^{L+2}$ is the same as $\mathcal{F}_{\mathsf{bCOT}}^{L+2}$ except that the outputs are VOLE correlations over $\mathbb{F}_{2^\rho}$ instead of COT correlations. Here γ is necessary to mask $b_i^* \cdot \Delta_\mathsf{B}$. In particular, a consistency check in DILO lets $\mathsf{P_B}$ send a hashing of values related to $b_i^* \cdot \Delta_\mathsf{B}$ to the malicious party $\mathsf{P_A}$, which may leak the bit b_i^* to $\mathsf{P_A}$. This attack would be prevented by using a uniform γ to mask $b_i^* \cdot \Delta_\mathsf{B}$. Given $[a]_{b_i^*\Delta_\mathsf{B}+\gamma}$ and $[a]_\gamma$ for any bit a held by $\mathsf{P_A}$, it is easy to locally compute $[ab_i^*]_{\Delta_\mathsf{B}}$ from the additive homomorphism of IT-MACs. Similarly, given $[a]_{\Delta_\mathsf{B}+\gamma}$ and $[a]_\gamma$, two parties can locally compute $[a]_{\Delta_\mathsf{B}}$.

4. $\mathsf{P_A}$ and $\mathsf{P_B}$ calls $\mathcal{F}_{\mathsf{bCOT}}^{L+2}$ to generate the vectors of authenticated bits $[\boldsymbol{a}], [\hat{\boldsymbol{a}}]$ as well as $[a_i \boldsymbol{b}^*]_{\Delta_\mathsf{B}}$ for each $i \in [1,n]$, where $\boldsymbol{a} \in \mathbb{F}_2^n$ (resp., $\hat{\boldsymbol{a}} \in \mathbb{F}_2^n$) is used as the vector of random masks $\{a_i\}$ (resp., $\{\hat{a}_k\}$) held by $\mathsf{P_A}$ on the output wires of all AND gates. Then, they can locally compute $[a_i b_j]_{\Delta_\mathsf{B}}$ and $[a_j b_i]_{\Delta_\mathsf{B}}$ for each AND gate $(i,j,k,\wedge)$ by calculating $\mathbf{M} \cdot [a_i \boldsymbol{b}^*]_{\Delta_\mathsf{B}}$. Both parties run the Fix procedure with input $\{a_{i,j}\}$ to obtain $\{[a_{i,j}]\}$, where $a_{i,j} = a_i \wedge a_j$ for each AND gate $(i,j,k,\wedge)$.

5. $\mathsf{P_A}$ and $\mathsf{P_B}$ call $\mathcal{F}_{\mathsf{bVOLE}}^{L+2}$ to get a vector of authenticated values $[\tilde{\boldsymbol{a}}]$ with a uniform vector $\tilde{\boldsymbol{a}} \in \mathbb{F}_{2^\rho}^n$. Both parties run the Fix procedure with input $(\Delta_\mathsf{A} \cdot \boldsymbol{a}, \Delta_\mathsf{A} \cdot \hat{\boldsymbol{a}}, \{\Delta_\mathsf{A} \cdot a_{i,j}\}, \Delta_\mathsf{A})$ to obtain authenticated values $[\Delta_\mathsf{A} \cdot \boldsymbol{a}]$, $[\Delta_\mathsf{A} \cdot \hat{\boldsymbol{a}}]$, $\{[\Delta_\mathsf{A} \cdot a_{i,j}]\}$ and $[\Delta_\mathsf{A}]_{\Delta_\mathsf{B}}$. The Fix procedure corresponds to calling $\mathcal{F}_{\mathsf{bVOLE}}^{L+2}$, and also outputs $[\Delta_\mathsf{A} a_i \boldsymbol{b}^*]_{\Delta_\mathsf{B}}$ for each $i \in [1,n]$ and $[\Delta_\mathsf{A}]_{b_i^*\Delta_\mathsf{B}}$ for each $i \in [1,L]$ to both parties. Note that $[\Delta_\mathsf{A}]_{\Delta_\mathsf{B}}$ and $[\Delta_\mathsf{A}]_{b_i^*\Delta_\mathsf{B}}$ can be written as $[\Delta_\mathsf{B}]$ and $[b_i^*\Delta_\mathsf{B}]$ respectively, where we also use $[B_i^*]$ to denote $[b_i^*\Delta_\mathsf{B}]$. Furthermore, $\mathsf{P_A}$ and $\mathsf{P_B}$ can locally compute $[\Delta_\mathsf{A} a_i b_j]_{\Delta_\mathsf{B}}$ and $[\Delta_\mathsf{A} a_j b_i]_{\Delta_\mathsf{B}}$ for each AND gate $(i,j,k,\wedge)$ by computing $\mathbf{M} \cdot [\Delta_\mathsf{A} a_i \boldsymbol{b}^*]_{\Delta_\mathsf{B}}$ for each $i \in [1,n]$.

6. Parties $\mathsf{P_A}$ and $\mathsf{P_B}$ call $\mathcal{F}_{\mathsf{DVZK}}$ to prove the following relations:
   - For each AND gate $(i,j,k,\wedge)$, given $([b_i],[b_j],[b_{i,j}])$, prove $b_{i,j} = b_i \wedge b_j$.
   - For each AND gate $(i,j,k,\wedge)$, given $([a_i],[a_j],[a_{i,j}])$, prove $a_{i,j} = a_i \wedge a_j$.
   - For each $i \in [1,L]$, given $([b_i^*],[\Delta_\mathsf{B}],[B_i^*])$, prove $B_i^* = b_i^* \cdot \Delta_\mathsf{B}$.

7. $\mathsf{P_B}$ also executes an efficient verification protocol to convince $\mathsf{P_A}$ that the same global keys are input to different functionalities $\mathcal{F}_{\mathsf{bCOT}}^{L+2}$ and $\mathcal{F}_{\mathsf{bVOLE}}^{L+2}$. It is unnecessary to check the consistency of $\Delta_\mathsf{A} \cdot \boldsymbol{a}, \Delta_\mathsf{A} \cdot \hat{\boldsymbol{a}}, \{\Delta_\mathsf{A} \cdot a_{i,j}\}, \Delta_\mathsf{A}$ input to Fix w.r.t. $\mathcal{F}_{\mathsf{bVOLE}}^{L+2}$. The resulting VOLE correlations on these inputs are used to compute the MAC tags of $\mathsf{P_B}$ on its shares. If these inputs are incorrect, this only leads to these MAC tags, which will be authenticated by $\mathsf{P_A}$, being incorrect. This is harmless for security.

8. For each AND gate $(i,j,k,\wedge)$, $\mathsf{P_A}$ and $\mathsf{P_B}$ locally compute $[\tilde{b}_k]_{\Delta_\mathsf{B}} := [a_{i,j}] + [a_i b_j] + [a_j b_i] + [\hat{a}_k]$ and $[\tilde{B}_k]_{\Delta_\mathsf{B}} := [\Delta_\mathsf{A} a_{i,j}] + [\Delta_\mathsf{A} a_i b_j] + [\Delta_\mathsf{A} a_j b_i] + [\Delta_\mathsf{A} \hat{a}_k] + [\tilde{a}_k]$, where all values are authenticated under Δ_B. Then, $\mathsf{P_A}$ sends a pair of MAC tags $(\mathsf{M_A}[\tilde{b}_k], \mathsf{M_A}[\tilde{B}_k])$ to $\mathsf{P_B}$, who computes the following over $\mathbb{F}_{2^\kappa}$

$$\tilde{b}_k := (\mathsf{K_B}[\tilde{b}_k] + \mathsf{M_A}[\tilde{b}_k]) \cdot \Delta_\mathsf{B}^{-1} \text{ and } \tilde{B}_k := (\mathsf{K_B}[\tilde{B}_k] + \mathsf{M_A}[\tilde{B}_k]) \cdot \Delta_\mathsf{B}^{-1}.$$

   It is easy to see that $\tilde{b}_k = a_{i,j} \oplus a_i b_j \oplus a_j b_i \oplus \hat{a}_k \in \{0,1\}$ and $\tilde{B}_k = (a_{i,j} + a_i b_j + a_j b_i + \hat{a}_k) \cdot \Delta_\mathsf{A} + \tilde{a}_k \in \mathbb{F}_{2^\rho}$, where the randomness $\tilde{a}_k \in \mathbb{F}_{2^\rho}$ is crucial to prevent that $\tilde{B}_k$ directly reveals Δ_A in the case of $\tilde{b}_k = 1$. We observe that both parties now obtain an authenticated bit $[\tilde{b}_k]_{\Delta_\mathsf{A}}$ by defining its local key $\mathsf{K_A}[\tilde{b}_k] = \tilde{a}_k$ and MAC tag $\mathsf{M_B}[\tilde{b}_k] = \tilde{B}_k$.

9. For each AND gate $(i, j, k, \wedge)$, $\mathsf{P_A}$ and $\mathsf{P_B}$ locally compute an authenticated bit $[\hat{b}_k]_{\Delta_\mathsf{A}} := [\tilde{b}_k]_{\Delta_\mathsf{A}} \oplus [b_{i,j}]_{\Delta_\mathsf{A}}$. Now, both parties obtain an authenticated triple $([a_i], [b_i], [a_j], [b_j], [\hat{a}_k], [\hat{b}_k])$ for each AND gate $(i, j, k, \wedge)$.

3.2 Our Solution for Generating Authenticated AND Triples

In the DILO protocol [DILO22a], the one-way communication cost of generating the authenticated triple $([a_i], [b_i], [a_j], [b_j], [\hat{a}_k], [\hat{b}_k])$ for each AND gate $(i, j, k, \wedge)$ is brought about by producing an authenticated bit $[\tilde{b}_k]$ under Δ_A that is in turn used to locally compute $[\hat{b}_k]$ with $\hat{b}_k = \tilde{b}_k \oplus b_i b_j$. DILO generates the authenticated bit $[\tilde{b}_k] = (\mathsf{K_A}[\tilde{b}_k], \mathsf{M_B}[\tilde{b}_k], \tilde{b}_k)$ under Δ_A by computing authenticated values on $\tilde{b}_k$ and $\mathsf{M_B}[\tilde{b}_k]$ under Δ_B. Specifically, we have the following two parts:

- $\mathsf{P_B}$ computes the bit $\tilde{b}_k$ from the authenticated bit on $\tilde{b}_k$ under Δ_B and corresponding MAC tag sent by $\mathsf{P_A}$ in communication of $\rho + 1$ bits.

- $\mathsf{P_B}$ computes the MAC tag $\mathsf{M_B}[\tilde{b}_k]$ by generating the authenticated value on $\mathsf{M_B}[\tilde{b}_k]$ under Δ_B and corresponding MAC tag sent by $\mathsf{P_A}$ in communication of 4ρ bits.

We observe that the communication cost of the first part can be further reduced to only 2 bits by setting $\mathsf{lsb}(\Delta_\mathsf{B}) = 1$. In particular, $\mathsf{P_A}$ can send the LSB x_k of the MAC tag w.r.t. $[\tilde{b}_k]_{\Delta_\mathsf{B}}$ to $\mathsf{P_B}$ who can compute $\tilde{b}_k$ by XORing x_k with the LSB of the local key w.r.t. $[\tilde{b}_k]_{\Delta_\mathsf{B}}$. The authentication of $\{\tilde{b}_k\}$ can be done in a batch by hashing the MAC tags on these bits. However, the communication cost of the second part is inherent due to the DILO approach of generating the MAC tag $\mathsf{M_B}[\tilde{b}_k]$. This leaves us a challenge problem: *how to generate authenticated bit* $[\tilde{b}_k]_{\Delta_\mathsf{A}}$ *without the* ρ*-time blow-up in communication.*

The crucial point for solving the above problem is to generate the MAC tag $\mathsf{M_B}[\tilde{b}_k]$ with constant communication per triple. In a straightforward way, $\mathsf{P_A}$ and $\mathsf{P_B}$ can run the Fix procedure to generate $[\tilde{b}_k]_{\Delta_\mathsf{A}}$ by taking one-bit communication after $\mathsf{P_B}$ has obtained $\tilde{b}_k$. However, $\mathsf{P_A}$ has no way to check the correctness of $\tilde{b}_k$ implied in $[\tilde{b}_k]_{\Delta_\mathsf{A}}$, where $[\tilde{b}_k]_{\Delta_\mathsf{B}}$ generated by both parties only allow $\mathsf{P_B}$ to check the correctness of $\tilde{b}_k$. We introduce the notion of dual-key authentication to allow both parties to check the correctness of $\tilde{b}_k$, where the bit $\tilde{b}_k$ is authenticated under global key $\Delta_\mathsf{A} \cdot \Delta_\mathsf{B}$ and thus no party can change the bit $\tilde{b}_k$ without being detected. We present an efficient approach to generate the dual-key authenticated bit $\langle \tilde{b}_k \rangle$ with communication of only one bit. By checking the consistency of all values input to the block-COT functionality, we can guarantee the correctness of $\langle \tilde{b}_k \rangle$, i.e., $\tilde{b}_k$ is a valid bit authenticated by both parties. When setting $\mathsf{lsb}(\Delta_\mathsf{A} \cdot \Delta_\mathsf{B}) = 1$, $\mathsf{P_B}$ can obtain the bit $\tilde{b}_k$ by letting $\mathsf{P_A}$ send one-bit message to $\mathsf{P_B}$ (see below for details). By using Fix, $\mathsf{P_A}$ and $\mathsf{P_B}$ can generate $[\tilde{b}_k]$ under Δ_A. Now, $\mathsf{P_B}$ can check the correctness of $\tilde{b}_k$ obtained, and $\mathsf{P_A}$ can verify the correctness of $\tilde{b}_k$ implied in $[\tilde{b}_k]$, by using the correctness of $\langle \tilde{b}_k \rangle$. Particularly, we propose a batch-check technique that enables both parties to check the correctness of $\{\tilde{b}_k\}$ in all triples with essentially no communication. In addition, we present two new checking protocols to verify the correctness of global keys and the consistency of values across different functionalities (see below for an overview). Overall, our techniques allow to achieve one-way communication of only 2 bits per triple, and are described below.

Dual-key authentication. We propose the notion of dual-key authentication, meaning that a bit is authenticated by two global keys $\Delta_\mathsf{A}, \Delta_\mathsf{B} \in \mathbb{F}_{2^\kappa}$ held by $\mathsf{P_A}$ and $\mathsf{P_B}$ respectively. In particular, a dual-key authenticated bit $\langle x \rangle = (\mathsf{D_A}[x], \mathsf{D_B}[x], x)$ lets $\mathsf{P_A}$ hold $\mathsf{D_A}[x]$ and $\mathsf{P_B}$ hold $\mathsf{D_B}[x]$ such that $\mathsf{D_A}[x] + \mathsf{D_B}[x] = x \cdot \Delta_\mathsf{A} \cdot \Delta_\mathsf{B} \in \mathbb{F}_{2^\kappa}$, where $x \in \{0, 1\}$ can be known by either $\mathsf{P_A}$ or $\mathsf{P_B}$, or unknown for both parties. From the definition, we have that dual-key authenticated bits are also *additively homomorphic*, which enables us to use the random-linear-combination approach to perform consistency checks associated with such bits. We are also able to generalize dual-key authenticated bits to dual-key authenticated values in which x is defined over any field $\mathbb{F}$ and $\mathsf{D_A}[x], \mathsf{D_B}[x], \Delta_\mathsf{A}, \Delta_\mathsf{B}$ are defined over an extension field $\mathbb{K}$ with $\mathbb{F} \subseteq \mathbb{K}$. This generalization may be useful for the design of subsequent protocols. A useful property is that $\langle x \rangle$ can be *locally* converted into $[x\Delta_\mathsf{A}]_{\Delta_\mathsf{B}}$ or $[x\Delta_\mathsf{B}]_{\Delta_\mathsf{A}}$ and vice versa.

We consider that the bit x is shared as (a, b) with $x = a \wedge b$, where $\mathsf{P_A}$ holds $a \in \{0, 1\}$ and $\mathsf{P_B}$ holds $b \in \{0, 1\}$. Without loss of generality, we focus on the case that a is a secret bit. The bit b can be either a secret bit or a public bit 1, where the former means that no party knows x and the latter means that only $\mathsf{P_A}$ knows x. The DILO protocol [DILO22a] implicitly generates a dual-key authenticated bit by running $\mathsf{Fix}(a\Delta_\mathsf{A})$ w.r.t. global keys $b\Delta_\mathsf{B}+\gamma, \gamma$ to obtain $[a\Delta_\mathsf{A}]_{b\Delta_\mathsf{B}} = \langle ab \rangle = \langle x \rangle$, which incurs ρ-time blow-up in communication (even if a allows to be a random bit). Our approach can reduce the communication cost to at most one bit. In particular, we first let $\mathsf{P_A}$ and $\mathsf{P_B}$ generate a dual-key authenticated bit $\langle b \rangle = (\alpha, \beta)$ with $\alpha + \beta = b \cdot \Delta_\mathsf{A} \cdot \Delta_\mathsf{B} \in \mathbb{F}_{2^\kappa}$, where $\mathsf{P_A}$ gets α and $\mathsf{P_B}$ obtains β. Then, both parties initialize functionality $\mathcal{F}_\mathsf{bCOT}$ with a global key β. If $a \in \{0, 1\}$ allows to be random, both parties call $\mathcal{F}_\mathsf{bCOT}$ to generate $[a]_\beta$ without communication. Otherwise, both parties run Fix with input a to generate $[a]_\beta$ in communication of one bit. Given $[a]_\beta = (\mathsf{K_B}[a]_\beta, \mathsf{M_A}[a]_\beta, a)$, $\mathsf{P_A}$ and $\mathsf{P_B}$ can *locally* compute a dual-key authenticated bit $\langle a \rangle$ by letting $\mathsf{P_A}$ compute $\mathsf{D_A}[x] := \mathsf{M_A}[a]_\beta + a \cdot \alpha \in \mathbb{F}_{2^\kappa}$ and $\mathsf{P_B}$ set $\mathsf{D_B}[x] := \mathsf{K_B}[a]_\beta \in \mathbb{F}_{2^\kappa}$. We have that $\mathsf{D_A}[x] + \mathsf{D_B}[x] = (\mathsf{M_A}[a]_\beta + \mathsf{K_B}[a]_\beta) + a \cdot \alpha = a \cdot (\alpha + \beta) = a \cdot b \cdot \Delta_\mathsf{A} \cdot \Delta_\mathsf{B} \in \mathbb{F}_{2^\kappa}$. To guarantee correctness of $\langle x \rangle$, we need to check the consistency of β input to $\mathcal{F}_\mathsf{bCOT}$ and a input to Fix, which will be shown below.

Sampling global keys with correctness checking. As described above, we need to generate two global keys Δ_A and Δ_B such that $\mathsf{lsb}(\Delta_\mathsf{A} \cdot \Delta_\mathsf{B}) = 1$, which allows one party to get the bit $x = \mathsf{lsb}(\mathsf{D_A}[x]) \oplus \mathsf{lsb}(\mathsf{D_B}[x])$ from a dual-key authenticated bit $\langle x \rangle$. To do this, we let $\mathsf{P_A}$ sample $\Delta_\mathsf{A} \leftarrow \{0, 1\}^\kappa$ such that $\mathsf{lsb}(\Delta_\mathsf{A}) = 1$. Then, we let $\mathsf{P_B}$ sample $\Delta_\mathsf{B} \leftarrow \{0, 1\}^\kappa$, and make $\mathsf{P_A}$ and $\mathsf{P_B}$ run the Fix procedure w.r.t. Δ_A with input Δ_B to generate $[\Delta_\mathsf{B}]_{\Delta_\mathsf{A}}$ (i.e., $\langle 1 \rangle$), where $\alpha_0 \oplus \beta_0 = \Delta_\mathsf{A}\Delta_\mathsf{B}$. $\mathsf{P_A}$ and $\mathsf{P_B}$ can exchange $\mathsf{lsb}(\alpha_0)$ and $\mathsf{lsb}(\beta_0)$ to decide whether $\mathsf{lsb}(\alpha_0) \oplus \mathsf{lsb}(\beta_0) = 0$. If yes, then $\mathsf{lsb}(\Delta_\mathsf{A}\Delta_\mathsf{B}) = \mathsf{lsb}(\alpha_0) \oplus \mathsf{lsb}(\beta_0) = 0$. In this case, we let $\mathsf{P_B}$ update Δ_B as $\Delta_\mathsf{B} \oplus 1$, which makes $\Delta_\mathsf{A}\Delta_\mathsf{B}$ be updated as $\Delta_\mathsf{A}\Delta_\mathsf{B} \oplus \Delta_\mathsf{A}$, where $\mathsf{lsb}(\Delta_\mathsf{A}\Delta_\mathsf{B} \oplus \Delta_\mathsf{A}) = \mathsf{lsb}(\Delta_\mathsf{A}\Delta_\mathsf{B}) \oplus \mathsf{lsb}(\Delta_\mathsf{A}) = 1$. Since Δ_B is changed as $\Delta_\mathsf{B} \oplus 1$, α_0 needs to be updated as $\alpha_0 \oplus \Delta_\mathsf{A}$ in order to keep correct correlation.

While we adopt the KRRW authenticated garbling [KRRW18] in dual executions, some bit of global keys $\Delta_\mathsf{A}, \Delta_\mathsf{B} \in \{0, 1\}^\kappa$ is required to be fixed as 1. We often choose to define $\mathsf{lsb}(\Delta_\mathsf{A}) = 1$ and $\mathsf{lsb}(\Delta_\mathsf{B}) = 1$. While $\mathsf{lsb}(\Delta_\mathsf{A}) = 1$ has been satisfied, $\mathsf{lsb}(\Delta_\mathsf{B}) = 1$ does not always hold, as $\mathsf{P_B}$ may flip Δ_B depending on if $\mathsf{lsb}(\alpha_0) \oplus \mathsf{lsb}(\beta_0) = 0$. Thus, we let $\mathsf{P_B}$ set $\mathsf{msb}(\Delta_\mathsf{B}) = 1$ for ease of remembering. More importantly, $\mathsf{msb}(\Delta_\mathsf{B}) = 1$ has no impact on setting $\mathsf{lsb}(\Delta_\mathsf{A}\Delta_\mathsf{B}) = 1$.

To achieve active security, we need to guarantee that $\Delta_\mathsf{A} \cdot \Delta_\mathsf{B} \neq 0$ in the case that either $\mathsf{P_A}$ or $\mathsf{P_B}$ is corrupted. This can be assured by checking $\Delta_\mathsf{A} \neq 0$ and $\Delta_\mathsf{B} \neq 0$. We choose to check $\mathsf{lsb}(\Delta_\mathsf{A}) = 1$ and $\mathsf{msb}(\Delta_\mathsf{B}) = 1$ to realize the checking of $\Delta_\mathsf{A} \neq 0$ and $\Delta_\mathsf{B} \neq 0$. To enable $\mathsf{P_B}$ to check $\mathsf{lsb}(\Delta_\mathsf{A}) = 1$, both parties can generate random authenticated bits $[r_1]_\mathsf{B}, \ldots, [r_\rho]_\mathsf{B}$, and then $\mathsf{P_A}$ sends $\mathsf{lsb}(\mathsf{K_A}[r_i])$ for $i \in [1, \rho]$ to $\mathsf{P_B}$ who checks that $\mathsf{lsb}(\mathsf{K_A}[r_i]) \oplus \mathsf{lsb}(\mathsf{M_B}[r_i]) = r_i$ for all $i \in [1, \rho]$. A malicious $\mathsf{P_A}$ can cheat successfully if and only if it guesses correctly all random bits $r_1, \ldots, r_\rho$, which happens with probability $1/2^\rho$. The correctness check of $\mathsf{msb}(\Delta_\mathsf{B}) = 1$ can be done in a totally similar way. Furthermore, we need also to check $\mathsf{lsb}(\Delta_\mathsf{A}\Delta_\mathsf{B}) = 1$, and otherwise a selective failure attack may be performed on secret bit $\tilde{b}_k$. We first let $\mathsf{P_B}$ check $\mathsf{lsb}(\Delta_\mathsf{A}\Delta_\mathsf{B}) = 1$ by interacting with $\mathsf{P_A}$. We make $\mathsf{P_A}$ and $\mathsf{P_B}$ generate random dual-key authenticated bits $\langle s_1 \rangle, \ldots, \langle s_\rho \rangle$. Then, the check of $\mathsf{lsb}(\Delta_\mathsf{A}\Delta_\mathsf{B}) = 1$ can be done similarly, by letting $\mathsf{P_A}$ send $\mathsf{lsb}(\mathsf{D_A}[s_i])$ to $\mathsf{P_B}$ who checks that $\mathsf{lsb}(\mathsf{D_A}[s_i]) \oplus \mathsf{lsb}(\mathsf{D_B}[s_i]) = s_i$ for all $i \in [1, \rho]$. To produce $\langle s_1\rangle, \ldots, \langle s_\rho\rangle$, $\mathsf{P_A}$ and $\mathsf{P_B}$ can call $\mathcal{F}_{\mathsf{COT}}$ to generate random authenticated bits $[s_1]_{\Delta_\mathsf{A}}, \ldots, [s_\rho]_{\Delta_\mathsf{A}}$, and then run the Fix procedure w.r.t. Δ_A on input $(s_1\Delta_\mathsf{B}, \ldots, s_\rho\Delta_\mathsf{B})$ to generate $[s_1\Delta_\mathsf{B}]_{\Delta_\mathsf{A}}, \ldots, [s_\rho\Delta_\mathsf{B}]_{\Delta_\mathsf{A}}$ that are equivalent to $\langle s_1\rangle, \ldots, \langle s_\rho\rangle$. Then, the correctness of the input $(s_1\Delta_\mathsf{B}, \ldots, s_\rho\Delta_\mathsf{B})$ needs to be verified by $\mathsf{P_A}$ via letting $\mathsf{P_B}$ prove that $([s_i]_{\Delta_\mathsf{A}}, [\Delta_\mathsf{B}]_{\Delta_\mathsf{A}}, [s_i\Delta_\mathsf{B}]_{\Delta_\mathsf{A}})$ for all $i \in [1, \rho]$ satisfy the multiplication relationship using $\mathcal{F}_{\mathsf{DVZK}}$. Due to the dual execution, $\mathsf{P_A}$ needs also to symmetrically check $\mathsf{lsb}(\Delta_\mathsf{A}\Delta_\mathsf{B}) = 1$ by interacting with $\mathsf{P_B}$.

