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@northwestern.edu

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

Yu Yu
Shanghai Jiao Tong University
yuyu@yuyu.hk

December 20, 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 make significant progress in closing this gap by proposing two new actively secure constant-round 2PC protocols, one with one-way communication of $2\kappa+5$ bits per AND gate (for κ-bit computational security and any statistical security) and one with total communication of $2\kappa+\rho+5$ bits per AND gate (for ρ-bit statistical security). In particular, our first protocol essentially matches the one-way communication of semi-honest half-gates protocol. Our optimization is achieved by three 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.
3. In terms of total communication, we notice that the communication overhead of the consistency checking method by Dittmer et al. (Crypto 2022) can be optimized by adding one-round of interaction and utilizing the Free-XOR property. This reduces the online communication from $2\kappa+3\rho$ bits down to $2\kappa+\rho+1$ bits per AND gate. Combined with our first contribution, this yields total amortized communication of $2\kappa+\rho+5$ bits.

1 Introduction

Based on garbled circuits (GCs) [48], constant-round secure two-party computation (2PC) has obtained huge practical improvements in recent years in both communication [5, 33, 49, 39] and

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 $B = 3$.

2PC	Rounds		Communication per AND gate	
	Prep.	Online	one-way (bits)	two-way (bits)
Half-gates	1	2	2κ	2κ
HSS-PCG [28]	8	2	$8\kappa + 11$ (4.04×)	$16\kappa + 22$ (8.09×)
KRRW-PCG [32]	4	4	$5\kappa + 7$ (2.53×)	$8\kappa + 14$ (4.05×)
DILO [18]	7	2	$2\kappa + 8\rho + 1$ (2.25×)	$2\kappa + 8\rho + 5$ (2.27×)
DILOv2 [18]	3	4	$2\kappa + 2\rho + 2$ (1.32×)	$2\kappa + 4\rho + 2$ (1.63×)
This work, v.1	8	3	$2\kappa + 5$ (**1.02**×)	$4\kappa + 10$ (2.04×)
This work, v.2	8	2	$2\kappa + \rho + 3$ (1.17×)	$2\kappa + \rho + 4$ (**1.17**×)

computation [6, 26, 25]. However, compared to passively secure (a.k.a., semi-honest) 2PC protocols, their actively secure counterparts require significant overhead. Building upon the authenticated garbling framework [40, 41, 32, 46] and, more generally, working in the BMR family [5, 34, 35, 29, 27], the most recent work by Dittmer, Ishai, Lu and Ostrovsky [18] (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 [49]) to 1.5κ bits (three-halves [39]) 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 [24], and its concrete efficiency was studied in [1]. 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. [32] (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) [11, 10, 47, 15, 9], Katz et al. [32] can achieve $O(\kappa)$ bits per AND gate, while Dittmer et al. [18] can achieve $O(\rho)$ bits per AND gate. However, the latest advancement by Dittmer et al. [18] 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. We first propose an 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. Towards two-way communication, we optimize the consistency checking protocol in DILO (which is an optimized WRK checking protocol [40] with amortized 3ρ bits overhead) and reduce the consistency checking overhead down to ρ bits per AND gate as compared to the semi-honest half-gates protocol.

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. [18], 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 [31, 30] 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.

3. Towards total communication, we optimize the consistency checking procedure in WRK [40], resulting in a consistency checking protocol compatible with the compression technique [18] with amortized communication of ρ bits, which may be of independent interest. Recall that in WRK we use an additional garbled circuit to evaluate the MAC tag of the masked output wire value for each AND gate. First of all, we notice that in the secure computation scenario, we can settle for evaluating the *secret sharing* of the MAC tags, whose consistency can be verified using equality checking. This reduces ρ bits of communication. Moreover, notice that 1) we can perform batched MAC checking by checking the random linear combination of all AND gates, and 2) in Free-XOR compatible garbled circuits, the masked wire values of each wire is a public linear combination of previous AND gate outputs and circuit inputs. By changing the summation order, only one multiplication is needed per AND gate and input wire. Together with the distributed half-gate garbling scheme [32] and our preprocessing protocol, we get a circuit evaluation protocol with total amortized communication of $2\kappa + \rho + 5$ bits.

We provide a detailed comparison of our protocol with the literature in Table 1. Notice that in terms of amortized one-way communication we achieve constant additive overhead (5 bits per AND gate) as compared to the semi-honest half-gates protocol, while for amortized two-way communication the overhead is $\rho + 4$ bits. Under full-duplex networks (e.g., most wired communication) where communication in both directions can happen simultaneously, the one-way communication is more relevant and the first variant of our protocol effectively imposes no slow down compared to semi-honest half-gates; Nevertheless, even our two-way communication is minimal in the literature, for half-duplex networks (e.g., most wireless communication), we still cannot achieve the same desirable constant additive overhead in communication.

The DILOv2 protocol builds upon doubly authenticated multiplication triples [18]. Compared to DILO, the DILOv2 protocol is less efficient, as DILOv2 requires quasi-linear computational complexity. The reason is that the current instantiation of such doubly authenticated multiplication triples PCG based on Ring-LPN [12] is not on the same efficiency level as the random COT/VOLE PCGs. 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.

We also would like to stress that our protocol achieves adaptive security without relying on the random oracle model while all previous authenticated garbling protocols with adaptive security [40, 32, 29, 18] need the random oracle model.

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 P_A (resp., 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 [13, 23]. 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 Hash Functions

To instantiate our protocol without relying on the random oracle, we require different security properties for various hash functions that appear in our protocol. Here we recall their definitions. For the circular correlation robust under naturally derived keys (ccrnd) and tweakable correlation robust (tcr) hash functions we refer the work by Guo et al. [26] for the respective efficient instantiations in the ideal cipher model.

Tweakable correlation robustness (tcr) We first recall the tweakable correlation robustness property which is used in the reduction from correlated OT to string OT. The hash function has the syntax as $\mathsf{H} : \{0,1\}^\kappa \times \{0,1\}^\kappa \rightarrow \{0,1\}^\kappa$, where the first input is the hashed message while the second input is an index that ensures uniqueness of each hash function invocation. We recall the definitions as follows.

Definition 1. *Let $\mathsf{H} : \{0,1\}^{2\kappa} \rightarrow \{0,1\}^\kappa$ be a function, and let $\mathcal{R}$ be a distribution on $\{0,1\}^\kappa$. For $\Delta \in \{0,1\}^\kappa$, define $\mathcal{O}^{\mathsf{tcr}}_\Delta(x,i) = \mathsf{H}(x \oplus \Delta, i)$. For a distinguisher D, we define the following advantage*

$$\mathsf{Adv}^{\mathsf{tcr}}_{\mathsf{H},\mathcal{R}} := \left| \Pr_{\Delta \leftarrow \mathcal{R}}[\mathsf{D}^{\mathcal{O}^{\mathsf{tcr}}_\Delta(\cdot)}(1^\kappa) = 1] - \Pr_{f \leftarrow \mathcal{F}_{2\kappa,\kappa}}[\mathsf{D}^{f(\cdot)}(1^\kappa) = 1] \right| ,$$

where $\mathcal{F}_{2\kappa,\kappa}$ denotes the set of all functions mapping 2κ-bit inputs to κ-bit outputs. We call H (t,q,ρ,ϵ)-tweakable correlation robust if for all D running in time t and making at most q queries to the oracle and all $\mathcal{R}$ with min-entropy at least ρ, it holds that $\mathsf{Adv}^{\mathsf{tcr}}_{\mathsf{H},\mathcal{R}} \leq \epsilon$.

(Circular) correlation robustness under naturally derived keys (ccrnd). We require that the hash function H ensures the privacy property in the garbling scheme. During garbling, the tweaks of H are not adversarially chosen, but generated by the honest party in the garbling process. Tweaks generated in this way are referred to as "naturally derived" in the literature [49, 26].

Definition 2. *Let $\mathsf{H} : \{0,1\}^{2\kappa} \rightarrow \{0,1\}^\kappa$ be a function, and let $\mathcal{R}$ be a distribution on $\{0,1\}^\kappa$. For $\Delta \in \{0,1\}^\kappa$, define $\mathcal{O}^{\mathsf{ccrnd}}_\Delta(x,i,b) = \mathsf{H}(x \oplus \Delta, i) \oplus b \cdot \Delta$. A sequence of queries $\mathcal{Q} = (Q_1, ..., Q_q)$ is natural if each query Q_i with response x_i is one of the following:*

1. *$x_i \leftarrow \{0,1\}^\kappa$.*

2. *$x_i = x_{i_1} \oplus x_{i_2}$, where $i_i < i_2 < i$.*

3. *$x_i = \mathsf{H}(x_{i_1}, i)$, where $i_1 < i$.*

4. *$x_i = \mathcal{O}(x_{i_1}, i, b)$, where $i_1 < i$.*

Fix some natural sequence $\mathcal{Q}$ of length q. In the real-world experiment, denoted $\mathbf{Real}_{\mathsf{H},\mathcal{Q},\mathcal{R}}$, a key Δ is sampled from $\mathcal{R}$ and then the oracle $\mathcal{O}$ in step 4, above, is set to $\mathcal{O}^{\mathsf{crnd}}$ (resp. $\mathcal{O}^{\mathsf{ccrnd}}$). In the ideal-world experiment, denoted $\mathbf{Ideal}_{\mathsf{H},\mathcal{Q}}$, the oracle $\mathcal{O}$ is instead a function chosen uniformly from $\mathcal{F}_{2\kappa,\kappa}$ (resp. $\mathcal{F}_{2\kappa+1,\kappa}$) ($\mathcal{F}_{n,m}$ denotes the set of all functions mapping n-bit inputs to m-bit outputs). Either experiment defines a distribution (determined by executing the operations in $\mathcal{Q}$ in order) over values $x_1, ..., x_q$, which are output by the experiment.

For a distinguisher D, we define the following advantage and use superscript to differentiate the two cases.

$$\mathsf{Adv}_{\mathsf{H},\mathcal{Q},\mathcal{R}} := \left| \Pr_{\{x_i\} \leftarrow \mathbf{Real}_{\mathsf{H},\mathcal{Q},\mathcal{R}}}[\mathsf{D}(\{x_i\}) = 1] - \Pr_{\{x_i\} \leftarrow \mathbf{Ideal}_{\mathsf{H},\mathcal{Q}}}[\mathsf{D}(\{x_i\}) = 1] \right| ,$$

We call H (t,q,ρ,ϵ)-correlation robust (resp. circular correlation robust) for naturally derived keys if for all D running in time t and all Q of length q, and all $\mathcal{R}$ with min-entropy at least ρ, it holds that $\mathsf{Adv}^{\mathsf{crnd}}_{\mathsf{H},\mathcal{Q},\mathcal{R}} \leq \epsilon$ (resp. $\mathsf{Adv}^{\mathsf{ccrnd}}_{\mathsf{H},\mathcal{Q},\mathcal{R}} \leq \epsilon$).

Sum of Random Permutation. A common technique for checking the equality of two long strings is to perform hashing first and then check for equality in the digests. In some scenarios, the difference between the preimage of the honest party and the adversary is a secret unknown to the adversary. I.e., the adversary has to successfully guess the secret before passing the equality check.

Therefore, in the aforementioned setting, we can instantiate the hash function as the sum of applying random permutation π on each of its inputs. We note that this hash function was proposed previously by Damgård et al. [17].

Definition 3. *Let $\pi : \{0,1\}^\kappa \to \{0,1\}^\kappa$ be a random permutation. For each $\ell \in \mathbb{N}$, we define the hash function $\mathsf{H}^\pi : \{0,1\}^{\ell \cdot \kappa} \to \{0,1\}^\kappa$ as*

$$\mathsf{H}^\pi(x_1, ..., x_\ell) := \sum_{i=1}^{\ell} \pi(x_i) \enspace .$$

2.3 Information-Theoretic Message Authentication Codes

We use information-theoretic message authentication codes (IT-MACs) [7, 37] 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 [37, 17] 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. Notice that we hash the MAC tags to reduce communication [17].

- $\mathsf{CheckZero}([x_1], \ldots, [x_\ell])$: 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:
  1. $\mathsf{P_A}$ sends $h_\mathsf{A} := \mathsf{H}^\pi(\mathsf{M_A}[x_1], ..., \mathsf{M_A}[x_\ell])$ to $\mathsf{P_B}$, where H^π is defined in Definition 3.
  2. $\mathsf{P_B}$ computes $h_\mathsf{B} := \mathsf{H}^\pi(\mathsf{K_B}[x_1], ..., \mathsf{K_B}[x_\ell])$ and checks that $h_\mathsf{A} = h_\mathsf{B}$. If the check fails, $\mathsf{P_B}$ aborts.

Following previous works [17, 42], we have the following lemma, which we prove for completeness.

Functionality $\mathcal{F}_{\mathsf{bCOT}}^{L}$

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 $\mathsf{K_A}[\boldsymbol{u}]_{\Delta_i} \leftarrow \mathbb{F}_{2^\kappa}^{\ell}$; otherwise, receive $\mathsf{K_A}[\boldsymbol{u}]_{\Delta_i} \in \mathbb{F}_{2^\kappa}^{\ell}$ from the adversary.
- If $\mathsf{P_B}$ is honest, sample $\boldsymbol{u} \leftarrow \mathbb{F}_2^{\ell}$ and compute $\mathsf{M_B}[\boldsymbol{u}]_{\Delta_i} := \mathsf{K_A}[\boldsymbol{u}]_{\Delta_i} + \boldsymbol{u} \cdot \Delta_i \in \mathbb{F}_{2^\kappa}^{\ell}$ for $i \in [1, L]$. Otherwise, receive $\boldsymbol{u} \in \mathbb{F}_2^{\ell}$ and $\mathsf{M_B}[\boldsymbol{u}]_{\Delta_i} \in \mathbb{F}_{2^\kappa}^{\ell}$ for $i \in [1, L]$ from the adversary, and recomputes $\mathsf{K_A}[\boldsymbol{u}]_{\Delta_i} := \mathsf{M_B}[\boldsymbol{u}]_{\Delta_i} + \boldsymbol{u} \cdot \Delta_i \in \mathbb{F}_{2^\kappa}^{\ell}$ 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.

Lemma 1. *If $\Delta \in \mathbb{F}_{2^\kappa}$ is sampled uniformly at random and π is a random permutation, then the probability that there exists some $i \in [1, \ell]$ such that $x_i \neq 0$ and $\mathsf{P_B}$ accepts in the* CheckZero *procedure is bounded by $\frac{\tau+1}{2^\kappa}$, where τ upper bounds $\mathsf{P_A}$'s running time.*

Proof. Suppose $x_i \neq 0$ for some $i \in [\ell]$. Then the value $\mathsf{K_B}[x_i] = \mathsf{M_A}[x_i] + x_i\Delta$ appear uniformly random to $\mathsf{P_A}$. $\mathsf{P_A}$ learns at most τ input-output pairs of π by querying π and π^{-1} and the probability that $\mathsf{K_B}[x_i]$ falls into this set is at most $\frac{t}{2^\kappa}$.

Conditioned on this event not happening, $\pi(\mathsf{K_B}[x_i])$ is indistinguishable from uniform randomness for $\mathsf{P_A}$ and so is h_B, which implies that the probability that $h_\mathsf{A} = h_\mathsf{B}$ is at most $2^{-\kappa}$. Applying the union bound we get the desired soundness bound. □

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 $\frac{1}{|\mathcal{R}|} + 2^{-\kappa}$. Based on the CheckZero procedure, we define the following batch-opening procedure (denoted by Open):

- $\mathsf{Open}([x_1], \ldots, [x_\ell])$: On input authenticated values $[x_1], \ldots, [x_\ell]$ defined over field $\mathbb{F}_{2^\kappa}$, $\mathsf{P_A}$ opens these values as follows:
  1. $\mathsf{P_A}$ sends $(x_1, \ldots, x_\ell)$ 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 $\mathsf{CheckZero}([y_1], \ldots, [y_\ell])$ with $\mathsf{P_B}$. If $\mathsf{P_B}$ does not abort, it outputs $(x_1, \ldots, x_\ell)$.

2.4 Correlated Oblivious Transfer

Our 2PC protocol will adopt the standard functionality [10, 47] 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) [8], the recent protocols [10, 47, 15, 9] 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) [18], which allows to generate authenticated bits with the same choice bits and different global keys. Functionality $\mathcal{F}_{\mathsf{bCOT}}^{L}$ shown in Figure 1 is the same as the standard COT functionality, except that L vectors (rather than a single vector) of authenticated bits $[\boldsymbol{u}]_{\mathsf{B},\Delta_1}, \ldots, [\boldsymbol{u}]_{\mathsf{B},\Delta_L}$ are generated. Here the vector of choice bits

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.

$\boldsymbol{u}$ 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 $\mathcal{F}^L_{\mathsf{bCOT}}$ 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 [4].

- $([\boldsymbol{x}]_{\mathsf{B},\Delta_1}, \ldots, [\boldsymbol{x}]_{\mathsf{B},\Delta_L}) \leftarrow \mathsf{Fix}(sid, \boldsymbol{x})$: On input a session identifier sid of $\mathcal{F}_{\mathsf{bCOT}}$, and a vector $\boldsymbol{x} \in \mathbb{F}_2^\ell$ 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 $\boldsymbol{r} \in \mathbb{F}_2^\ell$ 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 $L = 1$.

2.5 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 [42, 20, 4, 45, 43, 2, 19, 44, 3]. 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 [45]).

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 [18].

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

Recently, Dittmer, Ishai, Lu and Ostrovsky [18] 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 [40], 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. [18] 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 2ρ 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. [18] 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 [32]. 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 [18], 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 $\mathsf{D}_\mathsf{A}[x], \mathsf{D}_\mathsf{B}[x], \Delta_\mathsf{A}, \Delta_\mathsf{B}$ 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 P_A holds $a \in \{0,1\}$ and 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 P_A knows x. The DILO protocol [18] 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 P_A and 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 P_A gets α and 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 $[a]_\beta = (\mathsf{K}_\mathsf{B}[a]_\beta, \mathsf{M}_\mathsf{A}[a]_\beta, a)$, P_A and P_B can *locally* compute a dual-key authenticated bit $\langle a \rangle$ by letting P_A compute $\mathsf{D}_\mathsf{A}[x] := \mathsf{M}_\mathsf{A}[a]_\beta + a \cdot \alpha \in \mathbb{F}_{2^\kappa}$ and P_B set $\mathsf{D}_\mathsf{B}[x] := \mathsf{K}_\mathsf{B}[a]_\beta \in \mathbb{F}_{2^\kappa}$. We have that $\mathsf{D}_\mathsf{A}[x] + \mathsf{D}_\mathsf{B}[x] = (\mathsf{M}_\mathsf{A}[a]_\beta + \mathsf{K}_\mathsf{B}[a]_\beta) + a \cdot \alpha = a \cdot (\alpha + \beta) = a \cdot b \cdot \Delta_\mathsf{A} \cdot \Delta_\mathsf{B} \in \mathbb{F}_{2^\kappa}$. 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 $x = \mathsf{lsb}(\mathsf{D}_\mathsf{A}[x]) \oplus \mathsf{lsb}(\mathsf{D}_\mathsf{B}[x])$ from a dual-key authenticated bit $\langle x \rangle$. To do this, we let P_A sample $\Delta_\mathsf{A} \leftarrow \{0,1\}^\kappa$ such that $\mathsf{lsb}(\Delta_\mathsf{A}) = 1$. Then, we let P_B sample $\Delta_\mathsf{B} \leftarrow \{0,1\}^\kappa$, and make P_A and 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 $\alpha_0 \oplus \beta_0 = \Delta_\mathsf{A}\Delta_\mathsf{B}$. P_A and 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 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 [32] 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 P_B may flip Δ_B depending on if $\mathsf{lsb}(\alpha_0) \oplus \mathsf{lsb}(\beta_0) = 0$. Thus, we let 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 P_A or 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 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 P_A sends $\mathsf{lsb}(\mathsf{K}_\mathsf{A}[r_i])$ for $i \in [1, \rho]$ to P_B who checks that $\mathsf{lsb}(\mathsf{K}_\mathsf{A}[r_i]) \oplus \mathsf{lsb}(\mathsf{M}_\mathsf{B}[r_i]) = r_i$ for all $i \in [1, \rho]$. A malicious 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 P_B check $\mathsf{lsb}(\Delta_\mathsf{A}\Delta_\mathsf{B}) = 1$ by interacting with P_A. We make P_A and 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 $\langle s_1 \rangle, \dots, \langle s_\rho \rangle$, $\mathsf{P_A}$ and $\mathsf{P_B}$ can call $\mathcal{F}_{\mathsf{COT}}$ to generate random authenticated bits $[s_1]_{\Delta_\mathsf{A}}, \dots, [s_\rho]_{\Delta_\mathsf{A}}$, and then run the Fix procedure w.r.t. Δ_A on input $(s_1\Delta_\mathsf{B}, \dots, s_\rho\Delta_\mathsf{B})$ to generate $[s_1\Delta_\mathsf{B}]_{\Delta_\mathsf{A}}, \dots, [s_\rho\Delta_\mathsf{B}]_{\Delta_\mathsf{A}}$ that are equivalent to $\langle s_1 \rangle, \dots, \langle s_\rho \rangle$. Then, the correctness of the input $(s_1\Delta_\mathsf{B}, \dots, s_\rho\Delta_\mathsf{B})$ 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 [18], 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 $\langle b_1^* \rangle, \dots, \langle b_L^* \rangle$ 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 $\beta_1, \dots, \beta_L, \Delta_\mathsf{B}$, and then call $\mathcal{F}_{\mathsf{bCOT}}^{L+1}$ to generate $[\boldsymbol{a}]_{\beta_1}, \dots, [\boldsymbol{a}]_{\beta_L}$ and $[\boldsymbol{a}]_{\Delta_\mathsf{B}}$. For each tuple $([a_i]_{\beta_1}, \dots, [a_i]_{\beta_L})$, 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 $\mathsf{P_B}$ *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 [20, 18] 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. [18], 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 $[y]_{\mathsf{B}} := \sum \chi^k \cdot [\tilde{b}_k]_{\mathsf{B}} + [r]_{\mathsf{B}}$. Then, $\mathsf{P_B}$ can open $[y]_{\mathsf{B}}$ that allows $\mathsf{P_A}$ to get y in an authenticated

[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.

