Technical Report: Secure End-to-End Publicly Auditable 2PC

Khai Hanh Tang[1] and Chan Nam Ngo[2]

[1] khaihanh.tang@ntu.edu.sg, Nanyang Technological University, Singapore
[2] namncc@pse.dev, Privacy Scaling Exploration

August 30, 2024

1 Introduction

Secure Multiparty Computation (MPC) allows n parties $P_1, \ldots, P_n$ to securely evaluate a joint function $f(x_1, \ldots, x_n)$ while keeping the secret input x_i private to its owner P_i. MPC comes with two main flavors of security [18], namely *Semi-honest* and *Active Security*:

Semi-Honest (SEM) only guarantees security if the adversary acts according to the protocol, i.e. one will not deviate from the protocol but only tries to break privacy from the MPC transcript that is available to it.

Active Security (ACT) guarantees the security of the MPC protocol even if the adversary acts arbitrarily. This is a stronger security setting.

Generic MPC is enabled through two approaches: Secret-Sharing (SS) and Garbled-Circuit [18] (GC). Our focus is on the latter due to its attractive property of constant round complexity.[3]

Efficiency of Actively Secure Garbled Circuit. ACT for GC is trivially available through the usage of Zero-Knowledge-Proof [22] yet even with optimizations it is necessary to prove a statement with 19 clauses conjunction per gate. Cut-and-Choose [18] is another approach for ACT in which the Garbler prepares k circuits in total and the Evaluator can choose t to test and only accepts if all t circuits are valid. Yet the preparation of many circuits and the testing for validity yield concrete inefficiency and become unreasonable when the circuit size is large. Another approach is through Authenticated Garbling [28] which will be our focus due to its efficiency compared with the previous two.[4]

Yet, these classical flavors of security only take into consideration the parties that participate in the MPC protocol itself, i.e. only $P_1, \ldots, P_n$ can verify the security of the MPC protocol.

[3] Round complexity is an important factor for blockchain model due to bottleneck in block generation time.

[4] We defer the efficiency analysis of such approaches in blockchain model to later of this Section. The concrete efficiency of this approach will be detailed in the Preliminary Section.

Verifiable MPC. We consider a somewhat "added-on" setting, in which, an external party, let us call it Verifier, can verify the security of the MPC protocol, and we call protocols in this setting Verifiable MPC (V-MPC), the security verification can come in 4 flavors:

Designated Verifier (DV-MPC): A single Verifier who potentially has some private state can verify the protocol execution to be secure.

Collaborative Verifiable (CV-MPC) A set of Verifiers who potentially have some private states can jointly (according to some access structure) verify the security of the protocol.[5]

Multi Verifier (MV-MPC) A set of Verifiers who potentially have some private states can individually verify the protocol's security.[6]

Publicly Verifiable (PV-MPC) Any external Verifier without any secret state can verify the security of the protocol (possibly with aid from some CRS).

We focus on the last setting which is also the strongest one, i.e. Publicly Verifiable MPC.

Verifiability vs Auditability. Without regard to the verifiability flavor, to make an MPC verifiable, there are two different approaches:

Verifiable Output (VO) The output of the MPC is made verifiable.

Auditable Transcript (AT) The transcript of the MPC is made verifiable.

We distinguish that the difference is that the VO flavor is independent of the MPC protocol while the VT one is dependent on the MPC protocol itself.

> **Example.**[*Verifiability vs Auditability*:] Let us consider the following Shuffle functionality.
>
> The Shuffle functionality runs between n parties $P_1, \ldots, P_n$ each with a secret input $x_1, \ldots, x_n$ starting as an ordered list of $O_0 = \{x_1, \ldots, x_n\}$. The functionality runs in n rounds, in each round R_i, party P_i sends a permutation π_i to Shuffle in which the functionality will apply to obtain $O_i = \pi_i(O_{i-1})$. Finally, the functionality outputs O_n.
>
> **Verifiability** VO only asks for the quality of the output, i.e. that O_n is a permutation of O_0 while **Auditability** VT asks for the correctness of the process, i.e. that O_n is a permutation of O_0 through a series of permutation $\pi_1, \ldots, \pi_n$.

Verifiable MPC. For VO-MPC, we can leverage Collaborative Zero-Knowledge Proof (co-ZKP, between $P_1, \ldots, P_n$) to produce a ZK proof that the output $y = f(x_1, \ldots, x_n)$ while keeping the $x_1, \ldots, x_n$ private to $P_1, \ldots, P_n$. In the literature, co-ZKP can be achieved by thresholdizing any existing ZKP, demonstrated by Collaborative Zero-Knowledge Succinct Argument of Knowledge (co-zkSNARKs) [25]. Yet, there are potential Co-ZKP based on other ZKP such as: MPC-In-The-Head (MPCitH: seminal work IKOS07 [21], Other MPCitH techniques [13,23,10,27,19,12,1,2,14]), VOLE-IZK (DVZK with fast prover time and small memory: [29,17,9,32,30,5,16,31,6]), or VOLE-In-The-Head [7].[7]

[5] CV-MPC can be considered as a thresholdized version of DV-MPC.

[6] Such verification flavor is also called Crowd-Verifiable MPC [4].

[7] VOLE-In-The-Head is the public coin Verifier version of VOLE-IZK

DV-MPC can leverage any private coin Verifier (Co-) ZKP, CV-MPC is achievable through thresholdizing the Verifier state in the private coin Verifier (Co-) ZKP, MV-MPC can be obtained by running multiple DV-MPC instances, and PV-MP can utilize any public coin verifier ZKP.

Fortunately, all existing Co-ZKP is constant rounds protocols, thus the round complexity of the VO-MPC only depends on the underlying MPC protocol, i.e. a Secret-Sharing-based [18] (SS) VO-MPC will yield linear round complexity while a Garbled-Circuit-based [18] (GC) VO-MPC will have constant-round complexity.

Auditable Transcript MPC. For AT-MPC, we have to make the transcript auditable while keeping the privacy of the protocol execution. In general, for both SS-based and GC-based AT-MPC, the transcript can be made auditable through Additively-Homomorphic Commitment (COM, such as Pedersen, [8]) or Verifiable MAC (V-MAC, [28]) on each computation step. If the COM or V-MAC in the AT-MPC is public verifiable [7] then we have PV-MPC. COM-based has worse concrete efficiency than V-MAC-based AT-MPC due to the first operating on groups while the latter only operates on rings.

One can consider ACT a special case of DV/MV AT-MPC where the Verifier is the party in the MPC protocol. This means that if the ACT mechanism can be publicly verifiable, we obtain PV-MPC as well.