Generating compressed authenticated AND triples. As described above, for generating a compressed authenticated AND triple $([a_i], [b_i], [a_j], [b_j], [\hat{a}_k], [\hat{b}_k])$, the crucial step is to generate a dual-key authenticated bit $\langle \tilde{b}_k\rangle$ with $\tilde{b}_k = \hat{b}_k \oplus b_i b_j$. From the definition of $\tilde{b}_k$, we know that $\langle \tilde{b}_k\rangle = \langle a_{i,j}\rangle \oplus \langle a_i b_j\rangle \oplus \langle a_j b_i\rangle \oplus \langle \hat{a}_k\rangle$. We use the above approach to generate the dual-key authenticated bits $\langle a_{i,j}\rangle, \langle \hat{a}_k\rangle$ and $\langle a_i \boldsymbol{b}^*\rangle$ for $i \in [1, n]$ that can be locally converted to $\langle a_i b_j\rangle, \langle a_j b_i\rangle$ by multiplying a public matrix $\mathbf{M}$. Then, we combine all the dual-key authenticated bits to obtain $\langle \tilde{b}_k\rangle$. From $\mathsf{lsb}(\Delta_\mathsf{A}\Delta_\mathsf{B}) = 1$, we can let $\mathsf{P_A}$ send $\mathsf{lsb}(\mathsf{D_A}[\tilde{b}_k])$ to $\mathsf{P_B}$ who is able to recover $\tilde{b}_k = \mathsf{lsb}(\mathsf{D_A}[\tilde{b}_k]) \oplus \mathsf{lsb}(\mathsf{D_B}[\tilde{b}_k])$. By running the Fix procedure with input $\tilde{b}_k$, both parties can generate $[\tilde{b}_k]$, which can be in turn locally converted into $[\hat{b}_k]$. More details are shown as follows.

1. As in the DILO protocol [DILO22a], we let $\mathsf{P_A}$ and $\mathsf{P_B}$ obtain $[\boldsymbol{b}^*]$ and $\{[b_{i,j}]\}$ by calling $\mathcal{F}_{\mathsf{COT}}$ and running Fix with input $b_{i,j} = b_i b_j$. Then, both parties compute $[\boldsymbol{b}] := \mathbf{M} \cdot [\boldsymbol{b}^*]$ to obtain $[b_i], [b_j]$ for each AND gate $(i, j, k, \wedge)$.

2. $\mathsf{P_A}$ and $\mathsf{P_B}$ have produced $\langle 1\rangle = (\alpha_0, \beta_0)$ such that $\alpha_0 + \beta_0 = \Delta_\mathsf{A} \cdot \Delta_\mathsf{B} \in \mathbb{F}_{2^\kappa}$. For each $i \in [1, L]$, both parties can further generate a dual-key authenticated bit $\langle b_i^*\rangle = (\alpha_i, \beta_i)$ with $\alpha_i + \beta_i = b_i^* \cdot \Delta_\mathsf{A} \cdot \Delta_\mathsf{B} \in \mathbb{F}_{2^\kappa}$ by running Fix w.r.t. Δ_A with input $B_i^* = b_i^*\Delta_\mathsf{B}$. The communication to generate $\langle b_1^*\rangle, \ldots, \langle b_L^*\rangle$ is $L\kappa$ bits and logarithmic to the number n of AND gates due to $L = O(\rho \log(n/\rho))$.

3. $\mathsf{P_B}$ and $\mathsf{P_A}$ initialize $\mathcal{F}_{\mathsf{bCOT}}^{L+1}$ with global keys $\beta_1, \ldots, \beta_L, \Delta_\mathsf{B}$, and then call $\mathcal{F}_{\mathsf{bCOT}}^{L+1}$ to generate $[\boldsymbol{a}]_{\beta_1}, \ldots, [\boldsymbol{a}]_{\beta_L}$ and $[\boldsymbol{a}]_{\Delta_\mathsf{B}}$. For each tuple $([a_i]_{\beta_1}, \ldots, [a_i]_{\beta_L})$, we can convert it to $\langle a_i \boldsymbol{b}^*\rangle$. By multiplying the public matrix $\mathbf{M}$, both parties can obtain $\langle a_i b_j\rangle$ and $\langle a_j b_i\rangle$ for each AND gate $(i, j, k, \wedge)$. From $[\boldsymbol{a}]_{\Delta_\mathsf{B}}$, both parties directly obtain $[a_i], [a_j]$ for each AND gate $(i, j, k, \wedge)$.

4. $\mathsf{P_B}$ and $\mathsf{P_A}$ initialize $\mathcal{F}_{\mathsf{bCOT}}^{2}$ with global keys $\beta_0, \Delta_\mathsf{B}$, and then call $\mathcal{F}_{\mathsf{bCOT}}^{2}$ to generate $[\hat{\boldsymbol{a}}]_{\beta_0}$ and $[\hat{\boldsymbol{a}}]_{\Delta_\mathsf{B}}$. Both parties further run the Fix procedure with input $a_{i,j} = a_i \wedge a_j$ to generate $[a_{i,j}]_{\beta_0}$ and $[a_{i,j}]_{\Delta_\mathsf{B}}$, where $[a_{i,j}]_{\Delta_\mathsf{B}}$ will be used to prove validity of $a_{i,j}$. The parties can convert $[\hat{\boldsymbol{a}}]_{\beta_0}$ and $\{[a_{i,j}]_{\beta_0}\}$ into $\langle \hat{a}_k\rangle$ and $\langle a_{i,j}\rangle$ for each AND gate $(i, j, k, \wedge)$.

5. Both parties can *locally* compute $\langle \tilde{b}_k\rangle := \langle a_{i,j}\rangle \oplus \langle a_i b_j\rangle \oplus \langle a_j b_i\rangle \oplus \langle \hat{a}_k\rangle$. Then, $\mathsf{P_A}$ can send $\mathsf{lsb}(\mathsf{D_A}[\tilde{b}_k])$ to $\mathsf{P_B}$, who computes $\tilde{b}_k := \mathsf{lsb}(\mathsf{D_A}[\tilde{b}_k]) \oplus \mathsf{lsb}(\mathsf{D_B}[\tilde{b}_k])$ due to $\mathsf{lsb}(\Delta_\mathsf{A}\Delta_\mathsf{B}) = 1$. Both parties run Fix on input $\tilde{b}_k$ to generate $[\tilde{b}_k]$.

6. $\mathsf{P_A}$ and $\mathsf{P_B}$ can *locally* compute $[\hat{b}_k] := [\tilde{b}_k] \oplus [b_{i,j}]$. Now, the parties hold $([a_i], [b_i], [a_j], [b_j], [\hat{a}_k], [\hat{b}_k])$ for each AND gate $(i, j, k, \wedge)$.

Consistency check. We have shown how to generate compressed authenticated AND triples. Below, we show how to verify their correctness. We only need to guarantee the consistency of all Fix inputs, all global keys input to the bCOT functionality and all bits sent by $\mathsf{P_A}$ to $\mathsf{P_B}$. When all messages and inputs are consistent, no malicious party can break the correctness of all triples. Specifically, we present the following checks to guarantee the consistency.

1. Check the correctness of the following authenticated AND triples:
   - $([b_i], [b_j], [b_{i,j}])$ *s.t.* $b_{i,j} = b_i \wedge b_j$ for each AND gate $(i, j, k, \wedge)$.
   - $([a_i], [a_j], [a_{i,j}])$ *s.t.* $a_{i,j} = a_i \wedge a_j$ for each AND gate $(i, j, k, \wedge)$.
   - $([b_i^*], [\Delta_{\mathsf{B}}], [B_i^*])$ *s.t.* $B_i^* = b_i^* \cdot \Delta_{\mathsf{B}}$ for each $i \in [1, L]$.
2. The keys $\beta_0, \beta_1, \ldots, \beta_L$ input to functionality $\mathcal{F}_{\mathsf{bCOT}}$ are consistent to the values defined in $\langle 1 \rangle, \langle b_1^* \rangle, \ldots, \langle b_L^* \rangle$.
3. $\mathsf{P_A}$ needs to check that two global keys $\Delta_{\mathsf{B}}^{(1)}$ and $\Delta_{\mathsf{B}}^{(2)}$ respectively input to functionalities $\mathcal{F}_{\mathsf{bCOT}}^{L+1}$ and $\mathcal{F}_{\mathsf{bCOT}}^{2}$ are consistent with Δ_{B} defined in $\langle 1 \rangle$.
4. $\mathsf{P_A}$ checks that the bit $\tilde{b}_k$ defined in $[\tilde{b}_k]$ is consistent to that defined in $\langle \tilde{b}_k \rangle$, and $\mathsf{P_B}$ checks that $\tilde{b}_k$ computed by itself is consistent to that defined in $\langle \tilde{b}_k \rangle$.

The first two checks guarantee the correctness of $\langle \tilde{b}_k \rangle$ and $[b_{i,j}]$, the third check verifies the consistency of the global keys in $[a_i], [a_j], [\hat{a}_k]$, and the final check assure the consistency of bits authenticated between $\langle \tilde{b}_k \rangle$ and $[\tilde{b}_k]$. Check 1 can be directly realized by calling functionality $\mathcal{F}_{\mathsf{DVZK}}$.

For Check 2, for each $i \in [0, L]$, we let $\mathsf{P_A}$ and $\mathsf{P_B}$ run the Fix procedure w.r.t. β_i on input Δ'_{A} to generate $[\Delta'_{\mathsf{A}}]_{\beta_i}$, which can be locally converted into $[\beta_i]_{\Delta'_{\mathsf{A}}}$, where $\Delta'_{\mathsf{A}} \in \mathbb{F}_{2^\kappa}$ is sampled uniformly at random by $\mathsf{P_A}$.[2] For $i \in [0, L]$, we present a new protocol to verify the consistency of β_i in the following equations:

$$\alpha_i + \beta_i = b_i^* \cdot \Delta_{\mathsf{A}} \cdot \Delta_{\mathsf{B}},$$
$$\mathsf{K}'_{\mathsf{A}}[\beta_i] + \mathsf{M}'_{\mathsf{A}}[\beta_i] = \beta_i \cdot \Delta'_{\mathsf{A}},$$

where b_0^* is defined as 1. We first multiply two sides of the first equation by Δ_{A}^{-1}, and obtain $\alpha_i \cdot \Delta_{\mathsf{A}}^{-1} + \beta_i \cdot \Delta_{\mathsf{A}}^{-1} = b_i^* \cdot \Delta_{\mathsf{B}}$. We rewrite the resulting equation as $\mathsf{K_A}[\beta_i] + \mathsf{M_B}[\beta_i] = \beta_i \cdot \Delta_{\mathsf{A}}^{-1}$ where $\mathsf{K_A}[\beta_i] = \alpha_i \cdot \Delta_{\mathsf{A}}^{-1}$ and $\mathsf{M_B}[\beta_i] = b_i^* \cdot \Delta_{\mathsf{B}}$. Below, we can adapt the known techniques [DIO21, DILO22a] to check the consistency of β_i authenticated under different global keys (i.e., $[\beta_i]_{\Delta_{\mathsf{A}}^{-1}}$ and $[\beta_i]_{\Delta'_{\mathsf{A}}}$) in a batch (see Section 4.3 for details).

For Check 3, we make $\mathsf{P_A}$ and $\mathsf{P_B}$ run the Fix procedure w.r.t. $\Delta_{\mathsf{B}}^{(1)}$ (resp., $\Delta_{\mathsf{B}}^{(2)}$) on input Δ'_{A} to obtain $[\Delta_{\mathsf{B}}^{(1)}]_{\Delta'_{\mathsf{A}}}$ (resp., $[\Delta_{\mathsf{B}}^{(2)}]_{\Delta'_{\mathsf{A}}}$). Authenticated values $[\Delta_{\mathsf{B}}^{(1)}]_{\Delta'_{\mathsf{A}}}$ and $[\Delta_{\mathsf{B}}^{(2)}]_{\Delta'_{\mathsf{A}}}$ are equivalent to $\langle 1_{\mathsf{B}}^{(1)} \rangle$ and $\langle 1_{\mathsf{B}}^{(2)} \rangle$ where $\Delta_{\mathsf{B}}^{(1)} \Delta'_{\mathsf{A}}$ and $\Delta_{\mathsf{B}}^{(2)} \Delta'_{\mathsf{A}}$ are used as the global keys in dual-key authentication. Both parties can invoke a relaxed equality-check functionality $\mathcal{F}_{\mathsf{EQ}}$ (shown in Appendix A) to check $1_{\mathsf{B}}^{(1)} - 1_{\mathsf{B}}^{(2)} = 0$. Using the checking technique by Dittmer et al. [DILO22a], we can also check the consistency of the values authenticated between $[\Delta_{\mathsf{B}}^{(1)}]_{\Delta'_{\mathsf{A}}}$ and $[\Delta_{\mathsf{B}}]_{\Delta_{\mathsf{A}}}$ generated during the sampling phase.

For Check 4, we use a random-linear-combination approach to perform the check in a batch. Specifically, we can let $\mathsf{P_A}$ and $\mathsf{P_B}$ call $\mathcal{F}_{\mathsf{COT}}$ to generate $[\boldsymbol{r}]_{\mathsf{B}}$ and then obtain $[r]_{\mathsf{B}} \leftarrow \mathsf{B2F}([\boldsymbol{r}]_{\mathsf{B}})$, where $r \in \mathbb{F}_{2^\kappa}$ is uniform. Then, both parties run Fix w.r.t. Δ_{A} on input $r\Delta_{\mathsf{B}}$ to generate $[r\Delta_{\mathsf{B}}]_{\Delta_{\mathsf{A}}}$ (i.e., $\langle r \rangle$). We can let the parties call a standard coin-tossing functionality $\mathcal{F}_{\mathsf{Rand}}$ to sample a random element $\chi \in \mathbb{F}_{2^\kappa}$. Then, both parties can locally compute $\langle y \rangle := \sum \chi^k \cdot \langle \tilde{b}_k \rangle + \langle r \rangle$ and

[2] An independent global key Δ'_{A} is necessary to perform the consistency check, and otherwise a malicious $\mathsf{P_B}$ will always pass the check if Δ_{A} is reused.

$[y]_\mathsf{B} := \sum \chi^k \cdot [\tilde{b}_k]_\mathsf{B} + [r]_\mathsf{B}$. Then, $\mathsf{P_B}$ can open $[y]_\mathsf{B}$ that allows $\mathsf{P_A}$ to get y in an authenticated way. Finally, both parties can use $\mathcal{F}_\mathsf{EQ}$ to verify that the opening of $\langle y \rangle - y \cdot \langle 1 \rangle$ is 0. Since χ is sampled uniformly at random after all authenticated values are determined, the consistency check will detect malicious behaviors except with probability at most $n/2^\kappa$.

3.3 Our Solution for Dual Execution without Leakage

While the evaluator's random masks are compressed, the state-of-the-art construction of authenticated garbling based on half-gates by Katz et al. [KRRW18] is no longer applied. The circuit authentication approach in [KRRW18] requires the evaluator to reveal all masked wire values, which is prohibitive for the compression technique. Therefore, based on the technique [WRK17a], Dittmer et al. [DILO22a] designed a new construction of authenticated garbling without revealing masked wire values. However, this construction incurs extra communication overhead of $3\rho - 1$ bits per AND gate, compared to the half-gates-based construction [KRRW18].

In duplex networks, communication cost is often measured by one-way communication. This allows us to adopt the idea of dual execution [MF06] to perform the authentication of circuit evaluation. In the original dual execution [MF06], the semi-honest Yao-2PC protocol [Yao86] is executed two times with the same inputs in parallel by swapping the roles of parties for the second execution, and then the correctness of the output is verified by checking that the two executions have the same output bits. However, an inherent problem of the above method is that selective failure attacks are allowed to leak one-bit information of the input by the honest party, even though there exists a protocol to check the consistency of inputs in two executions. For example, suppose that $\mathsf{P_A}$ is honest and $\mathsf{P_B}$ is malicious. When $\mathsf{P_A}$ is a garbler and $\mathsf{P_B}$ is an evaluator, both parties compute an output $f(x, y)$ where x is the $\mathsf{P_A}$'s input and y is the $\mathsf{P_B}$'s input. After swapping the roles, they compute another output $g(x, y)$ with $g \neq f$, as garbler $\mathsf{P_B}$ is malicious. If the output-equality check passes, then $g(x, y) = f(x, y)$, else $g(x, y) \neq f(x, y)$. In both cases, this leaks one-bit information on the input x.

In the authenticated garbling framework, we propose a new technique to circumvent the problem and eliminate the one-bit leakage. Together with our technique to generate compressed authenticated AND triples, we can achieve the cost of one-way communication that is almost the same as the semi-honest half-gates protocol [ZRE15]. Specifically, we let $\mathsf{P_A}$ and $\mathsf{P_B}$ execute the protocol, which combines the sub-protocol of generating authenticated AND triples as described above with the construction of distributed garbling [KRRW18], for two times with same inputs in the dual-execution way. For each wire w in the circuit, we need to check that the actual values z_w and z'_w in two executions are identical. We perform the checking by verifying $z_w \cdot (\Delta_\mathsf{A} \oplus \Delta_\mathsf{B}) = z'_w \cdot (\Delta_\mathsf{A} \oplus \Delta_\mathsf{B})$. Since $\Delta_\mathsf{A} \oplus \Delta_\mathsf{B}$ is unknown for the adversary, the probability that $z_w \neq z'_w$ but the check passes is negligible. Our approach allows two parties to check the correctness of all wire values in the circuit, and thus prevents selective failure attacks.

In more detail, for each wire w, let Λ_w and (a_w, b_w) be the masked value and wire masks in the first execution and (Λ'_w, a'_w, b'_w) be the values in the second execution. Thus, $\mathsf{P_A}$ and $\mathsf{P_B}$ need to check that $\Lambda_w \oplus a_w \oplus b_w = \Lambda'_w \oplus a'_w \oplus b'_w$ for each wire w, where the output wires of XOR gates are unnecessary to be checked as they are locally computed. Below, our task is to check that $(\Lambda_w \oplus a_w \oplus b_w) \cdot (\Delta_\mathsf{A} \oplus \Delta_\mathsf{B}) = (\Lambda'_w \oplus a'_w \oplus b'_w) \cdot (\Delta_\mathsf{A} \oplus \Delta_\mathsf{B})$ holds for each wire w. By two protocol executions, both parties hold $([a_w], [b_w], [a'_w], [b'_w])$ for each wire w. When $\mathsf{P_A}$ is a garbler and $\mathsf{P_B}$ is an evaluator, $\mathsf{P_A}$ holds a garbled label $\mathsf{L}_{w,0}$ and $\mathsf{P_B}$ holds $(\Lambda_w, \mathsf{L}_{w,\Lambda_w})$. Since $\mathsf{L}_{w,\Lambda_w} = \mathsf{L}_{w,0} \oplus \Lambda_w \Delta_\mathsf{A}$ has the form of IT-MACs, we can view $(\mathsf{L}_{w,0}, \mathsf{L}_{w,\Lambda_w}, \Lambda_w)$ as an authenticated bit $[\Lambda_w]_\mathsf{B}$, where $\mathsf{L}_{w,0}$ is considered as the local key and L_{w,Λ_w} plays the role of MAC tag. Similarly, when $\mathsf{P_A}$ is an evaluator and $\mathsf{P_B}$ is a garbler, two parties hold an authenticated bit $[\Lambda'_w]_\mathsf{A}$. Following the known observation (e.g., [KRRW18]), for any authenticated bit $[y]_{\mathsf{B}}$, $\mathsf{P_A}$ and $\mathsf{P_B}$ have an additive sharing of $y \cdot \Delta_{\mathsf{A}} = \mathsf{K_A}[y] \oplus \mathsf{M_B}[y]$. Therefore, for all cross terms, both parties can obtain their additive shares, and then can compute two values that are checked to be identical. In particular, both parties can compute the additive shares of all cross terms: $Z^{\mathsf{A}}_{w,1} \oplus Z^{\mathsf{B}}_{w,1} = \Lambda_w \Delta_{\mathsf{A}}, Z^{\mathsf{A}}_{w,2} \oplus Z^{\mathsf{B}}_{w,2} = \Lambda'_w \Delta_{\mathsf{B}}, Z^{\mathsf{A}}_{w,3} \oplus Z^{\mathsf{B}}_{w,3} = a_w \Delta_{\mathsf{B}}, Z^{\mathsf{A}}_{w,4} \oplus Z^{\mathsf{B}}_{w,4} = a'_w \Delta_{\mathsf{B}}, Z^{\mathsf{A}}_{w,5} \oplus Z^{\mathsf{B}}_{w,5} = b_w \Delta_{\mathsf{A}}, Z^{\mathsf{A}}_{w,6} \oplus Z^{\mathsf{B}}_{w,6} = b'_w \Delta_{\mathsf{A}}$. Then, for each wire w, $\mathsf{P_A}$ and $\mathsf{P_B}$ can respectively compute

$$
\begin{aligned}
V^{\mathsf{A}}_w &= (\oplus_{i \in [1,6]} Z^{\mathsf{A}}_{w,i}) \oplus a_w \Delta_{\mathsf{A}} \oplus \Lambda'_w \Delta_{\mathsf{A}} \oplus a'_w \Delta_{\mathsf{A}} \\
V^{\mathsf{B}}_w &= (\oplus_{i \in [1,6]} Z^{\mathsf{B}}_{w,i}) \oplus b_w \Delta_{\mathsf{B}} \oplus \Lambda_w \Delta_{\mathsf{B}} \oplus b'_w \Delta_{\mathsf{B}},
\end{aligned}
$$

such that $V^{\mathsf{A}}_w = V^{\mathsf{B}}_w$. Without loss of generality, we assume that only $\mathsf{P_B}$ obtains the output, and thus only $\mathsf{P_B}$ needs to check the correctness of all masked values. In this case, we make $\mathsf{P_A}$ send the hash value of all V^{A}_w to $\mathsf{P_B}$, who can check its correctness with V^{B}_w for each wire w.

Optimizations for processing inputs. Dittmer et al. [DILO22a] consider that the wire masks (i.e., $\boldsymbol{b}_{\mathcal{I}}$) on all wires in $\mathcal{I}_{\mathsf{B}}$ held by evaluator $\mathsf{P_B}$ is uniformly random and authenticated AND triples associated with $\boldsymbol{b}_{\mathcal{I}}$ are generated using the previous approach (e.g., [KRRW18]). This will require an independent preprocessing protocol, and also brings more preprocessing communication cost. We solve the problem by specially processing the input of evaluator $\mathsf{P_B}$. In particular, instead of making $\mathsf{P_B}$ send masked value $\Lambda_w := y_w \oplus b_w$ for each $w \in \mathcal{I}_{\mathsf{B}}$ to $\mathsf{P_A}$ where y_w is the input bit, we use an OT protocol to transmit L_{w,Λ_w} to $\mathsf{P_B}$. This allows to keep masked wire values $\Lambda_w := y_w \oplus b_w$ for all $w \in \mathcal{I}_{\mathsf{B}}$ secret. In this case, we can compress the wire masks using the technique as described in Section 3.2 and adopt the same preprocessing protocol to handle $\boldsymbol{b}_{\mathcal{I}}$. Since L is logarithm to the length n of vector $\boldsymbol{b}$ (now $n = |\mathcal{W}| + |\mathcal{I}_{\mathsf{B}}|$), this optimization essentially incurs no more overhead for the preprocessing phase. Furthermore, our preprocessing protocol to generate authenticated AND triples has already invoked functionality $\mathcal{F}_{\mathsf{COT}}$. Therefore, we can let two parties call $\mathcal{F}_{\mathsf{COT}}$ to generate random COT correlations in the preprocessing phase, and then transform them to OT correlations in the standard way. This essentially brings no more communication for the preprocessing phase, due to the sublinear communication of the recent protocols instantiating $\mathcal{F}_{\mathsf{COT}}$. Our optimization does not increase the rounds of online phase. As a trade-off, this optimization increases the online communication cost by $|\mathcal{I}_{\mathsf{B}}| \cdot \kappa$ bits.

In the second protocol execution (i.e., $\mathsf{P_A}$ as an evaluator and $\mathsf{P_B}$ as a garbler), we make a further optimization to directly guarantee that the masked values on all circuit-input wires are XOR of actual values and wire masks. In this case, it is unnecessary to check the correctness of masked values on all circuit-input wires between two protocol executions. The key idea is to utilize the authenticated bits and messages on circuit-input wires generated/sent during the first protocol execution along with the authenticated bits produced in the second protocol execution to generate the masked values on the wires in $\mathcal{I}_{\mathsf{A}} \cup \mathcal{I}_{\mathsf{B}}$. Due to the security of IT-MACs, we can guarantee the correctness of these masked values in the second execution. We postpone the details of this optimization to Section 5.