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. [32] is no longer applied. The circuit authentication approach in [32] requires the evaluator to reveal all masked wire values, which is prohibitive for the compression technique. Therefore, based on the technique [40], Dittmer et al. [18] 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 [32].

In duplex networks, communication cost is often measured by one-way communication. This allows us to adopt the idea of dual execution [36] to perform the authentication of circuit evaluation. In the original dual execution [36], the semi-honest Yao-2PC protocol [48] 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 [49]. 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 [32], 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., [32]), for any authenticated bit $[y]_{\mathsf{B}}$, $\mathsf{P_A}$ and $\mathsf{P_B}$ have an additive sharing of $y \cdot \Delta_{\mathsf{A}} = \mathsf{K}_{\mathsf{A}}[y] \oplus \mathsf{M}_{\mathsf{B}}[y]$. 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, P_{A} and P_{B} can respectively compute

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

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

Optimizations for processing inputs. Dittmer et al. [18] consider that the wire masks (i.e., $\boldsymbol{b}_{\mathcal{I}}$) on all wires in $\mathcal{I}_{\mathsf{B}}$ held by evaluator P_{B} is uniformly random and authenticated AND triples associated with $\boldsymbol{b}_{\mathcal{I}}$ are generated using the previous approach (e.g., [32]). 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 P_{B}. In particular, instead of making P_{B} send masked value $\Lambda_w := y_w \oplus b_w$ for each $w \in \mathcal{I}_{\mathsf{B}}$ to P_{A} where y_w is the input bit, we use an OT protocol to transmit L_{w,Λ_w} to 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., P_{A} as an evaluator and 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.

3.4 Optimization Towards Minimal Total Communication

We also make effort to minimize the total communication of two party computation protocols by optimizing the DILO-WRK protocol [18], achieving the two-way communication of $2\kappa + \rho + 5$ bits per AND gate. We explain the intuition behind our optimization as follows.

We first explain the consistency checking protocol in DILO-WRK, which substitutes the state-of-the-art checking technique [32] in the DILO protocol due to aforementioned security issues. In the original WRK protocol [40], the garbler essentially utilizes another garbled circuit AuthGC to compute the MAC tag of Λ_k for each AND gate $(i, j, k, \wedge)$, which ensures that the correct Λ_k is acquired by the evaluator. The authors of DILO observe that using the half-gates technique the original 4ρ bits communication of WRK (which corresponds to an un-optimized garbled circuit) can be optimized to 3ρ bits, resulting in a scheme we dub "DILO-WRK".

In essence, the goal of the WRK IT-MAC checking is to let the evaluator compute $(\lambda_k \oplus (\Lambda_i \oplus \lambda_i) \cdot (\Lambda_j \oplus \lambda_j)) \cdot \Delta_{\mathsf{B}}$ using the garbled circuit labels and preprocessing information, and compare it against the $\Lambda_k \cdot \Delta_{\mathsf{B}}$ that is locally computable. Since the former term is unalterable by the security of IT-MAC and is correct by definition, consistency follows when the equality check passes. Therefore, consistency checking reduces to an efficient comparison operation.

Our first insight is that unlike regular IT-MAC opening where the entire MAC tag has to be completely conveyed, in the secure computation setting we may settle for evaluating the additive share of $(\lambda_k \oplus (\Lambda_i \oplus \lambda_i) \cdot (\Lambda_j \oplus \lambda_j)) \cdot \Delta_{\mathsf{B}}$ since it is only used for subsequent equality checking. Therefore, we focus on the cross-terms $\Lambda_i \cdot \mathsf{M}[a_j]$ and $\Lambda_j \cdot \mathsf{M}[a_i]$ in the expanded equation below.

$$\begin{aligned}(\lambda_k \oplus (\Lambda_i \oplus \lambda_i) \cdot (\Lambda_j \oplus \lambda_j)) \cdot \Delta_{\mathsf{B}} &= \lambda_k \cdot \Delta_{\mathsf{B}} \oplus \Lambda_i \cdot \Lambda_j \cdot \Delta_{\mathsf{B}} \oplus \Lambda_i \cdot \lambda_j \cdot \Delta_{\mathsf{B}} \oplus \Lambda_j \cdot \lambda_i \cdot \Delta_{\mathsf{B}} \oplus \lambda_i \cdot \lambda_j \cdot \Delta_{\mathsf{B}} \\ &= \lambda_k \cdot \Delta_{\mathsf{B}} \oplus \Lambda_i \cdot \Lambda_j \cdot \Delta_{\mathsf{B}} \oplus \lambda_i \cdot \lambda_j \cdot \Delta_{\mathsf{B}} \oplus \Lambda_i \cdot b_j \cdot \Delta_{\mathsf{B}} \oplus \Lambda_i \cdot \mathsf{K}[a_j] \oplus \Lambda_j \cdot b_i \cdot \Delta_{\mathsf{B}} \oplus \Lambda_j \cdot \mathsf{K}[a_i] \\ &\oplus \Lambda_i \cdot \mathsf{M}[a_j] \oplus \Lambda_j \cdot \mathsf{M}[a_i] \ .\end{aligned}$$

In the DILO-WRK scheme, the two cross-terms are computed as follows. The evaluator sends two ciphertexts $G'_{k,1} := \mathsf{H}(\mathsf{L}_{i,0}) \oplus \mathsf{H}(\mathsf{L}_{i,1}) \oplus \mathsf{M}[a_j]$ and $G'_{k,2} := \mathsf{H}(\mathsf{L}_{j,0}) \oplus \mathsf{H}(\mathsf{L}_{j,1}) \oplus \mathsf{M}[a_i]$ to the evaluator and defines $C_1 := \mathsf{H}(\mathsf{L}_{i,0})$, $C_2 := \mathsf{H}(\mathsf{L}_{j,0})$. The evaluator computes $D_1 := \mathsf{H}(\mathsf{L}_{i,\Lambda_i}) \oplus G'_{k,1} = \Lambda_i \cdot \mathsf{M}[a_j] \oplus C_1$ and $D_2 := \mathsf{H}(\mathsf{L}_{j,\Lambda_j}) \oplus G'_{k,2} = \Lambda_j \cdot \mathsf{M}[a_i] \oplus C_2$, which constitute the additive sharing of the cross-terms with C_1 and C_2. In DILO-WRK the garbler sends an additional message that conveys the XOR of its local shares. Using our observation, this message can be omitted, leading to 2ρ bits of communication per AND gate.

In secure computation, the task of checking the aforementioned equality on every AND gate can be effectively reduced to only one equality check via random linear combination. The secure computation task therefore reduces to evaluating $\sum_{(i,j,k,\wedge) \in \mathcal{C}_{\mathsf{and}}} \chi^k \cdot (\Lambda_i \cdot \mathsf{M}[a_j] + \Lambda_j \cdot \mathsf{M}[a_i])$. Our second observation is that in free-XOR compatible garbling schemes every masked wire value Λ_w is a public linear combination of the masked values of previous AND gate output wires and input wires. More generally, we define the public binary vector $\boldsymbol{c}^w$ for every wire $w \in \mathcal{W}$ such that $\Lambda_w = \sum_{k \in \mathcal{W} \cup \mathcal{I}} c_k^w \cdot \Lambda_k$. Using this notation, we can expand the target expression as

$$\sum_{(i,j,k,\wedge) \in \mathcal{C}_{\mathsf{and}}} \chi^k \cdot \left((\sum_{k' \in \mathcal{W} \cup \mathcal{I}} c_{k'}^i \cdot \Lambda_{k'}) \cdot \mathsf{M}[a_j] + (\sum_{k' \in \mathcal{W} \cup \mathcal{I}} c_{k'}^j \cdot \Lambda_{k'}) \cdot \mathsf{M}[a_i] \right) \ .$$

By exchanging the summation order, the expression is transformed into a linear operation on all the masked Λ_w values, where the coefficients can be computed by the garbler. (Notice that the indices are renamed.)

$$\sum_{k \in \mathcal{W} \cup \mathcal{I}} \Lambda_k \cdot \sum_{(i',j',k',\wedge) \in \mathcal{C}_{\mathsf{and}}} \chi^{k'} \cdot (c_k^{i'} \cdot \mathsf{M}[a_{j'}] + c_k^{j'} \cdot \mathsf{M}[a_{i'}]) \ .$$

Using the half-gates technique, we can evaluate the target expression by sending $G'_k := \mathsf{H}(\mathsf{L}_{k,0}) \oplus \mathsf{H}(\mathsf{L}_{k,1}) \oplus \sum_{(i',j',k',\wedge) \in \mathcal{C}_{\mathsf{and}}} \chi^{k'} \cdot (c_k^{i'} \cdot \mathsf{M}[a_{j'}] + c_k^{j'} \cdot \mathsf{M}[a_{i'}])$ for each index $k \in \mathcal{W} \cup \mathcal{I}$, which has amortized communication cost of ρ bits per AND gate. We note that this checking method is applicable to all distributed garbling schemes that support Free-XOR. Therefore, we believe that this subprotocol Π_{GCCheck} (Figure 10) is of independent interest.

3.5 Adaptive Security without the Random Oracle

Compared with the conference version [16] we removed the reliance of the random oracle model in our security proof. In particular, we prove the adaptive security of the garbled circuit protocol without using the random oracle, unlike all previous authenticated garbling protocols with adaptive security [40, 32, 29, 18]. Recall that with adaptive security the garbler can send the garbled circuit ciphertexts in the offline phase before the inputs are specified, thus saving online communication.

The change is made possible by utilizing the garbler's wire masks a_w for each of the evaluator's input wire $w \in \mathcal{I}_\mathsf{B}$. Since the mask a_w is uniformly random in the view of the evaluator P_B, so is the masked value Λ_w for $w \in \mathcal{I}_\mathsf{B}$. Therefore, in the online phase we can let the evaluator first specify its input $\tilde{\Lambda}_w := y_w \oplus b_w$ and then let the garbler open its wire mask a_w, which allows the evaluator to learn $\Lambda_w := a_w \oplus \tilde{\Lambda}_w$[3].

In the security proof, the simulator can generate the garbled circuit using uniformly random Λ_w values in the offline phase. Then in the online phase, once it receives the evaluator's $\tilde{\Lambda}_w$ message, it can simulate the opening procedure accordingly so that the evaluator learns exactly the previously generated Λ_w values. Intuitively, can be done since the simulator knows the evaluator's authentication key corresponding to a_w and can flip a_w if necessary. Therefore, we can still argue adaptive security even if the garbled circuit is generated without the random oracle. We note that this technique is applicable to both of our online protocol in Section 5 and Section 6.

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

On the length of the input masks. In $\mathcal{F}_\mathsf{cpre}$ the garbler P_A has wire masks for each wire $w \in \mathcal{I}_\mathsf{A} \cup \mathcal{I}_\mathsf{B} \cup \mathcal{W}$. Nevertheless, the masks for $\mathcal{I}_\mathsf{B}$ is only required for the security proof without the random oracle model since in the security proof the garbled circuit has to be generated according to the active path and garbler's input mask for $w \in \mathcal{I}_\mathsf{B}$ allows randomized masked wire value in the view of P_B. Therefore, if we can settle for the random oracle model or the garbled circuit ciphertexts are sent during the online phase after the input has been specified, then the wire masks of P_A can be shortened to $|\mathcal{I}_\mathsf{A}| + |\mathcal{W}|$ bits.

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 4. *We use the notation $\langle x \rangle := (\mathsf{D}_\mathsf{A}[x], \mathsf{D}_\mathsf{B}[x], x)$ to denote the dual-key authenticated value $x \in \mathbb{F}_{2^\kappa}$, where $\mathsf{P}_\mathsf{A}, \mathsf{P}_\mathsf{B}$ holds $\mathsf{D}_\mathsf{A}[x], \mathsf{D}_\mathsf{B}[x]$ subject to $\mathsf{D}_\mathsf{A}[x] + \mathsf{D}_\mathsf{B}[x] = x\Delta_\mathsf{A}\Delta_\mathsf{B}$ and $\Delta_\mathsf{A}, \Delta_\mathsf{B}$ are the IT-MAC keys of $\mathsf{P}_\mathsf{A}, \mathsf{P}_\mathsf{B}$ respectively.*

[3] In more detail, due to the compression of P_B's wire masks, we cannot send $\tilde{\Lambda}_w$ directly but rather feed it to an oblivious transfer protocol.

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., $m := |\mathcal{W}| + |\mathcal{I}|$) 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 $\mathsf{P_B}$), and $L = \lceil 2\rho \log(\frac{en}{\sqrt{2\rho}}) + \frac{\log 2\rho}{2} \rceil$ be a compression parameter where $t = |\mathcal{W}|$. 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 $[b_w]$.
- 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 $w \in \mathcal{I}_{\mathsf{A}} \cup \mathcal{I}_{\mathsf{B}} \cup \mathcal{W}$, 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 $\mathcal{S}$. Let $\boldsymbol{b} = \mathbf{M} \cdot \boldsymbol{b}^*$. Run $\mathsf{Auth_B}(\boldsymbol{b}, n)$ to generate $[\boldsymbol{b}]$. 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.

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 $[x]_{\mathsf{A},\beta} = (\mathsf{K}_\mathsf{B}[x]_\beta, \mathsf{M}_\mathsf{A}[x]_\beta, x)$. 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 $\langle y \rangle$, P_A and P_B *locally* compute the following steps:
  - P_A outputs $\mathsf{D}_\mathsf{A}[xy] := \alpha \cdot x + \mathsf{M}_\mathsf{A}[x]_\beta \in \mathbb{F}_{2^\kappa}$.
  - P_B outputs $\mathsf{D}_\mathsf{B}[xy] := \mathsf{K}_\mathsf{B}[x]_\beta$.

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* $\langle x_1 \rangle := (\mathsf{D}_\mathsf{A}[x_1], \mathsf{D}_\mathsf{B}[x_1])$ *and* $\langle x_2 \rangle := (\mathsf{D}_\mathsf{A}[x_2], \mathsf{D}_\mathsf{B}[x_2])$*,* $\mathsf{P}_\mathsf{A}, \mathsf{P}_\mathsf{B}$ *can locally compute* $\langle x_1 + x_2 \rangle := (\mathsf{D}_\mathsf{A}[x_1] + \mathsf{D}_\mathsf{A}[x_2], \mathsf{D}_\mathsf{B}[x_1] + \mathsf{D}_\mathsf{B}[x_2])$*.*

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. P_A computes $h_\mathsf{A} := \mathsf{H}^\pi(\mathsf{D}_\mathsf{A}[x_1], \ldots, \mathsf{D}_\mathsf{A}[x_\ell])$, and P_B sets $h_\mathsf{B} := \mathsf{H}^\pi(\mathsf{D}_\mathsf{B}[x_1], \ldots, \mathsf{D}_\mathsf{B}[x_\ell])$, where H^π is defined in Definition 3.
  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 $\langle 1 \rangle = (\mathsf{D}_\mathsf{A}[1], \mathsf{D}_\mathsf{B}[1])$ 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 and* π *is a random permutation. Then the probability that there exists some* $i \in [1, \ell]$ *such that* $x_i \neq 0$ *and* P_A *or* P_B *accepts in the* $\mathsf{CheckZero2}$ *procedure is bounded by* $\frac{\tau+1}{2^\kappa}$ *where* τ *upper bounds the running time of* P_A *and* P_B*.*

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. *Let* π *be a random permutation inside the* $\mathsf{CheckZero2}$ *command, then 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* $\mathsf{lsb}(\Delta_\mathsf{A} \cdot \Delta_\mathsf{B}) = 1$ *in the honest case.*

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 $m^0_\mathsf{A} := (\mathsf{lsb}(\mathsf{K_A}[u_1]), \ldots, \mathsf{lsb}(\mathsf{K_A}[u_\rho]))$ to $\mathsf{P_B}$, who checks that $m^0_\mathsf{A} = (\mathsf{lsb}(\mathsf{M_B}[u_1]) \oplus u_1, \ldots, \mathsf{lsb}(\mathsf{M_B}[u_\rho]) \oplus u_\rho)$ 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 $m^1_\mathsf{A} = \mathsf{lsb}(\mathsf{K_A}[\tilde{\Delta}_\mathsf{B}])$ to $\mathsf{P_B}$, and $\mathsf{P_B}$ sends $m^1_\mathsf{B} = \mathsf{lsb}(\mathsf{M_B}[\tilde{\Delta}_\mathsf{B}])$ to $\mathsf{P_A}$ in parallel. If $m^1_\mathsf{A} \oplus m^1_\mathsf{B} = 0$, 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 $m^0_\mathsf{B} := (\mathsf{msb}(\mathsf{K_B}[v_1]), \ldots, \mathsf{msb}(\mathsf{K_B}[v_\rho]))$ to $\mathsf{P_A}$, who checks that $m^0_\mathsf{B} = (\mathsf{msb}(\mathsf{M_A}[v_1]) \oplus v_1, \ldots, \mathsf{msb}(\mathsf{M_A}[v_\rho]) \oplus v_\rho)$ 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 $m^2_\mathsf{A} := (\mathsf{lsb}(\mathsf{D_A}[x_1]), \ldots, \mathsf{lsb}(\mathsf{D_A}[x_\rho]))$ to $\mathsf{P_B}$, who checks that $m^2_\mathsf{A} = (\mathsf{lsb}(\mathsf{D_B}[x_1]) \oplus x_1, \ldots, \mathsf{lsb}(\mathsf{D_B}[x_\rho]) \oplus x_\rho)$.
   (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 $m^2_\mathsf{B} := (\mathsf{lsb}(\mathsf{D_B}[y_1]), \ldots, \mathsf{lsb}(\mathsf{D_B}[y_\rho]))$ to $\mathsf{P_A}$, who checks that $m^2_\mathsf{B} = (\mathsf{lsb}(\mathsf{D_A}[y_1]) \oplus y_1, \ldots, \mathsf{lsb}(\mathsf{D_A}[y_\rho]) \oplus y_\rho)$.
   (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.

- *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* $\mathsf{lsb}(\Delta_\mathsf{A} \cdot \Delta_\mathsf{B}) \neq 1$ *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* $\mathsf{lsb}(\Delta_\mathsf{A} \cdot \Delta_\mathsf{B}) \neq 1$ *then* $\mathsf{P_B}$ *aborts except with probability* $\frac{\tau+1}{2^\kappa} + 2^{-\rho}$, *where* τ *upper bounds the running time of* $\mathsf{P_B}$.

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 $\frac{\tau+1}{2^\kappa}$ probability. By union bound the soundness of the second check is $\frac{\tau+1}{2^\kappa} + 2^{-\rho}$. □

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 [20].

- $[y]_{\mathsf{B},\Delta_\mathsf{A}^{-1}} \leftarrow \mathsf{Invert}([x]_\mathsf{B})$: 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 [18], which we summarize in the $\mathsf{EQCheck}$ macro. We recall its soundness in Lemma 5 and prove it in Appendix D.2.In the following, we assume that $\mathcal{F}_{\mathsf{COT}}$ has been initialized with (sid, Δ_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 $[y'_i]_{\Delta'_\mathsf{A}} = (k'_i, m'_i, y'_i)$. $\mathsf{P_A}$ 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 $h_\mathsf{B} := \mathsf{H}^\pi(W_1, ..., W_\ell)$ to $\mathsf{P_A}$, who computes $h_\mathsf{A} := \mathsf{H}^\pi(V_1, ..., V_\ell)$ and verifies that $h_\mathsf{A} = h_\mathsf{B}$. If the check fails, $\mathsf{P_A}$ aborts. Here H^π is defined in Definition 3.