As in VO-MPC, an SS-based AT-MPC will yield linear round complexity while a GC-based VT-MPC can potentially yield a constant round complexity. This last one is also our main research question:

> **Research Question.** *Can we pick a GC-based protocol that is actively secure (such as WRK17 [23], KRRW18 [24], DILO22 [16], or CWYY23 [15]) and make such active security public verifiable and obtain constant round publicly auditable MPC?*

We answer the aforementioned question with definitive. We picked the basic blueprint of the actively secure Garbled Circuit protocol, namely WRK17 [23], which relies on V-MAC, we then adopted VOLEitH [7] (with SoftSpokenOT [26]) to make the utilized V-MAC public auditable. We obtained a Constant Round Publicly Auditable Garbled Circuit Protocol with inherited security and efficiency from both WRK17 and VOLEitH (minimal overheads).[8]

1.1 Blockchain-Public-Auditable-MPC

Boostrapping GC with Blockchain. Our protocol only requires the blockchain as a public bulletin board, i.e. for commitments of inputs before the computation and such commitments become a part of the statement for the publicly verifiable zero-knowledge proof, as in other off-chain protocols.

[8] We disregard the improvements in KRRW18 [24], DILO22 [16], or CWYY23 [15], due to non-trivial composition of such improvements and VOLEitH, for example they may require Learning-Parity-With-Noise assumption while SoftSpokenOT is incompatible with LPN.

Computation of GC on Blockchain. Blockchain applications such as decentralized applications built on Ethereum are sensitive to round complexity because each interaction round latency is capped by the blockchain's consensus mechanism, i.e. block generation time. Our protocol which is a constant round protocol is particularly suitable for building such applications. Furthermore, while the MPC protocol is run among a set of participants, to have a meaningful impact on the blockchain, the result has to be made verifiable to the set of blockchain verifiers that are public nodes. Our protocol provides an auditable transcript which is an even stronger property. There is only one caveat that is the communication cost is linear to the size of the circuit. In Ethereum, it was a problem in the past due to calldata cost but, with data blobs made available, such an issue is elevated. Furthermore, the concrete computational complexity of our (public) verification protocol is low due to we only require calls to PRF (hashes) and operations on rings.

Exploiting the characteristics of the blockchain for blockchain-MPC: Together with the bootstrapping, the necessary computation in MPC, such as data sanitization check, e.g. the input should be positive, can be replaced with ZKP leveraging the pre-committed data, i.e. the inputs into the MPC can be committed along with a proof of quality, as such, there is no need for checking inside the MPC hence the circuit in MPC can focus only on the application logic.[9]

Security in blockchain model: Blockchain is utilized as a public bulletin board outside of the protocol hence we only analyze the security of our protocol in its own hybrid model.

2 Preliminaries

2.1 Authenticated Bits

Assume that P_{A} holds a secret bit b. Then, in an interactive protocol between P_{A} and P_{B}, b is authenticated if P_{A} holds $\mathsf{M}[b]$ while P_{B} holds $\mathsf{K}[b]$ and a global value Δ_{B} satisfying

$$\mathsf{M}[b] = \mathsf{K}[b] \oplus b \cdot \Delta_{\mathsf{B}}.$$

If b, kept by P_{A} (respectively, P_{B}), is authenticated with respect to Δ_{B} (respectively, Δ_{A}), we write $[\![b; \Delta_{\mathsf{B}}]\!]_{\mathsf{A}}$ (respectively, $[\![b; \Delta_{\mathsf{A}}]\!]_{\mathsf{B}}$).

Protocol Π_{fix}. We recall from [9] the protocol Π_{fix} in the following Figure 1.

2.2 Publicly Verifiable VOLEs via SoftSpokenOT [26]

SoftSpokenOT proposed by [26] allows two parties to generate an authenticated bit, e.g., $[b]_{\mathsf{A}}$, in a way that each party only locally interacts with the random oracle (RO). This hence leads to publicly verifiable zero-knowledge proofs [7,14,11] which recently attracted a lot of attention from the research community. We

[9] Such an approach is similar to sanitizing SPDZ shares using ZKP, `https://github.com/Yoii-Inc/zk-mpc`.

Inputs: b and $[\![a; \Delta_{p'}]\!]_p$. Here $[\![a; \Delta_{p'}]\!]_p$ is parsed as $\mathsf{M}[a] = \mathsf{K}[a] \oplus a \cdot \Delta_{p'}$, for some $p \in \{\mathsf{A}, \mathsf{B}\}$, $p' = \mathsf{A}$ if $p = \mathsf{B}$, otherwise, $p' = \mathsf{B}$, and $a \in \{0, 1\}$. P_p keeps a and $\mathsf{M}[a]$ while $P_{p'}$ keeps $\mathsf{K}[a]$ and $\Delta_{p'}$.
Output: $[\![b; \Delta_{p'}]\!]_p$ as $\mathsf{M}[b] = \mathsf{K}[b] \oplus b \cdot \Delta_{p'}$ where $n \in \{0, 1\}$.
Execution:
1. Party P_p sends $d := a \oplus b$ to the remaining party $P_{p'}$.
2. P_p locally sets $\mathsf{M}[b] := \mathsf{M}[a]$.
3. $P_{p'}$ locally sets $\mathsf{K}[b] := \mathsf{K}[a] \oplus d \cdot \Delta_{p'}$.

Fig. 1. Sub-Protocol Π_{fix}.

recall the functionality $\mathcal{F}_{\mathsf{sVOLE}}^{\mathcal{P},\mathcal{V},\tau,N}$ and the technique from SoftSpokenOT for realizing $\mathcal{F}_{\mathsf{sVOLE}}^{\mathcal{P},\mathcal{V},\tau,N}$ for binary computations, over $\mathbb{F}_{2^k}$ for some $k \in \mathbb{N}$, as follows.

Functionality $\mathcal{F}_{\mathsf{sVOLE}}^{\mathcal{P},\mathcal{V},\tau,N}$. We recall the subspace VOLE functionality $\mathcal{F}_{\mathsf{sVOLE}}^{\mathcal{P},\mathcal{V},\tau,N}$ [7], adapted from [14], in the following Figure 2.

This functionality is parameterized by $\mathbb{F}_2$ and its extension $\mathbb{F}_{2^\tau}$ for some parameter $\tau \in \mathbb{N}$. We denote by κ the repetition parameter and N the number of random sVOLE correlations.
1. Upon receiving (init) from $\mathcal{P}$ and $\mathcal{V}$, it works as follows:
   (a) Sample $\Delta \xleftarrow{\$} \mathbb{F}_{2^\tau}$. Then, sample $\mathbf{u} \xleftarrow{\$} \mathbb{F}_2^N$ and $\mathbf{v} \xleftarrow{\$} \mathbb{F}_{2^\tau}^N$ and compute $\mathbf{w} := \mathbf{v} \oplus \mathbf{u} \cdot \Delta$.
   (b) If $\mathcal{P}$ is corrupted, receive $\mathbf{u}, \mathbf{v}$ from $\mathcal{P}$ and compute $\mathbf{w} := \mathbf{v} \oplus \mathbf{u} \cdot \Delta$.
   (c) If $\mathcal{V}$ is corrupted, receive $\Delta, \mathbf{w}$ from $\mathcal{V}$ and compute $\mathbf{v} := \mathbf{w} \oplus \mathbf{u} \cdot \Delta$.
   (d) Send $(\mathbf{u}, \mathbf{v})$ to $\mathcal{P}$.
2. Upon receiving (get) from $\mathcal{P}$ and $\mathcal{V}$, it sends $(\Delta, \mathbf{w})$ to both parties.