4 Preprocessing with Compressed Wire Masks

In this section we introduce the compressed preprocessing functionality $\mathcal{F}_{\mathsf{cpre}}$ (shown in Figure 3) for two party computation as well as an efficient protocol Π_{cpre} (shown in Figure 5 and Figure 6) to realize it. In a modular fashion we first introduce the sub-components which are called in the main preprocessing protocol. The security of the protocol is also argued similarly: we first prove

Functionality $\mathcal{F}_{\mathsf{cpre}}$

This functionality is parameterized by a Boolean circuit $\mathcal{C}$ consisting of a list of gates in the form of (i, j, k, T). Let $n := |\mathcal{W}| + |\mathcal{I}_{\mathsf{B}}|$ (resp., $m := |\mathcal{W}| + |\mathcal{I}_{\mathsf{A}}|$) be the number of all AND gates as well as circuit-input gates corresponding to the input of $\mathsf{P_B}$ (resp., $\mathsf{P_A}$), and $L = \lceil \rho \log \frac{2en}{\rho} + \frac{\log \rho}{2} \rceil$ be a compression parameter. It runs with parties $\mathsf{P_A}$, $\mathsf{P_B}$ and the ideal-world adversary $\mathcal{S}$, and operates as follows:

Initialize. Sample two global keys $\Delta_{\mathsf{A}}, \Delta_{\mathsf{B}} \in \mathbb{F}_{2^\kappa}$ as follows:

- If $\mathsf{P_A}$ is honest, sample $\Delta_{\mathsf{A}} \leftarrow \mathbb{F}_{2^\kappa}$ such that $\mathsf{lsb}(\Delta_{\mathsf{A}}) = 1$. Otherwise, receive $\Delta_{\mathsf{A}} \in \mathbb{F}_{2^\kappa}$ with $\mathsf{lsb}(\Delta_{\mathsf{A}}) = 1$ from $\mathcal{S}$.
- If $\mathsf{P_B}$ is honest, sample $\Delta_{\mathsf{B}} \leftarrow \mathbb{F}_{2^\kappa}$ such that $\mathsf{lsb}(\Delta_{\mathsf{A}}\Delta_{\mathsf{B}}) = 1$ and $\mathsf{msb}(\Delta_{\mathsf{B}}) = 1$. Otherwise, receive $\Delta_{\mathsf{B}} \in \mathbb{F}_{2^\kappa}$ with $\mathsf{msb}(\Delta_{\mathsf{B}}) = 1$ from $\mathcal{S}$, and then re-sample $\Delta_{\mathsf{A}} \leftarrow \mathbb{F}_{2^\kappa}$ such that $\mathsf{lsb}(\Delta_{\mathsf{A}}\Delta_{\mathsf{B}}) = 1$ and $\mathsf{lsb}(\Delta_{\mathsf{A}}) = 1$.
- Store $(\Delta_{\mathsf{A}}, \Delta_{\mathsf{B}})$, and output Δ_{A} and Δ_{B} to $\mathsf{P_A}$ and $\mathsf{P_B}$, respectively.

Macro. $\mathsf{Auth_A}(\boldsymbol{x}, \ell)$ (this is an internal subroutine only)

- If $\mathsf{P_B}$ is honest, sample $\mathsf{K_B}[\boldsymbol{x}] \leftarrow \mathbb{F}_{2^\kappa}^{\ell}$; otherwise, receive $\mathsf{K_B}[\boldsymbol{x}] \in \mathbb{F}_{2^\kappa}^{\ell}$ from $\mathcal{S}$.
- If $\mathsf{P_A}$ is honest, compute $\mathsf{M_A}[\boldsymbol{x}] := \mathsf{K_B}[\boldsymbol{x}] + \boldsymbol{x} \cdot \Delta_{\mathsf{B}} \in \mathbb{F}_{2^\kappa}^{\ell}$. Otherwise, receive $\mathsf{M_A}[\boldsymbol{x}] \in \mathbb{F}_{2^\kappa}^{\ell}$ from $\mathcal{S}$, and recompute $\mathsf{K_B}[\boldsymbol{x}] := \mathsf{M_A}[\boldsymbol{x}] + \boldsymbol{x} \cdot \Delta_{\mathsf{B}} \in \mathbb{F}_{2^\kappa}^{\ell}$.
- Send $(\boldsymbol{x}, \mathsf{M_A}[\boldsymbol{x}])$ to $\mathsf{P_A}$ and $\mathsf{K_B}[\boldsymbol{x}]$ to $\mathsf{P_B}$.

$\mathsf{Auth_B}(\boldsymbol{x}, \ell)$ can be defined similarly by swapping the roles of $\mathsf{P_A}$ and $\mathsf{P_B}$.

Preprocess the circuit with compressed wire masks. Sample $\mathbf{M} \leftarrow \mathbb{F}_2^{n \times L}$, and then execute as follows:

- For $w \in \mathcal{I}_{\mathsf{A}}$, set $b_w = 0$ and define $[b_w]$; for $w \in \mathcal{I}_{\mathsf{B}}$, set $a_w = 0$ and define $[a_w]$.
- If $\mathsf{P_A}$ is honest, sample $\boldsymbol{a} \leftarrow \mathbb{F}_2^m$; otherwise, receive $\boldsymbol{a} \in \mathbb{F}_2^m$ from $\mathcal{S}$. Then, execute $\mathsf{Auth_A}(\boldsymbol{a}, m)$ to generate $[\boldsymbol{a}]$. For each wire $w \in \mathcal{I}_{\mathsf{A}} \cup \mathcal{W}$, define a_w as the wire mask held by $\mathsf{P_A}$.
- If $\mathsf{P_B}$ is honest, sample $\boldsymbol{b}^* \leftarrow \mathbb{F}_2^L$; otherwise, receive $\boldsymbol{b}^* \in \mathbb{F}_2^L$ from $\mathcal{S}$. Run $\mathsf{Auth_B}(\boldsymbol{b}^*, L)$ to generate $[\boldsymbol{b}^*]$, and then compute $[\boldsymbol{b}] := \mathbf{M} \cdot [\boldsymbol{b}^*]$ with $\boldsymbol{b} \in \mathbb{F}_2^n$. For each wire $w \in \mathcal{I}_{\mathsf{B}} \cup \mathcal{W}$, define b_w as the wire mask held by $\mathsf{P_B}$.
- In a topological order, for each gate (i, j, k, T), do the following:
  - If $T = \oplus$, compute $[a_k] := [a_i] \oplus [a_j]$ and $[b_k] := [b_i] \oplus [b_j]$.
  - If $T = \wedge$, execute as follows:
    1. If $\mathsf{P_A}$ is honest, then sample $\hat{a}_k \leftarrow \{0, 1\}$, else receive $\hat{a}_k \in \{0, 1\}$ from $\mathcal{S}$.
    2. If $\mathsf{P_B}$ is honest, then compute $\hat{b}_k := (a_i \oplus b_i) \wedge (a_j \oplus b_j) \oplus \hat{a}_k$. Otherwise, receive $\hat{b}_k \in \{0, 1\}$ from $\mathcal{S}$, and re-compute $\hat{a}_k := (a_i \oplus b_i) \wedge (a_j \oplus b_j) \oplus \hat{b}_k$.

  Let $\hat{\boldsymbol{a}}$ and $\hat{\boldsymbol{b}}$ be the vectors consisting of bits $\hat{a}_k$ and $\hat{b}_k$ for $k \in \mathcal{W}$. Run $\mathsf{Auth_A}(\hat{\boldsymbol{a}})$ and $\mathsf{Auth_B}(\hat{\boldsymbol{b}})$ to generate $[\hat{\boldsymbol{a}}]$ and $[\hat{\boldsymbol{b}}]$, respectively.
- Output $\mathbf{M}$ and $([\boldsymbol{a}], [\hat{\boldsymbol{a}}], [\boldsymbol{b}^*], [\hat{\boldsymbol{b}}])$ to $\mathsf{P_A}$ and $\mathsf{P_B}$.

Figure 3: Compressed preprocessing functionality for authenticated triples.

in separate lemmas the respective security properties of sub-components and then utilize these lemmas to prove the main theorem.

4.1 Dual-Key Authentication

In this subsection we define the format of dual-key authentication and list some of its properties that we utilize in the upper level preprocessing protocol.

Definition 1. *We use the notation* $\langle x \rangle := (\mathsf{D_A}[x], \mathsf{D_B}[x], x)$ *to denote the dual-key authenticated value* x*, where* $\mathsf{P_A}, \mathsf{P_B}$ *holds* $\mathsf{D_A}[x], \mathsf{D_B}[x]$ *subject to* $\mathsf{D_A}[x] + \mathsf{D_B}[x] = x\Delta_\mathsf{A}\Delta_\mathsf{B}$ *and* $\Delta_\mathsf{A}, \Delta_\mathsf{B}$ *are the IT-MAC keys of* $\mathsf{P_A}, \mathsf{P_B}$ *respectively.*

We remark that for any $x \in \mathbb{F}_{2^\kappa}$ the IT-MAC authentication $[x\Delta_\mathsf{A}]_{\Delta_\mathsf{B}}$ can be locally transformed to $\langle x \rangle$, which we summarize in the following macro (the case for $[\Delta_\mathsf{B}]_{\Delta_\mathsf{A}}$ can be defined analogously). In particular, by computing $[\Delta_\mathsf{B}]_{\Delta_\mathsf{A}}$ we implicitly have $\langle 1 \rangle$, i.e., authentication of the constant $1 \in \mathbb{F}_{2^\kappa}$.

- $\langle x \rangle \leftarrow \mathsf{Convert1}_{[\cdot]\to\langle\cdot\rangle}([x\Delta_\mathsf{B}]_{\Delta_\mathsf{A}})$: Set $\mathsf{D_A}[x] := \mathsf{M_A}[x\Delta_\mathsf{B}]$ and $\mathsf{D_B}[x] := \mathsf{K_B}[x\Delta_\mathsf{B}]$.

For the ease of presentation, we also define the following macro that generates dual key authentication of cross terms $\langle xy \rangle$ assuming the existence of $\langle y \rangle := (\alpha, \beta)$ and $[x]_{\mathsf{A},\beta} = (\mathsf{K_B}[x]_\beta, \mathsf{M_A}[x]_\beta, x)$. The correctness can be verified straightforwardly.

- $\langle xy \rangle \leftarrow \mathsf{Convert2}_{[\cdot]\to\langle\cdot\rangle}([x]_{\mathsf{A},\beta}, \langle y \rangle)$: Given IT-MAC $[x]_{\mathsf{A},\beta}$ and dual-key authentication $\langle y \rangle$, $\mathsf{P_A}$ and $\mathsf{P_B}$ *locally* compute the following steps:
  - $\mathsf{P_A}$ outputs $\mathsf{D_A}[xy] := \alpha \cdot x + \mathsf{M_A}[x]_\beta \in \mathbb{F}_{2^\kappa}$.
  - $\mathsf{P_B}$ outputs $\mathsf{D_B}[xy] := \mathsf{K_B}[x]_\beta$.

In our protocol we utilize the following properties of dual key authentication. Since they are straightforward we only provide brief explanation and refrain from providing detailed description.

Claim 1. *The dual-key authentication is additively homomorphic. In particular, given* $\langle x_1 \rangle := (\mathsf{D_A}[x_1], \mathsf{D_B}[x_1])$ *and* $\langle x_2 \rangle := (\mathsf{D_A}[x_2], \mathsf{D_B}[x_2])$*,* $\mathsf{P_A}, \mathsf{P_B}$ *can locally compute* $\langle x_1 + x_2 \rangle := (\mathsf{D_A}[x_1] + \mathsf{D_A}[x_2], \mathsf{D_B}[x_1] + \mathsf{D_B}[x_2])$*.*

The additive homomorphism of dual-key authentication implies that given public coefficients $c_0, c_1, \ldots, c_\ell \in \mathbb{F}_{2^\kappa}$, two parties can *locally* compute $\langle y \rangle := c_0 + \sum_{i=1}^{\ell} c_i \cdot \langle x_i \rangle$.

We define the zero-checking macro $\mathsf{CheckZero2}$ which ensures soundness for both parties. We note that this is simply the equality checking operations.

- $\mathsf{CheckZero2}(\langle x_1 \rangle, \ldots \langle x_\ell \rangle)$: On input dual-key authenticated values $\langle x_1 \rangle, \ldots \langle x_\ell \rangle$ both parties check $x_i = 0$ for $i \in [1, \ell]$ as follows:
  1. $\mathsf{P_A}$ computes $h_\mathsf{A} := \mathsf{H}(\mathsf{D_A}[x_1], \ldots, \mathsf{D_A}[x_\ell])$, and $\mathsf{P_B}$ sets $h_\mathsf{B} := \mathsf{H}(\mathsf{D_B}[x_1], \ldots, \mathsf{D_B}[x_\ell])$, where $\mathsf{H} : \{0,1\}^* \to \{0,1\}^\kappa$ is a random oracle.
  2. Both parties call functionality $\mathcal{F}_\mathsf{EQ}$ to check $h_\mathsf{A} = h_\mathsf{B}$. If $\mathcal{F}_\mathsf{EQ}$ outputs false, the parties abort.

Notice that the additive homomorphic and zero-checking properties allow us to check that a dual-key authenticated value $\langle x \rangle$ matches a public value x' assuming the existence of $\langle 1 \rangle = (\mathsf{D_A}[1], \mathsf{D_B}[1])$ by calling $\mathsf{CheckZero2}(\langle x \rangle - x' \langle 1 \rangle)$. Similar to $\mathsf{CheckZero}$ we have the following soundness lemma of $\mathsf{CheckZero2}$.

Lemma 2. *If* $\Delta_\mathsf{A}, \Delta_\mathsf{B} \in \mathbb{F}_{2^\kappa}$ *is sampled uniformly at random and are non-zero, then the probability that there exists some* $i \in [1, \ell]$ *such that* $x_i \neq 0$ *and* $\mathsf{P_A}$ *or* $\mathsf{P_B}$ *accepts in the* $\mathsf{CheckZero2}$ *procedure is bounded by* $\frac{2}{2^\kappa}$*.*

Protocol Π_{samp}

$\mathsf{P_A}$ samples $\Delta_\mathsf{A} \leftarrow \mathbb{F}_{2^\kappa}$ such that $\mathsf{lsb}(\Delta_\mathsf{A}) = 1$. $\mathsf{P_B}$ samples $\tilde{\Delta}_\mathsf{B} \leftarrow \mathbb{F}_{2^\kappa}$ such that $\mathsf{msb}(\tilde{\Delta}_\mathsf{B}) = 1$. Then, $\mathsf{P_A}$ and $\mathsf{P_B}$ execute the following steps.

1. $\mathsf{P_A}$ and $\mathsf{P_B}$ call functionality $\mathcal{F}_\mathsf{COT}$ on respective input $(\mathsf{init}, sid_0, \Delta_\mathsf{A})$ and (init, sid_0), and then call $\mathcal{F}_\mathsf{COT}$ on the same input $(\mathsf{extend}, sid_0, \rho)$ to generate random authenticated bits $[\boldsymbol{u}]_\mathsf{B}$.
2. Then $\mathsf{P_A}$ convinces $\mathsf{P_B}$ that $\mathsf{lsb}(\Delta_\mathsf{A}) = 1$ by sending a ρ-bit vector $m_\mathsf{A}^0 := (\mathsf{lsb}(\mathsf{K_A}[u_1]), \ldots, \mathsf{lsb}(\mathsf{K_A}[u_\rho]))$ to $\mathsf{P_B}$, who checks that $m_\mathsf{A}^0 = (\mathsf{lsb}(\mathsf{M_B}[u_1]) \oplus u_1, \ldots, \mathsf{lsb}(\mathsf{M_B}[u_\rho]) \oplus u_\rho)$ holds.
3. $\mathsf{P_B}$ runs $\mathsf{Fix}(sid_0, \tilde{\Delta}_\mathsf{B})$ to generate $[\tilde{\Delta}_\mathsf{B}]_{\Delta_\mathsf{A}}$. Then, $\mathsf{P_A}$ sends $m_\mathsf{A}^1 = \mathsf{lsb}(\mathsf{K_A}[\tilde{\Delta}_\mathsf{B}])$ to $\mathsf{P_B}$, and $\mathsf{P_B}$ sends $m_\mathsf{B}^1 = \mathsf{lsb}(\mathsf{M_B}[\tilde{\Delta}_\mathsf{B}])$ to $\mathsf{P_A}$ in parallel. If $m_\mathsf{A}^1 \oplus m_\mathsf{B}^1 = 0$, both parties compute $[\Delta_\mathsf{B}]_{\Delta_\mathsf{A}} := [\tilde{\Delta}_\mathsf{B}]_{\Delta_\mathsf{A}} \oplus 1$ where $\Delta_\mathsf{B} = \tilde{\Delta}_\mathsf{B} \oplus 1$; otherwise, the parties set $[\Delta_\mathsf{B}]_{\Delta_\mathsf{A}} := [\tilde{\Delta}_\mathsf{B}]_{\Delta_\mathsf{A}}$.
4. $\mathsf{P_A}$ and $\mathsf{P_B}$ calls $\mathcal{F}_\mathsf{COT}$ on respective input (init, sid'_0) and $(\mathsf{init}, sid'_0, \Delta_\mathsf{B})$, and then call $\mathcal{F}_\mathsf{COT}$ on the same input $(\mathsf{extend}, sid'_0, \rho)$ to generate random authenticated bits $[\boldsymbol{v}]_\mathsf{A}$.
5. Then $\mathsf{P_B}$ convinces $\mathsf{P_A}$ that $\mathsf{msb}(\Delta_\mathsf{B}) = 1$ by sending a ρ-bit vector $m_\mathsf{B}^0 := (\mathsf{msb}(\mathsf{K_B}[v_1]), \ldots, \mathsf{msb}(\mathsf{K_B}[v_\rho]))$ to $\mathsf{P_A}$, who checks that $m_\mathsf{B}^0 = (\mathsf{msb}(\mathsf{M_A}[v_1]) \oplus v_1, \ldots, \mathsf{msb}(\mathsf{M_A}[v_\rho]) \oplus v_\rho)$ holds.
6. $\mathsf{P_A}$ and $\mathsf{P_B}$ execute the following steps to mutually check that $\mathsf{lsb}(\Delta_\mathsf{A} \cdot \Delta_\mathsf{B}) = 1$.
   (a) Both parties call $\mathcal{F}_\mathsf{COT}$ on the same input $(\mathsf{extend}, sid_0, \rho)$ to generate random authenticated bits $[\boldsymbol{x}]_\mathsf{B}$, as well as run $\mathsf{Fix}(sid_0, \Delta_\mathsf{B} \cdot \boldsymbol{x})$ to generate $[\Delta_\mathsf{B} \cdot \boldsymbol{x}]_\mathsf{B}$. $\mathsf{P_B}$ proves to $\mathsf{P_A}$ that a set of authenticated triples $\{([x_i]_\mathsf{B}, [\Delta_\mathsf{B}]_\mathsf{B}, [x_i \Delta_\mathsf{B}]_\mathsf{B})\}_{i \in [1,\rho]}$ is valid by calling $\mathcal{F}_\mathsf{DVZK}$, and $\mathsf{P_A}$ aborts if it receives false from $\mathcal{F}_\mathsf{DVZK}$.
   (b) Both parties set $\langle \boldsymbol{x} \rangle := \mathsf{Convert1}_{[\cdot] \to \langle \cdot \rangle}([\Delta_\mathsf{B} \cdot \boldsymbol{x}]_\mathsf{B})$. Then, $\mathsf{P_A}$ sends $m_\mathsf{A}^2 := (\mathsf{lsb}(\mathsf{D_A}[x_1]), \ldots, \mathsf{lsb}(\mathsf{D_A}[x_\rho]))$ to $\mathsf{P_B}$, who checks that $m_\mathsf{A}^2 = (\mathsf{lsb}(\mathsf{D_B}[x_1]) \oplus x_1, \ldots, \mathsf{lsb}(\mathsf{D_B}[x_\rho]) \oplus x_\rho)$.
   (c) The parties run $\mathsf{Fix}(sid'_0, \Delta_\mathsf{A})$ to generate $[\Delta_\mathsf{A}]_\mathsf{A}$.
   (d) Both parties call $\mathcal{F}_\mathsf{COT}$ on the same input $(\mathsf{extend}, sid'_0, \rho)$ to generate random authenticated bits $[\boldsymbol{y}]_\mathsf{A}$, as well as run $\mathsf{Fix}(sid'_0, \Delta_\mathsf{A} \cdot \boldsymbol{y})$ to generate $[\Delta_\mathsf{A} \cdot \boldsymbol{y}]_\mathsf{A}$. $\mathsf{P_B}$ proves to $\mathsf{P_A}$ that a set of authenticated triples $\{([y_i]_\mathsf{A}, [\Delta_\mathsf{A}]_\mathsf{A}, [y_i \Delta_\mathsf{A}]_\mathsf{A})\}_{i \in [1,\rho]}$ is valid by calling $\mathcal{F}_\mathsf{DVZK}$, and $\mathsf{P_B}$ aborts if it receives false from $\mathcal{F}_\mathsf{DVZK}$.
   (e) Both parties set $\langle \boldsymbol{y} \rangle := \mathsf{Convert1}_{[\cdot] \to \langle \cdot \rangle}([\Delta_\mathsf{A} \cdot \boldsymbol{y}]_\mathsf{A})$. Then, $\mathsf{P_B}$ sends $m_\mathsf{B}^2 := (\mathsf{lsb}(\mathsf{D_B}[y_1]), \ldots, \mathsf{lsb}(\mathsf{D_B}[y_\rho]))$ to $\mathsf{P_A}$, who checks that $m_\mathsf{B}^2 = (\mathsf{lsb}(\mathsf{D_A}[y_1]) \oplus y_1, \ldots, \mathsf{lsb}(\mathsf{D_A}[y_\rho]) \oplus y_\rho)$.
   (f) Both parties locally compute two dual-key authenticated bits $\langle 1_\mathsf{B} \rangle := \mathsf{Convert1}_{[\cdot] \to \langle \cdot \rangle}([\Delta_\mathsf{B}]_\mathsf{B})$ and $\langle 1_\mathsf{A} \rangle := \mathsf{Convert1}_{[\cdot] \to \langle \cdot \rangle}([\Delta_\mathsf{A}]_\mathsf{A})$.
   (g) The parties run $\mathsf{CheckZero2}(\langle 1_\mathsf{B} \rangle - \langle 1_\mathsf{A} \rangle)$, and abort if the check fails.
7. $\mathsf{P_A}$ outputs $(\Delta_\mathsf{A}, \alpha_0)$ and $\mathsf{P_B}$ outputs $(\Delta_\mathsf{B}, \beta_0)$, such that $\mathsf{lsb}(\Delta_\mathsf{A}) = 1$, $\mathsf{msb}(\Delta_\mathsf{B}) = 1$, $\mathsf{lsb}(\Delta_\mathsf{A} \cdot \Delta_\mathsf{B}) = 1$ and $\alpha_0 + \beta_0 = \Delta_\mathsf{A} \cdot \Delta_\mathsf{B} \in \mathbb{F}_{2^\kappa}$.

Figure 4: Sub-protocol for sampling global keys.

4.2 Global-Key Sampling

We require $\Delta_\mathsf{A} \neq 0$, $\Delta_\mathsf{B} \neq 0$, and $\mathsf{lsb}(\Delta_\mathsf{A}\Delta_\mathsf{B}) = 1$ in the preprocessing phase to facilitate dual-key authentication. Considering the requirement of half-gates garbling, we have the constraints $\mathsf{lsb}(\Delta_\mathsf{A}) = 1$, $\mathsf{msb}(\Delta_\mathsf{B}) = 1$, and $\mathsf{lsb}(\Delta_\mathsf{A}\Delta_\mathsf{B}) = 1$ in $\mathcal{F}_\mathsf{cpre}$. We design the protocol Π_samp in Figure 4 and argue in Lemma 3 that the key constraints are satisfied.

Lemma 3. *The protocol Π_samp satisfies the following properties:*

- *The outputs satisfy that $\mathsf{lsb}(\Delta_\mathsf{A}) = 1$, $\mathsf{msb}(\Delta_\mathsf{B}) = 1$, and $\mathsf{lsb}(\Delta_\mathsf{A}\Delta_\mathsf{B}) = 1$ in the honest case.*

- *If* $\mathsf{lsb}(\Delta_\mathsf{A}) \neq 1$ *then* $\mathsf{P_B}$ *aborts except with probability* $2^{-\rho}$*. Conditioned on* $\Delta_\mathsf{A} \neq 0$*, if* $\mathsf{lsb}(\Delta_\mathsf{A}\Delta_\mathsf{B}) \neq 1$ *then* $\mathsf{P_B}$ *aborts except with probability* $2^{-\rho}$*.*
- *If* $\mathsf{msb}(\Delta_B) \neq 1$ *then* $\mathsf{P_A}$ *aborts except with probability* $2^{-\rho}$*. Conditioned on* $\Delta_\mathsf{B} \neq 0$*, if* $\mathsf{lsb}(\Delta_\mathsf{A}\Delta_\mathsf{B}) \neq 1$ *then* $\mathsf{P_B}$ *aborts except with probability* $2 \cdot 2^{-\kappa} + 2^{-\rho}$*.*

Proof. For the honest case since $\mathsf{P_A}$ and $\mathsf{P_B}$ follow the protocol instruction when sampling keys, the constraints on Δ_A and Δ_B are satisfied automatically. Moreover, notice that $\mathsf{lsb}(\Delta_\mathsf{A}\tilde{\Delta}_\mathsf{B}) = \mathsf{lsb}(\mathsf{K_A}[\tilde{\Delta}_\mathsf{B}]) \oplus \mathsf{lsb}(\mathsf{M_B}[\tilde{\Delta}_\mathsf{B}])$ and $\mathsf{lsb}(\Delta_\mathsf{A}) = 1$. If the parties discover in step 6b that $\mathsf{lsb}(\Delta_\mathsf{A}\tilde{\Delta}_\mathsf{B}) = 0$, $\mathsf{P_B}$ sets $\Delta_\mathsf{B} := \tilde{\Delta}'_\mathsf{B} \oplus 1$ and $\mathsf{lsb}(\Delta_\mathsf{A}\Delta_\mathsf{B}) = \mathsf{lsb}(\Delta_\mathsf{A}\tilde{\Delta}_\mathsf{B} + \Delta_\mathsf{A}) = 1$.