Lemma 5. *Let π be a random permutation. 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* EQCheck *procedure is bounded by $\frac{\tau+2}{2^\kappa}$ where τ upper bounds the running time of $\mathsf{P_B}$.*

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 EQCheck to complete our goal.

We list the differences that inverse key authentication induces to EQCheck. Recall that $\mathcal{F}_\mathsf{COT}$ has been initialized with (sid, Δ_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 $[\boldsymbol{r}_1]_{\Delta_\mathsf{A}}, \ldots, [\boldsymbol{r}_\ell]_{\Delta_\mathsf{A}}$ 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. *Let π be a random permutation. 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* EQCheck *procedure is bounded by $\frac{\tau+2}{2^\kappa}$ where τ upper bounds the running time of $\mathsf{P_B}$.*

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 D.3.

Theorem 1. *Let π be a random permutation and H_π be defined as in Definition 3. Protocol Π_cpre shown in Figure 5 and Figure 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.*

Protocol Π_{cpre}

Inputs: A Boolean circuit $\mathcal{C}$ that consists of a list of gates of the form (i, j, k, T). Let $n = |\mathcal{W}| + |\mathcal{I}_{\mathsf{B}}|$, $m = |\mathcal{W}| + |\mathcal{I}|$, $L = \lceil 2\rho \log(\frac{en}{\sqrt{2}\rho}) + \frac{\log 2\rho}{2} \rceil$ and $t = |\mathcal{W}|$. Let H_π be defined as in Definition 3.

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, $\mathcal{F}_{\mathsf{COT}}$ 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 each wire $w \in \mathcal{I} \cup \mathcal{W}$, 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 $\mathcal{C}$.

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 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

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.

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 [22], 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 [10, 47].

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. We utilize the distributed half-gates garbling scheme [32] for both of our protocols. For the protocol in this section that optimizes towards one-way communication we apply the dual-execution technique to check consistency whereas for the second protocol in the next section that optimizes towards total communication we design a novel consistency checking procedure that achieves ρ bits of communication per AND gate. Since our second protocol is inspired by the optimized WRK garbling of Dittmer et al. [18], we recall that scheme as well as the distributed half-gates garbling at Appendix E.1 and Appendix E.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 circuit and gets $\{\Lambda_w, \mathsf{L}_{w,\Lambda_w}\}_{w\in\mathcal{W}\cup\mathcal{O}}$.

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 5. *For a distributed garbling scheme with preprocessing defined by* Garble *and* Eval*, consider the event* Bad *where the evaluator aborts* or[4] *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* $\mathsf{P_B}$*'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 Appendix E.1 and Appendix E.2.) has 0-selective failure resilience [40, 32] since the inputs Λ_w to Eval are completely masked and independent of the real input. In Lemma 7 we show that for the KRRW scheme that we use in this paper, replacing the evaluator's mask to 2ρ-wise independent randomness induces $2^{-\rho}$-selective failure resilience.[5] The proof is given in Appendix D.4.

Lemma 7. *For the KRRW distributed garbling schemes (see details at Appendix E.1) by sampling the wire masks* $\boldsymbol{a}, \boldsymbol{b}$ *using the compressed preprocessing functionality* $\mathcal{F}_{\mathsf{cpre}}$ *(recall that* $\boldsymbol{b} := \mathbf{M} \cdot \boldsymbol{b}^*$ *is compressed randomness), the resulting schemes have* $2^{-\rho}$*-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 8. *After running* Eval*, the evaluator holds the 'color bits'* Λ_w *for every wire* $w \in \mathcal{W}$*. The garbler* $\mathsf{P_A}$ *and evaluator* $\mathsf{P_B}$ *also hold* $\mathsf{K_A}[\Lambda_w], \mathsf{M_B}[\Lambda_w]$ *subject to* $\mathsf{M_B}[\Lambda_w] = \mathsf{K_A}[\Lambda_w] + \Lambda_w \Delta_\mathsf{A}$*.*

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

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

It is easy to verify $\mathsf{M_B}[\Lambda_w] = \mathsf{K_A}[\Lambda_w] + \Lambda_w \cdot \Delta_\mathsf{A}$, 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 8 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 [31, 30].

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.

---

[4] Different garbling schemes vary in the choice of where to place the consistency check. For the WRK scheme the evaluator can detect consistency during the evaluation whereas for the half-gates garbling the circuit values are effectively committed after evaluation and a subsequent checking phase is dedicated to ensure that the circuit values are correctly computed. Therefore we define the event Bad as disjunction of the two cases.

[5] We note that in the proof of Ishai et al. [18] the mask $\boldsymbol{b}$ is assumed to be ρ-wise independent and the argument is that one corrupted table entry is equivalent to a coin toss with $1/2$ probability of failure. If there are less than ρ corrupted table entries then evaluator's abort probability is input-independent, otherwise (more than ρ entries are corrupted) the evaluator would abort with overwhelming probability. Their argument is not very precise so we choose to focus on the probability that the input-output correlation on each AND gate being falsified, which implies that the evaluator would abort. Thus, we require the stricter 2ρ-wise independence in $\boldsymbol{b}$, but this does not affect amortized performance of our protocol.

Intuitions of Consistency Checking. The privacy of 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 active 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 8, we may define Λ_w using the wire labels $\mathsf{L}_{w,0}, \mathsf{L}_{w,\Lambda_w}$ and the global key Δ. Our goal is to check the following equations where the left-hand (resp. right-hand) side is the evaluation result of $\mathsf{P_A}$ (resp. $\mathsf{P_B}$). Here Equation 1 is the corrupted $\mathsf{P_A}$ case while Equation 2 is the corrupted $\mathsf{P_B}$ case.

$$(\mathsf{L}'_{w,\Lambda'_w} \oplus \mathsf{L}'_{w,0}) \cdot \Delta_\mathsf{B}^{-1} \oplus a'_w \oplus b'_w = \Lambda_w \oplus a_w \oplus b_w \tag{1}$$

$$\Lambda'_w \oplus a'_w \oplus b'_w = (\mathsf{L}_{w,\Lambda_w} \oplus \mathsf{L}_{w,0}) \cdot \Delta_\mathsf{A}^{-1} \oplus a_w \oplus b_w \ . \tag{2}$$

Multiplying the first equation by Δ_B, the second by Δ_A and do summation[6] gives the $V_w^\mathsf{A}, V_w^\mathsf{B}$ values in the consistency checking.

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

Communication complexity. In our dual execution protocol, $\mathsf{P_A}$ and $\mathsf{P_B}$ sends $(2\kappa+1)t + (\kappa+2)|\mathcal{I}_\mathsf{A}| + (2\kappa+1)|\mathcal{I}_\mathsf{B}| + 2\kappa + |\mathcal{O}|$ and $(2\kappa+1)t + (\kappa+2)|\mathcal{I}_\mathsf{B}| + (2\kappa+1)|\mathcal{I}_\mathsf{A}| + \kappa$ 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.

5.3 Security Analysis

We state the security of our 2PC protocol in Theorem 2 and prove it in Appendix D.5. As an intermediate step, we prove in Lemma 9 that the difference of the V_w^A and V_w^B values in the consistency checking phase actually captures the error on the wire w (indicating whether the result of w is flipped).

Lemma 9. *Let $e_w := (a_w \oplus b_w \oplus \Lambda_w) \oplus (a'_w \oplus b'_w \oplus \Lambda'_w)$ be the error on wire $w \in \mathcal{W} \cup \mathcal{I}$ after the execution of $\Pi_\mathsf{2PC\text{-}1way}$. Then the checking values $V_w^\mathsf{A}, V_w^\mathsf{B}$ satisfy that $V_w^\mathsf{A} \oplus V_w^B = e_w \cdot (\Delta_\mathsf{A} \oplus \Delta_\mathsf{B})$.*

Theorem 2. *Let $\mathsf{H_{ccrnd}}$ be a $(\mathsf{poly}(\kappa), 2|\mathcal{W}|, \kappa, \epsilon_\mathsf{ccrnd})$-circular correlation robust hash function under naturally derived keys, $\mathsf{H_{tcr}}$ be a $(\mathsf{poly}(\kappa), |\mathcal{I}|, \kappa, \epsilon_\mathsf{tcr})$-tweakable correlation robust hash function, and π be a random permutation. 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 $\mathcal{F}_\mathsf{cpre}, \mathcal{F}_\mathsf{COT}$-hybrid model and the random oracle model.*

6 Optimization Towards Two-Way Communication

In this section we propose an optimization to the DILO-WRK online protocol, reducing the amortized online communication cost from $2\kappa + 3\rho$ bits to $2\kappa + \rho + 1$ bits per AND gate. We mainly focus on reducing the overhead with regard to consistency checking. In particular, our technique

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

Protocol $\Pi_{\mathsf{2PC\text{-}1way}}$

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 $\mathsf{H_{tcr}} : \{0,1\}^{2\kappa} \to \{0,1\}^\kappa$ be a tweakable correlation robust hash function, $\mathsf{H_{ccrnd}} : \{0,1\}^{2\kappa} \to \{0,1\}^\kappa$ be a circular correlation robust hash function under naturally derived keys, π be a random permutation, and H_π be as defined in Definition 3.

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.
3. $\mathsf{P_A}$ and $\mathsf{P_B}$ execute the KRRW Garble (using $\mathsf{H_{ccrnd}}$ with tweaks $\{w\|00, w\|01\}$ to instantiate the hash function H inside Garble) to generate a distributed garbled circuit $(\mathcal{GC}_\mathsf{A}, \mathcal{GC}_\mathsf{B})$. In particular, For each wire w, two labels $\mathsf{L}_{w,0}, \mathsf{L}_{w,1} \in \{0,1\}^\kappa$ are generated and satisfy $\mathsf{L}_{w,1} = \mathsf{L}_{w,0} \oplus \Delta_\mathsf{A}$. $\mathsf{P_A}$ sends $\mathcal{GC}_\mathsf{A}$ to $\mathsf{P_B}$.

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 $\Lambda_w := x_w \oplus a_w \in \{0,1\}$, 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 inputs $(\mathsf{init}, sid, \Gamma_\mathsf{A})$ and (init, sid). For each $w \in \mathcal{I}_\mathsf{B}$, $\mathsf{P_B}$ computes $\tilde{\Lambda}_w := y_w \oplus b_w$ and then the two parties call $\mathsf{Fix}(\tilde{\Lambda}_w)$ to get $[\tilde{\Lambda}_w]_{\Gamma_\mathsf{A}}$. Then $\mathsf{P_A}$ sends $m_{w,0} := \mathsf{H_{tcr}}(\mathsf{K}[\Lambda'_w], w\|0) \oplus \mathsf{L}_{w,a_w}$ and $m_{w,1} := \mathsf{H_{tcr}}(\mathsf{K}[\tilde{\Lambda}_w] \oplus \Gamma_\mathsf{A}, w\|0) \oplus \mathsf{L}_{w,\bar{a}_w}$ for $w \in \mathcal{I}_\mathsf{B}$ to $\mathsf{P_B}$, who computes $\mathsf{L}_{w,\Lambda_w} := m_{w,\tilde{\Lambda}_w} \oplus \mathsf{H_{tcr}}(\mathsf{M}[\tilde{\Lambda}_w], w\|0)$.
6. $\mathsf{P_A}$ and $\mathsf{P_B}$ run $\mathsf{Open}(a_w)$ for $w \in \mathcal{I}_\mathsf{B}$, which allows $\mathsf{P_B}$ to learn a_w and computes $\Lambda_w := \tilde{\Lambda}_w \oplus a_w$.
7. $\mathsf{P_B}$ runs $\mathsf{Eval}(\mathcal{GC}_\mathsf{A}, \mathcal{GC}_\mathsf{B}, \{(\Lambda_w, \mathsf{L}_{w,\Lambda_w})\}_{w \in \mathcal{I}_\mathsf{A} \cup \mathcal{I}_\mathsf{B}})$ (once again, using $\mathsf{H_{ccrnd}}$ with respective tweaks to instantiate the hash function H inside Eval) 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 $(\mathcal{F}_\mathsf{cpre}, \mathcal{F}_\mathsf{COT})$-hybrid model.

is to perform random linear combination prior to hashing so that the cross-terms from different AND gates can be combined. Then, using the half-gate multiplication technique we can securely evaluate the cross-term using ρ bits for each AND gate as compared to 3ρ bits in the DILO-WRK scheme. We present the protocol in Figure 9.

We note that in $\Pi_{\mathsf{2PC\text{-}2way}}$ we only use the hash function a la half-gates. Therefore, we only require the hash function to be circular correlation robust under naturally derived keys (ccrnd), which can be instantiated in the ideal cipher model by calling one random permutation [26].

Since the consistency checking phase only relies on the Free-XOR properties, we formulate it as an independent procedure that can be coupled with any distributed garbling protocol that supports Free-XOR. This may be of independent interest to future protocols that requires consistency checking without revealing the masked values for each wire.

On the length of Δ_B. We remark that since the key of $\mathsf{P_B}$ no longer serves to garble a circuit, its length can be reduced to ρ bits. Since the preprocessing protocol has constant amortized communication overhead, we can re-use the same preprocessing protocol (thus using the same functionality $\mathcal{F}_\mathsf{cpre}$) but truncate all MAC tags and keys related to Δ_B, including Δ_B itself down to

Protocol $\Pi_{\mathsf{2PC\text{-}1way}}$, 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 $[b'_w]$.
9. $\mathsf{P_A}$ and $\mathsf{P_B}$ run Garble (using $\mathsf{H_{ccrnd}}$ with tweaks $\{w\|10, w\|11\}$ to instantiate the hash function H inside Garble) to generate a distributed garbled circuit $(\mathcal{GC}'_{\mathsf{A}}, \mathcal{GC}'_{\mathsf{B}})$. For each wire w, two labels $\mathsf{L}'_{w,0}, \mathsf{L}'_{w,1} \in \{0,1\}^\kappa$ are generated and satisfy $\mathsf{L}'_{w,1} = \mathsf{L}'_{w,0} \oplus \Delta_{\mathsf{B}}$. $\mathsf{P_B}$ sends $\mathcal{GC}'_{\mathsf{B}}$ to $\mathsf{P_A}$.
10. 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 (using fresh tweaks). In particular:
    (a) For each $w \in \mathcal{I}_{\mathsf{B}}$, $\mathsf{P_B}$ computes $\Lambda'_w := y_w \oplus b'_w$ and then sends $(\Lambda'_w, \mathsf{L}'_{w,\Lambda'_w})$ to $\mathsf{P_B}$.
    (b) $\mathsf{P_A}$ and $\mathsf{P_B}$ call $\mathcal{F}_{\mathsf{COT}}$ on respective inputs (init, sid') and $(\mathsf{init}, sid', \Gamma_{\mathsf{B}})$. For each $w \in \mathcal{I}_{\mathsf{A}}$, $\mathsf{P_A}$ computes $\tilde{\Lambda}'_w := x_w \oplus a'_w$ and then the two parties call $\mathsf{Fix}(\tilde{\Lambda}'_w)$ to get $[\tilde{\Lambda}'_w]_{\Gamma_{\mathsf{B}}}$. Then $\mathsf{P_B}$ sends $m'_{w,0} := \mathsf{H_{tcr}}(\mathsf{K}[\tilde{\Lambda}'_w], w\|1) \oplus \mathsf{L}_{w,b'_w}$ and $m'_{w,1} := \mathsf{H_{tcr}}(\mathsf{K}[\tilde{\Lambda}'_w] \oplus \Gamma_{\mathsf{B}}, w\|1) \oplus \mathsf{L}_{w,\bar{b}'_w}$ for $w \in \mathcal{I}_{\mathsf{A}}$ to $\mathsf{P_A}$, who computes $\mathsf{L}'_{w,\Lambda'_w} := m'_{w,\tilde{\Lambda}'_w} \oplus \mathsf{H_{tcr}}(\mathsf{M}[\tilde{\Lambda}'_w], w\|1)$.
    (c) $\mathsf{P_A}$ and $\mathsf{P_B}$ run $\mathsf{Open}(b'_w)$ for $w \in \mathcal{I}_{\mathsf{A}}$, which allows $\mathsf{P_A}$ to learn b'_w and computes $\Lambda'_w := \tilde{\Lambda}'_w \oplus b'_w$.
    (d) $\mathsf{P_A}$ runs $\mathsf{Eval}(\mathcal{GC}'_{\mathsf{A}}, \mathcal{GC}'_{\mathsf{B}}, \{(\Lambda'_w, \mathsf{L}'_{w,\Lambda'_w})\}_{w \in \mathcal{I}_{\mathsf{A}} \cup \mathcal{I}_{\mathsf{B}}})$ (once again, using $\mathsf{H_{tccr}}$ to instantiate the hash function H inside Eval) to obtain . For each $w \in \mathcal{W}$, both parties define $[\Lambda'_w]_{\mathsf{A}} = (\mathsf{L}'_{w,0}, \mathsf{L}'_{w,\Lambda'_w}, \Lambda'_w)$.
11. $\mathsf{P_A}$ 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 $w \in \mathcal{W} \cup \mathcal{I}$, $\mathsf{P_A}$ and $\mathsf{P_B}$ respectively compute
    $$V_w^{\mathsf{A}} = (a_w \oplus a'_w \oplus \Lambda'_w)\Delta_{\mathsf{A}} \oplus \mathsf{M_A}[a_w] \oplus \mathsf{M_A}[a'_w] \oplus \mathsf{M_A}[\Lambda'_w] \oplus \mathsf{K_A}[b_w] \oplus \mathsf{K_A}[b'_w] \oplus \mathsf{K_A}[\Lambda_w],$$
    $$V_w^{\mathsf{B}} = (b_w \oplus b'_w \oplus \Lambda_w)\Delta_{\mathsf{B}} \oplus \mathsf{M_B}[b_w] \oplus \mathsf{M_B}[b'_w] \oplus \mathsf{M_B}[\Lambda_w] \oplus \mathsf{K_B}[a_w] \oplus \mathsf{K_B}[a'_w] \oplus \mathsf{K_B}[\Lambda'_w].$$
    (b) $\mathsf{P_A}$ computes $h_{\mathsf{A}} := \mathsf{H}^\pi(\{V_w^{\mathsf{A}}\}_{w \in \mathcal{I} \cup \mathcal{W}})$, and then sends it to $\mathsf{P_B}$ who computes $h_{\mathsf{B}} := \mathsf{H}^\pi(\{V_w^{\mathsf{B}}\}_{w \in \mathcal{I} \cup \mathcal{W}})$ and checks that $h_{\mathsf{A}} = h_{\mathsf{B}}$. If the check fails then $\mathsf{P_B}$ 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 $(\mathcal{F}_{\mathsf{cpre}}, \mathcal{F}_{\mathsf{COT}})$-hybrid model, continued.

ρ bits. This can be done by simply discarding the respective $\kappa - \rho$ higher bits since the messages that they authenticate are single bits. We use the original notations in the following descriptions but remind the readers that the MAC keys and tags are truncated and of ρ-bit length.

Intuitions of the consistency checking. Our starting point is the KRRW scheme, where all masked wire values are made public so that the checking equation $(\Lambda_i \oplus \lambda_i) \cdot (\Lambda_j \oplus \lambda_j) \cdot \Delta_{\mathsf{B}} = (\Lambda_k \oplus \lambda_k) \cdot \Delta_{\mathsf{B}}$ reduces to equality checking (recall that $\lambda_i \cdot \lambda_j \cdot \Delta_{\mathsf{B}}$ is already shared after preprocessing). With compressed preprocessing, the masked values must be kept secret due to not being fully masked. Therefore, we must securely compute the secret sharing of the multiplication between the masked wire values and values known to $\mathsf{P_A}$.

In more detail, we need to check for every AND gate $(i, j, k, \wedge)$ the following equation,

$$(\Lambda_i \oplus \lambda_i) \cdot (\Lambda_j \oplus \lambda_j) \cdot \Delta_{\mathsf{B}} = (\Lambda_k \oplus \lambda_k) \cdot \Delta_{\mathsf{B}} \ .$$

Protocol $\Pi_{\mathsf{2PC\text{-}2way}}$

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 $\mathsf{H_{tcr}} : \{0,1\}^{2\kappa} \to \{0,1\}^{\kappa}$ be a tweakable correlation robust hash function and $\mathsf{H_{ccrnd}} : \{0,1\}^{2\kappa} \to \{0,1\}^{\kappa}$ be a hash function that is circular correlation robust under naturally derived keys.

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.
3. $\mathsf{P_A}$ and $\mathsf{P_B}$ execute the KRRW Garble (using $\mathsf{H_{ccrnd}}$ with tweaks $\{w\|00, w\|01\}$ to instantiate the hash function H inside Garble) to generate a distributed garbled circuit $(\mathcal{GC}_\mathsf{A}, \mathcal{GC}_\mathsf{B})$. In particular, For each wire w, two labels $\mathsf{L}_{w,0}, \mathsf{L}_{w,1} \in \{0,1\}^{\kappa}$ are generated and satisfy $\mathsf{L}_{w,1} = \mathsf{L}_{w,0} \oplus \Delta_\mathsf{A}$. $\mathsf{P_A}$ then sends $\mathcal{GC}_\mathsf{A}$ to $\mathsf{P_B}$.