Fig. 2. The subspace VOLE functionality $\mathcal{F}_{\mathsf{sVOLE}}^{\mathcal{P},\mathcal{V},\tau,N}$.

SoftSpokenOT. Let $\tau \in \mathbb{N}$. Let $t_x \in \{0, 1\}$ for all $x \in \mathbb{F}_{2^\tau}$. SoftSpokenOT works by the observation that

$$\bigoplus_{x \in \mathbb{F}_{2^\tau} \setminus \{\Delta_{\mathsf{B}}\}} t_x \cdot (\Delta_{\mathsf{B}} \oplus x) = \bigoplus_{x \in \mathbb{F}_{2^\tau}} t_x \cdot (\Delta_{\mathsf{B}} \oplus x) = \bigoplus_{x \in \mathbb{F}_{2^\tau}} t_x \cdot x \oplus \Delta_{\mathsf{B}} \cdot \bigoplus_{x \in \mathbb{F}_{2^\tau}} t_x.$$

By the above observation, we can set $b = \bigoplus_{x \in \mathbb{F}^{2^\tau}} t_x$, $\mathsf{M}[b] = \bigoplus_{x \in \mathbb{F}_{2^\tau}} t_x \cdot x$ and $\mathsf{K}[b] = \bigoplus_{x \in \mathbb{F}_{2^\tau} \setminus \{\Delta_{\mathsf{B}}\}} t_x \cdot (\Delta_{\mathsf{B}} \oplus x)$. We have the following discussions.

- For $x \in \mathbb{F}_{2^\tau} \setminus \{0\}$, t_x can be generated randomly while t_0 is uniquely determined by $t_0 := b \oplus \bigoplus_{x \in \mathbb{F}_{2^\tau} \setminus \{0\}} t_x$.
- The computation of $\mathsf{M}[b]$ does not involve Δ_{B}.
- Party P_{B}, when computing $\mathsf{K}[b]$, requires the knowledge of $2^\tau - 1$ (out of 2^τ) elements in $\{t_x\}_{x \in \mathbb{F}_{2^\tau}}$. The only element excluded is $t_{\Delta_{\mathsf{B}}}$. Moreover, P_{B} is not allowed to know $t_{\Delta_{\mathsf{B}}}$. Otherwise, P_{B} can recover b which is a secret of P_{A}. Therefore, P_{A} sends to P_{B} a vector commitment vc that commits to $\{t_x\}_{x \in \mathbb{F}_{2^\tau}}$. P_{B} then obtains Δ_{B} from RO with input containing vc. Once Δ_{B} is determined, P_{A} opens $2^\tau - 1$ (out of 2^τ) positions in vc with respect to $\{t_x\}_{x \in \mathbb{F}_{2^\tau} \setminus \{\Delta_{\mathsf{B}}\}}$. Hence, P_{B} now has sufficient information to determine $\mathsf{K}[b]$.

3 Technical Overview

We first recall the authenticated garbling WRK protocol [28] in Section 3.1. Then, we discuss our technique for making WRK protocol publicly auditable in Section 3.2.

3.1 Authenticated Garbling from WRK Protocol [28]

We recall the authenticated garbling technique from WRK protocol [28] (recalled in [24]) without clearly discussing it in detail. Consider a circuit containing AND ($\cdot$) and XOR ($\oplus$) gates. We denote each gate by the tuple $(\alpha, \beta, \gamma, \mathsf{op})$ where $\mathsf{op} \in \{\cdot, \oplus\}$ is the operation, and α, β, and γ are the indices of the left, right, and output wires, respectively. As remarked in [28,24], XOR gates can be handled for free. Regarding AND gates, consider the tuple $(\alpha, \beta, \gamma, \cdot)$. Let P_{A} and P_{B} respectively play the roles of garbler and authenticator. We first recall a few notations before discussing the garbled circuits.

The value at each each wire w is denoted by $z_w \in \{0,1\}$. To mask z_w, we use a mask $\lambda_w \in \{0,1\}$ and computed the masked value $\hat{z}_w = z_w \oplus \lambda_w$. In [28], at each wire w, λ_w is split into shared random masks r_w and s_w held by P_{A} and P_{B}, respectively, such that $\lambda_w = r_w \oplus s_w$. Hence, $\hat{z}_w = z_w \oplus \lambda_w$.

The input wires to the circuit are split into two separate sets $\mathcal{I}_{\mathsf{A}}$ and $\mathcal{I}_{\mathsf{B}}$ such that P_{A} (respectively, P_{B}) knows each z_w for $w \in \mathcal{I}_{\mathsf{A}}$ (respective, $w \in \mathcal{I}_{\mathsf{B}}$). Therefore, in the beginning (at the phase **Input Preprocessing**), P_{B} (respectively, P_{A}) must send s_w (respectively, r_w) to P_{A} (respectively, P_{B}) for computing $\hat{z}_w$ for $w \in \mathcal{I}_{\mathsf{A}}$ (respectively, $w \in \mathcal{I}_{\mathsf{B}}$). P_{A} and P_{B} then broadcast all $\hat{z}_w$ for $w \in \mathcal{I}_{\mathsf{A}} \cup \mathcal{I}_{\mathsf{B}}$. Then, the evaluation of the circuit, following a topological ordering, is hence considered in two types of gates:

- For each XOR gate $(\alpha, \beta, \gamma, \oplus)$, the computation is free. Having $\hat{z}_\alpha$ and $\hat{z}_\beta$, each party (P_{A} and P_{B}) can locally compute $\hat{z}_\gamma := \hat{z}_\alpha \oplus \hat{z}_\beta = (z_\alpha \oplus z_\beta) \oplus (\lambda_\alpha \oplus \lambda_\beta)$. This is hence understood that $z_\gamma = z_\alpha \oplus z_\beta$ and $\lambda_\gamma = \lambda_\alpha \oplus \lambda_\beta$.
- For each AND gate $(\alpha, \beta, \gamma, \cdot)$, the computation is not that simple. In fact, having $\hat{z}_\alpha = z_\alpha \oplus \lambda_\alpha$ and $\hat{z}_\beta = z_\beta \oplus \lambda_\beta$, the parties need to obtain $\hat{z}_\gamma = z_\alpha \cdot z_\beta \oplus \lambda_\gamma$ where $\lambda_\gamma = r_\gamma \oplus s_\gamma$ with r_γ and s_γ independently held by P_{A} and P_{B}, respectively. To do this, we first observe that

$$\begin{aligned}\hat{z}_\alpha \cdot \hat{z}_\beta &= (z_\alpha \oplus \lambda_\alpha) \cdot (z_\beta \oplus \lambda_\beta) \\ &= z_\alpha \cdot z_\beta \oplus \lambda_\alpha \cdot z_\beta \oplus \lambda_\beta \cdot z_\alpha \oplus \lambda_\alpha \cdot \lambda_\beta \\ &= z_\alpha \cdot z_\beta \oplus \lambda_\alpha \cdot \underbrace{(z_\beta \oplus \lambda_\beta)}_{\hat{z}_\beta} \oplus \lambda_\beta \cdot \underbrace{(z_\alpha \oplus \lambda_\alpha)}_{\hat{z}_\alpha} \oplus \lambda_\alpha \cdot \lambda_\beta.\end{aligned}$$