For the case of a corrupted $\mathsf{P_A}$, notice that $\mathsf{lsb}(\mathsf{K_A}[r]) \oplus \mathsf{lsb}(\mathsf{M_B}[r]) = r \cdot \mathsf{lsb}(\Delta_\mathsf{A})$ and $\mathsf{lsb}(\mathsf{D_A}[r]) \oplus \mathsf{lsb}(\mathsf{D_B}[r]) = r \cdot \mathsf{lsb}(\Delta_\mathsf{A}\Delta_\mathsf{B})$ for $r \in \mathbb{F}_2$. If $\mathsf{lsb}(\Delta_\mathsf{A}) = 0$ then $\mathsf{P_A}$ passing the test is equivalent to $m^0_\mathsf{A} \oplus (\mathsf{lsb}(\mathsf{K_A}[u_1]), \ldots, \mathsf{lsb}(\mathsf{K_A}[u_\rho])) = \boldsymbol{u}$ which happens with $2^{-\rho}$ probability since $\boldsymbol{u}$ is sampled independently from the left-hand side of the equation. Conditioned on $\Delta_\mathsf{A} \neq 0$, the second test passes when $\mathsf{lsb}(\Delta_\mathsf{A}\Delta_\mathsf{B}) = 0$ except with $2^{-\rho}$ probability from similar argument.

For the case of a corrupted $\mathsf{P_B}$, the checks in step 5 and step 6e are equivalent to the corrupted $\mathsf{P_A}$ case. Thus the soundness of the first check is $2^{-\rho}$. Also Lemma 2 guarantees that inconsistent Δ_B will be detected except with $2 \cdot 2^{-\kappa}$ probability. By union bound the soundness of the second check is $2 \cdot 2^{-\kappa} + 2^{-\rho}$. □

4.3 Consistency Check Between Values and MAC Tags

In our protocol to generate dual-key authentication, we need a party (e.g., $\mathsf{P_B}$) to use the MAC tags (denoted as $\{\beta_i\}$) of some existing IT-MAC authenticated values as the global keys of another $\mathcal{F}_{\mathsf{bCOT}}$ instance (denoted as $\{\beta'_i\}$). We enforce this constraint by checking equality between values authenticated by different keys. Our first observation is that the MAC tags are already implicitly authenticated by Δ_A^{-1}.

Authentication under inverse key. We define the Invert macro to *locally* convert $[x]_\mathsf{B} = (\mathsf{K_A}[x], \mathsf{M_B}[x], x)$ to $[y]_{\mathsf{B},\Delta_\mathsf{A}^{-1}} := (\mathsf{K_A}[y]_{\Delta_\mathsf{A}^{-1}}, \mathsf{M_B}[y]_{\Delta_\mathsf{A}^{-1}}, y)$. We note that this technique appeared previously in the certified VOLE protocols [DIO21].

- $[y]_{\mathsf{B},\Delta_\mathsf{A}^{-1}} \leftarrow \mathsf{Invert}([x]_\mathsf{B})$: On input $[x]_\mathsf{B}$ for $x \in \mathbb{F}_{2^\kappa}$, $\mathsf{P_A}$ and $\mathsf{P_B}$ execute the following:
  - $\mathsf{P_B}$ outputs $y := \mathsf{M_B}[x]$ and $\mathsf{M_B}[y]_{\Delta_\mathsf{A}^{-1}} := x$.
  - $\mathsf{P_A}$ outputs $\mathsf{K_A}[y]_{\Delta_\mathsf{A}^{-1}} := \mathsf{K_A}[x] \cdot \Delta_\mathsf{A}^{-1} \in \mathbb{F}_{2^\kappa}$.

We demonstrate the correctness of the Invert macro as follows.

Lemma 4. *Let* $[x]_\mathsf{B} = (\alpha, \beta, x)$ *where* $x \in \mathbb{F}_{2^\kappa}$ *then the MAC tag of* $\mathsf{P_B}$*,* β*, is implicitly authenticated by* Δ_A^{-1}*, i.e., the inverse of* $\mathsf{P_A}$*'s global key over* $\mathbb{F}_{2^\kappa}$*.*

This claim can be verified by multiplying both side of the equation by Δ_A^{-1}.

$$\underbrace{\beta}_{\mathsf{M_B}[x]} = \underbrace{\alpha}_{\mathsf{K_A}[x]} + x \cdot \Delta_\mathsf{A} \implies \underbrace{x}_{\mathsf{M_B}[\beta]_{\Delta_A^{-1}}} = \underbrace{\alpha \cdot \Delta_\mathsf{A}^{-1}}_{\mathsf{K_A}[\beta]_{\Delta_A^{-1}}} + \beta \cdot \Delta_\mathsf{A}^{-1} \ .$$

Random inverse key authentication. Notice that in the Invert macro, if we require the input $[x]$ to be uniformly random, i.e., $x \leftarrow \mathbb{F}_{2^\kappa}$, then the output value $y := \mathsf{M_A}[x] = x\Delta_\mathsf{A} - \mathsf{K_B}[x]$ is also uniformly random in the view of $\mathsf{P_A}$. Using this method we can generate random $\mathbb{F}_{2^\kappa}$ elements authenticated by Δ_A^{-1}.

Equality check across different keys. We recall a known technique to verify equality between two values authenticated by respective independent keys [DILO22a], which we summarize in the $\mathsf{EQCheck}$ macro. We recall its soundness in Lemma 5 and prove it in Appendix C.2.In the following, we assume that $\mathcal{F}_{\mathsf{COT}}$ has been initialized with $(sid, \Delta_{\mathsf{A}})$ and $(sid', \Delta'_{\mathsf{A}})$.

- $\mathsf{EQCheck}(\{[y_i]_{\Delta_{\mathsf{A}}}\}_{i\in[1,\ell]}, \{[y'_i]_{\Delta'_{\mathsf{A}}}\}_{i\in[1,\ell]})$: On input two sets of authenticated values under different keys $\Delta_{\mathsf{A}}, \Delta'_{\mathsf{A}}$, $\mathsf{P_A}$ and $\mathsf{P_B}$ check that $y_i = y'_i$ for all $i \in [1, \ell]$ as follows:
  1. Let $[y_i]_{\Delta_{\mathsf{A}}} = (k_i, m_i, y_i)$ and $[y'_i]_{\Delta'_{\mathsf{A}}} = (k'_i, m'_i, y'_i)$. Two parties $\mathsf{P_A}$ and $\mathsf{P_B}$ run $\mathsf{Fix}(sid, \{m'_i\}_{i\in[1,\ell]})$ to obtain a set of authenticated values $\{[m'_i]_{\Delta_{\mathsf{A}}}\}_{i\in[1,\ell]}$, and also run $\mathsf{Fix}(sid', \{m_i\}_{i\in[1,\ell]})$ to get another set of authenticated values $\{[m_i]_{\Delta'_{\mathsf{A}}}\}_{i\in[1,\ell]}$.
  2. For each $i \in [1, \ell]$, $\mathsf{P_A}$ computes $V_i := k_i \cdot \Delta'_{\mathsf{A}} + k'_i \cdot \Delta_{\mathsf{A}} + \mathsf{K_A}[m_i]_{\Delta'_{\mathsf{A}}} + \mathsf{K_A}[m'_i]_{\Delta_{\mathsf{A}}} \in \mathbb{F}_{2^\kappa}$, and $\mathsf{P_B}$ computes $W_i := \mathsf{M_B}[m_i]_{\Delta'_{\mathsf{A}}} + \mathsf{M_B}[m'_i]_{\Delta_{\mathsf{A}}} \in \mathbb{F}_{2^\kappa}$.
  3. $\mathsf{P_B}$ sends $h := \mathsf{H}(W_1, \ldots, W_\ell)$ to $\mathsf{P_A}$, who verifies that $h = \mathsf{H}(V_1, \ldots, V_\ell)$. If the check fails, $\mathsf{P_A}$ aborts.

Lemma 5. *If Δ_{A} and Δ'_{A} are independently sampled from $\mathbb{F}_{2^\kappa}$, then the probability that there exists some $i \in [1, \ell]$ such that $y_i \neq y'_i$ and $\mathsf{P_A}$ accepts in the $\mathsf{EQCheck}$ procedure is bounded by $\frac{3}{2^\kappa}$.*

The consistency check. The observation in Lemma 4 suggests that the MAC tags $\{\beta_i\}$ are already implicitly authenticated by Δ_{A}^{-1}. Moreover, by calling $\mathsf{Fix}(\Delta'_{\mathsf{A}})$, $\mathsf{P_A}$ and $\mathsf{P_B}$ can acquire $\{[\Delta'_{\mathsf{A}}]_{\beta'_i}\}$ and locally convert them to $\{[\beta'_i]_{\Delta'_{\mathsf{A}}}\}$. Since Δ_{A} and Δ'_{A} are independent, we can apply $\mathsf{EQCheck}$ to complete our goal.

We list the differences that inverse key authentication induces to $\mathsf{EQCheck}$. Recall that $\mathcal{F}_{\mathsf{COT}}$ has been initialized with $(sid, \Delta_{\mathsf{A}})$ and $(sid', \Delta'_{\mathsf{A}})$.

- $\mathsf{EQCheck}(\{[\beta_i]_{\Delta_{\mathsf{A}}^{-1}}\}_{i\in[1,\ell]}, \{[\beta'_i]_{\Delta'_{\mathsf{A}}}\}_{i\in[1,\ell]})$: On input two sets of authenticated values under different keys $\Delta_{\mathsf{A}}^{-1}, \Delta'_{\mathsf{A}}$, $\mathsf{P_A}$ and $\mathsf{P_B}$ check that $\beta_i = \beta'_i$ for all $i \in [1, \ell]$ as follows:
  1. $\mathsf{P_A}$ and $\mathsf{P_B}$ call $\mathcal{F}_{\mathsf{COT}}$ on the same input $(\mathsf{extend}, sid, \ell\kappa)$ to get authenticated bits $[\boldsymbol{r}_1]_{\Delta_{\mathsf{A}}}, \ldots, [\boldsymbol{r}_\ell]_{\Delta_{\mathsf{A}}}$ with $\boldsymbol{r}_i \in \mathbb{F}_2^\kappa$. Then, for $i \in [1, \ell]$, both parties define $[r_i]_{\Delta_{\mathsf{A}}} := \mathsf{B2F}([\boldsymbol{r}_i]_{\Delta_{\mathsf{A}}})$ with $r_i \in \mathbb{F}_{2^\kappa}$, and set $[s_i]_{\Delta_{\mathsf{A}}^{-1}} := \mathsf{Invert}([r_i]_{\Delta_{\mathsf{A}}})$.
  2. $\mathsf{P_A}$ and $\mathsf{P_B}$ run $\mathsf{EQCheck}(\{[\beta_i]_{\Delta_{\mathsf{A}}^{-1}}\}_{i\in[1,\ell]}, \{[\beta'_i]_{\Delta'_{\mathsf{A}}}\}_{i\in[1,\ell]})$ as described above, except that they use random authenticated values $[s_i]_{\Delta_{\mathsf{A}}^{-1}}$ for $i \in [1, \ell]$ to generate chosen authenticated values under Δ_{A}^{-1} in the Fix procedure.

It is straightforward to verify the soundness is not affected by changing to the inverse key. Thus we omit the proof of the following lemma.

Lemma 6. *If Δ_{A} and Δ'_{A} are independently sampled from $\mathbb{F}_{2^\kappa}$, then the probability that there exists some $i \in [1, \ell]$ such that $\beta_i \neq \beta'_i$ and $\mathsf{P_A}$ accepts in the $\mathsf{EQCheck}$ procedure is bounded by $\frac{3}{2^\kappa}$.*

4.4 Circuit Dependent Compressed Preprocessing

We now describe the protocol to realize the functionality $\mathcal{F}_{\mathsf{cpre}}$. Following the conventions of previous works, we defer all consistency checks to the end of the protocol. Notice that step 1 to step 5 corresponds to the circuit-independent phase (where we only require the scale rather than the topology information of the circuit) while the rest is the circuit-dependent phase (where the entire circuit is known). The protocol is shown in Figure 5 and Figure 6. We then analyze its security in Theorem 1. The proof is presented in Appendix C.3.

Protocol Π_{cpre}

Inputs: A Boolean circuit $\mathcal{C}$ that consists of a list of gates of the form (i, j, k, T). Let $n = |\mathcal{W}| + |\mathcal{I}_{\mathsf{B}}|$, $m = |\mathcal{W}| + |\mathcal{I}_{\mathsf{A}}|$, $L = \lceil \rho \log \frac{2en}{\rho} + \frac{\log \rho}{2} \rceil$ and $t = |\mathcal{W}|$.

Initialize: $\mathsf{P_A}$ and $\mathsf{P_B}$ execute sub-protocol Π_{samp} (Figure 4) to obtain $(\Delta_{\mathsf{A}}, \alpha_0)$ and $(\Delta_{\mathsf{B}}, \beta_0)$ respectively, such that $\mathsf{lsb}(\Delta_{\mathsf{A}}) = 1$, $\mathsf{msb}(\Delta_{\mathsf{B}}) = 1$, $\mathsf{lsb}(\Delta_{\mathsf{A}} \cdot \Delta_{\mathsf{B}}) = 1$ and $\alpha_0 + \beta_0 = \Delta_{\mathsf{A}} \cdot \Delta_{\mathsf{B}} \in \mathbb{F}_{2^\kappa}$. Thus, both parties hold $\langle 1 \rangle$ (i.e., $[\Delta_{\mathsf{B}}]_{\Delta_{\mathsf{A}}}$). After the sub-protocol execution, $\mathcal{F}_{\mathsf{COT}}$ was initialized by session identifier sid_0 and Δ_{A}.

Generate authenticated AND triples: $\mathsf{P_A}$ and $\mathsf{P_B}$ execute as follows:

1. $\mathsf{P_B}$ samples a matrix $\mathbf{M} \leftarrow \mathbb{F}_2^{n \times L}$ and sends it to $\mathsf{P_A}$.
2. Both parties call $\mathcal{F}_{\mathsf{COT}}$ on input $(\mathsf{extend}, sid_0, L)$ to generate random authenticated bits $[\boldsymbol{b}^*]$ where $\boldsymbol{b}^* \in \mathbb{F}_2^L$ and compute $[\boldsymbol{b}] := \mathbf{M} \cdot [\boldsymbol{b}^*]$ with $\boldsymbol{b} \in \mathbb{F}_2^n$.
3. Both parties run $\mathsf{Fix}(sid_0, \{b_i^* \Delta_{\mathsf{B}}\}_{i \in [1,L]})$ to generate authenticated values $[b_i^* \Delta_{\mathsf{B}}]_{\mathsf{B}}$. The parties locally run $\langle b_i^* \rangle \leftarrow \mathsf{Convert1}_{[\cdot] \to \langle \cdot \rangle}([b_i^* \Delta_{\mathsf{B}}]_{\Delta_{\mathsf{A}}})$. Let $\alpha_i, \beta_i \in \mathbb{F}_{2^\kappa}$ such that $\alpha_i + \beta_i = b_i^* \cdot \Delta_{\mathsf{A}} \cdot \Delta_{\mathsf{B}}$ for each $i \in [1, L]$.
4. $\mathsf{P_B}$ and $\mathsf{P_A}$ call $\mathcal{F}_{\mathsf{bCOT}}^{L+1}$ on respective inputs $(\mathsf{init}, sid_1, \beta_1, ..., \beta_L, \Delta_{\mathsf{B}})$ and (init, sid_1). Then, both parties send $(\mathsf{extend}, sid_1, m)$ to $\mathcal{F}_{\mathsf{bCOT}}^{L+1}$, which returns $([\boldsymbol{a}]_{\beta_1}, \ldots, [\boldsymbol{a}]_{\beta_L}, [\boldsymbol{a}]_{\Delta_{\mathsf{A}}})$ where $\boldsymbol{a} \in \mathbb{F}_2^m$. Then, $\mathsf{P_A}$ samples $\Delta'_{\mathsf{A}} \leftarrow \mathbb{F}_{2^\kappa}$, and then two parties run $\mathsf{Fix}(sid_1, \Delta'_{\mathsf{A}})$ to obtain $([\Delta'_{\mathsf{A}}]_{\beta_1}, \ldots, [\Delta'_{\mathsf{A}}]_{\beta_L}, [\Delta'_{\mathsf{A}}]_{\Delta_{\mathsf{B}}})$. $\mathsf{P_A}$ and $\mathsf{P_B}$ set $\langle 1_{\mathsf{B}}^{(1)} \rangle := \mathsf{Convert1}_{[\cdot] \to \langle \cdot \rangle}([\Delta_{\mathsf{B}}]_{\Delta'_{\mathsf{A}}})$ where $[\Delta_{\mathsf{B}}]_{\Delta'_{\mathsf{A}}}$ is equivalent to $[\Delta'_{\mathsf{A}}]_{\Delta_{\mathsf{B}}}$, and define $[\beta_i]_{\Delta'_{\mathsf{A}}} = [\Delta'_{\mathsf{A}}]_{\beta_i}$ for $i \in [1, L]$.
5. $\mathsf{P_B}$ and $\mathsf{P_A}$ call $\mathcal{F}_{\mathsf{bCOT}}^2$ on respective input $(\mathsf{init}, sid_2, \beta_0, \Delta_{\mathsf{B}})$ and (init, sid_2). Then, both parties send $(\mathsf{extend}, sid_2, t)$ to $\mathcal{F}_{\mathsf{bCOT}}^2$, which returns $([\hat{\boldsymbol{a}}]_{\beta_0}, [\hat{\boldsymbol{a}}]_{\Delta_{\mathsf{B}}})$ to the parties. $\mathsf{P_A}$ and $\mathsf{P_B}$ run $\mathsf{Fix}(sid_2, \Delta'_{\mathsf{A}})$ to get $[\Delta'_{\mathsf{A}}]_{\beta_0}$ and $[\Delta'_{\mathsf{A}}]_{\Delta_{\mathsf{B}}}$, and then locally convert to $[\beta_0]_{\Delta'_{\mathsf{A}}}$ and $[\Delta_{\mathsf{B}}]_{\Delta'_{\mathsf{A}}}$. Then, both parties set $\langle 1_{\mathsf{B}}^{(2)} \rangle := \mathsf{Convert1}_{[\cdot] \to \langle \cdot \rangle}([\Delta_{\mathsf{B}}]_{\Delta'_A})$.
6. For $w \in \mathcal{I}_{\mathsf{A}}$, $\mathsf{P_A}$ and $\mathsf{P_B}$ set $[b_w] = [0]$; for $w \in \mathcal{I}_{\mathsf{B}}$, both parties set $[a_w] = [0]$. For each wire $w \in \mathcal{I}_{\mathsf{A}} \cup \mathcal{W}$, two parties define $[a_w]$ in $[\boldsymbol{a}]$ as the authenticated bit on wire w; for each wire $w \in \mathcal{I}_{\mathsf{B}} \cup \mathcal{W}$, define $[b_w]$ in $[\boldsymbol{b}]$ as the authenticated bit on wire w. In a topological order, for each gate (i, j, k, T), $\mathsf{P_A}$ and $\mathsf{P_B}$ do the following:
   - If $T = \oplus$, compute $[a_k] := [a_i] \oplus [a_j]$ and $[b_k] := [b_i] \oplus [b_j]$.
   - If $T = \wedge$, $\mathsf{P_A}$ computes $a_{i,j} := a_i \wedge a_j$, and $\mathsf{P_B}$ computes $b_{i,j} := b_i \wedge b_j$.
7. Both parties run $\mathsf{Fix}(sid_0, \{b_{i,j}\}_{(i,j,*,\wedge) \in \mathcal{C}_{\mathsf{and}}})$ to generate a set of authenticated bits $\{[b_{i,j}]\}$, and also execute $\mathsf{Fix}(sid_2, \{a_{i,j}\}_{(i,j,*,\wedge) \in \mathcal{C}_{\mathsf{and}}})$ to generate a set of authenticated bits $\{[a_{i,j}]\}$.
8. For $i \in [1, n], j \in [1, L]$, $\mathsf{P_A}$ and $\mathsf{P_B}$ set $\langle a_i b_j^* \rangle := \mathsf{Convert2}_{[\cdot] \to \langle \cdot \rangle}([a_i]_{\beta_j}, \langle b_j^* \rangle)$. Then, both parties collect these dual-key authenticated bits to obtain $\langle a_i \boldsymbol{b}^* \rangle$, and compute $\langle a_i b_j \rangle$ and $\langle a_j b_i \rangle$ for each AND gate $(i, j, k, \wedge)$ from $\mathbf{M} \cdot \langle a_i \boldsymbol{b}^* \rangle$ for $i \in [1, n]$. Further, both parties set $\langle \hat{a}_k \rangle := \mathsf{Convert2}_{[\cdot] \to \langle \cdot \rangle}([\hat{a}_k]_{\beta_0}, \langle 1 \rangle)$ and $\langle a_{i,j} \rangle \leftarrow \mathsf{Convert2}_{[\cdot] \to \langle \cdot \rangle}([a_{i,j}]_{\beta_0}, \langle 1 \rangle)$.

Figure 5: The compressed preprocessing protocol for a Boolean circuit $\mathcal{C}$.

Theorem 1. *Protocol Π_{cpre} shown in Figures 5 and 6 securely realizes functionality $\mathcal{F}_{\mathsf{cpre}}$ (Figure 3) against malicious adversaries in the $(\mathcal{F}_{\mathsf{COT}}, \mathcal{F}_{\mathsf{bCOT}}, \mathcal{F}_{\mathsf{DVZK}}, \mathcal{F}_{\mathsf{EQ}}, \mathcal{F}_{\mathsf{Rand}})$-hybrid model.*

Consistency checks. We explain the rationale of the consistency checks in Π_{cpre}.

- The $\mathcal{F}_{\mathsf{DVZK}}$ in step 11 checks that the Fix inputs of $\mathsf{P_A}$ in step 6 and those of $\mathsf{P_B}$ in step 6 and step 3 are well-formed.
- The $\mathsf{CheckZero2}$ and $\mathsf{EQCheck}$ in step 12 ensure to $\mathsf{P_A}$ that the multiple instances of Δ_{B} in Π_{samp} (Figure 4) and Π_{cpre} (step 4 and step 5 in Figure 5) are identical. Also, $\mathsf{P_B}$ can make sure that

Protocol Π_{cpre}, continued

9. For each AND gate $(i, j, k, \wedge)$, $\mathsf{P_A}$ and $\mathsf{P_B}$ locally compute $\langle \tilde{b}_k \rangle := \langle a_{i,j} \rangle \oplus \langle a_i b_j \rangle \oplus \langle a_j b_i \rangle \oplus \langle \hat{a}_k \rangle$. Then, for each $k \in \mathcal{W}$, $\mathsf{P_A}$ sends $\mathsf{lsb}(\mathsf{D_A}[\tilde{b}_k])$ to $\mathsf{P_B}$, who computes $\tilde{b}_k := \mathsf{lsb}(\mathsf{D_A}[\tilde{b}_k]) \oplus \mathsf{lsb}(\mathsf{D_B}[\tilde{b}_k])$. For each AND gate $(i, j, k, \wedge)$, $\mathsf{P_B}$ computes $\hat{b}_k := \tilde{b}_k \oplus b_{i,j}$.
10. Both parties run $\mathsf{Fix}(sid_0, \{\hat{b}_k\}_{k \in \mathcal{W}})$ to obtain $[\hat{b}_k]$ for each $k \in \mathcal{W}$.

Consistency check: $\mathsf{P_A}$ and $\mathsf{P_B}$ perform the following consistency-check steps:

11. Let $[B_i^*] = [b_i^* \Delta_\mathsf{B}]_{\Delta_\mathsf{A}}$ produced in the previous phase. Both parties call $\mathcal{F}_{\mathsf{DVZK}}$ to prove the following statements hold:
    - For each AND gate $(i, j, k, \wedge)$, for $([b_i], [b_j], [b_{i,j}])$, $b_{i,j} = b_i \wedge b_j$.
    - For each AND gate $(i, j, k, \wedge)$, for $([a_i], [a_j], [a_{i,j}])$, $a_{i,j} = a_i \wedge a_j$.
    - For each $i \in [1, L]$, for $([b_i^*], [\Delta_\mathsf{B}], [B_i^*])$, $B_i^* = b_i^* \cdot \Delta_\mathsf{B}$.
12. $\mathsf{P_A}$ and $\mathsf{P_B}$ call $\mathcal{F}_{\mathsf{COT}}$ on respective input $(\mathsf{init}, sid_3, \Delta'_\mathsf{A})$ and (init, sid_3). Then they run $[\Delta_\mathsf{B}]_{\Delta'_\mathsf{A}} := \mathsf{Fix}(sid_3, \Delta_\mathsf{B})$ and $\langle 1_\mathsf{B}^{(3)} \rangle := \mathsf{Convert1}_{[\cdot] \to \langle \cdot \rangle}([\Delta_\mathsf{B}]_{\Delta'_\mathsf{A}})$. $\mathsf{P_A}$ and $\mathsf{P_B}$ run $\mathsf{CheckZero2}(\langle 1_\mathsf{B}^{(1)} \rangle - \langle 1_\mathsf{B}^{(2)} \rangle, \langle 1_\mathsf{B}^{(2)} \rangle - \langle 1_\mathsf{B}^{(3)} \rangle)$ and $\mathsf{EQCheck}([\Delta_\mathsf{B}]_{\Delta_\mathsf{A}}, [\Delta_\mathsf{B}]_{\Delta'_\mathsf{A}})$ to check that $\Delta'_\mathsf{A}, \Delta_\mathsf{B}$ are consistent when it is used in different functionalities. Both parties run $[\beta_i]_{\Delta_\mathsf{A}^{-1}} \leftarrow \mathsf{Invert}([b_i^* \Delta_\mathsf{B}]_{\Delta_\mathsf{A}})$ for each $i \in [0, L]$, and then execute $\mathsf{EQCheck}(\{[\beta_i]_{\Delta_\mathsf{A}^{-1}}\}_{i \in [0,L]}, \{[\beta_i]_{\Delta'_\mathsf{A}}\}_{i \in [0,L]})$.
13. $\mathsf{P_A}$ and $\mathsf{P_B}$ call $\mathcal{F}_{\mathsf{COT}}$ on input $(\mathsf{extend}, sid_0, \kappa)$ to generate a vector of random authenticated bits $[\boldsymbol{r}]_\mathsf{B}$ with $\boldsymbol{r} \in \mathbb{F}_2^\kappa$, and run $[r]_\mathsf{B} \leftarrow \mathsf{B2F}([\boldsymbol{r}]_\mathsf{B})$ where $r = \sum_{i \in [0,\kappa)} r_i \cdot X^i \in \mathbb{F}_{2^\kappa}$. Then both parties run $\mathsf{Fix}(sid_0, r \cdot \Delta_\mathsf{B})$ to obtain $[r \cdot \Delta_\mathsf{B}]_{\Delta_\mathsf{A}}$. The parties execute $\langle r \rangle \leftarrow \mathsf{Convert1}_{[\cdot] \to \langle \cdot \rangle}([r \cdot \Delta_\mathsf{B}]_{\Delta_\mathsf{A}})$.
14. $\mathsf{P_A}$ and $\mathsf{P_B}$ call $\mathcal{F}_{\mathsf{Rand}}$ to sample a random element $\chi \in \mathbb{F}_{2^\kappa}$.
15. $\mathsf{P_A}$ convinces $\mathsf{P_B}$ that $\tilde{b}_k$ is correct (and thus $\hat{b}_k$ is correct) for $k \in \mathcal{W}$ as follows.
    (a) Both parties compute $\langle y \rangle := \sum_{k \in \mathcal{W}} \chi^k \cdot \langle \tilde{b}_k \rangle + \langle r \rangle$. Then $\mathsf{P_B}$ sends y to $\mathsf{P_A}$.
    (b) The parties execute $\mathsf{CheckZero2}(\langle y \rangle - y \cdot \langle 1 \rangle)$.
16. $\mathsf{P_B}$ convinces $\mathsf{P_A}$ that $[\hat{b}_k]$ is correct for $k \in \mathcal{W}$ as follows:
    (a) For each AND gate $(i, j, k, \wedge)$, $\mathsf{P_A}$ and $\mathsf{P_B}$ compute $[\tilde{b}_k]_\mathsf{B} := [\hat{b}_k]_\mathsf{B} \oplus [b_{i,j}]_\mathsf{B}$.
    (b) Both parties compute $[y]_\mathsf{B} := \sum_{k \in \mathcal{W}} \chi^k \cdot [\tilde{b}_k]_\mathsf{B} + [r]_\mathsf{B}$.
    (c) $\mathsf{P_A}$ and $\mathsf{P_B}$ run $\mathsf{CheckZero}([y]_\mathsf{B} - y)$.