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 $\Lambda_w := x_w \oplus a_w \in \{0,1\}$, 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 inputs $(\mathsf{init}, sid, \Gamma_\mathsf{A})$ and (init, sid). For each $w \in \mathcal{I}_\mathsf{B}$, $\mathsf{P_B}$ computes $\tilde{\Lambda}_w := y_w \oplus b_w$ and then the two parties call $\mathsf{Fix}(\tilde{\Lambda}_w)$ to get $[\tilde{\Lambda}_w]_{\Gamma_\mathsf{A}}$. Then $\mathsf{P_A}$ sends $m_{w,0} := \mathsf{H_{tcr}}(\mathsf{K}[\Lambda'_w], w\|0) \oplus \mathsf{L}_{w,a_w}$ and $m_{w,1} := \mathsf{H_{tcr}}(\mathsf{K}[\tilde{\Lambda}_w] \oplus \Gamma_\mathsf{A}, w\|0) \oplus \mathsf{L}_{w,\bar{a}_w}$ for $w \in \mathcal{I}_\mathsf{B}$ to $\mathsf{P_B}$, who computes $\mathsf{L}_{w,\Lambda_w} := m_{w,\tilde{\Lambda}_w} \oplus \mathsf{H_{tcr}}(\mathsf{M}[\tilde{\Lambda}_w], w\|0)$.
6. $\mathsf{P_A}$ and $\mathsf{P_B}$ run $\mathsf{Open}([a_w])$ for $w \in \mathcal{I}_\mathsf{B}$, which allows $\mathsf{P_B}$ to learn a_w and computes $\Lambda_w := \tilde{\Lambda}_w \oplus a_w$.
7. $\mathsf{P_B}$ runs $\mathsf{Eval}(\mathcal{GC}_\mathsf{A}, \mathcal{GC}_\mathsf{B}, \{(\Lambda_w, \mathsf{L}_{w,\Lambda_w})\}_{w \in \mathcal{I}_\mathsf{A} \cup \mathcal{I}_\mathsf{B}})$ (once again, using $\mathsf{H_{ccrnd}}$ with tweaks $\{w\|00, w\|01\}$ to instantiate the hash function H inside Eval) 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 $[\Lambda_w]_\mathsf{B} = (\mathsf{L}_{w,0}, \mathsf{L}_{w,\Lambda_w}, \Lambda_w)$.
8. $\mathsf{P_A}$ and $\mathsf{P_B}$ run Π_GCCheck to check for consistency. If the check succeeds then the parties proceed to the next step. Otherwise, they abort.

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 9: Actively secure 2PC protocol in the $(\mathcal{F}_\mathsf{cpre}, \mathcal{F}_\mathsf{COT})$-hybrid model that optimizes towards minimal two-way communication. The differences as compared to Π_2PC are marked in blue.

We can re-write the equation as follows (notice that $B'_k \in \mathbb{F}_2$ can be locally computed by $\mathsf{P_B}$),

$$(\underbrace{\Lambda_i \cdot \Lambda_j \oplus \Lambda_k \oplus b_k \oplus \hat{b}_k \oplus \Lambda_i \cdot b_j \oplus \Lambda_j \cdot b_i}_{B'_k} \oplus a_k \oplus \hat{a}_k \oplus \Lambda_i \cdot a_j \oplus \Lambda_j \cdot a_i) \cdot \Delta_\mathsf{B} = 0 \ .$$

By expanding the terms and utilizing the IT-MAC relation $a_k \cdot \Delta_\mathsf{B} = \mathsf{M}[a_k] + \mathsf{K}[a_k]$ we have,

$$B'_k \cdot \Delta_\mathsf{B} \oplus \mathsf{M}[a_k] \oplus \mathsf{K}[a_k] \oplus \mathsf{M}[\hat{a}_k] \oplus \mathsf{K}[\hat{a}_k] \oplus \Lambda_i \cdot (\mathsf{M}[a_j] \oplus \mathsf{K}[a_j]) \oplus \Lambda_j \cdot (\mathsf{M}[a_i] \oplus \mathsf{K}[a_i]) = 0 \ .$$

By re-arranging the terms according to their membership, we have:

$$\underbrace{B'_k \cdot \Delta_{\mathsf{B}} \oplus \mathsf{K}[a_k] \oplus \mathsf{K}[\hat{a}_k] \oplus \Lambda_i \cdot \mathsf{K}[a_j] \oplus \Lambda_j \cdot \mathsf{K}[a_i]}_{B_k} \oplus \underbrace{\mathsf{M}[a_k] \oplus \mathsf{M}[\hat{a}_k]}_{A_{k,0}} \oplus \Lambda_i \cdot \mathsf{M}[a_j] \oplus \Lambda_j \cdot \mathsf{M}[a_i] = 0 \ .$$

Notice that the value B_k and $A_{k,0}$ are both locally computable by $\mathsf{P_B}$ and $\mathsf{P_A}$ respectively, so we only have to securely compute the rest of the terms. The previous method is to utilize the fact that the masked value Λ_i, Λ_j are already authenticated by the wire labels. Given such authentication we can evaluate the multiplication of Λ_i with any value $X \in \mathbb{F}_{2^\rho}$ known to $\mathsf{P_A}$ as follows. $\mathsf{P_A}$ sends $G_i := \mathsf{H}(\mathsf{L}_{i,0}) \oplus \mathsf{H}(\mathsf{L}_{i,1}) \oplus X$ to $\mathsf{P_B}$ who later recovers $\mathsf{H}(\mathsf{L}_{i,\Lambda_i}) \oplus \Lambda_i \cdot G_i = \mathsf{H}(\mathsf{L}_{i,0}) \oplus \Lambda_i \cdot X$. Clearly this forms an additive sharing of $\Lambda_i \cdot X$, and since there are two multiplications, at least 2ρ-bit of communication is needed for every AND gate.

Our insight is that in garbled circuits with free-XOR optimization, the masked value Λ_i on any wire i is a *public* linear combination of the masked wire values on all the AND gate output wires and input wires. We formalize this by defining the following public vector $\boldsymbol{c}^i \in \mathbb{F}_2^{|\mathcal{I}|+|\mathcal{W}|}$ for every wire i s.t. $\Lambda_i = \sum_k c_k^i \cdot \Lambda_k$. This allows us to collapse the two terms into one by exchanging the summation order with the random linear combination. In particular, to check the correctness of every AND gate $(i, j, k, \wedge)$ we perform random linear combination using public randomness $\chi_1, ..., \chi_t$, and our checking equation becomes the following. (Recall that $t = |\mathcal{W}|$.)

$$\sum_{k \in \mathcal{W}} \chi_k \cdot B_k \oplus \sum_{k \in \mathcal{W}} \chi_k \cdot A_{k,0} \oplus \sum_{(i',j',k',\wedge) \in \mathcal{C}_{\mathsf{and}}} \chi_{k'} \cdot (\Lambda_{i'} \cdot \mathsf{M}[a_{j'}] \oplus \Lambda_{j'} \cdot \mathsf{M}[a_{i'}]) = 0 \ .$$

Using the aforementioned notation, we have,

$$\sum_{k \in \mathcal{W}} \chi_k \cdot B_k \oplus \sum_{k \in \mathcal{W}} \chi_k \cdot A_{k,0} \oplus \sum_{(i',j',k',\wedge) \in \mathcal{C}_{\mathsf{and}}} \chi_{k'} \cdot \big((\sum_{k \in \mathcal{W} \cup \mathcal{I}} c_k^{i'} \cdot \Lambda_k) \cdot \mathsf{M}[a_{j'}] \oplus (\sum_{k \in \mathcal{W} \cup \mathcal{I}} c_k^{j'} \cdot \Lambda_k) \cdot \mathsf{M}[a_{i'}] \big) = 0 \ .$$

By exchanging the summation order, we have,

$$\sum_{k \in \mathcal{W}} \chi_k \cdot B_k \oplus \sum_{k \in \mathcal{W}} \chi_k \cdot A_{k,0} \oplus \sum_{k \in \mathcal{W} \cup \mathcal{I}} \Lambda_k \cdot \underbrace{\sum_{(i',j',k',\wedge) \in \mathcal{C}_{\mathsf{and}}} \chi_{k'} \cdot (c_k^{i'} \cdot \mathsf{M}[a_{j'}] \oplus c_k^{j'} \cdot \mathsf{M}[a_{i'}])}_{A_{k,1}} = 0 \ .$$

Using the half-gates technique, $\mathsf{P_A}$ sends $G'_k := \mathsf{H}_{\mathsf{ccrnd}}(\mathsf{L}_{k,0}, k\|2) \oplus \mathsf{H}_{\mathsf{ccrnd}}(\mathsf{L}_{k,1}, k\|2) \oplus A_{k,1}$ for every $k \in \mathcal{W} \cup \mathcal{I}$ to evaluate the additive sharing of $\Lambda_k \cdot A_{k,1}$. Therefore, we reduce the consistency checking of AND gate correlation to equality checking using ρ bits of amortized communication.

Now we formulate this as an independent procedure $\mathsf{GCCheck}$ in Figure 10.

Communication Complexity. In the online phase $\mathsf{P_A}$ and $\mathsf{P_B}$ sends $(2\kappa + \rho + 1)t + (\kappa + \rho + 1)|\mathcal{I}_{\mathsf{A}}| + (2\kappa + \rho + 1)|\mathcal{I}_{\mathsf{B}}| + \kappa + |\mathcal{O}|$ and $|\mathcal{I}_{\mathsf{B}}|$ bits respectively. Since in this protocol we only need to invoke Π_{cpre} once, we conclude that the total two-way communication of $\Pi_{\mathsf{2PC\text{-}2way}}$ is $2\kappa + \rho + 5$ bits per AND gate.

6.1 Security Analysis

We first claim in Lemma 10 that the soundness error of the consistency checking phase can be bounded by $\frac{t+1}{2^\rho}$. Then, the main security theorem is shown in Theorem 3. The respective proofs are shown in Appendix D.6 and Appendix D.7.

Protocol GCCheck

Inputs: $\mathsf{P_A}$ and $\mathsf{P_B}$ holds the wire masks $\boldsymbol{a}, \hat{\boldsymbol{a}}, \boldsymbol{b}, \hat{\boldsymbol{b}}$ authenticated by $\Delta_\mathsf{B}, \Delta_\mathsf{A}$ respectively. Moreover, $\mathsf{P_B}$ holds the evaluated masked wire bits and the corresponding labels $\{\Lambda_w, \mathsf{L}_{w,\Lambda_w}\}$ for each wire w in the circuit, while the garbler holds $\{\mathsf{L}_{w,0}, \mathsf{L}_{w,1}\}$. Let $\mathsf{H}'_{\mathsf{ccrnd}} : \{0,1\}^{2\kappa} \to \{0,1\}^\rho$ be the hash function that evaluates $\mathsf{H}_{\mathsf{ccrnd}}$ and truncates the output down to ρ bits. Here $\mathsf{H}_{\mathsf{ccrnd}}$ is a circular correlation robust hash function under naturally derived keys.

Consistency check:

1. $\mathsf{P_B}$ samples a random challenge $\chi_1, ..., \chi_t \in \mathbb{F}_{2^\rho}$ and sends it to $\mathsf{P_A}$. (Recall that $|\mathcal{W}| = t$.)
2. The parties locally compute the following values.
   - $\mathsf{P_A}$ computes $A_{k,0} := \mathsf{M}[a_k] \oplus \mathsf{M}[\hat{a}_k]$ for $k \in \mathcal{W}$ and $A_{k,1} := \sum_{(i',j',k',\wedge)\in\mathcal{C}_{\mathsf{and}}} \chi_{k'} \cdot (c_k^{i'} \cdot \mathsf{M}[a_{j'}] \oplus c_k^{j'} \cdot \mathsf{M}[a_{i'}])$ for $k \in \mathcal{W} \cup \mathcal{I}$, where $\boldsymbol{c}^i$ is a public vector such that $\sum_{k\in\mathcal{W}\cup\mathcal{I}} c_k^i \cdot \Lambda_k = \Lambda_i$.
   - $\mathsf{P_B}$ computes $B'_k := \Lambda_i \cdot \Lambda_j \oplus \Lambda_k \oplus b_k \oplus \hat{b}_k \oplus \Lambda_i \cdot b_j \oplus \Lambda_j \cdot b_i$ and $B_k := B'_k \cdot \Delta_\mathsf{B} \oplus \mathsf{K}[a_k] \oplus \mathsf{K}[\hat{a}_k] \oplus \Lambda_i \cdot \mathsf{K}[a_j] \oplus \Lambda_j \cdot \mathsf{K}[a_i]$ for $(i, j, k, \wedge) \in \mathcal{C}_{\mathsf{and}}$.
3. For each AND gate $(i, j, k, \wedge)$, $\mathsf{P_A}$ sends $G'_k := \mathsf{H}'_{\mathsf{ccrnd}}(\mathsf{L}_{k,0}, k\|2) \oplus \mathsf{H}'_{\mathsf{ccrnd}}(\mathsf{L}_{k,1}, k\|2) \oplus A_{k,1}$ to $\mathsf{P_B}$. $\mathsf{P_A}$ locally defines $C_k := \mathsf{H}'_{\mathsf{ccrnd}}(\mathsf{L}_{k,0}, k\|2)$ while $\mathsf{P_B}$ computes $D_k := \mathsf{H}'_{\mathsf{ccrnd}}(\mathsf{L}_{k,\Lambda_k}, k\|2) \oplus \Lambda_k \cdot G'_k$.
4. $\mathsf{P_A}$ computes $h_\mathsf{A} := \sum_{k\in\mathcal{W}} \chi_k \cdot A_{k,0} \oplus C_k$ and sends it to $\mathsf{P_B}$. $\mathsf{P_B}$ computes $h_\mathsf{B} := \sum_{k\in\mathcal{W}} \chi_k \cdot B_k \oplus D_k$ and aborts if $h_\mathsf{A} \neq h_\mathsf{B}$. Otherwise $\mathsf{P_B}$ continues.

Figure 10: The consistency checking procedure that keeps the privacy of the evaluator's masked bits.

Lemma 10. *After the equality check* GCCheck *(Figure 10), except with probability* $\frac{2}{2^\rho}$, $\mathsf{P_B}$ *either aborts or evaluates the garbled circuit exactly according to* $\mathcal{C}$.

Theorem 3. *Let* $\mathsf{H}_{\mathsf{ccrnd}}$ *be a* $(\mathsf{poly}(\kappa), 3|\mathcal{W}| + |\mathcal{I}|, \kappa, \epsilon_{\mathsf{ccrnd}})$*-circular correlation robust hash function under naturally derived keys,* $\mathsf{H}_{\mathsf{tcr}}$ *be a* $(\mathsf{poly}(\kappa), |\mathcal{I}_\mathsf{B}|, \kappa, \epsilon_{\mathsf{tcr}})$*-tweakable correlation robust hash function. Protocol* $\Pi_{\mathsf{2PC\text{-}2way}}$ *shown in Figure 9 securely realizes functionality* $\mathcal{F}_{\mathsf{2PC}}$ *in the presence of malicious adversary in the* $(\mathcal{F}_{\mathsf{cpre}}, \mathcal{F}_{\mathsf{COT}})$*-hybrid model.*

Acknowledgements

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, 62022018). Yu Yu is 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. Xiao Wang is supported by DARPA under Contract No. HR001120C0087, NSF award #2016240, #2236819, 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. We thank the anonymous reviewers for their helpful comments.

References

[1] 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.

[2] 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.

[3] 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.

[4] 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.

[5] 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.

[6] 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.

[7] 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.

[8] 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.

[9] 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.

[10] 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.

[11] 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.

[12] Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, Lisa Kohl, and Peter Scholl. Efficient pseudorandom correlation generators from ring-LPN. In *Advances in Cryptology—Crypto 2020, Part II*, volume 12171 of *LNCS*, pages 387–416. Springer, 2020.

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

[14] 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.

[15] 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.

[16] Hongrui Cui, Xiao Wang, Kang Yang, and Yu Yu. Actively secure half-gates with minimum overhead under duplex networks. LNCS, pages 35–67. Springer, 2023.

[17] 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.

[18] 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.

[19] 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.

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

[21] 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.

[22] 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.

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

[24] 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.

[25] 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.

[26] 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.

[27] 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.

[28] 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.

[29] 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.

[30] 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.

[31] 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.

[32] 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.

[33] 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.

[34] 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.

[35] 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.

[36] 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.

[37] 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.

[38] Jacques Patarin. The "coefficients H" technique (invited talk). In *Annual International Workshop on Selected Areas in Cryptography (SAC) 2008*, LNCS, pages 328–345. Springer, 2009.

[39] 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.

[40] 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.

[41] 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.

[42] 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.

[43] 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.

[44] 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.

[45] 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.

[46] 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.

[47] 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.

[48] 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.

[49] 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 [14]. 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 $\mathcal{F}_{\mathsf{EQ}}$ [37] that is recalled in Figure 11. 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 11: 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 12, 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 12: 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 $n = \|\mathcal{W}\| + \|\mathcal{I}_\mathsf{B}\|, m = \|\mathcal{W}\| + \|\mathcal{I}_\mathsf{A}\|, t = \|\mathcal{W}\|$
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 Instantiation of Extended Tweakable Correlation Robustness in the Random Permutation Model

Lemma 11. *Let $p < 2^k/2$. If π is modeled as a random permutation then TMMO^π is (t, q, ρ, ϵ)-extended tweakable correlation robust, where $\epsilon = \frac{4tq}{2^k} + \frac{5q^2}{2^{k+1}} + \frac{tq}{2^\rho} + \frac{q}{2^k}$.*

The original proof of [26] can be applied almost verbatim. Here we recall the proof for completeness. The proof utilizes the H-coefficient technique [38]. In particular, the notation $\pi \vdash Q_\pi$ indicates that the permutation π is consistent with the transcript Q_π.

Proof. Fix a deterministic distinguisher D making queries to two oracles. The first is a random permutation on $\{0,1\}^\kappa$ (and its inverse); in the real world, the second oracle is $\mathcal{O}_\Delta^{\mathsf{etcr}}(w, i, j) = \mathsf{TMMO}^\pi(\Delta \cdot j \oplus w, i)$ (for Δ sampled from $\mathcal{R}$), but in the ideal world it is a random function from

$\{0,1\}^{3\kappa}$ to $\{0,1\}^{\kappa}$. Following the notation from [26, Section VII-A], we denote the transcript of D's interaction by $Q = (Q_\pi, Q_\mathcal{O}, \Delta)$. We only consider attainable transcripts.

We say a transcript $(Q_\pi, Q_\mathcal{O}, \Delta)$ is bad if:

- (B-1) There is a query $(w_k, i_k, j_k, z_k) \in Q_\mathcal{O}$ and a query of the form $(\Delta \cdot j_k \oplus w_k, \star) \in Q_\pi$.
- (B-2) There is a query $(w_k, i_k, j_k, z_k) \in Q_\mathcal{O}$ such that $z_k = 0$.
- (B-3) There are distinct (w_k, i_k, j_k, z_k), $(w_\ell, i_\ell, j_\ell, z_\ell) \in Q_\mathcal{O}$ such that $z_j = z_\ell$.

Since we require that $j_k \neq 0$, multiplying by j_k over $\mathbb{F}_{2^\kappa}$ does not change the min-entropy of Δ and thus the probability of (B-1) in the ideal world is at most $tq/2^\rho$. Since each z_k is uniform and independent of Δ, it is similarly easy to see that the probability of (B-2) in the ideal world is at most $q/2^\kappa$, and the probability of (B-3) in the ideal world is at most $q^2/2^{\kappa+1}$.

Fix a good transcript $Q = (Q_\pi, Q_\mathcal{O}, \Delta)$. Letting $Q_\mathcal{O} = \{(w_k, i_k, j_k, z_k)\}$ as above, define $u_k = \Delta \cdot j_k \oplus w_k$ for $1 \leq k \leq q$,and set $\mathcal{U} = \{u_1, ..., u_q\}$. Fixing some $\pi \vdash Q_\pi$,we may define $v_k = \pi(u_k)$, $s_k = v_k \oplus i_k$, and $t_k = z_k \oplus v_k$; set $\mathcal{V} = \{v_1, ..., v_q\}$. Define a predicate $\mathsf{Bad}(\pi)$ on π, which is true if any of the following hold:

- (C-1) For some $1 \leq k \leq q$,there is a query of the form $(s_k, \star) \in Q_\pi$, or $s_k \in \mathcal{U}$.
- (C-2) For some $1 \leq k \leq q$,there is a query of the form $(\star, t_k) \in Q_\pi$, or $t_k \in \mathcal{V}$.
- (C-3) There are distinct k, ℓ with $1 \leq k < \ell \leq q$, such that $s_k = s_\ell$ or $t_k = t_\ell$.

We bound the probability of the above events when π is a uniform permutation, conditioned on $\pi \vdash Q_\pi$.