  Hence, $\lambda_\alpha \cdot \lambda_\beta \oplus \lambda_\gamma \oplus \lambda_\alpha \cdot \hat{z}_\beta \oplus \lambda_\beta \cdot \hat{z}_\alpha \oplus \hat{z}_\alpha \cdot \hat{z}_\beta = z_\alpha \cdot z_\beta \oplus \lambda_\gamma$. Notice that, $\lambda_\alpha \cdot \hat{z}_\beta = r_\alpha \cdot \hat{z}_\beta \oplus s_\alpha \cdot \hat{z}_\beta$, $\lambda_\beta \cdot \hat{z}_\alpha = r_\beta \cdot \hat{z}_\alpha \oplus s_\beta \cdot \hat{z}_\alpha$, and $\lambda_\gamma = r_\gamma \oplus s_\gamma$ such that P_{A} can computes $r_\alpha \cdot \hat{z}_\beta$ and $r_\beta \cdot \hat{z}_\alpha$ while P_{B} can compute $s_\alpha \cdot \hat{z}_\beta$ and $s_\beta \cdot \hat{z}_\alpha$. However, regarding $\lambda_\alpha \cdot \lambda_\beta$, [28] suggests to use additional shared random masks r'_γ and s'_γ respectively held by P_{A} and P_{B} such that $r'_\gamma \oplus s'_\gamma =$

$\lambda_\alpha \cdot \lambda_\beta$. Then, P_A can construct the garbled table corresponding to the four possibilities of $(\hat{z}_\alpha, \hat{z}_\beta)$, i.e., by encrypting to the following four rows where the k-th row is

$$r_{\gamma,k} = r'_\gamma \oplus r_\gamma \oplus r_\alpha \cdot k_1 \oplus r_\beta \cdot k_0 \tag{1}$$

for $k \in [0,3]$ with $(k_0, k_1) = \mathsf{bin}(k)$ (i.e., $k = k_0 + 2k_1$ and $k_0, k_1 \in \{0,1\}$). Here, the k-th row assumes that $(\hat{z}_\alpha, \hat{z}_\beta) = (k_0, k_1) = \mathsf{bin}(k)$. Regarding P_B, if $k \in [0,3]$, i.e., $(\hat{z}_\alpha, \hat{z}_\beta) = (k_0, k_1) = k$, is the selected row, then P_B can compute

$$s_{\gamma,k} = s'_\gamma \oplus s_\gamma \oplus s_\alpha \cdot k_1 \oplus r_\beta \cdot k_0 \oplus k_0 \cdot k_1 \tag{2}$$

and decrypt the corresponding row to obtain $r'_\gamma \oplus r_\gamma \oplus r_\alpha \cdot k_1 \oplus r_\beta \cdot k_0$. Taking sum achieves $\hat{z}_\gamma = z_\alpha \cdot z_\beta \oplus \lambda_\gamma$ as desired.

We also note that encrypting requires the associated labels $\mathsf{L}_{w,0}$ and $\mathsf{L}_{w,1}$ corresponding to the assumptions $\hat{z}_w = 0$ and $\hat{z}_w = 1$. To be compatible with free XORs (i.e., $\hat{z}_\gamma = \hat{z}_\alpha \oplus \hat{z}_\beta$ for any XOR gate $(\alpha, \beta, \gamma, \oplus)$), [28] enforces $\mathsf{L}_{w,0} \oplus \Delta_\mathsf{A} = \mathsf{L}_{w,1}$ where Δ_A is only known by P_A. This means that, for each wire w, P_B can achieve either $\mathsf{L}_{w,0}$ or $\mathsf{L}_{w,1}$ and cannot achieve both. Hence, for each XOR gate $(\alpha, \beta, \gamma, \oplus)$, by enforcing $\mathsf{L}_{\gamma,0} = \mathsf{L}_{\alpha,0} \oplus \mathsf{L}_{\beta,0}$, we can see that

$$\begin{aligned}
\mathsf{L}_{\gamma,\hat{z}_\gamma} &= \mathsf{L}_{\gamma,0} \oplus \hat{z}_\gamma \cdot \Delta_\mathsf{A} \\
&= \mathsf{L}_{\alpha,0} \oplus \mathsf{L}_{\beta,0} \oplus (\hat{z}_\alpha \oplus \hat{z}_\beta) \cdot \Delta_\mathsf{A} \\
&= (\mathsf{L}_{\alpha,0} \oplus \hat{z}_\alpha \cdot \Delta_\mathsf{A}) \oplus (\mathsf{L}_{\beta,0} \oplus \hat{z}_\beta \cdot \Delta_\mathsf{A}) \\
&= \mathsf{L}_{\alpha,\hat{z}_\alpha} \oplus \mathsf{L}_{\beta,\hat{z}_\beta}.
\end{aligned}$$

However, for each AND gate $(\alpha, \beta, \gamma, \cdot)$, we cannot have $\mathsf{L}_{\gamma,0}$ related to $\mathsf{L}_{\alpha,0}$ and $\mathsf{L}_{\alpha,1}$. Hence, $\mathsf{L}_{\gamma,0}$ is hence independently sampled by P_A. We also would like to maintain that $\mathsf{L}_{\gamma,\hat{z}_\gamma} = \mathsf{L}_{\gamma,0} \oplus \hat{z}_\gamma \cdot \Delta_\mathsf{A}$. At this point, it seems impossible to P_B to obtain $\mathsf{L}_{\gamma,\hat{z}_\gamma}$ without the knowledge of Δ_A. We will discuss how to deal with this after discussing authenticating shared random masks below.