Output: $\mathsf{P_A}$ and $\mathsf{P_B}$ output a matrix $\mathbf{M}$ along with $([\boldsymbol{a}], [\hat{\boldsymbol{a}}], [\boldsymbol{b}^*], [\hat{\boldsymbol{b}}])$.

Figure 6: The compressed preprocessing protocol for a Boolean circuit $\mathcal{C}$, continued.

Δ'_A in step 4 and step 5 of Π_{cpre} (Figure 5) are identical.

- $\mathsf{P_B}$ checks that the message in step 9 of Π_{cpre} from $\mathsf{P_A}$ are correct. To do this, $\mathsf{P_B}$ checks its locally computed value against the dual-key authenticated value, which is unalterable. Moreover, we reduce the communication using random linear combination. This is done in step 14 and step 15 of Π_{cpre} (Figure 6).
- $\mathsf{P_A}$ checks that the Fix inputs of $\mathsf{P_B}$ in step 10 of Π_{cpre} (Figure 6) are correct. This is done by checking the IT-MAC authenticated values against the dual-key authenticated ones in step 16 of Π_{cpre} (Figure 6).

Optimization based on Fiat-Shamir. In the protocol Π_{cpre}, both parties choose random public challenges by calling functionality $\mathcal{F}_{\mathsf{Rand}}$. Based on the Fiat-Shamir heuristic [FS87], both parties can generate the challenges by hashing the protocol transcript up until this point, which is secure in the random oracle model. This optimization can save one communication round, and has also been used in previous work such as [BCG+19a, YWL+20].

Communication complexity. As recent PCG-like COT protocols have communication complexity sublinear to the number of resulting correlations, we can ignore the communication cost of generating random COT correlations when counting the communication amortized to every triple. Our checking protocols only introduce a negligibly small communication overhead. Therefore, the Fix procedure brings the main communication cost where Fix is used to transform random COT to chosen COT. Also, since parameter L is logarithmic to the number n of triples, we only need to consider the Fix procedures related to n.

This includes IT-MAC generation of $a_{i,j}$ (from $\mathsf{P_A}$ to $\mathsf{P_B}$ in step 6 of Figure 5), $b_{i,j}$ (from $\mathsf{P_B}$ to $\mathsf{P_A}$ in the same step), $\hat{b}_k$ (from $\mathsf{P_B}$ to $\mathsf{P_A}$ in step 10 of Figure 6). In addition, for each triple, $\mathsf{P_A}$ needs to send $\mathsf{lsb}(\mathsf{D}[\tilde{b}_k])$ to $\mathsf{P_B}$ in step 9 of Figure 6. Overall, the one-way communication cost is 2 bits per triple.

5 Authenticated Garbling from COT

Now we describe the online phase of our two-party computation protocol. We first introduce a generalized distributed garbling syntax which can be instantiated by different schemes and then introduce the complete Boolean circuit evaluation protocol Π_{2PC}.

5.1 Distributed Garbling

We define the format of distributed garbling using two macros Garble and Eval, assuming that the preprocessing information is ready. Notice that these two macros can be instantiated by different garbling schemes. In our main protocol that optimizes towards one-way communication we instantiate it using the distributed half-gates garbling [KRRW18] whereas we use the optimized WRK garbling of Dittmer et al. [DILO22a] for the version that optimizes towards two-way communication. We recall the respective schemes at Appendices D.1 and D.2.

- $\mathsf{Garble}(\mathcal{C})$: $\mathsf{P_A}$ and $\mathsf{P_B}$ perform *local* operations as follows:
  - $\mathsf{P_A}$ computes and outputs $(\mathcal{GC}_{\mathsf{A}}, \{\mathsf{L}_{w,0}, \mathsf{L}_{w,1}\}_{w\in\mathcal{I}_{\mathsf{A}}\cup\mathcal{I}_{\mathsf{B}}\cup\mathcal{W}\cup\mathcal{O}})$.
  - $\mathsf{P_B}$ computes and outputs $\mathcal{GC}_{\mathsf{B}}$.
- $\mathsf{Eval}(\mathcal{GC}_{\mathsf{A}}, \mathcal{GC}_{\mathsf{B}}, \{(\Lambda_w, \mathsf{L}_{w,\Lambda_w})\}_{w\in\mathcal{I}_{\mathsf{A}}\cup\mathcal{I}_{\mathsf{B}}})$: $\mathsf{P_B}$ evaluates the garbled circuit and obtain $\{\Lambda_w, \mathsf{L}_{w,\Lambda_w}\}_{w\in\mathcal{W}\cup\mathcal{O}}$.

The addition of evaluator's random masks is to decouple the abort probability with the real input values (recall that the Eval function only requires masked values). The following definition captures this security property.

Definition 2. *For a distributed garbling scheme with preprocessing defined by* Garble *and* Eval*, consider the event* Bad *where the evaluator aborts or outputs masked wire value* Λ_w *that is incorrect (wrt. the input values of* Eval *and the masks of preprocessing). We call a distributed garbling scheme to be* ϵ-selective failure resilience*, if conditioned on the garbled circuit* $\mathcal{GC}_{\mathsf{A}}, \mathcal{GC}_{\mathsf{B}}$*, the evaluator's* *candidate input wire labels* $\{(\mathsf{L}_{w,0}, \mathsf{L}_{w,1})\}_{w \in \mathcal{I}_\mathsf{B}}$ *and the garbler's input wire masked values and labels* $\{(\Lambda_w, \mathsf{L}_w)\}_{w \in \mathcal{I}_\mathsf{A}}$*, for any two pairs of* P_B*'s inputs* $\boldsymbol{y}, \boldsymbol{y}'$*, we have*

$$|\Pr[\mathsf{Bad}|\boldsymbol{y}] - \Pr[\mathsf{Bad}|\boldsymbol{y}']| \leq \epsilon \ ,$$

where $\Pr[\mathsf{Bad}|\boldsymbol{y}]$ *denotes the probability that the event* Bad *happens when the evaluator's input value is* $\boldsymbol{y}$ *and with aforementioned conditions.*

With uncompressed preprocessing the DILO-WRK and KRRW distributed garbling (recalled at Appendices D.1 and D.2.) has 0-selective failure resilience [WRK17a, KRRW18] since the inputs Λ_w to Eval are completely masked and independent of the real input. In Lemma 9 we show that for the DILO-WRK and KRRW schemes, replacing the evaluator's mask to ρ-wise independent randomness induces $2^{-\rho}$-selective failure resilience.

The next lemma states that after evaluating the garbled circuit the garbler and evaluator implicitly holds the authentication of the masked public wire values (color/permutation bits). To the best of our knowledge we are the first to apply this observation in the consistency check of authenticated garbling.

Lemma 7. *After running* Eval*, the evaluator holds the 'color bits'* Λ_w *for every wire* $w \in \mathcal{W}$*. The garbler* P_A *and evaluator* P_B *also hold* $\mathsf{K}_\mathsf{A}[\Lambda_w], \mathsf{M}_\mathsf{B}[\Lambda_w]$ *subject to* $\mathsf{M}_\mathsf{B}[\Lambda_w] = \mathsf{K}_\mathsf{A}[\Lambda_w] + \Lambda_w \Delta_\mathsf{A}$*.*

Proof. We can define the following values using only wire labels:

$$\Lambda_w := (\mathsf{L}_{w,0} \oplus \mathsf{L}_{w,\Lambda_w}) \cdot \Delta_\mathsf{A}^{-1}, \quad \mathsf{M}_\mathsf{B}[\Lambda_w] := \mathsf{L}_{w,\Lambda_w}, \quad \mathsf{K}_\mathsf{A}[\Lambda_w] := \mathsf{L}_{w,0} \ .$$

It is easy to verify $\mathsf{M}_\mathsf{B}[\Lambda_w] = \mathsf{K}_\mathsf{A}[\Lambda_w] + \Lambda_w \cdot \Delta_\mathsf{A}$, which implies that $[\Lambda_w]_\mathsf{B} := (\mathsf{L}_{w,0}, \mathsf{L}_{w,\Lambda_w}, \Lambda_w)$ is a valid IT-MAC. □

5.2 A Dual Execution Protocol Without Leakage

We describe a malicious secure 2PC protocol with almost the same one-way communication as half-gates garbling. We achieve this by adapting the dual execution technique to the distributed garbling setting. Intuitively, our observation in Lemma 7 allows us to check the consistency of every wire of the circuit. Together with some IT-MAC techniques to ensure input consistency, our protocol circumvents the one-bit leakage of previous dual execution protocols [HKE12, HsV20].

In the following descriptions, we denote the actual value induced by the input on each wire w of the circuit $\mathcal{C}$ by z_w. The masked value on that wire is denoted as $\Lambda_w := z_w \oplus a_w \oplus b_w$ which is revealed to the evaluator during evaluation. The protocol is described in Figure 7 and Figure 8.

Intuitions of Consistency Checking. The security of the semi-honest garbled circuit guarantees that when the garbled circuit is correctly computed, then except with negligible probability the evaluator can only acquire one of the two labels (corresponding to the execution path) for each wire in the circuit. Thus, we can check the color bits of the honest party against the labels that the corrupted party acquires (in the separate execution) to verify consistency.

Using the notations from Lemma 7, let $\bar{\Lambda}_w := (\mathsf{L}_{w,\Lambda_w} \oplus \mathsf{L}_{w,0}) \cdot \Delta_\mathsf{A}^{-1}, \bar{\Lambda}'_w := (\mathsf{L}'_{w,\Lambda'_w} \oplus \mathsf{L}'_{w,0}) \cdot \Delta_\mathsf{B}^{-1}$ for $w \in \mathcal{W}$. Our goal is to check the following equations where the left-hand (resp. right-hand) side is the evaluation result of P_A (resp. P_B).

$$\bar{\Lambda}'_w \oplus a'_w \oplus b'_w = \Lambda_w \oplus a_w \oplus b_w \text{ for the corrupted } \mathsf{P}_\mathsf{A} \text{ case,} \quad (1)$$
$$\Lambda'_w \oplus a'_w \oplus b'_w = \bar{\Lambda}_w \oplus a_w \oplus b_w \text{ for the corrupted } \mathsf{P}_\mathsf{B} \text{ case.} \quad (2)$$

Protocol Π_{2PC}

Inputs: In the preprocessing phase, $\mathsf{P_A}$ and $\mathsf{P_B}$ agree on a Boolean circuit $\mathcal{C}$ with circuit-input wires $\mathcal{I}_\mathsf{A} \cup \mathcal{I}_\mathsf{B}$, output wires of all AND gates $\mathcal{W}$ and circuit-output wires $\mathcal{O}$. In the online phase, $\mathsf{P_A}$ holds an input $x \in \{0,1\}^{|\mathcal{I}_\mathsf{A}|}$ and $\mathsf{P_B}$ holds an input $y \in \{0,1\}^{|\mathcal{I}_\mathsf{B}|}$; $\mathsf{P_B}$ will receive the output $z = \mathcal{C}(x, y)$. Let $\mathsf{H} : \{0,1\}^{2\kappa} \to \{0,1\}^\kappa$ and $\mathsf{H}' : \{0,1\}^* \to \{0,1\}^\kappa$ be two random oracles.

Preprocessing: $\mathsf{P_A}$ plays the role of a garbler and $\mathsf{P_B}$ acts as an evaluator, and two parties execute as follows:

1. Both parties call $\mathcal{F}_\mathsf{cpre}$ to obtain a matrix $\mathbf{M}$ and vectors of authenticated bits $([\boldsymbol{a}], [\hat{\boldsymbol{a}}], [\boldsymbol{b}^*], [\hat{\boldsymbol{b}}])$. The parties locally compute $[\boldsymbol{b}] := \mathbf{M} \cdot [\boldsymbol{b}^*]$.
2. Following a predetermined topological order, $\mathsf{P_A}$ and $\mathsf{P_B}$ use $([\boldsymbol{a}], [\hat{\boldsymbol{a}}], [\boldsymbol{b}], [\hat{\boldsymbol{b}}])$ to obtain authenticated masks $[a_w], [b_w]$ for each wire w and other authenticated bits that will be used in the construction of authenticated garbling.
3. Using the authenticated bits from the previous step and the KRRW garbling scheme, $\mathsf{P_A}$ and $\mathsf{P_B}$ run Garble to generate a distributed garbled circuit $(\mathcal{GC}_\mathsf{A}, \mathcal{GC}_\mathsf{B})$, and $\mathsf{P_A}$ sends $\mathcal{GC}_\mathsf{A}$ to $\mathsf{P_B}$. For each wire w, two garbled labels $\mathsf{L}_{w,0}, \mathsf{L}_{w,1} \in \{0,1\}^\kappa$ are generated and satisfy $\mathsf{L}_{w,1} = \mathsf{L}_{w,0} \oplus \Delta_\mathsf{A}$. $\mathsf{P_A}$ knows the label $\mathsf{L}_{w,0}$ for each wire w as well as Δ_A.

Online: In the following steps, $\mathsf{P_A}$ securely transmits one label on each circuit-input wire to $\mathsf{P_B}$, and $\mathsf{P_B}$ evaluates the circuit.

4. For each $w \in \mathcal{I}_\mathsf{A}$, $\mathsf{P_A}$ computes a masked value $\Lambda_w := x_w \oplus a_w \in \{0,1\}$, and then sends $(\Lambda_w, \mathsf{L}_{w,\Lambda_w})$ to $\mathsf{P_B}$.
5. $\mathsf{P_A}$ and $\mathsf{P_B}$ call $\mathcal{F}_\mathsf{COT}$ on respective input $(\mathsf{init}, sid, \Delta_\mathsf{A})$ and (init, sid), and then send $(\mathsf{extend}, sid, |\mathcal{I}_\mathsf{B}|)$ to $\mathcal{F}_\mathsf{COT}$, which returns random authenticated bits $[\boldsymbol{r}]_\mathsf{B}$ to the parties.
6. For each $w \in \mathcal{I}_\mathsf{B}$, $\mathsf{P_B}$ computes $\Lambda_w := y_w \oplus b_w$ and then sends $d_w := \Lambda_w \oplus r_w$ to $\mathsf{P_A}$. Both parties set $[\Lambda_w]_\mathsf{B} := [r_w]_\mathsf{B} \oplus d_w$. For each $w \in \mathcal{I}_\mathsf{B}$, $\mathsf{P_A}$ sends $m_{w,0} := \mathsf{H}(\mathsf{K_A}[\Lambda_w], w\|1) \oplus \mathsf{L}_{w,0}$ and $m_{w,1} := \mathsf{H}(\mathsf{K_A}[\Lambda_w] \oplus \Delta_\mathsf{A}, w\|1) \oplus \mathsf{L}_{w,1}$ to $\mathsf{P_B}$, who computes $\mathsf{L}_{w,\Lambda_w} := m_{w,\Lambda_w} \oplus \mathsf{H}(\mathsf{M_B}[\Lambda_w], w\|1)$.
7. $\mathsf{P_B}$ runs $\mathsf{Eval}(\mathcal{GC}_\mathsf{A}, \mathcal{GC}_\mathsf{B}, \{(\Lambda_w, \mathsf{L}_{w,\Lambda_w})\}_{w \in \mathcal{I}_\mathsf{A} \cup \mathcal{I}_\mathsf{B}})$ to obtain $(\Lambda_w, \mathsf{L}_{w,\Lambda_w})$ for each wire $w \in \mathcal{W} \cup \mathcal{O}$. For each $w \in \mathcal{W}$, both parties define $[\Lambda_w]_\mathsf{B} = (\mathsf{L}_{w,0}, \mathsf{L}_{w,\Lambda_w}, \Lambda_w)$.

Figure 7: Actively secure 2PC protocol in the $\mathcal{F}_\mathsf{cpre}$-hybrid model.

Multiplying the first equation by Δ_B, the second by Δ_A and do summation[3] gives the $\tilde{V}_w^\mathsf{A}, \tilde{V}_w^\mathsf{B}$ values in the consistency checking.

$$\begin{matrix} (a_w + a'_w + \Lambda'_w)\Delta_\mathsf{A} + \mathsf{M_A}[a_w + a'_w] \\ +\mathsf{M_A}[\bar{\Lambda}'_w] + \mathsf{K_A}[b_w + b'_w + \bar{\Lambda}_w] \end{matrix} = \begin{matrix} (b_w + b'_w + \Lambda_w)\Delta_\mathsf{B} + \mathsf{M_B}[b_w + b'_w] \\ +\mathsf{M_B}[\bar{\Lambda}_w] + \mathsf{K_B}[a_w + a'_w + \bar{\Lambda}'_w] \end{matrix}$$

Communication complexity. In our dual execution protocol, $\mathsf{P_A}$ and $\mathsf{P_B}$ sends $(2\kappa+1)t + (\kappa+1)|\mathcal{I}_\mathsf{A}| + 2\kappa|\mathcal{I}_\mathsf{B}| + \kappa + |\mathcal{O}|$ and $(2\kappa+1)t + (\kappa+2)|\mathcal{I}_\mathsf{B}| + 2\kappa|\mathcal{I}_\mathsf{A}|$ bits respectively. Therefore the amortized one-way communication is $2\kappa + 1$ bits per AND gate. Since we need to call $\mathcal{F}_\mathsf{cpre}$ twice in Π_2PC, we conclude that the amortized one-way (resp. two-way) communication in the ($\mathcal{F}_\mathsf{COT}$, $\mathcal{F}_\mathsf{bCOT}$, $\mathcal{F}_\mathsf{DVZK}$, $\mathcal{F}_\mathsf{EQ}$, $\mathcal{F}_\mathsf{Rand}$)-hybrid model is $2\kappa + 5$ (resp. $4\kappa + 10$) bits.

For the second version that combines Π_cpre and the optimized WRK online protocol, the amortized one-way (resp. two-way) communication is $2\kappa + 3\rho + 2$ (resp. $2\kappa + 3\rho + 4$) bits in the same hybrid model.

[3] We define a_w, a'_w, b_w, b'_w by the MAC tag and keys to implicitly authenticate them.

Protocol Π_{2PC}, continued

Dual execution and consistency check:

8. Re-using the initialization procedure of functionality $\mathcal{F}_{\mathsf{cpre}}$ (i.e., the same global keys Δ_{A} and Δ_{B} are adopted), $\mathsf{P_A}$ and $\mathsf{P_B}$ execute the preprocessing phase as described above again by swapping the roles (i.e., $\mathsf{P_A}$ is an evaluator and $\mathsf{P_B}$ is a garbler). Thus, for each $w \in \mathcal{W}$, $\mathsf{P_A}$ and $\mathsf{P_B}$ hold $[a'_w]$ and $[b'_w]$. For each wire w, $\mathsf{P_B}$ has also the label $\mathsf{L}'_{w,0}$.
9. Swapping the roles (i.e., $\mathsf{P_A}$ is the evaluator and $\mathsf{P_B}$ is the garbler), $\mathsf{P_A}$ and $\mathsf{P_B}$ execute the online phase as described above again, except for the following differences of processing inputs:
   (a) For each $w \in \mathcal{I}_{\mathsf{B}}$, $\mathsf{P_A}$ and $\mathsf{P_B}$ run $\mathsf{Open}([b_w] \oplus [b'_w] \oplus [r_w]_{\mathsf{B}} \oplus d_w)$ that enables $\mathsf{P_A}$ to obtain the masked value $\Lambda'_w = y_w \oplus b'_w$, and $\mathsf{P_B}$ sends $\mathsf{L}'_{w,\Lambda'_w}$ to $\mathsf{P_A}$.
   (b) For each $w \in \mathcal{I}_{\mathsf{A}}$, both parties set $[\Lambda'_w]_{\mathsf{A}} := [a_w] \oplus [a'_w] \oplus \Lambda_w$, and then garbler $\mathsf{P_B}$ sends $m'_{w,0} := \mathsf{H}(\mathsf{K_B}[\Lambda'_w], w\|2) \oplus \mathsf{L}'_{w,0}$ and $m'_{w,1} := \mathsf{H}(\mathsf{K_B}[\Lambda'_w] \oplus \Delta_{\mathsf{B}}, w\|2) \oplus \mathsf{L}'_{w,1}$ to $\mathsf{P_A}$, who computes $\mathsf{L}'_{w,\Lambda'_w} := m'_{w,\Lambda'_w} \oplus \mathsf{H}(\mathsf{M_A}[\Lambda'_w], w\|2)$.

   After the 2th execution of online phase, $\mathsf{P_A}$ and $\mathsf{P_B}$ obtain $[\Lambda'_w]_{\mathsf{A}}$ for all $w \in \mathcal{W}$.
10. $\mathsf{P_A}$ and $\mathsf{P_B}$ check that $(\Lambda_w \oplus a_w \oplus b_w) \cdot (\Delta_{\mathsf{A}} \oplus \Delta_{\mathsf{B}}) = (\Lambda'_w \oplus a'_w \oplus b'_w) \cdot (\Delta_{\mathsf{A}} \oplus \Delta_{\mathsf{B}})$ holds by performing the following steps.
    (a) For each $w \in \mathcal{W}$, $\mathsf{P_A}$ and $\mathsf{P_B}$ respectively compute
    $$\begin{aligned} V_w^{\mathsf{A}} &= (a_w \oplus a'_w \oplus \Lambda'_w)\Delta_{\mathsf{A}} \oplus \mathsf{M_A}[a_w] \oplus \mathsf{M_A}[a'_w] \oplus \mathsf{M_A}[\Lambda'_w] \oplus \\ &\quad \mathsf{K_A}[b_w] \oplus \mathsf{K_A}[b'_w] \oplus \mathsf{K_A}[\Lambda_w], \\ V_w^{\mathsf{B}} &= (b_w \oplus b'_w \oplus \Lambda_w)\Delta_{\mathsf{B}} \oplus \mathsf{M_B}[b_w] \oplus \mathsf{M_B}[b'_w] \oplus \mathsf{M_B}[\Lambda_w] \oplus \\ &\quad \mathsf{K_B}[a_w] \oplus \mathsf{K_B}[a'_w] \oplus \mathsf{K_B}[\Lambda'_w]. \end{aligned}$$
    (b) $\mathsf{P_A}$ computes $h := \mathsf{H}'(V_1^{\mathsf{A}}, \ldots, V_t^{\mathsf{A}})$, and then sends it to $\mathsf{P_B}$ who checks that $h = \mathsf{H}'(V_1^{\mathsf{B}}, \ldots, V_t^{\mathsf{B}})$. If the check fails, $\mathsf{P_B}$ aborts.

Output processing: For each $w \in \mathcal{O}$, $\mathsf{P_A}$ and $\mathsf{P_B}$ run $\mathsf{Open}([a_w])$ such that $\mathsf{P_B}$ receives a_w, and then $\mathsf{P_B}$ computes $z_w := \Lambda_w \oplus (a_w \oplus b_w)$.

Figure 8: Actively secure 2PC protocol in the $\mathcal{F}_{\mathsf{cpre}}$-hybrid model, continued.

5.3 Security Analysis

We first give two useful lemmas about the equality checking (following the proofs of [WRK17a, KRRW18, DILO22b]) and postpone their proofs to Appendix C.4.We state the security of our 2PC protocol in Theorem 2 and prove it in Appendix C.5.