Consider (C-1). Fixing some index k, recall that $s_k = v_k \oplus i_k = \pi(\Delta \cdot j_k \oplus w_k) \oplus i_k$. Since Q is good, $\pi(\Delta \cdot j_k \oplus w_k)$ is uniform in a set of size at least $2^\kappa - t$ (and thus so is s_k). Therefore, $\Pr[\exists (x, y) \in Q_\pi : s_k = x] \leq \frac{t}{2^\kappa - t} \leq \frac{2t}{2^\kappa}$, using $t \leq 2^{\kappa-1}$. Similarly, we have $\Pr[s_k \in \mathcal{U}] \leq \frac{|\mathcal{U}|}{2^\kappa - t} \leq \frac{2q}{2^\kappa}$. Taking a union bound over all k, we have that the probability of (C-1) is at most $\frac{2q(t+q)}{2^\kappa}$.

Next consider (C-2). Fixing some index k, recall that $t_k = z_k \oplus v_k = z_k \oplus \pi(\Delta \cdot j_k \oplus w_k)$ and so, arguing as above, we have $\Pr[\exists (x, y) \in Q_\pi : t_k = y] \leq \frac{t}{2^\kappa - t} \leq < \frac{2t}{2^\kappa}$.

Fixing some $v_\ell \in \mathcal{V}$, we have $t_k = v_\ell$ iff $z_k \oplus \pi(\Delta \cdot j_k \oplus w_k) = \pi(\Delta \cdot j_\ell \oplus w_\ell)$. The above can only possibly occur if $\Delta \cdot j_k \oplus w_k \neq \Delta \cdot j_\ell \oplus w_\ell$ since, if not, then $z_k = 0$ in contradiction to (B-2). But if $\Delta \cdot j_k \oplus w_k \neq \Delta \cdot j_\ell \oplus w_\ell$ then $\pi(\Delta \cdot j_\ell \oplus w_\ell)$ is uniform in a set of size at least $2^\kappa - t - 1$ even conditioned on the value of $\pi(\Delta \cdot j_k \oplus w_k)$ and thus $\Pr[t_k = v_\ell] \leq \frac{1}{2^\kappa - t - 1} \leq \frac{2}{2^\kappa}$ (once again, using $t < 2^{\kappa-1}$). Taking a union bound over all $v_\ell \in \mathcal{V}$ we see that the probability that $t_k \in \mathcal{V}$ is at most $\frac{2q}{2^\kappa}$. Finally, taking a union bound over all k (and considering both sub-cases above) shows that the probability of (C-2) is at most $\frac{2q(t+q)}{2^\kappa}$.

To analyze (C-3), fix distinct k, ℓ. Then $s_k = s_\ell$ iff $\pi(\Delta \cdot j_k \oplus w_k) \oplus i_k = \pi(\Delta \cdot j_\ell \oplus w_\ell) \oplus i_\ell$. If $\Delta \cdot j_k \oplus w_k = \Delta \cdot j_\ell \oplus w_\ell$ then $i_k \neq i_\ell$ and so $s_k = s_\ell$ is impossible. Otherwise, $\pi(\Delta \cdot j_k \oplus w_k)$ is uniform in $\geq 2^\kappa - t - 1$ values even conditioned on the value of $\pi(\Delta \cdot j_\ell \oplus w_\ell)$, and thus $\Pr[s_k = s_\ell] \leq \frac{1}{2^\kappa - t - 1} \leq \frac{2}{2^\kappa}$.

The event $t_k = t_\ell$ occurs iff $z_k \oplus \pi(\Delta \cdot j_k \oplus w_k) = z_\ell \oplus \pi(\Delta \cdot j_\ell \oplus w_\ell)$. The above can only possibly occur if $\Delta \cdot j_k \oplus w_k \neq \Delta \cdot j_\ell \oplus w_\ell$ since, if not, then $z_k = z_\ell$ in contradiction to (B-3). But if $\Delta \cdot j_k \oplus w_k \neq \Delta \cdot j_\ell \oplus w_\ell$ then $\pi(\Delta \cdot j_k \oplus w_k)$ is uniform in a set of at least $2^\kappa - t - 1$ values, even conditioned on $\pi(\Delta \cdot j_\ell \oplus w_\ell)$, and so $\Pr[s_k = s_\ell] \leq \frac{2}{2^\kappa}$. Taking a union bound over all distinct k, ℓ shows that the probability of (C-3) is at most $2t^2/2^\kappa$. In summary, we have $\Pr[\mathsf{Bad} \mid \pi \vdash Q_\pi] \leq \frac{4t(t+q)+2q^2}{2^\kappa}$.

The probability that the ideal world is consistent with the good transcript Q is $\frac{\Pr[\mathcal{R}=\Delta]}{(2^\kappa)_t \cdot 2^{kq}}$, where $(2^\kappa)_t$ denotes $2^\kappa \cdot (2^\kappa - 1) \cdot \ldots \cdot (2^\kappa - t + 1)$. Now we bound the probability that the real world is consistent with Q.

The probability that the real world is consistent with the transcript is

$$\frac{\Pr[\forall (w,i,j,z) \in Q_{\mathcal{O}} : \mathcal{O}^{\mathsf{etcr}}_{\Delta}(w,i,j) = z \mid \pi \vdash Q_\pi]}{(2^\kappa)_t} \cdot \Pr[\mathcal{R} = \Delta].$$

Let $\pi \vdash_k Q$ if $\pi \vdash Q_\pi$ and $\mathcal{O}^{\mathsf{etcr}}(w_\ell, i_\ell, j_\ell) = z_\ell$ for all $\ell \le k$. The numerator above is at least

$$\Pr[\pi \vdash_q Q \wedge \neg\mathsf{Bad}(\pi) \mid \pi \vdash Q_\pi] \ge (1 - \Pr[\mathsf{Bad}(\pi) \mid \pi \vdash Q_\pi]) \cdot \prod_{k=1}^{q} \Pr[\pi \vdash_k Q \mid \neg\mathsf{Bad}(\pi) \wedge \pi \vdash_{k-1} Q]$$

Consider any π such that $\pi \vdash Q_\pi$ and $\neg\mathsf{Bad}(\pi)$. Note that $\mathcal{O}^{\mathsf{etcr}}(w_k, i_k, j_k) = z_k$ iff $\pi(s_k) = t_k$ (for Δ, s_k, t_k as defined before). If $\neg\mathsf{Bad}(\pi)$, there is no query of the form $(s_k, \star)$ or of the form $(\star, t_k)$ in Q_π. Moreover, since neither (C-1) nor (C-2) occur, neither $\pi(s_k)$ nor $\pi^{-1}(t_k)$ is determined by the input/output relations $\{\pi(u_\ell) = v_\ell\}_{\ell=1,\ldots,q}$. Furthermore, since (C-3) does not occur, neither $\pi(s_k)$ nor $\pi^{-1}(t_k)$ is determined by the fact that $\pi \vdash_{j-1} Q$ or, equivalently, the fact that $\pi(s_\ell) = t_\ell$ for all $\ell < k$. Thus, for all k we have

$$\Pr[\pi \vdash_k Q \mid \neg\mathsf{Bad}(\pi) \wedge \pi \vdash_{k-1} Q] \ge 1/2^\kappa,$$

and therefore

$$\Pr[\pi \vdash_q Q \mid \neg\mathsf{Bad}(\pi) \wedge \pi \vdash_{k-1} Q] \ge 1/2^{\kappa q}.$$

Thus the ratio of the probability that the real world is consistent with Q to the probability that the ideal world is consistent with Q is at least $(1 - \Pr[\mathsf{Bad}(\pi) \mid \pi \vdash Q_\pi])$. Using the bound on the probability of $\mathsf{Bad}(\pi)$ we can conclude the adversary's advantage is bounded by $\epsilon = \frac{4tq}{2^k} + \frac{5q^2}{2^{k+1}} + \frac{tq}{2^\rho} + \frac{q}{2^k}$. □

D Proofs of Security

D.1 Row-independence of Random Matrix

Let $L = \lceil \rho + m \cdot \log(\frac{en}{m}) + \frac{\log m}{2} \rceil$ 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 (m, L)-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:

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

Using Markov's inequality we have

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

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

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

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

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

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

D.2 Proof of Lemma 5

Proof. Suppose $y_i \neq y'_i$ for some $i \in [\ell]$. Then we have

$$\begin{aligned} V_i &= k_i \Delta'_{\mathsf{A}} \oplus k'_i \Delta_{\mathsf{A}} \oplus \mathsf{K}_{\mathsf{A}}[\tilde{m}_i]_{\Delta'_{\mathsf{A}}} \oplus \mathsf{K}_{\mathsf{A}}[\tilde{m}'_i]_{\Delta_{\mathsf{A}}} \\ &= (m_i \oplus y_i \Delta_{\mathsf{A}})\Delta'_{\mathsf{A}} \oplus (m'_i \oplus y'_i \Delta'_{\mathsf{A}})\Delta_{\mathsf{A}} \oplus (\mathsf{M}_{\mathsf{B}}[\tilde{m}_i]_{\Delta'_{\mathsf{A}}} \oplus \tilde{m}_i \Delta'_{\mathsf{A}}) \oplus (\mathsf{M}_{\mathsf{B}}[\tilde{m}'_i]_{\Delta_{\mathsf{A}}} \oplus \tilde{m}'_i \Delta_{\mathsf{A}}) \\ &= (y_i \oplus y'_i)\Delta_{\mathsf{A}}\Delta'_{\mathsf{A}} \oplus (m_i \oplus \tilde{m}_i)\Delta'_{\mathsf{A}} \oplus (m'_i \oplus \tilde{m}'_i)\Delta_{\mathsf{A}} \oplus \mathsf{M}_{\mathsf{B}}[\tilde{m}'_i]_{\Delta_{\mathsf{A}}} \oplus \mathsf{M}_{\mathsf{B}}[\tilde{m}_i]_{\Delta'_{\mathsf{A}}} \ . \end{aligned}$$

Fixing Δ_{A}, the probability that $(y_i \oplus y'_i)\Delta_{\mathsf{A}}\Delta'_{\mathsf{A}} \oplus (m_i \oplus \tilde{m}_i)\Delta'_{\mathsf{A}} \oplus (m'_i \oplus \tilde{m}'_i)\Delta_{\mathsf{A}} = 0$ is at most $2^{-\kappa}$. Conditioned on this event not happening, V_i is uniformly random to P_{B} and therefore the probability that it's learned by P_{B} is at most $\frac{\tau}{2^\kappa}$. Once again, conditioned on this event not happening, h_{A} appears uniformly random to P_{B} and the probability of $h_{\mathsf{A}} = h_{\mathsf{B}}$ is bounded by $2^{-\kappa}$.

Using the union bound, we conclude that the soundness error of $\mathsf{EQCheck}$ is bounded by $\frac{\tau+2}{2^\kappa}$. ☐

D.3 Proof of Theorem 1

Proof. **Correctness.** 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 $\mathsf{lsb}(\mathsf{D}_{\mathsf{A}}[x]) \oplus \mathsf{lsb}(\mathsf{D}_{\mathsf{B}}[x]) = x$ for any $x \in \mathbb{F}_2$. This implies that all the $\tilde{b}_k$ values that 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 proof there are multiple instances where the same keys Δ_{A} or Δ_{B} are used. We use different superscripts to differentiate those keys received from the adversary.

Corrupted 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^1_{A} by simulating $\mathcal{F}_{\mathsf{COT}}$.

2. $\mathcal{S}_{\mathsf{A}}$ receives m^0_{A} 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^1_{B} message. It also receives the message m^1_{A} from $\mathcal{A}$.

4. $\mathcal{S}_{\mathsf{A}}$ simulates the init and extend commands of $\mathcal{F}_{\mathsf{COT}}$ internally.

5. $\mathcal{S}_{\mathsf{A}}$ sends m^0_{B} 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^2_A from the adversary. If $m^2_\mathsf{A} \neq \mathsf{lsb}(\mathsf{D}_\mathsf{A}[x_1]), ..., \mathsf{lsb}(\mathsf{D}_\mathsf{A}[x_\rho])$ then $\mathcal{S}_\mathsf{A}$ aborts.

   (c) $\mathcal{S}_\mathsf{A}$ extracts $\mathcal{A}$'s input of Fix as Δ^2_A.

   (d) $\mathcal{S}_\mathsf{A}$ samples $\boldsymbol{y}$ as the output of extend and extracts $\mathcal{A}$'s input to the Fix command. If the multiplication correlation does not hold then $\mathcal{S}_\mathsf{A}$ aborts.

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

   (f)–(g) If $\Delta^1_\mathsf{A} \neq \Delta^2_\mathsf{A}$ 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 $\mathbf{M} \leftarrow \mathbb{F}_2^{n\times L}$ and sends it to $\mathcal{A}$.

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

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 receives $\boldsymbol{a} \leftarrow \mathbb{F}_2^m$ and $\hat{\boldsymbol{a}} \leftarrow \mathbb{F}_2^t$ from $\mathcal{A}$ in $\mathcal{F}^{L+1}_\mathsf{bCOT}$ and $\mathcal{F}^2_\mathsf{bCOT}$. 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 $a_{i,j}$ 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 $\mathcal{A}$ and evaluates the $\hat{b}_k$ values.

10. $\mathcal{S}_\mathsf{A}$ simulates Fix using previously computed $\hat{b}_k$ values.

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 $a_{i,j} \neq a_i \cdot a_j$ 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 and samples $r \leftarrow \mathbb{F}_{2^\kappa}$.

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

Hybrid 3 We make explicit the use of Δ_B in this hybrid and mark them in blue.

- In step 6g we compute $h_B = \pi(\mathsf{D}_A[1_B] \oplus \mathsf{D}_A[1_A] \oplus (\Delta_A^1 \oplus \Delta_A^2)\Delta_B)$ where Δ_A^2 is defined as in the simulation above.
- We compute h_B in $\mathsf{CheckZero2}(\langle 1_B^{(1)}\rangle - \langle 1_B^{(2)}\rangle, \langle 1_B^{(2)}\rangle - \langle 1_B^{(3)}\rangle)$ in step 12 as $\pi(\mathsf{D}_A[1_B^{(1)}] \oplus \mathsf{D}_A[1_B^{(2)}] \oplus ((\Delta_A')^{(1)} \oplus (\Delta_A')^{(2)})\Delta_B), \pi(\mathsf{D}_A[1_B^{(2)}] \oplus \mathsf{D}_A[1_B^{(3)}] \oplus ((\Delta_A')^{(2)} \oplus (\Delta_A')^{(3)})\Delta_B)$.
- In step 15 we compute h_B as $\pi(\mathsf{D}_A[y] \oplus y \cdot \mathsf{D}_A[1_B] \oplus \sum_k \chi^k e_k \Delta_A^1 \Delta_B)$ where e_k is the error in $\tilde{b}_k$ that $\mathcal{A}$ sends in step 9.

Since we merely re-write the input inside the H^π function, $\mathbf{Hybrid_2}$ and $\mathbf{Hybrid_3}$ are identical.

Hybrid 4 $\mathcal{S}_A$ replaces the blue terms in the previous hybrid with uniform randomness. Since the preimage in the blue terms are uniformly random to $\mathcal{A}$, except with probability $\frac{\tau}{2^\kappa}$ the blue values are uniformly random (τ upper bounds the running time of $\mathcal{A}$). Thus, $\mathbf{Hybrid_3}$ and $\mathbf{Hybrid_4}$ are $\frac{\tau}{2^\kappa}$-indistinguishable.

Hybrid 5 $\mathcal{S}_A$ 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 6 $\mathcal{S}_A$ receives the message m_A^2 from $\mathcal{A}$. If $m_A^2 \neq \mathsf{lsb}(\mathsf{D}_A[x_1]), ..., \mathsf{lsb}(\mathsf{D}_A[x_\rho])$ then $\mathcal{S}_A$ aborts. By Lemma 3 the two hybrids are $2^{-\rho}$-indistinguishable.

Hybrid 7 If $\Delta_A^1 \neq \Delta_A^2$ in step 6c then $\mathcal{S}_A$ aborts. Since h_B is uniformly random to $\mathcal{A}$ if $\Delta_A^1 \neq \Delta_A^2$, we have that $\mathbf{Hybrid_6}$ and $\mathbf{Hybrid_7}$ are $2^{-\kappa}$-indistinguishable.

Hybrid 8 Let $(\Delta_A')^{(1)}$ and $(\Delta_A')^{(2)}$ be the Fix command input of $\mathcal{A}$ in step 4 and step 5 respectively. Let $(\Delta_A')^{(3)}$ be the input of $\mathcal{F}_{\mathsf{COT}}$ in step 12. If $(\Delta_A')^{(1)} \neq (\Delta_A')^{(2)}$ or $(\Delta_A')^{(2)} \neq (\Delta_A')^{(3)}$ then $\mathcal{S}_A$ aborts to simulate $\mathsf{CheckZero2}$. Using the previous argument, the two hybrids are $2^{-\kappa}$-indistinguishable.

Hybrid 8 If $\mathcal{A}$ sends incorrect $\mathsf{lsb}(\mathsf{D}_A[\tilde{b}_k])$ values in step 9 then $\mathcal{S}_A$ simulates the $\mathsf{CheckZero2}$ command using previous strategy. By the Schwartz-Zippel lemma the two hybrids are $(\frac{t+1}{2^\kappa})$-indistinguishable. This is the ideal world execution.

Therefore, the ideal world and real world executions are $(\frac{2}{2^\rho} + \frac{t+\tau+3}{2^\kappa})$-indistinguishable in the corrupted P_A case.

Corrupted P_B. $\mathcal{S}_B$ first simulates the key sampling protocol Π_{samp} as follows:

1. $\mathcal{S}_B$ simulates the init and extend command internally.
2. $\mathcal{S}_B$ sends m_A^0 following protocol instruction.
3. $\mathcal{S}_B$ extracts $\tilde{\Delta}_B^1$ from the Fix macro, sends m_A^1 and receives m_B^1. It fixes $[\tilde{\Delta}_B^1]_B$ according to protocol instruction.

4. $\mathcal{S}_{\mathsf{B}}$ extracts Δ_{B}^2 from the init command.

5. $\mathcal{S}_{\mathsf{B}}$ receives m_{B}^0 from $\mathcal{A}$ and aborts if $\mathsf{msb}(\Delta_{\mathsf{B}}^2) \neq 1$.

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 the input from the Fix command. If the multiplication correlation does not hold then it aborts.

   (b) $\mathcal{S}_{\mathsf{B}}$ sends m_{A}^2 according to the 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_{B}^2 from $\mathcal{A}$ and aborts if $m_{\mathsf{B}}^2 \neq \mathsf{lsb}(\mathsf{D}_{\mathsf{B}}[y_1]), ..., \mathsf{lsb}(\mathsf{D}_{\mathsf{B}}[y_\rho])$ then $\mathcal{S}_{\mathsf{B}}$ aborts.

(f)–(g) If $\tilde{\Delta}_{\mathsf{B}}^1 \neq \Delta_{\mathsf{B}}^2$ 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. $\mathcal{S}_{\mathsf{B}}$ receives $\boldsymbol{b}^*$ from $\mathcal{A}$ 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 $(\beta_1, ..., \beta_L, \Delta_{\mathsf{B}}^3)$ and $(\beta_0, \Delta_{\mathsf{B}}^4)$ 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 and sends $\mathsf{lsb}(\mathsf{D}_{\mathsf{A}}[\tilde{b}_k])$.

10. $\mathcal{S}_{\mathsf{B}}$ extracts the input $\{\hat{b}_k^2\}$ 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 Δ_{B}^5 from the Fix command. If $\Delta_{\mathsf{B}}^3 \neq \Delta_{\mathsf{B}}^4$ or $\Delta_{\mathsf{B}}^4 \neq \Delta_{\mathsf{B}}^5$ then $\mathcal{S}_{\mathsf{B}}$ sends $h \leftarrow \mathbb{F}_{2^\kappa}$ to $\mathcal{F}_{\mathsf{EQ}}$ and aborts to simulate $\mathsf{CheckZero2}$. If $\Delta_{\mathsf{B}}^5 \neq \Delta_{\mathsf{B}}^1$ (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. $\mathcal{S}_{\mathsf{B}}$ receives $\boldsymbol{r} \leftarrow \mathbb{F}_2^\kappa$ to simulate extend. Define $r := \mathsf{B2F}(\boldsymbol{r})$. Then it extracts $r \cdot \Delta_{\mathsf{B}}^6$ 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 $y := \sum_{k\in\mathcal{W}} \chi^k \cdot \tilde{b}_k + r$.

15. $\mathcal{S}_{\mathsf{B}}$ receives y^2 from $\mathcal{A}$. If $y \neq y^2$ or $\Delta_{\mathsf{B}}^6 \neq \Delta_{\mathsf{B}}^1$ 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 then $\mathcal{S}_{\mathsf{B}}$ aborts.

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

Hybrid 1 This is the real-world execution.