Authenticating Shared Random Masks. As noted by [28], this design's privacy is guaranteed against malicious P_A. Nevertheless, P_A can violate the correctness of the garbled table by changing the orders of rows in garbled tables or encrypting incorrect rows. To cope with this issue, [28] enforces every shared random mask from each party to be authenticated by using subfield VOLE. In particular, P_A and P_B keep Δ_A and Δ_B, respectively, such that, for each r_w and s_w held by P_A and P_B, respectively, it should hold that

$$\mathsf{M}[r_w] = \mathsf{K}[r_w] \oplus r_w \cdot \Delta_\mathsf{B} \text{ and } \mathsf{M}[s_w] = \mathsf{K}[s_w] \oplus s_w \cdot \Delta_\mathsf{A}$$

where $(r_w, \mathsf{M}[r_w], \mathsf{K}[s_w])$ is held by P_A while $(s_w, \mathsf{M}[s_w], \mathsf{K}[r_w])$ is held by P_B. By applying protocol Π_{fix} in Figure 1, for each garbled table corresponding to each AND gate $(\alpha, \beta, \gamma, \cdot)$, for $k \in [0,3]$, P_A and P_B can respectively achieve

$$(r_{\gamma,k}, \mathsf{M}[r_{\gamma,k}], \mathsf{K}[s_{\gamma,k}]) \text{ and } (s_{\gamma,k}, \mathsf{M}[s_{\gamma,k}], \mathsf{K}[r_{\gamma,k}]).$$

With the above employment of subfield VOLE to authenticate shared random masks, we are guaranteed that a malicious P_{A} cannot take advantage in causing the protocol to deviate from its proper purpose. We now turn back to discussing how to make P_{B} obtain $\mathsf{L}_{\gamma,\hat{z}_\gamma} = \mathsf{L}_{\gamma,0} \oplus \hat{z}_\gamma \cdot \Delta_{\mathsf{A}}$ by leveraging the authenticated shared random masks. Hence, in each row of the garbled table for each AND gate $(\alpha, \beta, \gamma, \cdot)$, P_{A} additionally encrypts $\mathsf{M}[r_{\gamma,k}]$ for P_{B} to subsequently verify the authenticity of the decrypted $r_{\gamma,k}$.

Computing Labels from Authenticated Shared Random Masks. Since $\mathsf{L}_{\gamma,\hat{z}_\gamma} = \mathsf{L}_{\gamma,0} \oplus \hat{z}_\gamma \cdot \Delta_{\mathsf{A}}$, P_{A} holds $r_{\gamma,k}$ and $\mathsf{K}[s_{\gamma,k}]$ while P_{B} holds $\mathsf{M}[s_{\gamma,k}]$, P_{A} can compute

$$\mathsf{L}_{\gamma,0} \oplus r_{\gamma,k} \cdot \Delta_{\mathsf{A}} \oplus \mathsf{K}[s_{\gamma,k}].$$

Since $\mathsf{M}[s_{\gamma,k}] = \mathsf{K}[s_{\gamma,k}] \oplus s_{\gamma,k} \cdot \Delta_{\mathsf{A}}$, this implies that

$$\begin{aligned} \underbrace{\mathsf{L}_{\gamma,0} \oplus r_{\gamma,k} \cdot \Delta_{\mathsf{A}} \oplus \mathsf{K}[s_{\gamma,k}]}_{\text{computed by } P_{\mathsf{A}} \text{ above}} \oplus \mathsf{M}[s_{\gamma,k}] &= \mathsf{L}_{\gamma,0} \oplus r_{\gamma,k} \cdot \Delta_{\mathsf{A}} \oplus s_{\gamma,k} \cdot \Delta_{\mathsf{A}} \\ &= \mathsf{L}_{\gamma,0} \oplus \hat{z}_{\gamma,k} \cdot \Delta_{\mathsf{A}}. \end{aligned} \tag{3}$$

Hence, in each row of the garbled table for each AND gate $(\alpha, \beta, \gamma, \cdot)$, P_{A} additionally encrypts $\mathsf{L}_{\gamma,0} \oplus r_{\gamma,k} \cdot \Delta_{\mathsf{A}} \oplus \mathsf{K}[s_{\gamma,k}]$ for P_{B} to subsequently compute $\mathsf{L}_{\gamma,0} \oplus \hat{z}_{\gamma,k} \cdot \Delta_{\mathsf{A}}$ according to (3). Notice that P_{B} does not know $\mathsf{L}_{\gamma,0}$, Δ_{A}, and $\mathsf{K}[s_{\gamma,k}]$. Therefore, even P_{B} can decrypt to obtain $\mathsf{L}_{\gamma,\hat{z}_\gamma}$, P_{B} only knows either $\mathsf{L}_{\gamma,0}$ or $\mathsf{L}_{\gamma,1}$ (depending on the value of $\hat{z}_\gamma$) and cannot know both. Knowing both helps P_{B} to decrypt all four rows of the garbled table which is strictly prohibited since it compromises the privacy of P_{A}.

Putting All Together. With the above discussions, for each garbled table corresponding to each AND gate $(\alpha, \beta, \gamma, \cdot)$, each encrypted row must include $r_{\gamma,k}$, $\mathsf{M}[r_{\gamma,k}]$, and $\mathsf{L}_{\gamma,0} \oplus r_{\gamma,k} \cdot \Delta_{\mathsf{A}} \oplus \mathsf{K}[s_{\gamma,k}]$. The discussion is realized in Figure 4, namely, WRK protocol. This protocol employs functionality $\mathcal{F}_{\mathsf{pre}}$ in Figure 3 as a subroutine for generating authenticated shared random masks and obtaining shared authenticated AND triples, i.e., authenticated r_1, r_2, r_3 (held by P_{A}) and s_1, s_2, s_3 (held by P_{A}) satisfying

$$(r_1 \oplus s_1) \cdot (r_2 \oplus s_2) = r_3 \oplus s_3.$$

This functionality is described as follows:

1. Assuming no $\Delta_{\mathsf{A}}, \Delta_{\mathsf{B}}$ stored, upon receiving Δ_{A} from P_{A} and init from P_{B}, sample $\Delta_{\mathsf{B}} \xleftarrow{\$} \{0,1\}^\rho$, where ρ is the statistical security parameter, and store Δ_{A} and Δ_{B}. Then, send Δ_{B} to P_{B}.
2. Upon receiving $(\mathsf{random}, r, \mathsf{M}[r], \mathsf{K}[s])$ from P_{A} and random from P_{B}, sample $s \xleftarrow{\$} \{0,1\}$, and compute $\mathsf{M}[s] := \mathsf{K}[s] \oplus s \cdot \Delta_{\mathsf{A}}$ and $\mathsf{K}[r] := \mathsf{M}[r] \oplus r \cdot \Delta_{\mathsf{B}}$. Then, send $(s, \mathsf{M}[s], \mathsf{K}[r])$ to P_{B}.
3. Upon receiving $(\mathsf{and}, (r_1, \mathsf{M}[r_1], \mathsf{K}[s_1]), (r_2, \mathsf{M}[r_2], \mathsf{K}[s_2]), (r_3, \mathsf{M}[r_3], \mathsf{K}[s_3]))$ from P_{A} and $(\mathsf{and}, (s_1, \mathsf{M}[s_1], \mathsf{K}[r_1]), (s_2, \mathsf{M}[s_2], \mathsf{K}[r_2]))$ from P_{B}, send cheat to P_{B} if it does not hold that $\mathsf{M}[r_i] = \mathsf{K}[r_i] \oplus r_i \cdot \Delta_{\mathsf{B}}$ and $\mathsf{M}[s_i] = \mathsf{K}[s_i] \oplus s_i \cdot \Delta_{\mathsf{A}}$ for $i \in \{1,2\}$. Otherwise, compute $s_3 = ((r_1 \oplus s_1) \cdot (r_2 \oplus s_2)) \oplus r_3$ and $\mathsf{M}[s_3] := \mathsf{K}[s_3] \oplus s_3 \cdot \Delta_{\mathsf{A}}$. Send $(s_3, \mathsf{M}[s_3], \mathsf{K}[r_3])$ to P_{B}.