Lemma 8. *After the equality check, except with probability $\frac{2+\mathsf{poly}(\kappa)}{2^\kappa}$, $\mathsf{P_B}$ either aborts or evaluates the garbled circuit exactly according to $\mathcal{C}(\boldsymbol{x}, \boldsymbol{y})$, where we canonically define the circuit input $\boldsymbol{x}, \boldsymbol{y}$ using the messages in step 4, step 6, and the randomness from the preprocessing phase.*

Lemma 9. *For the DILO-WRK and KRRW distributed garbling schemes (see details at Appendices D.2 and D.1.) by sampling the wire masks $\boldsymbol{a}, \boldsymbol{a}', \boldsymbol{b}, \boldsymbol{b}'$ using the compressed preprocessing functionality $\mathcal{F}_{\mathsf{cpre}}$ (recall that $\boldsymbol{b} := \mathbf{M} \cdot \boldsymbol{b}^*, \boldsymbol{a}' := \mathbf{M} \cdot (\boldsymbol{a}^*)'$ are compressed randomness), the resulting schemes have $2^{-\rho}$-selective failure resilience.*

Theorem 2. *Protocol Π_{2PC} shown in Figure 7 and Figure 8 securely realizes functionality $\mathcal{F}_{\mathsf{2PC}}$ in the presence of malicious adversary in the $\mathcal{F}_{\mathsf{cpre}}$-hybrid model and the random oracle model.*

Acknowledgements

Work of Kang Yang is supported by the National Key Research and Development Program of China (Grant No. 2022YFB2702000), and by the National Natural Science Foundation of China (Grant Nos. 62102037, 61932019). Work of Xiao Wang is supported by DARPA under Contract No. HR001120C0087, NSF award #2016240, and research awards from Meta and Google. The views, opinions, and/or findings expressed are those of the author(s) and should not be interpreted as representing the official views or policies of the Department of Defense or the U.S. Government. Yu Yu was supported by the National Natural Science Foundation of China (Grant Nos. 62125204 and 92270201), the National Key Research and Development Program of China (Grant No. 2018YFA0704701), and the Major Program of Guangdong Basic and Applied Research (Grant No. 2019B030302008). Yu Yu also acknowledges the support from the XPLORER PRIZE.

References

[ASH+20] Jackson Abascal, Mohammad Hossein Faghihi Sereshgi, Carmit Hazay, Yuval Ishai, and Muthuramakrishnan Venkitasubramaniam. Is the classical GMW paradigm practical? The case of non-interactive actively secure 2PC. In *ACM Conf. on Computer and Communications Security (CCS) 2020*, pages 1591–1605. ACM Press, 2020.

[BBMH+21] Carsten Baum, Lennart Braun, Alexander Munch-Hansen, Benoît Razet, and Peter Scholl. Appenzeller to brie: Efficient zero-knowledge proofs for mixed-mode arithmetic and Z2k. In *ACM Conf. on Computer and Communications Security (CCS) 2021*, pages 192–211. ACM Press, 2021.

[BBMHS22] Carsten Baum, Lennart Braun, Alexander Munch-Hansen, and Peter Scholl. Moz$\mathbb{Z}_{2^k}$arella: Efficient vector-OLE and zero-knowledge proofs over $\mathbb{Z}_{2^k}$. In *Advances in Cryptology—Crypto 2022, Part IV*, volume 13510 of *LNCS*, pages 329–358. Springer, 2022.

[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 *ACM Conf. on Computer and Communications Security (CCS) 2019*, pages 291–308. ACM Press, 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 *Advances in Cryptology—Crypto 2019, Part III*, volume 11694 of *LNCS*, pages 489–518. Springer, 2019.

[BCG+22] Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, Lisa Kohl, Nicolas Resch, and Peter Scholl. Correlated pseudorandomness from expand-accumulate codes. In *Advances in Cryptology—Crypto 2022, Part II*, volume 13508 of *LNCS*, pages 603–633. Springer, 2022.

[BDOZ11] Rikke Bendlin, Ivan Damgård, Claudio Orlandi, and Sarah Zakarias. Semi-homomorphic encryption and multiparty computation. In *Advances in Cryptology—Eurocrypt 2011*, volume 6632 of *LNCS*, pages 169–188. Springer, 2011.

[BFKL94] Avrim Blum, Merrick L. Furst, Michael J. Kearns, and Richard J. Lipton. Cryptographic primitives based on hard learning problems. In *Advances in Cryptology—Crypto 1993*, volume 773 of *LNCS*, pages 278–291. Springer, 1994.

[BHKR13] Mihir Bellare, Viet Tung Hoang, Sriram Keelveedhi, and Phillip Rogaway. Efficient garbling from a fixed-key blockcipher. In *IEEE Symposium on Security and Privacy (S&P) 2013*, pages 478–492, 2013.

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

[BMRS21] Carsten Baum, Alex J. Malozemoff, Marc B. Rosen, and Peter Scholl. Mac'n'cheese: Zero-knowledge proofs for boolean and arithmetic circuits with nested disjunctions. In *Advances in Cryptology—Crypto 2021, Part IV*, volume 12828 of *LNCS*, pages 92–122. Springer, 2021.

[Can00] Ran Canetti. Security and composition of multiparty cryptographic protocols. *J. Cryptology*, 13(1):143–202, January 2000.

[Cle86] Richard Cleve. Limits on the security of coin flips when half the processors are faulty (extended abstract). In *18th Annual ACM Symposium on Theory of Computing (STOC)*, pages 364–369. ACM Press, 1986.

[CRR21] Geoffroy Couteau, Peter Rindal, and Srinivasan Raghuraman. Silver: Silent VOLE and oblivious transfer from hardness of decoding structured LDPC codes. In *Advances in Cryptology—Crypto 2021, Part III*, volume 12827 of *LNCS*, pages 502–534. Springer, 2021.

[DILO22a] Samuel Dittmer, Yuval Ishai, Steve Lu, and Rafail Ostrovsky. Authenticated garbling from simple correlations. In *Advances in Cryptology—Crypto 2022, Part IV*, volume 13510 of *LNCS*, pages 57–87. Springer, 2022.

[DILO22b] Samuel Dittmer, Yuval Ishai, Steve Lu, and Rafail Ostrovsky. Improving line-point zero knowledge: Two multiplications for the price of one. In *ACM Conf. on Computer and Communications Security (CCS) 2022*, pages 829–841. ACM Press, 2022.

[DIO21] Samuel Dittmer, Yuval Ishai, and Rafail Ostrovsky. Line-point zero knowledge and its applications. In *2nd Conference on Information-Theoretic Cryptography*, 2021.

[DK99] Yevgeniy Dodis and Sanjeev Khanna. Space time tradeoffs for graph properties. In *Intl. Colloquium on Automata, Languages, and Programming (ICALP)*, volume 1644 of *LNCS*, pages 291–300. Springer, 1999.

[DNNR17] Ivan Damgård, Jesper Buus Nielsen, Michael Nielsen, and Samuel Ranellucci. The TinyTable protocol for 2-party secure computation, or: Gate-scrambling revisited. In *Advances in Cryptology—Crypto 2017, Part I*, volume 10401 of *LNCS*, pages 167–187. Springer, 2017.

[FS87] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In *Advances in Cryptology—Crypto 1986*, volume 263 of *LNCS*, pages 186–194. Springer, 1987.

[GKW+20] Chun Guo, Jonathan Katz, Xiao Wang, Chenkai Weng, and Yu Yu. Better concrete security for half-gates garbling (in the multi-instance setting). In *Advances in Cryptology—Crypto 2020, Part II*, volume 12171 of *LNCS*, pages 793–822. Springer, 2020.

[GKWY20] Chun Guo, Jonathan Katz, Xiao Wang, and Yu Yu. Efficient and secure multiparty computation from fixed-key block ciphers. In *IEEE Symposium on Security and Privacy (S&P) 2020*, pages 825–841, 2020.

[GMW87] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game or A completeness theorem for protocols with honest majority. In *19th Annual ACM Symposium on Theory of Computing (STOC)*, pages 218–229. ACM Press, 1987.

[Gol04] Oded Goldreich. *Foundations of Cryptography: Basic Applications*, volume 2. Cambridge University Press, Cambridge, UK, 2004.

[HIV17] Carmit Hazay, Yuval Ishai, and Muthuramakrishnan Venkitasubramaniam. Actively secure garbled circuits with constant communication overhead in the plain model. In *Theory of Cryptography Conference (TCC) 2017*, volume 10678 of *LNCS*, pages 3–39. Springer, 2017.

[HKE12] Yan Huang, Jonathan Katz, and David Evans. Quid-Pro-Quo-tocols: Strengthening semi-honest protocols with dual execution. In *IEEE Symposium on Security and Privacy (S&P) 2012*, pages 272–284, 2012.

[HSS17] Carmit Hazay, Peter Scholl, and Eduardo Soria-Vazquez. Low cost constant round MPC combining BMR and oblivious transfer. In *Advances in Cryptology—Asiacrypt 2017, Part I*, volume 10624 of *LNCS*, pages 598–628. Springer, 2017.

[HSS20] Carmit Hazay, Peter Scholl, and Eduardo Soria-Vazquez. Low cost constant round MPC combining BMR and oblivious transfer. *J. Cryptology*, 33(4):1732–1786, October 2020.

[HsV20] Carmit Hazay, abhi shelat, and Muthuramakrishnan Venkitasubramaniam. Going beyond dual execution: MPC for functions with efficient verification. In *Intl. Conference on Theory and Practice of Public Key Cryptography 2020, Part II*, volume 12111 of *LNCS*, pages 328–356. Springer, 2020.

[KRRW18] Jonathan Katz, Samuel Ranellucci, Mike Rosulek, and Xiao Wang. Optimizing authenticated garbling for faster secure two-party computation. In *Advances in Cryptology—Crypto 2018, Part III*, volume 10993 of *LNCS*, pages 365–391. Springer, 2018.

[KS08] Vladimir Kolesnikov and Thomas Schneider. Improved garbled circuit: Free XOR gates and applications. In *Intl. Colloquium on Automata, Languages, and Programming (ICALP)*, volume 5126 of *LNCS*, pages 486–498. Springer, 2008.

[LPSY15] Yehuda Lindell, Benny Pinkas, Nigel P. Smart, and Avishay Yanai. Efficient constant round multi-party computation combining BMR and SPDZ. In *Advances in Cryptology—Crypto 2015, Part II*, volume 9216 of *LNCS*, pages 319–338. Springer, 2015.

[LSS16] Yehuda Lindell, Nigel P. Smart, and Eduardo Soria-Vazquez. More efficient constant-round multi-party computation from BMR and SHE. In *Theory of Cryptography Conference (TCC) 2016*, volume 9985 of *LNCS*, pages 554–581. Springer, 2016.

[MF06] Payman Mohassel and Matthew Franklin. Efficiency tradeoffs for malicious two-party computation. In *Intl. Conference on Theory and Practice of Public Key Cryptography*, volume 3958 of *LNCS*, pages 458–473. Springer, 2006.

[NNOB12] Jesper Buus Nielsen, Peter Sebastian Nordholt, Claudio Orlandi, and Sai Sheshank Burra. A new approach to practical active-secure two-party computation. In *Advances in Cryptology—Crypto 2012*, volume 7417 of *LNCS*, pages 681–700. Springer, 2012.

[RR21] Mike Rosulek and Lawrence Roy. Three halves make a whole? Beating the half-gates lower bound for garbled circuits. In *Advances in Cryptology—Crypto 2021, Part I*, volume 12825 of *LNCS*, pages 94–124. Springer, 2021.

[WRK17a] Xiao Wang, Samuel Ranellucci, and Jonathan Katz. Authenticated garbling and efficient maliciously secure two-party computation. In *ACM Conf. on Computer and Communications Security (CCS) 2017*, pages 21–37. ACM Press, 2017.

[WRK17b] Xiao Wang, Samuel Ranellucci, and Jonathan Katz. Global-scale secure multiparty computation. In *ACM Conf. on Computer and Communications Security (CCS) 2017*, pages 39–56. ACM Press, 2017.

[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 *IEEE Symposium on Security and Privacy (S&P) 2021*, pages 1074–1091, 2021.

[WYX+21] Chenkai Weng, Kang Yang, Xiang Xie, Jonathan Katz, and Xiao Wang. Mystique: Efficient conversions for zero-knowledge proofs with applications to machine learning. In *USENIX Security Symposium 2021*, pages 501–518. USENIX Association, 2021.

[WYY+22] Chenkai Weng, Kang Yang, Zhaomin Yang, Xiang Xie, and Xiao Wang. AntMan: Interactive zero-knowledge proofs with sublinear communication. In *ACM Conf. on Computer and Communications Security (CCS) 2022*, pages 2901–2914. ACM Press, 2022.

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

[YSWW21] Kang Yang, Pratik Sarkar, Chenkai Weng, and Xiao Wang. QuickSilver: Efficient and affordable zero-knowledge proofs for circuits and polynomials over any field. In *ACM Conf. on Computer and Communications Security (CCS) 2021*, pages 2986–3001. ACM Press, 2021.

[YWL+20] Kang Yang, Chenkai Weng, Xiao Lan, Jiang Zhang, and Xiao Wang. Ferret: Fast extension for correlated OT with small communication. In *ACM Conf. on Computer and Communications Security (CCS) 2020*, pages 1607–1626. ACM Press, 2020.

[YWZ20] Kang Yang, Xiao Wang, and Jiang Zhang. More efficient MPC from improved triple generation and authenticated garbling. In *ACM Conf. on Computer and Communications Security (CCS) 2020*, pages 1627–1646. ACM Press, 2020.

[ZRE15] Samee Zahur, Mike Rosulek, and David Evans. Two halves make a whole - reducing data transfer in garbled circuits using half gates. In *Advances in Cryptology—Eurocrypt 2015, Part II*, volume 9057 of *LNCS*, pages 220–250. Springer, 2015.

Supplementary Material

A Security Model and Functionalities

A.1 Security Model

We say that a two-party protocol Π *securely realizes* an ideal functionality $\mathcal{F}$ if for any probabilistic polynomial time (PPT) adversary $\mathcal{A}$, there exists a PPT adversary (a.k.a., simulator) $\mathcal{S}$, such that the joint distribution of the outputs of the honest party and $\mathcal{A}$ in the *real-world* execution where the party interacts with $\mathcal{A}$ and execute Π is computationally indistinguishable from that of the outputs of the honest party and $\mathcal{S}$ in the *ideal-world* execution where the party interacts with $\mathcal{S}$ and $\mathcal{F}$. We adopt the notion of security with abort, where fairness is not achieved in the two-party setting [Cle86]. For all our functionalities, the adversary can send abort to these functionalities at any time, and then the execution is aborted. For the sake of simplicity, we omit the description in these functionalities.

A.2 The Equality-Check Functionality

Our protocol will invoke a relaxed equality-checking functionality $\mathcal{F}_{\mathsf{EQ}}$ [NNOB12] that is recalled in Figure 9. This functionality can be securely realized by committing to the input and then opening it, as we allow to leak the inputs if two inputs are different. The protocol realizing $\mathcal{F}_{\mathsf{EQ}}$ needs two rounds and takes $2\kappa + \ell$ bits of one-way communication for ℓ-bit inputs.

Functionality $\mathcal{F}_{\mathsf{EQ}}$

Upon receiving $(\mathsf{eq}, sid, \ell, x)$ from $\mathsf{P_A}$ and $(\mathsf{eq}, sid, \ell, y)$ from $\mathsf{P_B}$, where $x, y \in \{0,1\}^\ell$, this functionality executes as follows:

- If $x = y$, then send (sid, true) to both parties.
- Otherwise, send (sid, false) to both parties, and also send the input of the honest party to the adversary.

Figure 9: Two-party equality-checking functionality.

A.3 The Coin-Tossing Functionality

Our protocol will use a standard coin-tossing functionality $\mathcal{F}_{\mathsf{Rand}}$ shown in Figure 10, which samples a uniform element in $\mathbb{F}_{2^\kappa}$. This can be securely realized by having every party commit to a random element via calling $\mathcal{F}_{\mathsf{Com}}$, and then open the commitments and use the sum of all random elements as the output.

Functionality $\mathcal{F}_{\mathsf{Rand}}$

Upon receiving (Rand, sid) from two parties $\mathsf{P_A}$ and $\mathsf{P_B}$, sample $r \leftarrow \mathbb{F}_{2^\kappa}$ and sends (sid, r) to both parties.

Figure 10: Two-party coin-tossing functionality.

B Table of Notation

In Table 2, we summarize the notation and macros used in our protocols to help the reader retrieving the definition of each notation fast. The notation and macros were also described in the previous sections.

Notation	Definitions
κ	Computational security parameter
ρ	Statistical security parameter
$x \leftarrow S$	Sample x uniformly at random from S
$[a, b)$ and $[a, b]$	$\{a, \ldots, b-1\}$ and $\{a, \ldots, b\}$
$\boldsymbol{a}$, a_i, $\mathbf{A}$	Vector, the i-th component of $\boldsymbol{a}$, matrix
$\{x_i\}$	A set without specifying the indices
$\mathsf{lsb}(x), \mathsf{msb}(x)$	Least significant bit of x, most significant bit of x.
$\mathsf{B2F}$	Macro to convert from $\mathbb{F}_2^{\kappa}$ to $\mathbb{F}_{2^{\kappa}}$
$\mathsf{F2B}$	Macro to convert from $\mathbb{F}_{2^{\kappa}}$ to $\mathbb{F}_2^{\kappa}$
$\mathcal{C}$, $\mathcal{O}$	A Boolean circuit, the set of circuit-output wires in $\mathcal{C}$
$\mathcal{I}_{\mathsf{A}}, \mathcal{I}_{\mathsf{B}}$	The sets of circuit-input wires of P_{A} and P_{B}
$\mathcal{C}_{\mathsf{and}}, \mathcal{W}$	The set of all AND gates and set of their output wires
n, m, t	Parameters $n = \lvert\mathcal{W}\rvert + \lvert\mathcal{I}_{\mathsf{B}}\rvert, m = \lvert\mathcal{W}\rvert + \lvert\mathcal{I}_{\mathsf{A}}\rvert, t = \lvert\mathcal{W}\rvert$
L	Compression parameter $L = \lceil \rho \log \frac{2en}{\rho} + \frac{\log \rho}{2} \rceil$
$[x]_{\mathsf{A},G}$	IT-MAC where x held by P_{A} is authenticated under G
$\langle x \rangle$	Dual-key authenticated value on x under $\Delta_{\mathsf{A}}\Delta_{\mathsf{B}}$
$\mathsf{CheckZero}([x])$	Check that x is equal to 0
$\mathsf{CheckZero2}(\langle x \rangle)$	Check that x is equal to 0
$\mathsf{Open}([x]_{\mathsf{A}})$	P_{A} opens x to P_{B} in an authenticated way
$\mathsf{Convert1}_{[\cdot] \to \langle \cdot \rangle}([x\Delta_{\mathsf{B}}]_{\Delta_{\mathsf{A}}})$	Convert $[x\Delta_{\mathsf{B}}]_{\Delta_{\mathsf{A}}}$ to a dual-key authenticated bit $\langle x \rangle$
$\mathsf{Convert2}_{[\cdot] \to \langle \cdot \rangle}([x]_{\mathsf{A},\beta}, \langle y \rangle)$	Convert $[x]_{\mathsf{A},\beta}$ along with $\langle y \rangle$ to $\langle xy \rangle$
$\mathsf{EQCheck}$	Check equality of values *auth.* under different global keys
$\mathsf{Garble}, \mathsf{Eval}$	Generation and evaluation of distributed garbling

Table 2: Definitions of the notation and macros used in this paper.

C Proofs of Security

C.1 Row-independence of Random Matrix

Let $L = \lceil \rho \log(\frac{2en}{\rho}) + \frac{\log \rho}{2} \rceil$ and let $\mathbf{M} \leftarrow \mathbb{F}_2^{n \times L}$ be a uniformly random matrix. In the following we show that $\mathbf{M}$ satisfies the (n, ρ)-independent property except with probability $2^{-\rho}$.

Recall that the property states that any ρ rows of the matrix are linearly independent. Since we are working in the binary field, a set of vectors in $\mathbb{F}_2^L$ being linearly dependent implies that they XOR to 0, which happens with probability 2^{-L} for uniformly random vectors. Therefore, denote $\mathcal{R}$ as the random variable counting the number of linearly dependent sets with size no more than ρ, then by the linearity of expectation we have:

$$\mathbb{E}[\mathcal{R}] = \sum_{k=1}^{\rho} \binom{n}{k} 2^{-L} \quad .$$

Using Markov's inequality we have

$$\Pr[\mathcal{R} \geq 1] \leq \mathbb{E}[\mathcal{R}] = \sum_{k=1}^{\rho} \binom{n}{k} 2^{-L} \quad .$$

In our secure computation setting n is the number of circuit input gates and AND gates so we may assume $n > 2\rho$. Thus we have

$$\Pr[\mathcal{R} \geq 1] \leq \frac{n^{\rho}}{\rho!} \cdot \frac{\rho}{2^L} \quad .$$

Using Stirling's approximation and taking $L \geq \lceil \rho \log(\frac{2en}{\rho}) + \frac{\log \rho}{2} \rceil$ we have

$$\begin{aligned} \Pr[\mathcal{R} \geq 1] &\leq \frac{n^{\rho}}{2\sqrt{\rho}(\frac{\rho}{e})^{\rho}} \cdot \frac{\rho}{(\frac{2en}{\rho})^{\rho} \cdot \sqrt{\rho}} \\ &\leq 2^{-(\rho+1)} < 2^{-\rho}, \end{aligned}$$

which implies $\Pr[\mathcal{R} = 0] \geq 1 - 2^{-\rho}$.

C.2 Proof of Lemma 5

Proof. Suppose $y_i \neq y_i'$. Let δ be the i-th component h's preimage, then $\mathsf{P_B}$ passing the check is equivalent to $(y_i\Delta_\mathsf{A} - m_i)\Delta_\mathsf{A}' - (y_i'\Delta_\mathsf{A}' - m_i')\Delta_\mathsf{A} + \tilde{m}_i\Delta_\mathsf{A}' - \tilde{m}_i'\Delta_\mathsf{A} = \delta + \mathsf{M_B}[\tilde{m}_i] - \mathsf{M_B}[\tilde{m}_i']$, which implies that $(y_i - y_i')\Delta_\mathsf{A}\Delta_\mathsf{A}' + (\tilde{m}_i - m_i)\Delta_\mathsf{A}' - (\tilde{m}_i' - m_i')\Delta_\mathsf{A} + \mathsf{M_B}[\tilde{m}_i'] - \mathsf{M_B}[\tilde{m}_i] - \delta = 0$. Since this is a bivariate polynomial whose coefficients are independent of the evaluation point $\Delta_\mathsf{A}, \Delta_\mathsf{A}'$, it evaluates to 0 with at most $2 \cdot 2^{-\kappa}$ probability. Except with probability $2^{-\kappa}$, $\mathsf{P_B}$ accepts due to hash collision. Applying union bound we conclude $\mathsf{P_B}$ rejects false proof except with $\frac{3}{2^\kappa}$ probability. □

C.3 Proof of Theorem 1

Proof. **Completeness.** Lemma 3 shows that the key sampling procedure Π_samp returns keys subject to $\mathsf{lsb}(\Delta_\mathsf{A}\Delta_\mathsf{B}) = 1$, which ensures $\mathsf{lsb}(\mathsf{D_A}[x]) \oplus \mathsf{lsb}(\mathsf{D_B}[x]) = x$ for any $x \in \mathbb{F}_2$. This implies that all the $\tilde{b}_k$ values that $\mathsf{P_B}$ computes in step 9 are correct.

Now we argue security. We first present the sampling simulation as a separate process and then describe the simulation for the main protocol Π_cpre. In the following, we simulate the random oracle by recording all the query-answer pairs and answer the queries from $\mathcal{A}$ consistently.

Corrupted $\mathsf{P_A}$. $\mathcal{S}_\mathsf{A}$ first simulates the key sampling protocol Π_samp as follows:

1. $\mathcal{S}_\mathsf{A}$ receives the input key Δ_A by simulating $\mathcal{F}_\mathsf{COT}$.
2. $\mathcal{S}_\mathsf{A}$ receives m_A^0 of $\mathcal{A}$. If $\mathsf{lsb}(\Delta_\mathsf{A}) \neq 1$ then it aborts.
3. $\mathcal{S}_\mathsf{A}$ samples $\tilde{\Delta}_\mathsf{B}$ s.t. $\mathsf{msb}(\tilde{\Delta}_\mathsf{B}) = 1$, to handle the Fix command and m_B^1 message. It also fixes $[\tilde{\Delta}_\mathsf{B}]_\mathsf{B}$ accordingly.
4. $\mathcal{S}_\mathsf{A}$ simulates the init and extend commands of $\mathcal{F}_\mathsf{COT}$ internally.
5. $\mathcal{S}_\mathsf{A}$ sends m_B^0 following the protocol instructions.
6. $\mathcal{S}_\mathsf{A}$ then simulates the checking procedure as follows:

(a) $\mathcal{S}_{\mathsf{A}}$ simulates extend and Fix using previously sampled keys. It also sends true to $\mathcal{A}$ to simulate $\mathcal{F}_{\mathsf{DVZK}}$.

(b) $\mathcal{S}_{\mathsf{A}}$ receives m_{A}^2 from the adversary. If $\mathsf{lsb}(\Delta_{\mathsf{A}}\Delta_{\mathsf{B}}) \neq 1$ then $\mathcal{S}_{\mathsf{A}}$ aborts.

(c) $\mathcal{S}_{\mathsf{A}}$ extracts $\mathcal{A}$'s input of Fix as $\tilde{\Delta}_{\mathsf{A}}$.

(d) $\mathcal{S}_{\mathsf{A}}$ samples $\boldsymbol{y}$ as the output of extend and extracts $\mathcal{A}$'s input $\tilde{\Delta}_{\mathsf{A}} \cdot \tilde{\boldsymbol{y}}$ to the Fix command. If $\boldsymbol{y} \neq \tilde{\boldsymbol{y}}$ then $\mathcal{S}_{\mathsf{A}}$ aborts.

(e) $\mathcal{S}_{\mathsf{A}}$ sends m_{B}^2 according to protocol instruction.

(f)–(g) If $\Delta_{\mathsf{A}} \neq \tilde{\Delta}_{\mathsf{A}}$ then $\mathcal{S}_{\mathsf{A}}$ sends $h \leftarrow \mathbb{F}_{2^\kappa}$ to $\mathcal{A}$ and aborts to simulate $\mathsf{CheckZero2}$. Otherwise it follows the protocol instruction.

Then $\mathcal{S}_{\mathsf{A}}$ simulates the main protocol Π_{cpre}.

1. $\mathcal{S}_{\mathsf{A}}$ samples $\mathbf{M}$ and sends it to $\mathcal{A}$.