Hybrid 2 $\mathcal{S}_{\mathsf{B}}$ extracts $\tilde{\Delta}_{\mathsf{B}}^1$ from the Fix command in step 3. If $\mathsf{msb}(\tilde{\Delta}[\mathsf{B}]^1) \neq 1$ then $\mathcal{S}_{\mathsf{B}}$ aborts. By Lemma 3 the two hybrids are $2^{-\rho}$-indistinguishable.

Hybrid 3 We make explicit the usage of Δ_{A} in this hybrid and mark them in blue.

- In step 6g we compute $h_{\mathsf{A}} = \pi(\mathsf{D}_{\mathsf{B}}[1_{\mathsf{B}}] \oplus \mathsf{D}_{\mathsf{B}}[1_{\mathsf{A}}] \oplus (\tilde{\Delta}_{\mathsf{B}}^1 \oplus \Delta_{\mathsf{B}}^2)\Delta_{\mathsf{A}})$ where Δ_{B}^2 is extracted as in the previous simulation.
- In step 12 we compute h_{A} in $\mathsf{CheckZero2}(\langle 1_{\mathsf{B}}^{(1)}\rangle - \langle 1_{\mathsf{B}}^{(2)}\rangle, \langle 1_{\mathsf{B}}^{(2)}\rangle - \langle 1_{\mathsf{B}}^{(3)}\rangle)$ as $\pi(\mathsf{D}_{\mathsf{B}}[1_{\mathsf{B}}^{(1)}] \oplus \mathsf{D}_{\mathsf{B}}[1_{\mathsf{B}}^{(2)}] \oplus (\Delta_{\mathsf{B}}^3 \oplus \Delta_{\mathsf{B}}^4)\Delta_{\mathsf{A}}), \pi(\mathsf{D}_{\mathsf{B}}[1_{\mathsf{B}}^{(2)}] \oplus \mathsf{D}_{\mathsf{B}}[1_{\mathsf{B}}^{(3)}] \oplus (\Delta_{\mathsf{B}}^4 \oplus \Delta_{\mathsf{B}}^5)\Delta_{\mathsf{A}})$. For the EQCheck commands we simulate them as $\pi(\mathsf{M}[\tilde{v}_1]_{\Delta} \oplus \mathsf{M}[\tilde{v}_2]_{\Delta'} \oplus (v_1 \oplus v_2)\Delta\Delta')$ where v_1, v_2 are two values to be checked and Δ, Δ' are two keys.
- In step 15 we compute h_{A} as $\pi(\mathsf{D}_{\mathsf{B}}[y] \oplus y \cdot \mathsf{D}_{\mathsf{B}}[1] \oplus (e\Delta_{\mathsf{B}}^1 \oplus r(\Delta_{\mathsf{B}}^1 \oplus \Delta_{\mathsf{B}}^6))\Delta_{\mathsf{A}})$ where e is the error in y that $\mathcal{A}$ sends in step 15 and Δ_{B}^6 is defined as in the above simulation.

Since we merely re-write the input inside the $\mathsf{H}_{\mathsf{etcr}}$ function, $\mathbf{Hybrid_2}$ and $\mathbf{Hybrid_3}$ are identical.

Hybrid 4 $\mathcal{S}_{\mathsf{B}}$ replaces the blue terms in the previous hybrid with uniform randomness. Since the preimage in the blue terms are uniformly random to $\mathcal{A}$, except with probability $\frac{\tau+1}{2^\kappa}$ the blue values are uniformly random (τ upper bounds the running time of $\mathcal{A}$). Thus, $\mathbf{Hybrid_3}$ and $\mathbf{Hybrid_4}$ are $\frac{\tau+1}{2^\kappa}$-indistinguishable[7].

Hybrid 5 $\mathcal{S}_{\mathsf{B}}$ 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 6 $\mathcal{S}_{\mathsf{B}}$ receives the message m_{B}^2 from $\mathcal{A}$. If $m_{\mathsf{B}}^2 \neq \mathsf{lsb}(\mathsf{D}_{\mathsf{B}}[y_1]), ..., \mathsf{lsb}(\mathsf{D}_{\mathsf{B}}[y_\rho])$ then $\mathcal{S}_{\mathsf{A}}$ aborts. By Lemma 3 the two hybrids are $2^{-\rho}$-indistinguishable.

Hybrid 7 If $\tilde{\Delta}_{\mathsf{B}}^1 \neq \Delta_{\mathsf{B}}^2$ in step 3 then $\mathcal{S}_{\mathsf{B}}$ aborts. Since h_{A} is uniformly random to $\mathcal{A}$, $\mathbf{Hybrid_6}$ and $\mathbf{Hybrid_7}$ are $2^{-\kappa}$-indistinguishable.

Hybrid 8 Let Δ_{B}^3 and Δ_{B}^4 be the init command input of $\mathcal{A}$ in step 4 and step 5 respectively. Let $(\Delta_{\mathsf{B}})^5$ be the input of Fix in step 12. If $\Delta_{\mathsf{B}}^3 \neq \Delta_{\mathsf{B}}^4$ or $\Delta_{\mathsf{B}}^4 \neq \Delta_{\mathsf{B}}^5$ then $\mathcal{S}_{\mathsf{B}}$ aborts to simulate CheckZero2. Using the previous argument, the two hybrids are $2 \cdot 2^{-\kappa}$-indistinguishable.

Hybrid 9 In step 12 if the input to EQCheck does not equal then $\mathcal{S}_{\mathsf{B}}$ aborts. Since the h_{A} values are uniformly random to $\mathcal{A}$, $\mathbf{Hybrid_8}$ and $\mathbf{Hybrid_9}$ are $2 \cdot 2^{-\kappa}$-indistinguishable.

Hybrid 10 If the y^2 that $\mathcal{A}$ sends in step 15 does not satisfy $y^2 = \sum_{k \in \mathcal{W}} \chi^k \cdot \tilde{b}_k + r$ or $\Delta_{\mathsf{B}}^6 \neq \Delta_{\mathsf{B}}^1$ then $\mathcal{S}_{\mathsf{B}}$ aborts. The two hybrids are $2^{-\kappa}$-indistinguishable.

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

Therefore, in the corrupted P_{B} case the real world and ideal world executions are $(\frac{2}{2^\rho} + \frac{t+\tau+8}{2^\kappa})$-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.

□

D.4 Proofs of Security Lemmas in Dual Execution

In this subsection we prove that with compressed preprocessing $\mathcal{F}_{\mathsf{cpre}}$ the KRRW distributed garbling scheme still has $2^{-\rho}$-selective failure resilience. This is essentially a formalization of the results in the work of Dittmer et al. [18, Appendix B.2].

Proof of Lemma 7. We first recall the notion of (m, L)-independence. We call a matrix $\mathbf{M} \in \mathbb{F}_2^{n \times L}$ (m, L)-independent if any m 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 m-wise independence. This notion first appears in [21] and is applied in the authenticated garbling setting in [18]. We show that if we set $L = \lceil \rho + m \cdot \log(\frac{en}{m}) + \frac{\log m}{2} \rceil$ then a uniformly random $\mathbf{M}$ satisfies (m, L)-independence except with probability $2^{-\rho}$. We give the proof in Appendix D.1. Therefore in the following we set $L = \lceil 2\rho \log(\frac{en}{\sqrt{2}\rho}) + \frac{\log 2\rho}{2} \rceil$ and assume that a uniformly random $n \times L$ matrix satisfies $(2\rho, L)$-independence.

Proof. We consider the corrupted $\mathsf{P_A}$ case and the case for corrupted $\mathsf{P_B}$ can be derived analogously. Observe the equation $z_w = a_w \oplus b_w \oplus \Lambda_w$. We analyze the event Bad inductively. Consider the following pebbling game, where we consider every input wire, internal gate, and output wire as nodes on a DAG and place a pebble on that node once the wire label and masked value for that wire/gate output is known. Specifically, we place a blue pebble if the masked value of that wire is always correct and a red pebble if the probability that the wire value is inconsistent with respect to its two predecessor wires (denote this event as Bad') is non-zero.

Initially, the wire labels $\{\mathsf{L}_w\}$ and masked wire values $\{\Lambda_w\}$ for $w \in \mathcal{I}$, we can place blue pebbles on all input wires, since they are correct by definition.

As an inductive step, given the preprocessing information and garbler's share of the garbled circuit $\mathcal{GC}_{\mathsf{A}}$ (possibly malformed), the evaluator's share $\mathcal{GC}_{\mathsf{B}}$, we can pebble those gates whose two input wires are both pebbled. If this is an XOR gate, we place a blue pebble. Otherwise (this is an AND gate), we can identify the errors in the bit position where the evaluator extract the masked wire value (usually this is the LSB). Denote this gate as $(i, j, k, \wedge)$, if there are errors in $G_{k,0}$ *or* $G_{k,1}$ then we place a red pebble on this gate. If there are no errors regardless of the choice of Λ_i, Λ_j then we place a blue pebble.

Inductively, we can go through the entire DAG until all nodes are pebbled. Notice that the event Bad occurs if the event Bad' occurs on *any* of the nodes with red pebbles. Let ℓ be the number of red pebble nodes and consider the following two cases.

- $\ell \leq \rho$: In this case the $(2\rho,\ L)$-independence of $\mathbf{M}$ ensures that the Λ_w values that underlies all the Bad' events are completely masked by $\boldsymbol{b}$ and so probability of Bad is independent of the evaluator's input.
- $\ell > \rho$: In this case the event Bad implies that ρ consecutive coin flips all equal to head, which occurs except with probability $2^{-\rho}$.

[7] The additional $\frac{1}{2^\kappa}$ security less is due to the probability that Δ' cancels out the Δ terms, similar to the argument in Lemma 6.

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.

□

Proof of Lemma 9 We prove that the difference of the V_w^{A} and V_w^{B} values in the consistency checking phase actually captures the error on the wire w (indicating whether the result of w is flipped).

Proof. In particular, we can re-write V_w^{B} using the notations from Lemma 8.

$$\begin{aligned} V_w^{\mathsf{B}} &= (b_w \oplus b'_w \oplus \Lambda_w)\Delta_{\mathsf{B}} \oplus \mathsf{M}_{\mathsf{B}}[b_w \oplus b'_w \oplus \Lambda_w] \oplus \mathsf{K}_{\mathsf{B}}[a_w \oplus a'_w \oplus \Lambda'_w] \\ &= (b_w \oplus b'_w \oplus \Lambda_w)\Delta_{\mathsf{B}} \oplus (b_w \oplus b'_w \oplus \Lambda_w)\Delta_{\mathsf{A}} \oplus \mathsf{K}_{\mathsf{A}}[b_w \oplus b'_w \oplus \Lambda_w] \\ &\quad \oplus (a_w \oplus a'_w \oplus \Lambda'_w)\Delta_{\mathsf{B}} \oplus \mathsf{M}_{\mathsf{A}}[a_w \oplus a'_w \oplus \Lambda'_w] \\ &\quad \oplus (a_w \oplus a'_w \oplus \Lambda'_w)\Delta_{\mathsf{A}} \oplus (a_w \oplus a'_w \oplus \Lambda'_w)\Delta_{\mathsf{A}} \\ &= (a_w \oplus b_w \oplus \Lambda_w \oplus a'_w \oplus b'_w \oplus \Lambda'_w)\Delta_{\mathsf{B}} \\ &\quad \oplus (a_w \oplus b_w \oplus \Lambda_w \oplus a'_w \oplus b'_w \oplus \Lambda'_w)\Delta_{\mathsf{A}} \\ &\quad \oplus \mathsf{K}_{\mathsf{A}}[b_w \oplus b'_w \oplus \Lambda_w] \oplus \mathsf{M}_{\mathsf{A}}[a_w \oplus a'_w \oplus \Lambda'_w] \oplus (a_w \oplus a'_w \oplus \Lambda'_w)\Delta_{\mathsf{A}} \\ &= V_w^{\mathsf{A}} \oplus e_w \cdot (\Delta_{\mathsf{A}} \oplus \Delta_{\mathsf{B}}) \ . \end{aligned}$$

□

D.5 Proof of Theorem 2

In this subsection we prove the security of the two party computation protocol Π_{2PC} based on dual execution as shown in Figure 7 and Figure 8.

Proof. We first prove the security against a malicious $\mathsf{P_A}$ and then prove the case for a malicious $\mathsf{P_B}$. The running time of $\mathcal{A}$ is bounded by $\tau = \mathsf{poly}(\kappa)$. 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 $\mathsf{P_A}$

1–3 During the simulation of $\mathcal{F}_{\mathsf{cpre}}$, $\mathcal{S}_{\mathsf{A}}$ receives Δ_{A}, $\boldsymbol{a}$, $\hat{\boldsymbol{a}}$, $\mathsf{M}[\boldsymbol{a}]$, $\mathsf{M}[\hat{\boldsymbol{a}}]$, $\mathsf{K}[\boldsymbol{b}^*]$, and $\mathsf{K}[\hat{\boldsymbol{b}}]$ from $\mathcal{A}$ and locally records those values. Then $\mathcal{S}_{\mathsf{A}}$ internally samples $\boldsymbol{b}^*, \hat{\boldsymbol{b}}$ and computes $\mathsf{K}[\boldsymbol{b}^*], \mathsf{K}[\hat{\boldsymbol{b}}]$ accordingly. Then the wire masks a_w, b_w for each wire w in the circuit are derived according to the protocol. $\mathcal{S}_{\mathsf{A}}$ also receives $\mathcal{GC}_{\mathsf{A}}$ from $\mathcal{A}$.

4. $\mathcal{S}_{\mathsf{A}}$ receives the wire masks and labels $(\Lambda_w, \mathsf{L}_{w,\Lambda_w})$ for each $w \in \mathcal{I}_{\mathsf{A}}$ and extracts the input $\boldsymbol{x}$ of $\mathcal{A}$ by computing $x_w := \Lambda_w \oplus a_w$. $\mathcal{S}_{\mathsf{A}}$ sends the extracted input $\boldsymbol{x}$ to $\mathcal{F}_{\mathsf{2PC}}$.

5. $\mathcal{S}_{\mathsf{A}}$ uses the all-zero input $\boldsymbol{y}$ (i.e., $\Lambda_w = b_w$) to simulate the Fix command. Then it receives $m_{w,0}, m_{w,1}$ for $w \in \mathcal{I}_{\mathsf{B}}$ and computes the input label $\mathsf{L}_{w,\Lambda_w} := m_{w,\tilde{cbit}_w} \oplus \mathsf{H}_{\mathsf{tcr}}(\mathsf{M}_{\mathsf{B}}[\tilde{\Lambda}_w], w\|0)$.

6. $\mathcal{S}_{\mathsf{A}}$ simulates the Open command with $\mathcal{A}$.

7. Using the information from preprocessing and the adversary's random tape, $\mathcal{S}_{\mathsf{A}}$ defines the additive error for each AND gate k as $e^{\mathsf{A}}_{k,0}, e^{\mathsf{A}}_{k,1}$. $\mathcal{S}_{\mathsf{A}}$ then evaluates the garbled circuit using the information from previous simulation and derives the result $\{\Lambda_w, \mathsf{L}_{w,\Lambda_w}\}$.

8. $\mathcal{S}_\mathsf{A}$ simulates the preprocessing functionality $\mathcal{F}_\mathsf{cpre}$ by receiving $(\boldsymbol{a}^*)'$, $\hat{\boldsymbol{a}}'$, $\mathsf{M}[(\boldsymbol{a}^*)']$, $\mathsf{M}[\hat{\boldsymbol{a}}']$, $\mathsf{K}[\boldsymbol{b}']$, and $\mathsf{K}[\hat{\boldsymbol{b}}']$ from $\mathcal{A}$. $\mathcal{S}_\mathsf{A}$ randomly samples $\boldsymbol{b}', \hat{\boldsymbol{b}}'$ and computes $\mathsf{M}[\boldsymbol{b}']$, $\mathsf{M}[\hat{\boldsymbol{b}}']$ accordingly.

9. $\mathcal{S}_\mathsf{A}$ simulates the garbling process by generating $\mathcal{GC}'_\mathsf{B}$ and the active path (the labels to be acquired by $\mathcal{A}$) topologically as follows.

   - For an XOR gate $(i, j, k, \oplus)$, $\mathcal{S}_\mathsf{A}$ defines $\Lambda'_k = \Lambda'_i \oplus \Lambda'_j$ and $\mathsf{L}'_{k,\Lambda'_k} = \mathsf{L}'_{i,\Lambda'_i} \oplus \mathsf{L}'_{j,\Lambda'_j}$.
   - For an AND gate $(i, j, k, \wedge)$, $\mathcal{S}_\mathsf{A}$ samples $\Lambda'_k \leftarrow \mathbb{F}_2$ and generates $G'_{k,0} \leftarrow \mathbb{F}_{2^\kappa}$, $G'_{k,1} \leftarrow \mathbb{F}_{2^\kappa}$. Then $\mathcal{S}_\mathsf{A}$ evaluates $\mathsf{L}'_{k,\Lambda'_k}$ according to the protocol specification in KRRW and defines $c'_k = \mathsf{ExtBit}(\mathsf{L}'_{k,\Lambda'_k}) \oplus \Lambda'_k$.

   Finally, $\mathcal{S}_\mathsf{A}$ sends the simulated $\mathcal{GC}'_\mathsf{B}$ to $\mathcal{A}$.

10. $\mathcal{S}_\mathsf{A}$ simulates the online phase as follows.

    (a) For each $w \in \mathcal{I}_\mathsf{B}$, $\mathcal{S}_\mathsf{A}$ uses the all-zero input $\boldsymbol{y}$ and sends $(\Lambda'_w, \mathsf{L}'_{w,\Lambda'_w})$ to $\mathcal{A}$.

    (b) For each $w \in \mathcal{I}_\mathsf{A}$, $\mathcal{S}_\mathsf{A}$ extracts the input $\boldsymbol{x}'$ of $\mathcal{A}$ by simulating the Fix command. Then $\mathcal{S}_\mathsf{A}$ simulates the message $m'_{w,\Lambda'_w} := \mathsf{H}_\mathsf{tcr}(\mathsf{M}_\mathsf{A}[\Lambda'_w], w\|1) \oplus \mathsf{L}'_{w,\Lambda'_w}$ and $m'_{w,\bar{\Lambda}'_w} \leftarrow \mathbb{F}_{2^\kappa}$ for $w \in \mathcal{I}_\mathsf{A}$ and sends them to $\mathcal{A}$.

    (c) $\mathcal{S}_\mathsf{A}$ simulates the Open command by opening $\Lambda'_w \oplus \tilde{\Lambda}'_w$ for $w \in \mathcal{I}_\mathsf{A}$. Since $\mathcal{S}_\mathsf{A}$ knows the key Δ_A this can be done efficiently.

    (d) $\mathcal{S}_\mathsf{A}$ locally defines $\mathsf{M}_\mathsf{A}[\Lambda'_w]$ for $w \in \mathcal{W}$ using Eval.

11. $\mathcal{S}_\mathsf{A}$ receives $\tilde{h}_\mathsf{A}$ from $\mathcal{A}$ and locally computes h_A according to the protocol specification. Then we define e_w to be the error for wire $w \in \mathcal{W} \cup \mathcal{I}$ as follows. For each input wire $w \in \mathcal{I}_\mathsf{A}$ define $e_w := x_w \oplus x'_w$, for $w \in \mathcal{I}_\mathsf{B}$ define $e_w = 0$, and for the AND gate $(i, j, k, \wedge)$ define $e_k := (\Lambda_i \oplus a_i \oplus b_i) \cdot (\Lambda_j \oplus a_j \oplus b_j) \oplus \Lambda_k \oplus a_k \oplus b_k$. $\mathcal{S}_\mathsf{A}$ checks that $\tilde{h}_\mathsf{A} = h_\mathsf{A}$ and $e_w = 0$ for all $w \in \mathcal{W} \cup \mathcal{I}$. If not then $\mathcal{S}_\mathsf{A}$ sends abort to $\mathcal{F}_\mathsf{2PC}$, otherwise it sends $\mathsf{continue}$.

12. $\mathcal{S}_\mathsf{A}$ checks that $\mathcal{A}$ sends the correct MAC tag $\mathsf{M}[a_w]$ for $w \in \mathcal{O}$. If not it sends abort, otherwise it sends $\mathsf{continue}$.

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 P_B using the actual input $\boldsymbol{y}$.