Fig. 3. Functionality $\mathcal{F}_{\mathsf{pre}}$ (Figure 1 in [28]).

Inputs: Both parties P_{A} and P_{B} agree on a common circuit for function $f : \{0,1\}^{|\mathcal{I}_{\mathsf{A}}|} \times \{0,1\}^{|\mathcal{I}_{\mathsf{B}}|} \to \{0,1\}^{|\mathcal{O}|}$. In the input-preprocessing phase, P_{A} and P_{B} holds $\mathbf{x} = (z_i)_{i \in \mathcal{I}_{\mathsf{A}}} \in \{0,1\}^{|\mathcal{I}_{\mathsf{A}}|}$ and $\mathbf{y} = (z_i)_{i \in \mathcal{I}_{\mathsf{B}}} \in \{0,1\}^{|\mathcal{I}_{\mathsf{B}}|}$, respectively.

Function-Independent Preprocessing:

- P_{A} and P_{B} send init to $\mathcal{F}_{\mathsf{pre}}$ to receives Δ_{A} and Δ_{B}, respectively.
- For each $w \in \mathcal{I}_{\mathsf{A}} \cup \mathcal{I}_{\mathsf{B}} \cup \mathcal{W}$, both P_{A} and P_{B} send random to $\mathcal{F}_{\mathsf{pre}}$. In return, $\mathcal{F}_{\mathsf{pre}}$ sends $(r_w, \mathsf{M}[r_w], \mathsf{K}[s_w])$ and $(s_w, \mathsf{M}[s_w], \mathsf{K}[r_w])$ to P_{A} and P_{B}, respectively. Define $\lambda_w := r_w \oplus s_w$. P_{A} picks a uniform κ-bit string $\mathsf{L}_{w,0}$ where κ denotes the computational security parameter.

Function-Dependent Preprocessing:

- For each XOR gate $(\alpha, \beta, \gamma, \oplus)$, P_{A} computes $(r_\gamma, \mathsf{M}[r_\gamma], \mathsf{K}[s_\gamma]) := (r_\alpha \oplus r_\beta, \mathsf{M}[r_\alpha] \oplus \mathsf{M}[r_\beta], \mathsf{K}[s_\alpha] \oplus \mathsf{K}[s_\beta])$ and $\mathsf{L}_{\gamma,0} := \mathsf{L}_{\alpha,0} \oplus \mathsf{L}_{\beta,0}$. P_{B} computes $(s_\gamma, \mathsf{M}[s_\gamma], \mathsf{K}[r_\gamma]) := (s_\alpha \oplus s_\beta, \mathsf{M}[s_\alpha] \oplus \mathsf{M}[s_\beta], \mathsf{K}[r_\alpha] \oplus \mathsf{K}[r_\beta])$.
- For each AND gate $(\alpha, \beta, \gamma, \cdot)$:
  - P_{A} (respectively, P_{B}) sends $(\mathsf{and}, (r_\alpha, \mathsf{M}[r_\alpha], \mathsf{K}[s_\alpha]), (r_\beta, \mathsf{M}[r_\beta], \mathsf{K}[s_\beta]))$ (respectively, $(\mathsf{and}, (s_\alpha, \mathsf{M}[s_\alpha], \mathsf{K}[r_\alpha]), (s_\beta, \mathsf{M}[s_\beta], \mathsf{K}[r_\beta]))$) to $\mathcal{F}_{\mathsf{pre}}$. In return, $\mathcal{F}_{\mathsf{pre}}$ sends $(r'_\gamma, \mathsf{M}[r'_\gamma], \mathsf{K}[s'_\gamma])$ and $(s'_\gamma, \mathsf{M}[s'_\gamma], \mathsf{K}[r'_\gamma])$ respectively to P_{A} and P_{B}. Here, $r'_\gamma \oplus s'_\gamma = \lambda_\alpha \cdot \lambda_\beta$.
  - $\forall k \in [0,3]$, let $(k_0, k_1) := \mathsf{bin}(k) \in \{0,1\}^2$, i.e., $k = k_0 + 2 \cdot k_1$, and P_{A} computes
$$r_{\gamma,k} := r'_\gamma \oplus r_\gamma \oplus k_0 \cdot r_\alpha \oplus k_1 \cdot r_\beta,$$
$$\mathsf{M}[r_{\gamma,k}] := \mathsf{M}[r'_\gamma] \oplus \mathsf{M}[r_\gamma] \oplus k_0 \cdot \mathsf{M}[r_\beta] \oplus k_1 \cdot \mathsf{M}[r_\alpha],$$
$$\mathsf{K}[r_{\gamma,k}] := \mathsf{K}[r'_\gamma] \oplus \mathsf{K}[r_\gamma] \oplus k_0 \cdot \mathsf{K}[r_\beta] \oplus k_1 \cdot \mathsf{K}[r_\alpha] \oplus k_0 \cdot k_1 \cdot \Delta_{\mathsf{A}}.$$
  - $\forall k \in [0,3]$, let $(k_0, k_1) := \mathsf{bin}(k) \in \{0,1\}^2$, i.e., $k = k_0 + 2 \cdot k_1$, and P_{B} computes
$$s_{\gamma,k} := s'_\gamma \oplus s_\gamma \oplus k_0 \cdot s_\beta \oplus k_1 \cdot s_\alpha \oplus k_0 \cdot k_1,$$
$$\mathsf{M}[s_{\gamma,k}] := \mathsf{M}[s'_\gamma] \oplus \mathsf{M}[s_\gamma] \oplus k_0 \cdot \mathsf{M}[s_\beta] \oplus k_1 \cdot \mathsf{M}[s_\alpha],$$
$$\mathsf{K}[s_{\gamma,k}] := \mathsf{K}[s'_\gamma] \oplus \mathsf{K}[s_\gamma] \oplus k_0 \cdot \mathsf{K}[s_\beta] \oplus k_1 \cdot \mathsf{K}[s_\alpha].$$
  - P_{A} computes $\mathsf{L}_{\alpha,1} := \mathsf{L}_{\alpha,0} \oplus \Delta_{\mathsf{A}}$ and $\mathsf{L}_{\beta,1} := \mathsf{L}_{\beta,0} \oplus \Delta_{\mathsf{A}}$. P_{A} sends the following to P_{B}, for all $k \in [0,3]$ with $(k_0, k_1) := \mathsf{bin}(k) \in \{0,1\}^2$:
$$G_{\gamma,k} := H(\mathsf{L}_{\alpha,k_0}, \mathsf{L}_{\beta,k_1}, \gamma, k) \oplus (r_{\gamma,k}, \mathsf{M}[r_{\gamma,k}], \mathsf{L}_{\gamma,k} \oplus \mathsf{K}[s_{\gamma,k}] \oplus r_{\gamma,k} \cdot \Delta_{\mathsf{A}}).$$