2. $\mathcal{S}_{\mathsf{A}}$ locally simulates the extend command and gets $\boldsymbol{b}^*$.

3. $\mathcal{S}_{\mathsf{A}}$ simulates the Fix command using previously sampled $\boldsymbol{b}^*$ and Δ_{B}.

4–5 $\mathcal{S}_{\mathsf{A}}$ simulates the init command internally and sends $\boldsymbol{a} \leftarrow \mathbb{F}_2^m$ and $\hat{\boldsymbol{a}} \leftarrow \mathbb{F}_2^t$ to $\mathcal{A}$ to simulate $\mathcal{F}_{\mathsf{bCOT}}^{L+1}$ and $\mathcal{F}_{\mathsf{bCOT}}^{2}$. Then it extracts $(\Delta'_{\mathsf{A}})^{(1)}$ and $(\Delta'_{\mathsf{A}})^{(2)}$ respectively from the two Fix commands.

6. $\mathcal{S}_{\mathsf{A}}$ follows the protocol instruction.

7. $\mathcal{S}_{\mathsf{A}}$ simulates Fix using uniformly random messages. It also extracts $\tilde{a}_{i,j}$ from the Fix command from $\mathcal{A}$.

8. $\mathcal{S}_{\mathsf{A}}$ follows the protocol instruction.

9. $\mathcal{S}_{\mathsf{A}}$ receives the $\mathsf{lsb}(\mathsf{D}_{\mathsf{A}}[\tilde{b}_k])$ message from $\mathcal{A}$.

10. $\mathcal{S}_{\mathsf{A}}$ simulates Fix using uniformly random messages.

11. $\mathcal{S}_{\mathsf{A}}$ simulates $\mathcal{F}_{\mathsf{DVZK}}$ on $([b_i], [b_j], [b_{i,j}])$ for each AND gate $(i, j, k, \wedge)$ and $([b_i^*], [\Delta_{\mathsf{B}}], [B_i^*])$ by sending true to $\mathcal{A}$. If the previously extracted $\tilde{a}_{i,j} \neq a_i a_j$ then $\mathcal{S}_{\mathsf{A}}$ aborts.

12. $\mathcal{S}_{\mathsf{A}}$ extracts $\mathcal{A}$'s input to $\mathcal{F}_{\mathsf{COT}}$ as $(\Delta'_{\mathsf{A}})^{(3)}$. If $(\Delta'_{\mathsf{A}})^{(1)} \neq (\Delta'_{\mathsf{A}})^{(2)}$ or $(\Delta'_{\mathsf{A}})^{(2)} \neq (\Delta'_{\mathsf{A}})^{(3)}$ then $\mathcal{S}_{\mathsf{A}}$ sends $h \leftarrow \mathbb{F}_{2^\kappa}$ to simulate $\mathsf{CheckZero2}$ and aborts. Otherwise it follows the protocol instruction to simulate $\mathsf{EQCheck}$.

13. $\mathcal{S}_{\mathsf{A}}$ follows the protocol instruction.

14. $\mathcal{S}_{\mathsf{A}}$ simulates $\mathcal{F}_{\mathsf{Rand}}$ internally and sends $\chi \leftarrow \mathbb{F}_{2^\kappa}$ to $\mathcal{A}$.

15. $\mathcal{S}_{\mathsf{A}}$ sends $y := \sum_{k \in \mathcal{W}} \chi^k \cdot \tilde{b}_k + r$ to $\mathcal{A}$. If the previous $\mathsf{lsb}(\mathsf{D}_{\mathsf{A}}[\tilde{b}_k])$ messages are erroneous then $\mathcal{S}_{\mathsf{A}}$ sends $h \leftarrow \mathbb{F}_{2^\kappa}$ to $\mathcal{F}_{\mathsf{EQ}}$ and aborts to simulate $\mathsf{CheckZero2}$. Otherwise it follows protocol instructions.

16. $\mathcal{S}_{\mathsf{A}}$ follows the protocol instruction for $\mathsf{CheckZero}$.

Now we argue that the ideal world output and the real world output are indistinguishable using a series of hybrids.

Hybrid 1 This is the real-world execution.

Hybrid 2 $\mathcal{S}_\mathsf{A}$ extracts Δ_A in step 1. If $\mathsf{lsb}(\Delta_\mathsf{A}) \neq 1$ then $\mathcal{S}_\mathsf{A}$ aborts in step 2. By Lemma 3 the two hybrids are $2^{-\rho}$-indistinguishable.

Hybrid 3 $\mathcal{S}_\mathsf{A}$ samples independent $\tilde{\Delta}_\mathsf{B}$ for step 3. Since the functionality $\mathcal{F}_\mathsf{COT}$ is ideal, the two hybrids are identically distributed.

Hybrid 4 $\mathcal{S}_\mathsf{A}$ sends true to $\mathcal{A}$ and locally verify the multiplicative relation to simulate $\mathcal{F}_\mathsf{DVZK}$ in all subsequent hybrids. Since the functionality $\mathcal{F}_\mathsf{DVZK}$ is ideal, the two hybrids are identically distributed.

Hybrid 5 $\mathcal{S}_\mathsf{A}$ receives the message m_A^2 from $\mathcal{A}$. If $\mathsf{lsb}(\Delta_\mathsf{A}\Delta_\mathsf{B}) \neq 1$ then $\mathcal{S}_\mathsf{A}$ aborts. By Lemma 3 the two hybrids are $2^{-\rho}$-indistinguishable.

Hybrid 6 $\mathcal{S}_\mathsf{A}$ extracts $\tilde{\Delta}_\mathsf{A}$ in step 6c. If $\tilde{\Delta}_\mathsf{A} \neq \Delta_\mathsf{A}$ then $\mathcal{S}_\mathsf{A}$ sends $h \leftarrow \mathbb{F}_{2^\kappa}$ to $\mathcal{F}_\mathsf{EQ}$ and aborts to simulate $\mathsf{CheckZero2}$. In $\mathbf{Hybrid_5}$ we have $h = \mathsf{H}((\Delta_\mathsf{A} - \tilde{\Delta}_\mathsf{A})\Delta_\mathsf{B} + \mathsf{D}_\mathsf{A}[1_\mathsf{B}] - \mathsf{D}_\mathsf{A}[1_\mathsf{A}])$ which is computationally indistinguishable from uniform randomness. Together with Lemma 2 we conclude that the two hybrids are $(\frac{2+\mathsf{poly}(\kappa)}{2^\kappa})$-indistinguishable.

Hybrid 7 $\mathcal{S}_\mathsf{A}$ extracts the Fix command input $(\Delta'_\mathsf{A})^{(1)}$ and $(\Delta'_\mathsf{A})^{(2)}$ in step 4 and step 5 respectively. $\mathcal{S}_\mathsf{A}$ also extracts $(\Delta'_\mathsf{A})^{(3)}$ from $\mathcal{F}_\mathsf{COT}$ in step 12. If $(\Delta'_\mathsf{A})^{(1)} \neq (\Delta'_\mathsf{A})^{(2)}$ or $(\Delta'_\mathsf{A})^{(2)} \neq (\Delta'_\mathsf{A})^{(3)}$ then $\mathcal{S}_\mathsf{A}$ sends $h \leftarrow \mathbb{F}_{2^\kappa}$ and aborts to simulate $\mathsf{CheckZero2}$. Since the Fix messages in $\mathsf{EQCheck}$ are uniformly random, using the similar argument as in $\mathbf{Hybrid_6}$, the two hybrids are $(\frac{2+\mathsf{poly}(\kappa)}{2^\kappa})$-indistinguishable.

Hybrid 8 If $\mathcal{A}$ sends incorrect $\mathsf{lsb}(\mathsf{D}_\mathsf{A}[\tilde{b}_k])$ values in step 9 then $\mathcal{S}_\mathsf{A}$ simulates the $\mathsf{CheckZero2}$ command using previous strategy. Since χ is uniformly random, by the Schwartz-Zippel lemma the two hybrids are $(\frac{t+2+\mathsf{poly}(\kappa)}{2^\kappa})$-indistinguishable. This is the ideal world execution.

Therefore, the ideal world and real world executions are $(\frac{2}{2^\rho} + \frac{t+6+\mathsf{poly}(\kappa)}{2^\kappa})$-indistinguishable in the corrupted P_A case.

Corrupted P_B. $\mathcal{S}_\mathsf{B}$ first simulates the key sampling protocol Π_samp as follows:

1. $\mathcal{S}_\mathsf{B}$ simulates the init and extend command internally.
2. $\mathcal{S}_\mathsf{B}$ sends m_A^0 following protocol instruction.
3. $\mathcal{S}_\mathsf{B}$ extracts $\tilde{\Delta}_\mathsf{B}$ from the Fix macro, sends m_A^1 and receives m_B^1. It fixes $[\tilde{\Delta}_\mathsf{B}]_\mathsf{B}$ according to protocol instruction.
4. $\mathcal{S}_\mathsf{B}$ extracts $\hat{\Delta}_\mathsf{B}$ from the init command.
5. $\mathcal{S}_\mathsf{B}$ receives m_B^0 from $\mathcal{A}$ and aborts if $\mathsf{msb}(\hat{\Delta}_\mathsf{B}) \neq 1$.
6. $\mathcal{S}_\mathsf{B}$ simulates the checking procedure as follows:
   (a) $\mathcal{S}_\mathsf{B}$ sends $\boldsymbol{x} \leftarrow \mathbb{F}_{2^\rho}$ to simulate extend. It also extracts $\tilde{\Delta}_\mathsf{B} \cdot \tilde{\boldsymbol{x}}$ from the Fix command. If $\boldsymbol{x} \neq \tilde{\boldsymbol{x}}$ then it aborts.
   (b) $\mathcal{S}_\mathsf{B}$ sends m_A^2 according to protocol instruction.
   (c) $\mathcal{S}_\mathsf{B}$ samples Δ_A to simulate Fix.
   (d) $\mathcal{S}_\mathsf{B}$ samples $\boldsymbol{y}$ to simulate extend and Fix. It then sends true to $\mathcal{A}$ to simulate $\mathcal{F}_\mathsf{DVZK}$.

(e) $\mathcal{S}_{\mathsf{B}}$ receives m^2_{B} from $\mathcal{A}$ and aborts if $\mathsf{lsb}(\Delta_{\mathsf{A}}\hat{\Delta}_{\mathsf{B}}) \neq 1$.

(f)–(g) If $\tilde{\Delta}_{\mathsf{B}} \neq \hat{\Delta}_{\mathsf{B}}$ then $\mathcal{S}_{\mathsf{B}}$ sends $h \leftarrow \mathbb{F}_{2^\kappa}$ and aborts to simulate $\mathsf{CheckZero2}$.

$\mathcal{S}_{\mathsf{B}}$ then simulates the main protocol Π_{cpre} as follows.

1. $\mathcal{S}_{\mathsf{B}}$ receives the compression matrix $\mathbf{M}$ from $\mathcal{A}$.
2. $\mathcal{S}_{\mathsf{B}}$ samples $\boldsymbol{b}^*$ to simulate the extend command.
3. $\mathcal{S}_{\mathsf{B}}$ extracts the inputs $\{b^*_i \Delta_{\mathsf{B}}\}_{i\in[1,L]}$ from the Fix command of $\mathcal{A}$.

4–5 $\mathcal{S}_{\mathsf{B}}$ extracts the input $(\beta_1, ..., \beta_L, \Delta^{(1)}_{\mathsf{B}})$ and $(\beta_0, \Delta^{(2)}_{\mathsf{B}})$ from the init command. Then $\mathcal{S}_{\mathsf{B}}$ follows protocol instructions.

6–8 $\mathcal{S}_{\mathsf{B}}$ follows protocol specifications to generate $a_{i,j}$ for each AND gate $(i, j, k, \wedge)$. Then it extracts $b_{i,j}$ from $\mathcal{A}$'s input to Fix and generates $\langle \hat{a}_k \rangle, \langle a_{i,j} \rangle$ following protocol specifications.

9. $\mathcal{S}_{\mathsf{B}}$ follows protocol specifications.
10. $\mathcal{S}_{\mathsf{B}}$ extracts the input $\{\hat{b}_k\}$ of the Fix command.
11. $\mathcal{S}_{\mathsf{B}}$ simulates the $\mathcal{F}_{\mathsf{DVZK}}$ functionality by sending true to $\mathcal{A}$. If the extracted values in previous step 3 and step 6 do not satisfy the multiplicative relation then $\mathcal{S}_{\mathsf{B}}$ aborts.
12. $\mathcal{S}_{\mathsf{B}}$ extracts the $\mathcal{A}$'s input $\Delta^{(3)}_{\mathsf{B}}$ from the Fix command. If $\Delta^{(1)}_{\mathsf{B}} \neq \Delta^{(2)}_{\mathsf{B}}$ or $\Delta^{(2)}_{\mathsf{B}} \neq \Delta^{(3)}_{\mathsf{B}}$ then $\mathcal{S}_{\mathsf{B}}$ sends $h \leftarrow \mathbb{F}_{2^\kappa}$ to $\mathcal{F}_{\mathsf{EQ}}$ and aborts to simulate $\mathsf{CheckZero2}$. If $\Delta^{(3)}_{\mathsf{B}} \neq \Delta_{\mathsf{B}}$ (the latter one is from simulation of Π_{samp}) or the $\{\beta_i\}$ inputs from step 3 are inconsistent then $\mathcal{S}_{\mathsf{B}}$ aborts to simulate $\mathsf{EQCheck}$.
13. $\mathcal{S}_{\mathsf{B}}$ samples $\boldsymbol{r} \leftarrow \mathbb{F}^\kappa_2$ to simulate extend. Define $r := \mathsf{B2F}(\boldsymbol{r})$. Then it extracts $\tilde{r} \cdot \Delta_{\mathsf{B}}$ in the Fix command.
14. $\mathcal{S}_{\mathsf{B}}$ simulates $\mathcal{F}_{\mathsf{Rand}}$ by sending $\chi \leftarrow \mathbb{F}_{2^\kappa}$ to $\mathcal{A}$. Define $y := \sum_{k\in\mathcal{W}} \chi^k \cdot \tilde{b}_k + \tilde{r}$.
15. $\mathcal{S}_{\mathsf{B}}$ receives $\tilde{y}$ from $\mathcal{A}$. If $y \neq \tilde{y}$ then $\mathcal{S}_{\mathsf{B}}$ sends $h \leftarrow \mathbb{F}_{2^\kappa}$ to $\mathcal{F}_{\mathsf{EQ}}$ and aborts to simulate $\mathsf{CheckZero2}$.
16. If the $\hat{b}_k$ extracted in step 10 are incorrect or $r \neq \tilde{r}$ the $\mathcal{S}_{\mathsf{B}}$ aborts.

Now we argue the ideal world and real world are indistinguishable by a series of hybrid experiments.

Hybrid 1 This is the real-world execution.

Hybrid 2 $\mathcal{S}_{\mathsf{B}}$ uses independently sampled key Δ_{A} for the init command in step 2 and step 6c. Since $\mathcal{F}_{\mathsf{COT}}$ is perfect and messages in Fix are fully masked, we conclude that the two hybrids are identically distributed.

Hybrid 3 $\mathcal{S}_{\mathsf{B}}$ extracts $\tilde{\Delta}_{\mathsf{B}}$ from the Fix command and fix it following protocol instructions. It also extract Δ_{B} from the init command. If $\mathsf{msb}(\Delta_{\mathsf{B}}) \neq 1$ then $\mathcal{S}_{\mathsf{B}}$ aborts. By Lemma 3 the two hybrids are $2^{-\rho}$-indistinguishable.

Hybrid 4 In the following hybrids $\mathcal{S}_{\mathsf{B}}$ simulates $\mathcal{F}_{\mathsf{DVZK}}$ by sending true to $\mathcal{A}$ for $\mathsf{P_A}$'s multiplicative relation while aborts if the verification of $\mathsf{P_B}$'s multiplicative relation fails. Since $\mathcal{F}_{\mathsf{DVZK}}$ is ideal, this does not change the distribution.

Hybrid 5 $\mathcal{S}_{\mathsf{B}}$ aborts if $\mathsf{lsb}(\Delta_{\mathsf{A}}\Delta_{\mathsf{B}}) \neq 1$. By Lemma 3 the two hybrids are $(2 \cdot 2^{-\kappa} + 2^{-\rho})$-indistinguishable.

Hybrid 6 If $\Delta_{\mathsf{B}} \neq \tilde{\Delta}_{\mathsf{B}}$ then $\mathcal{S}_{\mathsf{B}}$ sends $h \leftarrow \mathbb{F}_{2^\kappa}$ to $\mathcal{F}_{\mathsf{EQ}}$ and aborts to simulate $\mathsf{CheckZero2}$. Otherwise it follows protocol specifications. If $\Delta_{\mathsf{B}} \neq \tilde{\Delta}_{\mathsf{B}}$ then $h = \mathsf{H}((\Delta_{\mathsf{B}} - \tilde{\Delta}_{\mathsf{B}})\Delta_{\mathsf{A}} + \mathsf{D}_{\mathsf{B}}[1_{\mathsf{B}}] - \mathsf{D}_{\mathsf{B}}[1_{\mathsf{A}}])$ in $\mathbf{Hybrid_5}$, which is computationally indistinguishable from uniform randomness. Together with Lemma 3 we conclude that the two hybrids are $(\frac{2+\mathsf{poly}(\kappa)}{2^\kappa})$-indistinguishable.

Hybrid 7 $\mathcal{S}_{\mathsf{B}}$ extracts the input $(\beta_1, ..., \beta_L, \Delta_{\mathsf{B}}^{(1)})$, $(\beta_0, \Delta_{\mathsf{B}}^{(2)})$ from the init command and $\Delta_{\mathsf{B}}^{(3)}$ from the Fix command. If $\Delta_{\mathsf{B}}^{(1)} \neq \Delta_{\mathsf{B}}^{(2)}$, $\Delta_{\mathsf{B}}^{(2)} \neq \Delta_{\mathsf{B}}^{(3)}$, or $\Delta_{\mathsf{B}}^{(3)} \neq \Delta_{\mathsf{B}}$ (the latter from the simulation of Π_{samp}) then $\mathcal{S}_{\mathsf{B}}$ sends $h \leftarrow \mathbb{F}_{2^\kappa}$ to $\mathcal{F}_{\mathsf{EQ}}$ and aborts in step 12. Following the same argument as the previous step the two hybrids are $(\frac{2+\mathsf{poly}(\kappa)}{2^\kappa})$-indistinguishable.

Hybrid 8 If the $\{\beta_i\}$ extracted in step 4 and step 5 are incorrect then $\mathcal{S}_{\mathsf{B}}$ aborts in step 12. By Lemma 5 and Lemma 6 the two hybrids are $3 \cdot 2^{-\kappa}$-indistinguishable.

Hybrid 9 If the $\tilde{y}$ that $\mathcal{A}$ sends in step 15 does not satisfy $\tilde{y} = \sum_{k \in \mathcal{W}} \chi^k \cdot \tilde{b}_k + \tilde{r}$ then $\mathcal{S}_{\mathsf{B}}$ follows the same simulation strategy in $\mathbf{Hybrid_6}$ for $\mathsf{CheckZero2}$. The two hybrids are $(\frac{2+\mathsf{poly}(\kappa)}{2^\kappa})$-indistinguishable.

Hybrid 10 If $\tilde{r} \neq r$ or the $\hat{b}_k$ are incorrect in step 10 then $\mathcal{S}_{\mathsf{B}}$ aborts in step 16. By Lemma 1 and the Schwartz-Zippel lemma, the two hybrids are $(\frac{t+2}{2^\kappa})$-indistinguishable. This is the ideal world execution.

Therefore, in the corrupted P_{B} case the real world and ideal world executions are $(\frac{2}{2^\rho} + \frac{t+13+\mathsf{poly}(\kappa)}{2^\kappa})$-indistinguishable.

We conclude that the protocol Π_{cpre} in Figure 5 and Figure 6 securely computes the Π_{cpre} functionality in Figure 3 in the $(\mathcal{F}_{\mathsf{COT}}, \mathcal{F}_{\mathsf{bCOT}}, \mathcal{F}_{\mathsf{DVZK}}, \mathcal{F}_{\mathsf{EQ}}, \mathcal{F}_{\mathsf{Rand}})$-hybrid model.

□

C.4 Proofs of Lemma 8 and Lemma 9

We first formalize the intuition of the consistency checking procedures in our protocol in the following lemma.

Lemma 10. *In the equality checking protocol if Equation 1 does not hold then* P_{B} *aborts except with probability* $\frac{2}{2^\kappa}$. *If Equation 2 does not hold then the message* h *from* P_{A} *is computationally indistinguishable from uniform randomness for* P_{B}.

Proof. Assuming that the hash function is a random oracle, the hash of one single point can be viewed as an obfuscated point function. Now we argue that if the equality does not hold then passing the test (resp. distinguishing h) is equivalent to correctly guessing the global IT-MAC key of P_{B} (resp. P_{A}).

In particular, we can re-write $\tilde{V}_w^{\mathsf{B}}$ as (recall that we use "bar" to denote the values derived from garbled circuit labels)

$$\begin{aligned}\tilde{V}_w^{\mathsf{B}} &= (b_w + b'_w + \Lambda_w)\Delta_{\mathsf{B}} + \mathsf{M}_{\mathsf{B}}[b_w + b'_w + \bar{\Lambda}_w] + \mathsf{K}_{\mathsf{B}}[a_w + a'_w + \bar{\Lambda}'_w] \\ &= (b_w + b'_w + \Lambda_w)\Delta_{\mathsf{B}} + (b_w + b'_w + \bar{\Lambda}_w)\Delta_{\mathsf{A}} + \mathsf{K}_{\mathsf{A}}[b_w + b'_w + \bar{\Lambda}_w] \\ &\quad + (a_w + a'_w + \bar{\Lambda}'_w)\Delta_{\mathsf{B}} + \mathsf{M}_{\mathsf{A}}[a_w + a'_w + \bar{\Lambda}'_w] \\ &\quad + (a_w + a'_w + \Lambda'_w)\Delta_{\mathsf{A}} + (a_w + a'_w + \Lambda'_w)\Delta_{\mathsf{A}} \\ &= (a_w + b_w + \Lambda_w + a'_w + b'_w + \bar{\Lambda}'_w)\Delta_{\mathsf{B}} \\ &\quad + (a_w + b_w + \bar{\Lambda}_w + a'_w + b'_w + \Lambda'_w)\Delta_{\mathsf{A}} \\ &\quad + \underbrace{\mathsf{K}_{\mathsf{A}}[b_w + b'_w + \bar{\Lambda}_w] + \mathsf{M}_{\mathsf{A}}[a_w + a'_w + \bar{\Lambda}'_w] + (a_w + a'_w + \Lambda'_w)\Delta_{\mathsf{A}}}_{\tilde{V}_w^{\mathsf{A}}}\end{aligned}$$

The difference between two sides of the equation that we check is multiplied by Δ_{B} (resp. Δ_{A}) in the corrupted P_{A} (resp. P_{B}) case, and thus unless the adversary correctly guesses the respective keys or hash collision occurs, it cannot pass this check or distinguishing h from uniform randomness in the real world. □

Before proving the two lemmas, we recall the notion of (ρ, L)-independence. We call a matrix $\mathbf{M} \in \mathbb{F}_2^{n\times L}$ (ρ, L)-independent if any ρ rows of $\mathbf{M}$ are linearly independent. Thus if we sample $\boldsymbol{b}^* \leftarrow \mathbb{F}_2^L$ and set $\boldsymbol{b} := \mathbf{M} \cdot \boldsymbol{b}^*$ then $\boldsymbol{b}$ satisfies ρ-wise independence. This notion first appears in [DK99] and is applied in the authenticated garbling setting in [DILO22a]. We show that if we set $L = \lceil \rho \log(2en/\rho) + \log \rho/2 \rceil$ then a uniformly random $\mathbf{M}$ satisfies this property except with probability $2^{-\rho}$. We defer the proof to Appendix C.1. Therefore in the following we assume a uniformly random $n \times L$ matrix satisfies (n, ρ)-independence.

of Lemma 8. In the $\mathcal{F}_{\mathsf{cpre}}$-hybrid model we can extract the inputs as $\boldsymbol{x}, \boldsymbol{y}$ in step 6 and step 4. Thus in step 9a P_{B} would generate a garbled circuit with labels and masked inputs that correspond to $\mathcal{C}(\boldsymbol{x}, \boldsymbol{y})$.

From the security property of the semi-honest half-gates garbling scheme [ZRE15], except when the adversary acquires a label outside the execution path induced by the input labels sent by P_{B} in step 9a, the label-defined outputs of the second execution $\bar{\Lambda}'_w := (\mathsf{L}'_{w,0} \oplus \mathsf{L}'_{w,\Lambda'_w}) \cdot \Delta_{\mathsf{B}}^{-1}$ correspond to the masked wire values in $\mathcal{C}(\boldsymbol{x}, \boldsymbol{y})$. Finally, Lemma 10 ensures that P_{B} also holds the same set of real wire values except with probability $\frac{2}{2^\kappa}$. This implies that P_{B} either adheres to the execution of $\mathcal{C}(\boldsymbol{x}, \boldsymbol{y})$ or aborts except with probability $\frac{2+\mathsf{poly}(\kappa)}{2^\kappa}$. □

of Lemma 9. We consider the corrupted P_{A} case and the case for corrupted P_{B} can be derived analogously. Observe the equation $a_w + b_w + \Lambda_w = a'_w + b'_w + \bar{\Lambda}'_w$. The only value over which a malicious P_{A} has control is the public masked wire value Λ_w that the honest P_{B} evaluates. To negate this value P_{A} can flip the c_w bit or corrupt the ciphertexts $G_{w,0}, G_{w,1}$. The first method will flip Λ_w regardless of the input value so we only consider the second case.