Hybrid 2 In this hybrid we make explicit the usage of the honest party's secret Δ_B and mark them in blue. In particular,

- In step 10b $\mathcal{S}_\mathsf{A}$ generates $m'_{w,\Lambda_w} = \mathsf{H}_\mathsf{tcr}(\mathsf{M}_\mathsf{A}[\tilde{\Lambda}'_w], w\|1) \oplus \mathsf{L}'_{w,\Lambda'_w}$ and $m'_{w,\bar{\Lambda}'_w} = \mathsf{H}_\mathsf{tcr}(\mathsf{M}_\mathsf{A}[\tilde{\Lambda}'_w] \oplus \Gamma_\mathsf{B}, w\|1) \oplus \Delta_\mathsf{B} \oplus \mathsf{L}'_{w,\Lambda'_w}$ for $w \in \mathcal{I}_\mathsf{A}$.
- In step 11b $\mathcal{S}_\mathsf{A}$ computes the checking value h_B as $\sum_i \pi(V_i^\mathsf{A} \oplus e_w \cdot \Delta_\mathsf{A} \oplus e_w \cdot \Delta_\mathsf{B})$, where e_w is the error for each wire $w \in \mathcal{W} \cup \mathcal{I}$ during Eval in step 7.
- In step 9 $\mathcal{S}_\mathsf{A}$ generates each gate in the garbled circuit $\mathcal{GC}'_\mathsf{B}$ as follows. The XOR gates are garbled as usual, while the AND gates are garbled as $G'_{k,0} = \mathsf{H}_\mathsf{ccrnd}(\mathsf{L}'_{i,\Lambda'_i}, w\|10) \oplus \mathsf{K}[a'_j] \oplus \mathsf{H}_\mathsf{ccrnd}(\mathsf{L}'_{i,\Lambda'_i} \oplus \Delta_\mathsf{B}, w\|10) \oplus b'_j \cdot \Delta_\mathsf{B}$ and $G'_{k,1} = \mathsf{H}_\mathsf{ccrnd}(\mathsf{L}'_{j,\Lambda'_j}, w\|11) \oplus \mathsf{K}[a'_i] \oplus \mathsf{L}'_{i,\Lambda'_i} \oplus \mathsf{H}_\mathsf{ccrnd}(\mathsf{L}'_{j,\Lambda'_j} \oplus \Delta_\mathsf{B}, w\|11) \oplus (b'_i \oplus \Lambda'_i) \cdot \Delta_\mathsf{B}$ while the output label $\mathsf{L}'_{k,\Lambda'_k}$ is derived using the Eval algorithm.

The first two changes make no difference to the view of the adversary since we just re-write the hash function input. Due to the observation in Lemma 9 the third change also brings no change to the adversary's view. Therefore, $\mathbf{Hybrid_1}$ and $\mathbf{Hybrid_2}$ are identically distributed.

Hybrid 3 In this hybrid we replace the first blue term with uniformly random values. Due to the tweakable correlation robust property, $\mathbf{Hybrid_2}$ and $\mathbf{Hybrid_3}$ are ϵ_{tcr}-indistinguishable.

Hybrid 4 In this hybrid we replace the second blue term with uniformly random values if $e_w \neq 0$ for some $w \in \mathcal{I} \cup \mathcal{W}$. Except when $\mathcal{A}$ queries the value $V_i^{\mathsf{A}} \oplus e_w \cdot \Delta_{\mathsf{A}} \oplus e_w \cdot \Delta_{\mathsf{B}}$ the random permuted output appears uniformly random to $\mathcal{A}$. Thus, $\mathbf{Hybrid_3}$ and $\mathbf{Hybrid_4}$ are $\frac{\tau}{2^\kappa}$-indistinguishable.

Hybrid 5 In this hybrid we replace the third blue term with uniformly random values. Due to the circular correlation robust under naturally derived keys property, $\mathbf{Hybrid_4}$ and $\mathbf{Hybrid_5}$ are $\epsilon_{\mathsf{ccrnd}}$-indistinguishable.

Hybrid 6 In this hybrid we change the simulation of the checking phase. Namely, $\mathcal{S}_{\mathsf{A}}$ sends abort whenever $e_w \neq 0$ for a wire $w \in \mathcal{W}$. If $e_w \neq 0$ in $\mathbf{Hybrid_4}$ then h_{B} is uniformly random in the view of $\mathcal{A}$, therefore an honest $\mathsf{P_B}$ would abort except with probability $2^{-\kappa}$. $\mathbf{Hybrid_5}$ and $\mathbf{Hybrid_6}$ are $2^{-\kappa}$-indistinguishable.

Hybrid 7 In this hybrid we change the input of $\mathsf{P_B}$ from $\boldsymbol{y}$ to all zeros. Since in step 5 $\mathsf{P_B}$'s input is completely masked the view of $\mathcal{A}$ is not changed. As for the honest party's output, due to Lemma 7 the probability that $\mathsf{P_B}$ aborts changes at most with probability $2^{-\rho}$. Therefore, $\mathbf{Hybrid_6}$ and $\mathbf{Hybrid_7}$ are $2^{-\rho}$-indistinguishable. This is exactly the ideal distribution.

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

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

1–3 During the simulation of $\mathcal{F}_{\mathsf{cpre}}$, $\mathcal{S}_{\mathsf{B}}$ receives Δ_{B}, $\boldsymbol{b}^*$, $\hat{\boldsymbol{b}}$, $\mathsf{M}[\boldsymbol{b}^*]$, $\mathsf{M}[\hat{\boldsymbol{b}}]$, $\mathsf{K}[\boldsymbol{a}]$, and $\mathsf{K}[\hat{\boldsymbol{a}}]$ from $\mathcal{A}$ and locally records those values. Then $\mathcal{S}_{\mathsf{B}}$ internally samples $\boldsymbol{a}, \hat{\boldsymbol{a}}$ and computes $\mathsf{K}[\boldsymbol{a}], \mathsf{K}[\hat{\boldsymbol{a}}]$ accordingly. Then the wire masks a_w, b_w for each wire w in the circuit are derived according to the protocol.

$\mathcal{S}_{\mathsf{B}}$ simulates the garbling phase by generating $\mathcal{GC}_{\mathsf{A}}$ and the active path (the labels to be acquired by $\mathcal{A}$) topologically as follows.

- For an XOR gate $(i, j, k, \oplus)$, $\mathcal{S}_{\mathsf{B}}$ defines $\Lambda_k = \Lambda_i \oplus \Lambda_j$ and $\mathsf{L}_{k,\Lambda_k} = \mathsf{L}_{i,\Lambda_i} \oplus \mathsf{L}_{j,\Lambda_j}$.
- For an AND gate $(i, j, k, \wedge)$, $\mathcal{S}_{\mathsf{A}}$ samples $\Lambda_k \leftarrow \mathbb{F}_2$ and generates $G_{k,0} \leftarrow \mathbb{F}_{2^\kappa}$, $G_{k,1} \leftarrow \mathbb{F}_{2^\kappa}$. Then $\mathcal{S}_{\mathsf{A}}$ evaluates L_{k,Λ_k} according to the protocol specification in KRRW and defines $c_k = \mathsf{ExtBit}(\mathsf{L}_{k,\Lambda_k}) \oplus \Lambda_k$.

Finally, $\mathcal{S}_{\mathsf{B}}$ sends the simulated $\mathcal{GC}_{\mathsf{A}}$ to $\mathcal{A}$.

4. $\mathcal{S}_{\mathsf{B}}$ sets $\Lambda_w = a_w$ (using all zero inputs) and then sends $\Lambda_w, \mathsf{L}_{w,\Lambda_w}$ for $w \in \mathcal{I}_{\mathsf{A}}$ to $\mathcal{A}$.

5. $\mathcal{S}_{\mathsf{B}}$ simulates the Fix command and extracts the input $\boldsymbol{y}$ of $\mathcal{A}$ and sends it to $\mathcal{F}_{\mathsf{2PC}}$. Then, $\mathcal{S}_{\mathsf{B}}$ generates the input messages $m_{w,\Lambda_w} = \mathsf{H}_{\mathsf{tcr}}(\mathsf{M_B}[\tilde{cbit}_w], w\|0) \oplus \mathsf{L}_{w,\Lambda_w}$ and $m_{\bar{\Lambda}_w} \leftarrow \mathbb{F}_{2^\kappa}$ and sends them to $\mathcal{A}$.

6. $\mathcal{S}_\mathsf{B}$ simulates the Open command by opening $\Lambda_w \oplus \tilde{\Lambda}_w$ for $w \in \mathcal{I}_\mathsf{B}$. Since $\mathcal{S}_\mathsf{B}$ knows the key Δ_B this can be done efficiently.

7. $\mathcal{S}_\mathsf{B}$ locally defines $\mathsf{M}_\mathsf{B}[\Lambda_w]$ for $w \in \mathcal{W}$ using Eval.

8–9 $\mathcal{S}_\mathsf{B}$ simulates the preprocessing functionality $\mathcal{F}_\mathsf{cpre}$ by receiving $\boldsymbol{b}'$, $\hat{\boldsymbol{b}}'$, $\mathsf{M}[\boldsymbol{b}']$, $\mathsf{M}[\hat{\boldsymbol{b}}']$, $\mathsf{K}[(\boldsymbol{a}^*)']$, and $\mathsf{K}[\hat{\boldsymbol{a}}']$ from $\mathcal{A}$. $\mathcal{S}_\mathsf{A}$ randomly samples $(\boldsymbol{a}^*)', \hat{\boldsymbol{a}}'$ and computes $\mathsf{M}[(\boldsymbol{a}^*)']$, $\mathsf{M}[\hat{\prime}]$ accordingly. $\mathcal{S}_\mathsf{B}$ receives the garbled circuit $\mathcal{GC}'_\mathsf{B}$ from $\mathcal{A}$. Using the information from preprocessing and the adversary's random tape, $\mathcal{S}_\mathsf{B}$ defines the additive error for each AND gate k as $(e')^\mathsf{B}_{k,0}, (e')^\mathsf{B}_{k,1}$.

10. $\mathcal{S}_\mathsf{B}$ simulates the online phase as follows.

    (a) For each $w \in \mathcal{I}_\mathsf{B}$ $\mathcal{S}_\mathsf{B}$ receives $\Lambda'_w, \mathsf{L}'_{w,\Lambda'_w}$ and recovers the input of $\mathcal{A}$ as $\boldsymbol{y}'$.

    (b) $\mathcal{S}_\mathsf{B}$ simulates the Fix command and recovers the input labels $\mathsf{L}'_{w,\Lambda'_w}$ for $w \in \mathcal{I}_\mathsf{A}$.

    (c) $\mathcal{S}_\mathsf{B}$ simulates the Open command with $\mathcal{A}$.

    (d) $\mathcal{S}_\mathsf{B}$ evaluates the garbled circuit using the information from previous simulation and derives the result $\{ \Lambda'_w, \mathsf{L}'_{w,\Lambda'_w} \}$ for $w \in \mathcal{W}$.

11. $\mathcal{S}_\mathsf{B}$ checks that the Λ'_w values derived from the evaluation of $\mathcal{GC}'_\mathsf{A}$ and $\mathcal{GC}'_\mathsf{B}$ are consistent with $\mathcal{C}(0, \boldsymbol{y})$ and that $\boldsymbol{y} = \boldsymbol{y}'$. In particular, for each AND gate $(i, j, k, \wedge)$, $\mathcal{S}_\mathsf{A}$ checks that $(\Lambda'_i \oplus a'_i \oplus b'_i) \cdot (\Lambda'_j \oplus a'_j \oplus b'_j) = \Lambda'_k \oplus a'_k \oplus b'_k$. If not then $\mathcal{S}_\mathsf{B}$ samples $h_\mathsf{A} \leftarrow \mathbb{F}_{2^\kappa}$ and sends it to $\mathcal{A}$. Otherwise, $\mathcal{S}_\mathsf{B}$ computes $h_\mathsf{B} = \mathsf{H}_\pi(\{V_w\}_{w \in \mathcal{I} \cup \mathcal{W}})$ according to protocol specification and sends it to $\mathcal{A}$.

12. If the previous step does not abort, then $\mathcal{S}_\mathsf{B}$ receives the correct output z_w from $\mathcal{F}_\mathsf{2PC}$ for $w \in \mathcal{O}$. Then $\mathcal{S}_\mathsf{B}$ sends the *corrected* MAC tag $\mathsf{M}[a_w] \oplus (z^0_w \oplus z_w) \cdot \Delta_\mathsf{B}$ to $\mathcal{A}$, simulating the Open command. Here $\boldsymbol{z}^0$ denotes the output value of $\mathcal{C}(0, \boldsymbol{y})$.

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

Hybrid 2 In this hybrid we make explicit the usage of the honest party's secret Δ_A and mark them in blue. In particular,

- In step 5 $\mathcal{S}_\mathsf{B}$ generates $m_{w,\Lambda_w} = \mathsf{H}_\mathsf{tcr}(\mathsf{M}_\mathsf{B}[\tilde{\Lambda}_w], w\|0) \oplus \mathsf{L}_{w,\Lambda_w}$ and $m_{w,\bar{\Lambda}_w} = \mathsf{H}_\mathsf{tcr}(\mathsf{M}_\mathsf{B}[c\tilde{bit}_w] \oplus \Gamma_\mathsf{B}, w\|1) \oplus \Delta_\mathsf{A} \oplus \mathsf{L}_{w,\Lambda_w}$ for $w \in \mathcal{I}_\mathsf{B}$.
- In step 11b $\mathcal{S}_\mathsf{B}$ computes the checking value h_A as $\sum_i \pi(V^\mathsf{B}_i \oplus e_w \cdot \Delta_\mathsf{B} \oplus e_w \cdot \Delta_\mathsf{B})$, where e_w is the error for $w \in \mathcal{W} \cup \mathcal{I}$ during Eval in step 10d.
- In step 3 $\mathcal{S}_\mathsf{B}$ generates each gate in the garbled circuit $\mathcal{GC}_\mathsf{A}$ as follows. The XOR gates are garbled as usual, while the AND gates are garbled as $G_{k,0} = \mathsf{H}_\mathsf{ccrnd}(\mathsf{L}_{i,\Lambda_i}, w\|00) \oplus \mathsf{K}[b_j] \oplus \mathsf{H}_\mathsf{ccrnd}(\mathsf{L}_{i,\Lambda_i} \oplus \Delta_\mathsf{A}, w\|00) \oplus a_j \cdot \Delta_\mathsf{A}$ and $G_{k,1} = \mathsf{H}_\mathsf{ccrnd}(\mathsf{L}_{j,\Lambda_j}, w\|01) \oplus \mathsf{K}[b_i] \oplus \mathsf{L}_{i,\Lambda_i} \oplus \mathsf{H}_\mathsf{ccrnd}(\mathsf{L}_{j,\Lambda_j} \oplus \Delta_\mathsf{A}, w\|01) \oplus (a_i \oplus \Lambda_i) \cdot \Delta_\mathsf{A}$ while the output label L_{k,Λ_k} is derived using the Eval algorithm.

Due to the observation in Lemma 9 the third change also brings no change to the adversary's view. Therefore, $\mathbf{Hybrid_1}$ and $\mathbf{Hybrid_2}$ are identically distributed.

Hybrid 3 In this hybrid we replace the first blue term in $\mathbf{Hybrid_2}$ with uniformly random values. Due to the tweakable correlation robust property of H_tcr, $\mathbf{Hybrid_2}$ and $\mathbf{Hybrid_3}$ are ϵ_tcr-indistinguishable.

Hybrid 4 In this hybrid we replace the second blue term in $\mathbf{Hybrid_2}$ with uniformly random values if $e_w \neq 0$ for some $w \in \mathcal{W} \cup \mathcal{I}$. Except when $\mathcal{A}$ queries the value $V_i^{\mathsf{B}} \oplus e_w \cdot \Delta_{\mathsf{A}} \oplus e_w \cdot \Delta_{\mathsf{B}}$ the random permuted output appears uniformly random to $\mathcal{A}$. Thus, $\mathbf{Hybrid_3}$ and $\mathbf{Hybrid_4}$ are $\frac{\mathcal{T}}{2^\kappa}$-indistinguishable.

Hybrid 5 In this hybrid we replace the third blue term in $\mathbf{Hybrid_2}$ with uniformly random values. Due to the circular correlation robust under naturally derived keys property, $\mathbf{Hybrid_4}$ and $\mathbf{Hybrid_5}$ are $\epsilon_{\mathsf{ccrnd}}$-indistinguishable.

Hybrid 6 In this hybrid we change the input of $\mathsf{P_A}$ from $\boldsymbol{x}$ to all zeros. Since in step 4 $\mathsf{P_A}$'s input is completely masked the view of $\mathcal{A}$ is not changed. As for the honest party's output, due to Lemma 7 the probabilities that $\exists w \in \mathcal{W}, e_w = 0$ under any pair of input values differ changes at most with probability $2^{-\rho}$. Therefore, $\mathbf{Hybrid_3}$ and $\mathbf{Hybrid_4}$ are $2^{-\rho}$-indistinguishable. This is exactly the ideal distribution.

Altogether, the ideal world and real world executions are $(\epsilon_{\mathsf{tcr}}+\epsilon_{\mathsf{ccrnd}}+\frac{\mathcal{T}}{2^\kappa}+2^{-\rho})$-indistinguishable in the corrupted $\mathsf{P_B}$ 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 $\mathcal{F}_{\mathsf{cpre}}, \mathcal{F}_{\mathsf{COT}}$-hybrid model. □

D.6 Proof of Lemma 10

In this subsection, we prove the soundness of the consistency checking procedure in Figure 10.

Proof. Let $e_k := (a_i \oplus b_i \oplus \Lambda_i) \cdot (a_j \oplus b_j \oplus \Lambda_j) \oplus (a_k \oplus b_k \oplus \Lambda_k)$ be the error of the output wire for an AND gate $(i, j, k, \wedge)$. Then we can show that the errors are captured in the XOR of h_{A} and h_{B}.

In particular, we can re-write h_{B} as follows.

$$\begin{aligned}
h_{\mathsf{B}} &= \sum_k \chi_k \cdot B_k \oplus D_k \\
&= \sum_k \chi_k (B'_k \Delta_{\mathsf{B}} \oplus \mathsf{K}[a_k] \oplus \mathsf{K}[\hat{a}_k] \oplus \Lambda_i \cdot \mathsf{K}[a_j] \oplus \Lambda_j \cdot \mathsf{K}[a_i]) \oplus D_k \\
&= \sum_k \chi_k ((\Lambda_i \cdot \Lambda_j \oplus \Lambda_k \oplus b_k \oplus \hat{b}_k \oplus \Lambda_i \cdot b_j \oplus \Lambda_j \cdot b_i) \cdot \Delta_{\mathsf{B}} \\
&\qquad \oplus \mathsf{K}[a_k] \oplus \mathsf{K}[\hat{a}_k] \oplus \Lambda_i \cdot \mathsf{K}[a_j] \oplus \Lambda_j \cdot \mathsf{K}[a_i]) \oplus D_k \\
&= \sum_k \chi_k ((e_k \cdot \Delta_{\mathsf{B}}) \oplus \mathsf{M}[a_k] \oplus \mathsf{M}[\hat{a}_k] \oplus \Lambda_i \cdot \mathsf{M}[a_j] \oplus \Lambda_j \cdot \mathsf{M}[a_i]) \oplus \Lambda_k \cdot A_{k,1} \oplus C_k \\
&= \sum_k \chi_k e_k \cdot \Delta_{\mathsf{B}} \oplus \sum_k (\chi_k \cdot A_{k,0} \oplus C_k) \oplus \sum_k \chi_k \cdot (\Lambda_i \cdot \mathsf{M}[a_j] \oplus \Lambda_j \cdot \mathsf{M}[a_i]) \oplus \sum_k \Lambda_k \cdot A_{k,1} \ .
\end{aligned}$$

Now we claim that the term marked blue is zero. Recall that we define $A_{k,1} := \sum_{(i',j',k',\wedge) \in \mathcal{C}} \chi_{k'} \cdot (c_k^{i'} \cdot \mathsf{M}[a_{j'}] \oplus c_k^{j'} \cdot \mathsf{M}[a_{i'}])$ and $\boldsymbol{c}_k^i$ is the public vector subject to $\sum_k c_k^i \cdot \Lambda_k = \Lambda_i$. Then by exchanging the summation order in the second part of the blue term, we can get the following result.

$$\begin{aligned}\sum_k \Lambda_k \cdot A_{k,1} &= \sum_k \Lambda_k \cdot (\sum_{(i',j',k',\wedge)\in\mathcal{C}} \chi_{k'} \cdot (c_k^{i'} \cdot \mathsf{M}[a_{j'}] \oplus c_k^{j'} \cdot \mathsf{M}[a_{i'}])) \\ &= \sum_{(i',j',k',\wedge)\in\mathcal{C}} \chi_{k'} \cdot \sum_k \Lambda_k \cdot (c_k^{i'} \cdot \mathsf{M}[a_{j'}] \oplus c_k^{j'} \cdot \mathsf{M}[a_{i'}]) \\ &= \sum_{k'} \chi_{k'} \cdot \sum_k \Lambda_k \cdot c_k^{i'} \cdot \mathsf{M}[a_{j'}] \oplus \sum_{k'} \chi_{k'} \cdot \sum_k \Lambda_k \cdot c_k^{j'} \cdot \mathsf{M}[a_{i'}] \\ &= \sum_{k'} \chi_{k'} \cdot \Lambda_{i'} \cdot \mathsf{M}[a_{j'}] \oplus \sum_{k'} \chi_{k'} \cdot \Lambda j' \cdot \mathsf{M}[a_{i'}] \ .\end{aligned}$$

This implies that the blue term is actually *zero*. Therefore, we have that $h_{\mathsf{B}} = h_{\mathsf{A}} \oplus \sum_k \chi_k \cdot e_k$. Recall that $\chi_1, ..., \chi_t$ are uniformly random. Suppose $e_k \neq 0$ for some AND gate $(i, j, k, \wedge)$ then except with probability $\frac{1}{2^\rho}$ we have $h_{\mathsf{A}} \neq h_{\mathsf{B}}$. In this case, suppose $\mathcal{A}$ sends a h_{A} that passes the test, then it implies that $\Delta_{\mathsf{B}} = (h_{\mathsf{A}} \oplus h_{\mathsf{B}}) \cdot (\sum_k \chi_k \cdot e_k)^{-1}$. Since Δ_{B} is uniformly random for $\mathcal{A}$ in the $\mathcal{F}_{\mathsf{cpre}}$-hybrid model, this happens except with probability $2^{-\rho}$.