Input Processing:

- For $w \in \mathcal{I}_{\mathsf{A}}$, P_{A} sends $(r_w, \mathsf{M}[r_w])$ to P_{B}. P_{B} checks whether $\mathsf{M}[r_w] = \mathsf{K}[r_w] \oplus r_w \cdot \Delta_{\mathsf{B}}$. P_{B} then sends $\hat{z}_w = z_w \oplus \lambda_w = z_w \oplus r_w \oplus s_w$ to P_{A}. Finally, P_{A} sends $\mathsf{L}_{w,\hat{z}_w}$ to P_{B}.
- For $w \in \mathcal{I}_{\mathsf{B}}$, P_{B} sends $(s_w, \mathsf{M}[s_w])$ to P_{A}. P_{A} checks whether $\mathsf{M}[s_w] = \mathsf{K}[s_w] \oplus s_w \cdot \Delta_{\mathsf{A}}$. P_{A} then sends $\hat{z}_w = z_w \oplus \lambda_w = z_w \oplus r_w \oplus s_w$ and $\mathsf{L}_{w,\hat{z}_w}$ to P_{B}.

Circuit Evaluation:

- P_{B} evaluates the circuit following a topological order. For each gate $(\alpha, \beta, \gamma, \mathsf{op})$ where $\mathsf{op} \in \{\cdot, \oplus\}$, P_{B} holds $(\hat{z}_\alpha, \mathsf{L}_{\alpha,\hat{z}_\alpha})$ and $(\hat{z}_\beta, \mathsf{L}_{\beta,\hat{z}_\beta})$ where $\hat{z}_\alpha = z_\alpha \oplus \lambda_\alpha$ and $\hat{z}_\beta = z_\beta \oplus \lambda_\beta$.
  - If $\mathsf{op} = \oplus$, P_{B} computes $\hat{z}_\gamma := \hat{z}_\alpha \oplus \hat{z}_\beta$ and $\mathsf{L}_{\gamma,\hat{z}_\gamma} := \mathsf{L}_{\alpha,\hat{z}_\alpha} \oplus \mathsf{L}_{\beta,\hat{z}_\beta}$.
  - If $\mathsf{op} = \cdot$, P_{B} computes $i := 2\hat{z}_\alpha + \hat{z}_\beta$. Then, P_{B} computes $(r_{\gamma,i}, \mathsf{M}[r_{\gamma,i}], \mathsf{L}_{\gamma,0} \oplus \mathsf{K}[s_{\gamma,i}] \oplus r_{\gamma,i} \cdot \Delta_{\mathsf{A}}) := G_{\gamma,i} \oplus H(\mathsf{L}_{\alpha,\hat{z}_\alpha}, \mathsf{L}_{\beta,\hat{z}_\beta}, \gamma, i)$. Then, P_{B} checks whether $\mathsf{M}[r_{\gamma,i}] = \mathsf{K}[r_{\gamma,i}] \oplus r_{\gamma,i} \cdot \Delta_{\mathsf{B}}$. If so, P_{B} computes $\hat{z}_\gamma = s_{\gamma,i} \oplus r_{\gamma,i}$ and $\mathsf{L}_{\gamma,\hat{z}_\gamma} := (\mathsf{L}_{\gamma,0} \oplus \mathsf{K}[s_{\gamma,i}] \oplus r_{\gamma,i} \cdot \Delta_{\mathsf{A}}) \oplus \mathsf{M}[s_{\gamma,i}]$.

Output Determination:

- For $w \in \mathcal{O}$, P_{A} sends $(r_w, \mathsf{M}[r_w])$ to P_{B} who checks whether $\mathsf{M}[r_w] = \mathsf{L}[r_w] \oplus r_w \cdot \Delta_{\mathsf{B}}$. If so, P_{B} computes $z_w := \hat{z}_w \oplus r_w \oplus s_w$.

Fig. 4. Protocol Π_{2PC} (Figure 2 in [28]).

Theorem 1 (Security of Π_{2PC}). *If H is modeled as a random oracle, the protocol Π_{2PC} in Figure 4, securely computes f against malicious adversaries with statistical security $2^{-\rho}$ in the $\mathcal{F}_{\mathsf{pre}}$-hybrid model.*

3.2 Making Authenticated Garbling from [28] Publicly Auditable

Our purpose is to make WRK protocol publicly auditable. Our approach is to follow the recent VOLEitH approach [7,14,11] by applying SoftSpokenOT [26] for making VOLE protocols (e.g., Wolverine [29] and Quicksilver [32]) publicly verifiable.

Verifier $\mathcal{V}$. To make WRK protocol publicly auditable, we modify the protocol in Figure 4 to include an additional party, called verifier $\mathcal{V}$. This party will play the role of verifying the transcript communicated between P_{A} and P_{B}.

Issues. We now discuss the employment of SoftSpokenOT [26] into WRK protocol. Assume that we transform all authenticated randomness of P_{A} and P_{B} into the forms specified by SoftSpokenOT (see Section 2.2). The transformation requires Δ_{A} and Δ_{B} to be public. However, making public Δ_{A} and Δ_{B} incurs the following issues:

- For each wire w, P_{B} is allowed to know only either $\mathsf{L}_{w,0}$ or $\mathsf{L}_{w,1}$. However, since Δ_{A} is public and $\mathsf{L}_{w,1} = \mathsf{L}_{w,0} \oplus \Delta_{\mathsf{B}}$, knowing either $\mathsf{L}_{w,0}$ or $\mathsf{L}_{w,1}$ implies knowledge of both labels. This hence violates the security of the original WRK protocol.
- Using only Δ_{A} and Δ_{B} for public verification is not convenient for amplifying soundness. We notice that VOLEitH systems [7,14,11] usually require repeating the protocol with a few distinct global keys to reduce soundness error.
- Verification shared AND triples, e.g., for an AND gate $(\alpha, \beta, \gamma, \cdot)$, both P_{A} and P_{B} should allow the verifier to believe that

$$(\underbrace{r_\alpha \oplus s_\alpha}_{\lambda_\alpha}) \cdot (\underbrace{r_\beta \oplus s_\beta}_{\lambda_\beta}) = \underbrace{r'_\gamma \oplus s'_\gamma}_{\lambda_\alpha \cdot \lambda_\beta}.$$

  Here, in WRK protocol, P_{A} and P_{B} rely on the outputs of $\mathcal{F}_{\mathsf{pre}}$ which guarantee the above identity holds. However, when making public auditability.