For the KRRW scheme, we can explicitly denote the errors for each AND gate $(i, j, k, \wedge) \in \mathcal{C}_{\mathsf{and}}$ as $E_k := E_1 \oplus E_2 \oplus \Lambda_i E_3 \oplus \Lambda_j E_4$ where E_1, E_2, E_3, E_4 denotes the errors in $\mathsf{H}(k, \mathsf{L}_{i,\Lambda_i}), \mathsf{H}(k, \mathsf{L}_{j,\Lambda_j}), G_{k,0} \oplus \mathsf{M}_{\mathsf{B}}[b_j], G_{k,1} \oplus \mathsf{M}_{\mathsf{B}}[b_i] \oplus \mathsf{L}_{i,\Lambda_i}$ respectively. We can arrange these equations into the matrix format as $\boldsymbol{E} \cdot \boldsymbol{\Lambda}$ where $\boldsymbol{\Lambda}$ denote all the Λ_w values and the event Bad occurs if $\boldsymbol{E} \cdot \boldsymbol{\Lambda} = \mathbf{0}$. Let ℓ be the number of rows in $\boldsymbol{E}$. We have the following cases.

- $\ell \leq \rho$: In this case the (n, ρ)-independence of $\mathbf{M}$ ensures that the vector $\boldsymbol{\Lambda}$ is completely masked by $\boldsymbol{b}$ and the abort probability is independent of the evaluator's input.

- $\ell > \rho$: In this case the event Bad implies that at least ρ coordinates in $\boldsymbol{E} \cdot \boldsymbol{\Lambda}$ are zeros, which occurs except with probability $2^{-\rho}$.

Therefore, for different evaluator's inputs $\boldsymbol{y}$ and $\boldsymbol{y}'$, the probability of Bad differs with at most $2^{-\rho}$ probability. In other words, the KRRW scheme with compressed preprocessing is $2^{-\rho}$-selective failure resilient.

For the DILO-WRK scheme, abort happens when $(\mathsf{H}'(k, \mathsf{L}_{i,\Lambda_i}) \oplus \mathsf{H}'(k, \mathsf{L}_{j,\Lambda_j}) \oplus G'_{w,0} \oplus \Lambda_i G'_{w,1} \oplus \Lambda_j G'_{w,2}) \notin \{0, \Delta_{\mathsf{B}}\}$. Thus, following the above analysis we conclude that the scheme is $2 \cdot 2^{-\rho}$-selective failure resilient. $\square$

C.5 Proof of Theorem 2

Proof. We first prove the security against a malicious $\mathsf{P_A}$ and then prove the case for a malicious $\mathsf{P_B}$. We first describe the simulator and then argue its effectiveness through a series of hybrid experiments. In the following, we simulate the random oracle by recording all the query-answer pairs and answer the queries from $\mathcal{A}$ consistently.

Simulator $\mathcal{S}_{\mathsf{A}}$ for malicious $\mathsf{P_A}$

1. $\mathcal{S}_{\mathsf{A}}$ first simulates $\mathcal{F}_{\mathsf{cpre}}$ locally by randomly choosing the global keys, wire mask shares, wire tags, etc.
2. In step 4 $\mathcal{S}_{\mathsf{A}}$ extracts the input $\boldsymbol{x}$ of $\mathcal{A}$ by computing $x_w := \Lambda_w \oplus a_w$ for $w \in \mathcal{I}_A$ and sends $\boldsymbol{x}$ to $\mathcal{F}_{\mathsf{2PC}}$.
3. In step 6, $\mathcal{S}_{\mathsf{A}}$ uses all-zero inputs to handle message $d_w := \Lambda_w \oplus r_w$.
4. $\mathcal{S}_{\mathsf{A}}$ evaluates the garbled circuit and derives the result $\{\Lambda_w, \mathsf{L}_{w,\Lambda_w}\}$.
5. In step 8 $\mathcal{S}_{\mathsf{A}}$ garbles the circuit using previously generated randomness. Then it sends $\mathcal{GC}'_{\mathsf{B}}$ to $\mathcal{A}$.
6. In step 9a ($\mathsf{P_A}$ as the evaluator) $\mathcal{S}_{\mathsf{A}}$ follows the protocol specifications, opening the masks and sending garbled labels.
7. In step 9b $\mathcal{S}_{\mathsf{A}}$ sends labels according to the protocol instructions.
8. Evaluate the circuit with extracted $\boldsymbol{x}$ and 0 to get $\{z_w\} \leftarrow \mathcal{C}(\boldsymbol{x}, 0)$ for each wire index $w \in \mathcal{W}$. If for any $w \in \mathcal{W}$, $z_w \neq \Lambda_w \oplus a_w \oplus b_w$ then $\mathcal{S}_{\mathsf{A}}$ sends abort to the ideal functionality. Otherwise, it sends $\mathsf{continue}$ and finishes the simulation.

Now consider the series of hybrids where the first one is the real protocol execution and the last one is the above simulated execution.

Hybrid 1 This is the real execution where $\mathcal{S}_{\mathsf{A}}$ plays the role of an honest $\mathsf{P_B}$ using the actual input $\boldsymbol{y}$.

Hybrid 2 The same as $\mathbf{Hybrid_1}$ except we replace the consistency checking procedure with the checking procedure in the last step of the previous simulation strategy. Lemma 8 states that except with $\frac{2+\mathsf{poly}(\kappa)}{2^\kappa}$ probability, $\mathsf{P_B}$ either correctly evaluates the circuit $\mathcal{C}(\boldsymbol{x}, \boldsymbol{y})$ or aborts in $\mathbf{Hybrid_1}$, which is identical to the behavior of $\mathsf{P_B}$ in $\mathbf{Hybrid_2}$ by definition. The view of $\mathcal{A}$ is identical. Thus the two hybrids are $(\frac{2+\mathsf{poly}(\kappa)}{2^\kappa})$-indistinguishable. (This step eliminates the inconsistent function attack.)

Hybrid 3 Now we are in the ideal world model. If the consistency check passes then $\mathcal{S}_\mathsf{A}$ sends the extracted $\boldsymbol{x}$ to $\mathcal{F}_\mathsf{2PC}$, otherwise it sends abort. In $\mathbf{Hybrid_2}$, $\mathsf{P_B}$ either correctly evaluate $\mathcal{C}(\boldsymbol{x}, \boldsymbol{y})$ or aborts. And thus in $\mathbf{Hybrid_3}$ if $\mathcal{S}_\mathsf{A}$ does not abort, $\mathsf{P_B}$ will output the correct circuit evaluation result sent by the ideal functionality $\mathcal{F}_\mathsf{2PC}$, identical to the output of $\mathsf{P_B}$ of $\mathbf{Hybrid_2}$. The two hybrids are identically distributed.

Hybrid 4 The same with $\mathbf{Hybrid_3}$ except that we replace the actual input $\boldsymbol{y}$ with dummy input 0. Since $\boldsymbol{y}$ is fully masked by $\boldsymbol{r}$ this does not change the distribution. Also Lemma 9 ensures that the abort probabilities are statistically close in the two cases. Thus the two hybrids are $2^{-\rho}$-indistinguishable. This is the ideal-world execution.

Altogether, the ideal world and real world executions are $(\frac{1}{2^\rho} + \frac{2+\mathsf{poly}(\kappa)}{2^\kappa})$-indistinguishable in the corrupted $\mathsf{P_A}$ case.

Simulator $\mathcal{S}_\mathsf{B}$ for malicious $\mathsf{P_B}$

1. $\mathcal{S}_\mathsf{B}$ first simulates $\mathcal{F}_\mathsf{cpre}$ by randomly choosing the global keys, wire mask shares, wire tags, etc.
2. In step 3 $\mathcal{S}_\mathsf{B}$ garbles the circuit using previously generated randomness and sends $\mathcal{GC}_\mathsf{A}$ to $\mathcal{A}$.
3. In step 4 of the protocol ($\mathsf{P_A}$ as the garbler), $\mathcal{S}_\mathsf{B}$ uses $\boldsymbol{x} = 0$ to handle message $\{ \Lambda_w, \mathsf{L}_{w,\Lambda_w} \}$ for $w \in \mathcal{I}_\mathsf{A}$.
4. In step 6 $\mathcal{S}_\mathsf{B}$ extracts the input $\boldsymbol{y}$ of $\mathcal{A}$ by computing $y_w := d_w \oplus b_w \oplus r_w$ for $w \in \mathcal{I}_B$ and sends $\boldsymbol{y}$ to $\mathcal{F}^\mathcal{C}_\mathsf{2pc}$. $\mathcal{S}_\mathsf{B}$ receives from $\mathcal{F}^\mathcal{C}_\mathsf{2pc}$ the evaluation result $\tilde{z}_w$ for $w \in \mathcal{O}$.
5. In step 9 ($\mathsf{P_A}$ as the evaluator) $\mathcal{S}_\mathsf{B}$ receives the messages from $\mathsf{P_B}$ and evaluates the circuit to get $\{ \Lambda'_w, \mathsf{L}'_{w,\Lambda'_w} \}$.
6. $\mathcal{S}_\mathsf{B}$ simulates the checking procedure as follows. If for any wire $w \in \mathcal{W}$ the evaluation result in the previous step $\Lambda'_w \oplus a'_w \oplus b'_w \neq z_w$ where z_w is value of wire w in $\mathcal{C}(0, \boldsymbol{y})$, then $\mathcal{S}_\mathsf{B}$ sends $h \leftarrow \mathbb{F}_{2^\kappa}$ to $\mathsf{P_B}$. Otherwise it follows the protocol instructions and send h.
7. In the output step, $\mathcal{S}_\mathsf{B}$ computes and sends the augmented mask $\tilde{a}_w := a_w \oplus z_w \oplus \tilde{z}_w$ for $w \in \mathcal{O}$ to $\mathsf{P_B}$. Recall that z_w is the evaluation result of $\mathcal{C}(0, \boldsymbol{y})$, $\tilde{z}_w$ is the real output from $\mathcal{F}^\mathcal{C}_\mathsf{2pc}$.

Hybrid 1 This is the real execution in the hybrid model.

Hybrid 2 Same as $\mathbf{Hybrid_1}$ except we replace the checking procedure with the previous simulation strategy, i.e., replacing h with uniform randomness if any wire value does not match between the two executions. Lemma 10 ensures that if $\mathsf{P_A}$'s garbled circuit evaluation deviates from the actual circuit $\mathcal{C}$ then the hash value h sent by an honest $\mathsf{P_A}$ ($\mathbf{Hybrid_1}$) is indistinguishable from uniform randomness ($\mathbf{Hybrid_2}$). Thus the they are $(\frac{\mathsf{poly}(\kappa)}{2^\kappa})$-indistinguishable.

Hybrid 3 Same as $\mathbf{Hybrid_2}$ except we extract the input $\boldsymbol{y}$ of $\mathcal{A}$ and program the output wire mask as in the previous simulation strategy. Since h is either uniformly random or chosen exactly that the check passes, changing the output mask $\{ a_w \}_{w \in \mathcal{O}}$ will not be noticed by $\mathcal{A}$. The two hybrids are identically distributed.

Hybrid 4 Same as $\mathbf{Hybrid_3}$ except we replace the actual input $\boldsymbol{x}$ with the dummy input 0. The second part of Lemma 9 states that the probability of failing the consistency equation (which determines whether h is uniformly random) is changed at most $2^{-\rho}$. Moreover, $\boldsymbol{x}$ is fully masked by $\boldsymbol{a}$. Therefore the two hybrids are $2^{-\rho}$-indistinguishable. This corresponds to the ideal world execution.

Altogether, the ideal world and real world executions are $(\frac{1}{2^\rho} + \frac{\mathsf{poly}(\kappa)}{2^\kappa})$-indistinguishable in the corrupted $\mathsf{P_A}$ case. This implies that the protocol Π_{2PC} shown in Figure 7 and Figure 8 securely realizes $\mathcal{F}_{\mathsf{2PC}}$ against malicious adversary in the $\mathcal{F}_{\mathsf{cpre}}$-hybrid model. □

D Construction of Distributed Garbling Schemes

In this section, we recall the constructions of two distributed garbling schemes.

D.1 KRRW Distributed Garbling

We recall the distributed half-gates garbling scheme by Katz et al. [KRRW18]. Let $\mathsf{H} : \{0,1\}^* \to \{0,1\}^\kappa$ be a random oracle and $\Delta_\mathsf{A} \in \{0,1\}^\kappa$ be the global key held by $\mathsf{P_A}$.

- $\mathsf{Garble}(\mathcal{C})$:
  1. For each circuit input wire $w \in \mathcal{I}$, $\mathsf{P_A}$ samples $\mathsf{L}_{w,0} \leftarrow \mathbb{F}_2^\kappa$ and sets $\mathsf{L}_{w,1} := \mathsf{L}_{w,0} \oplus \Delta_\mathsf{A}$.
  2. Process the gates topologically. For each XOR gate $(i,j,k,\oplus)$, $\mathsf{P_A}$ sets $\mathsf{L}_{k,0} := \mathsf{L}_{i,0} \oplus \mathsf{L}_{j,0}$ and $\mathsf{L}_{k,1} := \mathsf{L}_{i,0} \oplus \Delta_\mathsf{A}$. For each AND gate $(i,j,k,\wedge)$, $\mathsf{P_A}$ computes
  $$\begin{aligned} G_{k,0}^{(\mathsf{A})} &:= \mathsf{H}(k, \mathsf{L}_{i,0}) \oplus \mathsf{H}(k, \mathsf{L}_{i,1}) \oplus a_j \Delta_\mathsf{A} \oplus \mathsf{K_A}[b_j] \ , \\ G_{k,1}^{(\mathsf{A})} &:= \mathsf{H}(k, \mathsf{L}_{j,0}) \oplus \mathsf{H}(k, \mathsf{L}_{j,1}) \oplus a_i \Delta_\mathsf{A} \oplus \mathsf{K_A}[b_i] \oplus \mathsf{L}_{i,0} \ , \\ \mathsf{L}_{k,0} &:= \mathsf{H}(k, \mathsf{L}_{i,0}) \oplus \mathsf{H}(k, \mathsf{L}_{j,0}) \oplus (a_k \oplus \hat{a}_k)\Delta_\mathsf{A} \oplus \mathsf{K_A}[b_k] \oplus \mathsf{K_A}[\hat{b}_k] \ . \end{aligned}$$
  We also define $\mathsf{L}_{k,1} := \mathsf{L}_{k,0} \oplus \Delta_\mathsf{A}$ and $c_k := \mathsf{ExtBit}(\mathsf{L}_{k,0})$ where ExtBit is a bit selection normally instantiated by lsb.
  3. $\mathsf{P_A}$ outputs $\{\mathsf{L}_{w,0}, \mathsf{L}_{w,1}\}_{w \in \mathcal{I}_\mathsf{A} \cup \mathcal{I}_\mathsf{B} \cup \mathcal{W} \cup \mathcal{O}}$ and $\mathcal{GC}_\mathsf{A} = \{G_{w,0}^{(\mathsf{A})}, G_{w,1}^{(\mathsf{A})}, c_w\}_{w \in \mathcal{W}}$.
  4. For each $(i,j,k,\wedge) \in \mathcal{W}$, $\mathsf{P_B}$ defines
  $$\begin{aligned} G_{k,0}^{(\mathsf{B})} &:= \mathsf{M_B}[b_j] \ , \\ G_{k,1}^{(\mathsf{B})} &:= \mathsf{M_B}[b_i] \ . \end{aligned}$$
  5. $\mathsf{P_B}$ outputs $\mathcal{GC}_\mathsf{B} = \{G_{w,0}^{(\mathsf{B})}, G_{w,1}^{(\mathsf{B})}\}_{w \in \mathcal{W}}$.

- $\mathsf{Eval}(\mathcal{GC}_\mathsf{A}, \mathcal{GC}_\mathsf{B}, \{(\Lambda_w, \mathsf{L}_{w,\Lambda_w})\}_{w \in \mathcal{I}_\mathsf{A} \cup \mathcal{I}_\mathsf{B}})$:
  1. $\mathsf{P_B}$ processes the gates topologically. For each XOR gate $(i,j,k,\oplus)$ define $\Lambda_k := \Lambda_i \oplus \Lambda_j$ and $\mathsf{L}_{k,\Lambda_k} := \mathsf{L}_{i,\Lambda_i} \oplus \mathsf{L}_{j,\Lambda_j}$.
  2. For each AND gate $(i,j,k,\wedge)$ compute the output label
  $$\begin{aligned} G_{w,0} &:= G_{w,0}^{(\mathsf{A})} \oplus G_{w,0}^{(\mathsf{B})} \\ G_{w,1} &:= G_{w,1}^{(\mathsf{A})} \oplus G_{w,1}^{(\mathsf{B})} \oplus \mathsf{L}_{i,\Lambda_w} \\ \mathsf{L}_{k,\Lambda_k} &:= \mathsf{H}(k, \mathsf{L}_{i,\Lambda_i}) \oplus \mathsf{H}(k, \mathsf{L}_{j,\Lambda_j}) \oplus \mathsf{M_B}[b_k] \oplus \mathsf{M_B}[\hat{b}_k] \\ &\quad \oplus \Lambda_i (G_{k,0} \oplus \mathsf{M_B}[b_j]) \oplus \Lambda_j (G_{k,1} \oplus \mathsf{M_B}[b_i] \oplus \mathsf{L}_{i,\Lambda_i}) \ , \end{aligned}$$
  and the public value $\Lambda_k := \mathsf{ExtBit}(\mathsf{L}_{k,\Lambda_k}) \oplus c_k$.
  3. Output $\{\,(\Lambda_w, \mathsf{L}_{w,\Lambda_w})\,\}_{w \in \mathcal{W} \cup \mathcal{O}}$.

D.2 WRK Distributed Garbling with Optimization

We recall the optimized WRK distributed garbling scheme by Dittmer et al. [DILO22a]. Let $\mathsf{H} : \{0,1\}^* \to \{0,1\}^\kappa$ and $\mathsf{H}' : \{0,1\}^* \to \{0,1\}^\rho$ be two random oracles, and $\Delta_\mathsf{A} \in \mathbb{F}_{2^\kappa}, \Delta_\mathsf{B} \in \mathbb{F}_{2^\rho}$ be the global keys held by P_A and P_B respectively.

- $\mathsf{Garble}(\mathcal{C})$:

  1. For each circuit-input wire $w \in \mathcal{I}$, P_A samples $\mathsf{L}_{w,0} \leftarrow \mathbb{F}_2^\kappa$ and sets $\mathsf{L}_{w,1} := \mathsf{L}_{w,0} \oplus \Delta_\mathsf{A}$.
  2. Process the gates topologically. For each XOR gate $(i, j, k, \oplus)$, P_A computes $\mathsf{L}_{k,0} := \mathsf{L}_{i,0} \oplus \mathsf{L}_{j,0}$ and $\mathsf{L}_{k,1} := \mathsf{L}_{i,0} \oplus \Delta_\mathsf{A}$. For each AND gate $(i, j, k, \wedge)$, P_A computes

$$\begin{aligned} G_{k,0}^{(\mathsf{A})} &:= \mathsf{H}(k, \mathsf{L}_{i,0}) \oplus \mathsf{H}(k, \mathsf{L}_{i,1}) \oplus a_j \Delta_\mathsf{A} \oplus \mathsf{K}_\mathsf{A}[b_j] \ , \\ G_{k,1}^{(\mathsf{A})} &:= \mathsf{H}(k, \mathsf{L}_{j,0}) \oplus \mathsf{H}(k, \mathsf{L}_{j,1}) \oplus a_i \Delta_\mathsf{A} \oplus \mathsf{K}_\mathsf{A}[b_i] \oplus \mathsf{L}_{i,0} \ , \\ \mathsf{L}_{k,0} &:= \mathsf{H}(k, \mathsf{L}_{i,0}) \oplus \mathsf{H}(k, \mathsf{L}_{j,0}) \oplus (a_k \oplus \hat{a}_k) \Delta_\mathsf{A} \oplus \mathsf{K}_\mathsf{A}[b_k] \oplus \mathsf{K}_\mathsf{A}[\hat{b}_k] \ , \\ G_{k,0}^{\prime(\mathsf{A})} &:= \mathsf{H}'(k, \mathsf{L}_{i,0}) \oplus \mathsf{H}'(k, \mathsf{L}_{i,1}) \oplus \mathsf{M}_\mathsf{A}[a_k] \oplus \mathsf{M}_\mathsf{A}[\hat{a}_k] \ , \\ G_{k,1}^{\prime(\mathsf{A})} &:= \mathsf{H}'(k, \mathsf{L}_{i,0}) \oplus \mathsf{H}'(k, \mathsf{L}_{i,1}) \oplus \mathsf{M}_\mathsf{A}[a_j] \ , \\ G_{k,2}^{\prime(\mathsf{A})} &:= \mathsf{H}'(k, \mathsf{L}_{j,0}) \oplus \mathsf{H}'(k, \mathsf{L}_{j,1}) \oplus \mathsf{M}_\mathsf{A}[a_i] \ . \end{aligned}$$

     We also define $\mathsf{L}_{k,1} := \mathsf{L}_{k,0} \oplus \Delta_\mathsf{A}$.
  3. P_A outputs $\{\mathsf{L}_{w,0}, \mathsf{L}_{w,1}\}_{w \in \mathcal{I}_\mathsf{A} \cup \mathcal{I}_\mathsf{B} \cup \mathcal{W} \cup \mathcal{O}}$ and

$$\mathcal{GC}_\mathsf{A} = \{G_{w,0}^{(\mathsf{A})}, G_{w,1}^{(\mathsf{A})}, G'_{k,0}, G'_{k,1}, G'_{k,2}\}_{w \in \mathcal{W}} \ .$$

  4. For each $(i, j, k, \wedge) \in \mathcal{W}$, P_B defines

$$\begin{aligned} G_{k,0}^{(\mathsf{B})} &:= \mathsf{M}_\mathsf{B}[b_j] \ , \\ G_{k,1}^{(\mathsf{B})} &:= \mathsf{M}_\mathsf{B}[b_i] \ , \\ G_{k,0}^{\prime,(\mathsf{B})} &:= \mathsf{K}_\mathsf{B}[a_k] \oplus \mathsf{K}_\mathsf{B}[\hat{a}_k] \ , \\ G_{k,1}^{\prime,(\mathsf{B})} &:= \mathsf{K}_\mathsf{B}[a_j] \ , \\ G_{k,2}^{\prime,(\mathsf{B})} &:= \mathsf{K}_\mathsf{B}[a_i] \ . \end{aligned}$$

  5. P_B outputs $\mathcal{GC}_\mathsf{B} = \{G_{w,0}^{(\mathsf{B})}, G_{w,1}^{(\mathsf{B})}, G_{k,0}^{\prime,(\mathsf{B})}, G_{k,1}^{\prime,(\mathsf{B})}, G_{k,2}^{\prime,(\mathsf{B})}\}_{w \in \mathcal{W}}$.

- $\mathsf{Eval}(\mathcal{GC}_\mathsf{A}, \mathcal{GC}_\mathsf{B}, \{(\Lambda_w, \mathsf{L}_{w,\Lambda_w})\}_{w \in \mathcal{I}_\mathsf{A} \cup \mathcal{I}_\mathsf{B}})$:

  1. P_B processes the gates topologically. For each XOR gate $(i, j, k, \oplus)$ define $\Lambda_k := \Lambda_i \oplus \Lambda_j$ and $\mathsf{L}_{k,\Lambda_k} := \mathsf{L}_{i,\Lambda_i} \oplus \mathsf{L}_{j,\Lambda_j}$.
  2. For each AND gate $(i, j, k, \wedge)$ P_B first recovers the garbled table as:

$$\begin{aligned} G_{w,0} &:= G_{w,0}^{(\mathsf{A})} \oplus G_{w,0}^{(\mathsf{B})} \\ G_{w,1} &:= G_{w,1}^{(\mathsf{A})} \oplus G_{w,1}^{(\mathsf{B})} \oplus \mathsf{L}_{i,\Lambda_w} \\ G'_{w,0} &:= G_{w,0}^{\prime(\mathsf{A})} \oplus G_{w,0}^{\prime(\mathsf{B})} \\ G'_{w,1} &:= G_{w,1}^{\prime(\mathsf{A})} \oplus G_{w,1}^{\prime(\mathsf{B})} \\ G'_{w,2} &:= G_{w,2}^{\prime(\mathsf{A})} \oplus G_{w,2}^{\prime(\mathsf{B})} \ , \end{aligned}$$

Then $\mathsf{P_B}$ computes the label and masked wire value of the AND gate output wire as follows. Notice that if the value $(\mathsf{H}'(k, \mathsf{L}_{i,\Lambda_i}) \oplus \mathsf{H}'(k, \mathsf{L}_{j,\Lambda_j}) \oplus G'_{w,0} \oplus \Lambda_i G'_{w,1} \oplus \Lambda_j G'_{w,2}) \cdot \Delta_{\mathsf{B}}^{-1} \notin \mathbb{F}_2$ then $\mathsf{P_B}$ aborts.

$$\begin{aligned} \mathsf{L}_{k,\Lambda_k} &:= \mathsf{H}(k, \mathsf{L}_{i,\Lambda_i}) \oplus \mathsf{H}(k, \mathsf{L}_{j,\Lambda_j}) \oplus \mathsf{M_B}[b_k] \oplus \mathsf{M_B}[\hat{b}_k] \\ &\quad \oplus \Lambda_i (G_{k,0} \oplus \mathsf{M_B}[b_j]) \oplus \Lambda_j (G_{k,1} \oplus \mathsf{M_B}[b_i] \oplus \mathsf{L}_{i,\Lambda_i}) \ , \\ \Lambda_k &:= b_k \oplus \hat{b}_k \oplus \Lambda_i b_j \oplus \Lambda_j b_i \oplus \Lambda_i \Lambda_j \\ &\quad \oplus (\mathsf{H}'(k, \mathsf{L}_{i,\Lambda_i}) \oplus \mathsf{H}'(k, \mathsf{L}_{j,\Lambda_j}) \oplus G'_{w,0} \oplus \Lambda_i G'_{w,1} \oplus \Lambda_j G'_{w,2}) \cdot \Delta_{\mathsf{B}}^{-1} \ . \end{aligned}$$

3. $\mathsf{P_B}$ outputs $\{\, \mathsf{L}_{w,\Lambda_w}, \Lambda_w \,\}_{w \in \mathcal{W} \cup \mathcal{O}}$.