Using the union bound, we conclude that the soundness error of the consistency checking phase is bounded by $\frac{2}{2^\rho}$. □

D.7 Proof of Theorem 3

In this subsection we prove the security of the two party computation protocol $\Pi_{\mathsf{2PC\text{-}2way}}$ in Figure 9 that optimizes towards total communication complexity.

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.

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

1–3 During the simulation of $\mathcal{F}_{\mathsf{cpre}}$, $\mathcal{S}_{\mathsf{A}}$ receives Δ_{A}, $\boldsymbol{a}$, $\hat{\boldsymbol{a}}$, $\mathsf{M}[\boldsymbol{a}]$, $\mathsf{M}[\hat{\boldsymbol{a}}]$, $\mathsf{K}[\boldsymbol{b}^*]$, $\mathsf{K}[\hat{\boldsymbol{b}}]$, and $\mathcal{GC}_{\mathsf{A}}$ from $\mathcal{A}$ and locally records those values. Then $\mathcal{S}_{\mathsf{A}}$ internally samples $\boldsymbol{b}^*, \hat{\boldsymbol{b}}$ and computes $\mathsf{M}[\boldsymbol{b}^*], \mathsf{M}[\hat{\boldsymbol{b}}]$ accordingly. Then the wire masks a_w, b_w for each wire w in the circuit are derived according to the protocol.

4. $\mathcal{S}_{\mathsf{A}}$ receives the wire masks and labels $(\Lambda_w, \mathsf{L}_{w,\Lambda_w})$ for each $w \in \mathcal{I}_{\mathsf{A}}$ and extracts the input $\boldsymbol{x}$ of $\mathcal{A}$ by computing $x_w := \Lambda_w \oplus a_w$. $\mathcal{S}_{\mathsf{A}}$ sends the extracted input $\boldsymbol{x}$ to $\mathcal{F}_{\mathsf{2PC}}$.

5. $\mathcal{S}_{\mathsf{A}}$ simulates the Fix command using the all-zero input $\boldsymbol{y}$ (i.e., $\tilde{cbit}_w = b_w$). Then it receives $m_{w,0}, m_{w,1}$ for $w \in \mathcal{I}_{\mathsf{B}}$ from $\mathcal{A}$ and computes $\mathsf{L}_{w,\Lambda_w} = \mathsf{H}_{\mathsf{tcr}}(\mathsf{M_B}[\tilde{cbit}_w]) \oplus m_{w,\tilde{\Lambda}_w}$ for $w \in \mathcal{I}_{\mathsf{B}}$.

6. $\mathcal{S}_{\mathsf{A}}$ simulates the Open command and computes Λ_w for $w \in \mathcal{I}_{\mathsf{B}}$.

7. $\mathcal{S}_{\mathsf{A}}$ evaluates the garbled circuit using the information from previous simulation and derives the result $\{\Lambda_w, \mathsf{L}_{w,\Lambda_w}\}$.

8. $\mathcal{S}_{\mathsf{A}}$ simulates the protocol Π_{GCCheck} as follows.

   (a) $\mathcal{S}_{\mathsf{A}}$ samples a random challenge χ and sends it to $\mathcal{A}$.

   (b) $\mathcal{S}_{\mathsf{A}}$ locally computes $A_{k,0}, A_{k,1}$ from $\mathcal{A}$'s previous input to $\mathcal{F}_{\mathsf{cpre}}$.

(c) $\mathcal{S}_\mathsf{A}$ receives the G'_k messages from $\mathcal{A}$ and computes the D_k values for each AND gate $(i, j, k, \wedge)$ as well as the C_k values.

(d) $\mathcal{S}_\mathsf{A}$ receives the tag $\tilde{h}_\mathsf{A}$ from $\mathcal{A}$ and computes the h_A value according to the protocol specification. Let e_k be the error of the AND gate $(i, j, k, \wedge)$ defined as $e_k = (a_i \oplus b_i \oplus \Lambda_i) \cdot (a_j \oplus b_j \oplus \Lambda_j) \oplus (a_k \oplus b_k \oplus \Lambda_k)$. If $\tilde{h}_\mathsf{A} \neq h_\mathsf{A}$ or $e_k \neq 0$ for some k, then $\mathcal{S}_\mathsf{A}$ sends abort to $\mathcal{F}_\mathsf{2PC}$. Otherwise, it sends continue.

9. $\mathcal{S}_\mathsf{A}$ simulates the verification operation inside the Open command. For $w \in \mathcal{O}$, let $\tilde{a}_w$ be the message that $\mathcal{A}$ sends in this step. If $a_w \neq \tilde{a}_w$ for any $w \in \mathcal{O}$ then $\mathcal{S}_\mathsf{A}$ sends abort to $\mathcal{F}_\mathsf{2PC}$. Otherwise it sends continue.

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 In this hybrid we simulate the Open command as in the simulation described above, i.e., whenever $\mathcal{A}$ sends inconsistent messages $\mathcal{S}_\mathsf{A}$ sends abort to $\mathcal{F}_\mathsf{2PC}$. By the soundness of IT-MAC, $\mathbf{Hybrid_1}$ and $\mathbf{Hybrid_2}$ are $2^{-\rho}$-indistinguishable.

Hybrid 3 In this hybrid we simulate the consistency checking in step 4 as in the step 8d in simulation above. Due to Lemma 10, $\mathbf{Hybrid_2}$ and $\mathbf{Hybrid_3}$ are $\frac{t+1}{2^\rho}$-indistinguishable.

Hybrid 4 In this hybrid we change the input of $\mathsf{P_B}$ from $\boldsymbol{y}$ to all zeros. Since in step 5 $\mathsf{P_B}$'s input is completely masked the view of $\mathcal{A}$ is not changed. As for the honest party's output, due to Lemma 7 the probability that $\mathsf{P_B}$ aborts changes at most with probability $2^{-\rho}$. Therefore, $\mathbf{Hybrid_3}$ and $\mathbf{Hybrid_4}$ are $2^{-\rho}$-indistinguishable. This is exactly the ideal distribution.

Altogether, the ideal world and real world executions are $\frac{t+3}{2^\rho}$-indistinguishable in the corrupted $\mathsf{P_A}$ case.

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

1–2 During the simulation of $\mathcal{F}_\mathsf{cpre}$, $\mathcal{S}_\mathsf{B}$ receives Δ_B, $\boldsymbol{b}^*$, $\hat{\boldsymbol{b}}$, $\mathsf{M}[\boldsymbol{b}^*]$, $\mathsf{M}[\hat{\boldsymbol{b}}]$, $\mathsf{K}[\boldsymbol{a}]$, and $\mathsf{K}[\hat{\boldsymbol{a}}]$ from $\mathcal{A}$ and locally records those values. Then $\mathcal{S}_\mathsf{B}$ internally samples $\boldsymbol{a}, \hat{\boldsymbol{a}}$ and computes $\mathsf{M}[\boldsymbol{a}], \mathsf{M}[\hat{\boldsymbol{a}}]$ accordingly. Then the wire masks a_w, b_w for each wire w in the circuit are derived according to the protocol.

3. $\mathcal{S}_\mathsf{B}$ simulates the garbling phase by generating $\mathcal{GC}_\mathsf{A}$ and the active path (the labels to be acquired by $\mathcal{A}$) topologically as follows.

   - For the input wires $w \in \mathcal{I}_\mathsf{A}$, $\mathcal{S}_\mathsf{B}$ defines $\Lambda_w = a_w$ (using all zero inputs) and randomly samples $\mathsf{L}_{w,\Lambda_w} \leftarrow \mathbb{F}_{2^\kappa}$. Then it samples $\Lambda_w \leftarrow \mathbb{F}_2$ and $\mathsf{L}_{w,\Lambda_w} \leftarrow \mathbb{F}_{2^\kappa}$ for $w \in \mathcal{I}_\mathsf{B}$.
   - For an XOR gate $(i, j, k, \oplus)$, $\mathcal{S}_\mathsf{B}$ defines $\Lambda_k = \Lambda_i \oplus \Lambda_j$ and $\mathsf{L}_{k,\Lambda_k} = \mathsf{L}_{i,\Lambda_i} \oplus \mathsf{L}_{j,\Lambda_j}$.
   - For an AND gate $(i, j, k, \wedge)$, $\mathcal{S}_\mathsf{A}$ samples $\Lambda_k \leftarrow \mathbb{F}_2$ and generates $G_{k,0} \leftarrow \mathbb{F}_{2^\kappa}$, $G_{k,1} \leftarrow \mathbb{F}_{2^\kappa}$. Then $\mathcal{S}_\mathsf{A}$ evaluates L_{k,Λ_k} according to the protocol specification in KRRW and defines $c_k = \mathsf{ExtBit}(\mathsf{L}_{k,\Lambda_k}) \oplus \Lambda_k$.

   Finally, $\mathcal{S}_\mathsf{B}$ sends the simulated $\mathcal{GC}_\mathsf{A}$ to $\mathcal{A}$ and locally defines $\mathsf{M_B}[\Lambda_w]$ for $w \in \mathcal{W}$ using Eval.

4. $\mathcal{S}_\mathsf{B}$ sends $\Lambda_w, \mathsf{L}_{w,\Lambda_w}$ for $w \in \mathcal{I}_\mathsf{A}$ to $\mathcal{A}$.

5. $\mathcal{S}_\mathsf{B}$ simulates the Fix command, extracts the input $y_w = \Lambda'_w \oplus b_w$, and sends it to $\mathcal{F}_\mathsf{2PC}$. Then, $\mathcal{S}_\mathsf{B}$ generates the input messages $m_{w,\Lambda_w} = \mathsf{H}_\mathsf{tcr}(\mathsf{M}_\mathsf{B}[\tilde{cbit}_w], w\|0) \oplus \mathsf{L}_{w,\Lambda_w}$ and $m_{\bar{\Lambda}_w} \leftarrow \mathbb{F}_{2^\kappa}$ and sends them to $\mathcal{A}$.

6. $\mathcal{S}_\mathsf{B}$ simulates the Open command by opening the corrected input mask $\tilde{\Lambda}_w \oplus \Lambda_w$ for $w \in \mathcal{I}_\mathsf{B}$.

7. We don't need to simulate this step since it's non-interactive.

8. $\mathcal{S}_\mathsf{B}$ simulates the protocol Π_GCCheck as follows:
   (a) $\mathcal{S}_\mathsf{B}$ receives the random challenge χ from $\mathcal{A}$
   (b) $\mathcal{S}_\mathsf{B}$ locally computes B_k for each AND gate $(i, j, k, \wedge)$.
   (c) $\mathcal{S}_\mathsf{B}$ samples $G'_k \leftarrow \mathbb{F}_{2^\rho}$ for each AND gate $(i, j, k, \wedge)$ and sends them to P_B. Then it locally evaluates D_k according to the protocol specification.
   (d) $\mathcal{S}_\mathsf{B}$ computes $h_\mathsf{B} = \sum_k \chi_k \cdot B^k \oplus D_k$ and sends it to $\mathcal{A}$.

9. If the previous step does not abort, then $\mathcal{S}_\mathsf{B}$ receives the correct output z_w from $\mathcal{F}_\mathsf{2PC}$ for $w \in \mathcal{O}$. Then $\mathcal{S}_\mathsf{B}$ sends the *corrected* MAC tag $\mathsf{M}[a_w] \oplus (z_w^0 \oplus z_w) \cdot \Delta_\mathsf{B}$ to $\mathcal{A}$, simulating the Open command. Here $\boldsymbol{z}^0$ denotes the output value of $\mathcal{C}(0, \boldsymbol{y})$.

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

Hybrid 2 In this hybrid we generate the masked bits Λ_w for $w \in \mathcal{I}_\mathsf{B}$ as in the simulation above, i.e., by first sampling $\Lambda_w \leftarrow \mathbb{F}_2$ and then opening the *corrected* mask $a_w \oplus \Lambda_w \oplus \Lambda'_w$. Since a_w is uniformly random in the view of $\mathcal{A}$, $\mathbf{Hybrid_1}$ and $\mathbf{Hybrid_2}$ are identically distributed.

Hybrid 3 In this hybrid we make explicit the usage of the honest party's secret Δ_A and mark them in blue. In particular,

- In step 5 $\mathcal{S}_\mathsf{B}$ generates $m_{w,\Lambda_w} = \mathsf{H}_\mathsf{tcr}(\mathsf{M}_\mathsf{B}[\tilde{\Lambda}_w], w\|0) \oplus \mathsf{L}_{w,\Lambda_w}$ and $m_{w,\bar{\Lambda}_w} = \mathsf{H}_\mathsf{tcr}(\mathsf{M}_\mathsf{B}[\tilde{\Lambda}_w] \oplus \Gamma_\mathsf{B}, w\|0) \oplus \Delta_\mathsf{A} \oplus \mathsf{L}_{w,\Lambda_w}$ for $w \in \mathcal{I}_\mathsf{B}$.
- In step 3 of Figure 9 $\mathcal{S}_\mathsf{B}$ generates each gate in the garbled circuit $\mathcal{GC}_\mathsf{A}$ as follows. The XOR gates are garbled as usual, while the AND gates are garbled as $G_{k,0} = \mathsf{H}_\mathsf{ccrnd}(\mathsf{L}_{i,\Lambda_i}, k\|00) \oplus \mathsf{K}[b_j] \oplus \mathsf{H}_\mathsf{ccrnd}(\mathsf{L}_{i,\Lambda_i} \oplus \Delta_\mathsf{A}, k\|00) \oplus a_j \cdot \Delta_\mathsf{A}$ and $G_{k,1} = \mathsf{H}_\mathsf{ccrnd}(\mathsf{L}_{j,\Lambda_j}, k\|01) \oplus \mathsf{K}[b_i] \oplus \mathsf{L}_{i,\Lambda_i} \oplus \mathsf{H}_\mathsf{ccrnd}(\mathsf{L}_{j,\Lambda_j} \oplus \Delta_\mathsf{A}, k\|01) \oplus (a_i \oplus \Lambda_i) \cdot \Delta_\mathsf{A}$ while the output label L_{k,Λ_k} is derived using the Eval algorithm.
- In step 3 of Figure 10 $\mathcal{S}_\mathsf{B}$ computes $G'_k = \mathsf{H}'_\mathsf{ccrnd}(\mathsf{L}_{k,\Lambda_k}, k\|2) \oplus A_{k,1} \oplus \mathsf{H}'_\mathsf{ccrnd}(\mathsf{L}_{k,\Lambda_k} \oplus \Delta_\mathsf{A}, k\|2)$.

Both changes all make no difference to the view of the adversary since we just re-write the hash function inputs. Therefore, $\mathbf{Hybrid_2}$ and $\mathbf{Hybrid_3}$ are identically distributed.

Hybrid 4 In this hybrid we replace the first blue term in $\mathbf{Hybrid_3}$ with uniformly random values. Due to the tweakable correlation robust property of H_tcr, the two hybrids are ϵ_tcr-indistinguishable.

Hybrid 5 In this hybrid we replace the second and the third blue values in $\mathbf{Hybrid_3}$ with uniformly random values. Due to the circular correlation robust under naturally derived keys property of $\mathsf{H}_\mathsf{ccrnd}$, the two hybrids are ϵ_ccrnd-indistinguishable.

Hybrid 6 In this hybrid we change the input of $\mathsf{P_A}$ from $\boldsymbol{x}$ to all zeros. Since in step 4 $\mathsf{P_A}$'s input is completely masked the view of $\mathcal{A}$ is not changed. As for the honest party's output, due to Lemma 7 the probabilities that $\exists w \in \mathcal{W}, e_w = 0$ under any pair of input values differ changes at most with probability $2^{-\rho}$. Therefore, $\mathbf{Hybrid_4}$ and $\mathbf{Hybrid_5}$ are $2^{-\rho}$-indistinguishable. This is exactly the ideal distribution.

Therefore, the ideal world and real world executions are $\epsilon_{\mathsf{tcr}} + \epsilon_{\mathsf{ccrnd}} + 2^{-\rho}$-indistinguishable in the corrupted $\mathsf{P_A}$ case.

Altogether, the ideal world and real world executions are indistinguishable in the corrupted $\mathsf{P_B}$ case. This implies that the protocol $\Pi_{\mathsf{2PC\text{-}2way}}$ shown in Figure 9 securely realizes $\mathcal{F}_{\mathsf{2PC}}$ against malicious adversary in the $\mathcal{F}_{\mathsf{cpre}}, \mathcal{F}_{\mathsf{COT}}$-hybrid model. □

E Construction of Distributed Garbling Schemes

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

E.1 KRRW Distributed Garbling

We recall the distributed half-gates garbling scheme by Katz et al. [32]. Let $\mathsf{H} : \{0,1\}^\kappa \times \{0,1\}^\kappa \to \{0,1\}^\kappa$ be a hash function 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

  $$G_{k,0}^{(\mathsf{A})} := \mathsf{H}(\mathsf{L}_{i,0}, k\|0) \oplus \mathsf{H}(\mathsf{L}_{i,1}, k\|0) \oplus a_j\Delta_{\mathsf{A}} \oplus \mathsf{K_A}[b_j] \ ,$$
  $$G_{k,1}^{(\mathsf{A})} := \mathsf{H}(\mathsf{L}_{j,0}, k\|1) \oplus \mathsf{H}(\mathsf{L}_{j,1}, k\|1) \oplus a_i\Delta_{\mathsf{A}} \oplus \mathsf{K_A}[b_i] \oplus \mathsf{L}_{i,0} \ ,$$
  $$\mathsf{L}_{k,0} := \mathsf{H}(\mathsf{L}_{i,0}, k\|0) \oplus \mathsf{H}(\mathsf{L}_{j,0}, k\|1) \oplus (a_k \oplus \hat{a}_k)\Delta_{\mathsf{A}} \oplus \mathsf{K_A}[b_k] \oplus \mathsf{K_A}[\hat{b}_k] \ .$$

  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

  $$G_{k,0}^{(\mathsf{B})} := \mathsf{M_B}[b_j] \ ,$$
  $$G_{k,1}^{(\mathsf{B})} := \mathsf{M_B}[b_i] \ .$$

  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

$$
\begin{aligned}
G_{w,0} &:= G_{w,0}^{(\mathsf{A})} \oplus G_{w,0}^{(\mathsf{B})} \\
G_{w,1} &:= G_{w,1}^{(\mathsf{A})} \oplus G_{w,1}^{(\mathsf{B})} \oplus \mathsf{L}_{i,\Lambda_w} \\
\mathsf{L}_{k,\Lambda_k} &:= \mathsf{H}(\mathsf{L}_{i,\Lambda_i}, k\|0) \oplus \mathsf{H}(\mathsf{L}_{j,\Lambda_j}, k\|1) \oplus \mathsf{M}_{\mathsf{B}}[b_k] \oplus \mathsf{M}_{\mathsf{B}}[\hat{b}_k] \\
&\quad \oplus \Lambda_i(G_{k,0} \oplus \mathsf{M}_{\mathsf{B}}[b_j]) \oplus \Lambda_j(G_{k,1} \oplus \mathsf{M}_{\mathsf{B}}[b_i] \oplus \mathsf{L}_{i,\Lambda_i}) \ ,
\end{aligned}
$$

and the public value $\Lambda_k := \mathsf{ExtBit}(\mathsf{L}_{k,\Lambda_k}) \oplus c_k$.

3. Output $\{\, (\Lambda_w, \mathsf{L}_{w,\Lambda_w}) \,\}_{w \in \mathcal{W} \cup \mathcal{O}}$.

E.2 WRK Distributed Garbling with Optimization

We recall the optimized WRK distributed garbling scheme by Dittmer et al. [18]. Let $\mathsf{H} : \{0,1\}^\kappa \times \{0,1\}^\kappa \to \{0,1\}^\kappa$ and $\mathsf{H}' : \{0,1\}^\kappa \times \{0,1\}^\kappa \to \{0,1\}^\rho$ be two hash functions, 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

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

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. $\mathsf{P_B}$ 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 $(i, j, k, \wedge)$ $\mathsf{P_B}$ first recovers the garbled table as:

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

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

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

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