Our Approach. We briefly sketch our approach for coping with the abovementioned issues as follows.

Repurposing Δ_{A}. We first recall that Δ_{A} employed in WRK protocol is for two purposes: (i) authenticating P_{B}'s shared random mask and (ii) allowing P_{B} to compute $\mathsf{L}_{\gamma,\hat{z}_\gamma}$ for any AND gate $(\alpha, \beta, \gamma, \cdot)$. As discussed above, making Δ_{A} public violates P_{A}'s privacy. Therefore, in our design, we still keep Δ_{A} private to protect P_{A}'s privacy. To authenticate P_{B}'s shared random mask, we instead use Δ'_{A} generated from using SoftSpokenOT [26]. The idea is as follows:

- For each wire w, assume that s_w is authenticated in a way that P_{A} keeps Δ_{A} and $\mathsf{K}[s_w]$ while P_{B} keeps s_w and $\mathsf{M}[s_w]$.
- By applying SoftSpokenOT (c.f. Section 2.2), P_{B} can first obtain a random value $\overline{s}_w$ and $\mathsf{M}'[\overline{s}_w]$. Then, P_{B} and P_{A} apply steps 1 and 2 of protocol Π_{fix} presented in Figure 1 to obtain $\mathsf{M}'[\overline{s}_w]$. In particular, P_{B} and P_{A} respectively play the roles of P_p and $P_{p'}$. P_{B} first publishes $\hat{s}_w := s_w \oplus \overline{s}_w$ as in step 1. Then, P_{B} sets $\mathsf{M}'[s_w] := \mathsf{M}'[\overline{s}_w]$ as in step 1.
- At some point of the protocol, verifier $\mathcal{V}$ sends Δ'_{A} which helps every party to determine $\mathsf{K}'[\overline{s}_w]$. P_{A} hence can run step 3 by setting $\mathsf{K}'[s_w] := K'[\overline{s}_w] \oplus \hat{s}_w \cdot \Delta'_{\mathsf{A}}$.

We note that, as specified by SoftSpokenOT, both Δ'_{A} and $\mathsf{K}'[\overline{s}_w]$ are publicly known by every party. Since Δ'_{A} is not employed to determine $\mathsf{L}_{\gamma,\hat{z}_\gamma}$ for any AND gate $(\alpha,\beta,\gamma,\cdot)$, as long as Δ_{A} is not leaked, P_{A}'s privacy is hence guaranteed. On the other hand, any attempt from P_{A} to violate the correctness of the protocol is hence captured by the authenticated shared random masks via Δ'_{A}, instead of Δ_{A}, to make the protocol publicly auditable.

Symmetrically, we use Δ'_{B} to authenticate r_w for any wire w for public auditability purposes.

Recall that, for each AND gate $(\alpha,\beta,\gamma,\cdot)$, we additionally use r'_γ and s'_γ such that $r'_\gamma \oplus s'_\gamma = \lambda_\alpha \cdot \lambda_\beta$. We also use Δ'_{B} and Δ'_{A} to authenticate r'_γ and s'_γ, respectively.

Amplifying Soundness for Public Auditability. To amplify soundness for the public auditability version, we simply follow previous VOLEitH systems [7,14,11] repeat κ times where $\kappa \in \mathbb{N}$ is set sufficiently large to guarantee the soundness of the output transcript. Hence, rather than authenticating each s_w for any wire w by Δ'_{A}, we can authenticate by $\Delta'_{\mathsf{A},j}$ for $j \in [\kappa]$ with the associated IT-MAC tag $\mathsf{M}'_j[s_w]$ and key $\mathsf{K}'_j[s_w]$ such that

$$\mathsf{M}'_j[s_w] = \mathsf{K}'_j[s_w] \oplus s_w \cdot \Delta'_{\mathsf{A},j} \ \forall j \in [\kappa].$$

Similarly, we also authenticate r_w for any wire w by $\Delta'_{\mathsf{B},j}$, kept by P_{B}, to make it auditable and sound. Hence, we have

$$\mathsf{M}'_j[r_w] = \mathsf{K}'_j[r_w] \oplus s_w \cdot \Delta'_{\mathsf{B},j} \ \forall j \in [\kappa].$$

Here, $\mathsf{M}'_j[r_w]$ and $\mathsf{K}'_j[r_w]$ are the pair of IT-MAC tag and key kept by P_{A} and P_{B}, respectively.

Similarly, for any AND gate $(\alpha,\beta,\gamma,\cdot)$, we also authenticate r'_γ and s'_γ by $\{\Delta'_{\mathsf{B},j}\}_{j\in[\kappa]}$ and $\{\Delta'_{\mathsf{A},j}\}_{j\in[\kappa]}$, respectively.

Verifying Shared AND Triples. In our design above, for any AND gate $(\alpha,\beta,\gamma,\cdot)$, we use shared AND triples $(r_\alpha, r_\beta, r'_\gamma)$ and $(s_\alpha, s_\beta, s_\gamma)$, respectively kept by P_{A} and P_{B} and authenticated via $\{\Delta'_{\mathsf{B},j}\}_{j\in[\kappa]}$ and $\{\Delta'_{\mathsf{A},j}\}_{j\in[\kappa]}$, to ensure that

$$(\underbrace{r_\alpha \oplus s_\alpha}_{\lambda_\alpha}) \cdot (\underbrace{r_\beta \oplus s_\beta}_{\lambda_\beta}) = \underbrace{r_\gamma \oplus s_\gamma}_{\lambda_\alpha \cdot \lambda_\beta}. \tag{4}$$

Since we are trying to make the transcript publicly auditable, any public verifier $\mathcal{V}$ must believe that (4) holds. Our approach is as follows. For any $j \in [\kappa]$, assume that we have authenticated shared AND triples $(a_{\mathsf{A},j}, b_{\mathsf{A},j}, c_{\mathsf{A},j})$ and $(a_{\mathsf{B},j}, b_{\mathsf{B},j}, c_{\mathsf{B},j})$ respectively kept by P_{A} and P_{B} such that

$$(a_{\mathsf{A},j} \oplus a_{\mathsf{B},j}) \cdot (b_{\mathsf{A},j} \oplus b_{\mathsf{B},j}) = c_{\mathsf{A},j} \oplus c_{\mathsf{B},j}.$$

We adapt the checking multiplication trick [3,29] to check (4) as follows.

1. P_{A} (respectively, P_{B}) computes $d_{\mathsf{A},j} := r_\alpha \oplus a_{\mathsf{A},j}$ (respective, $d_{\mathsf{B},j} := s_\alpha \oplus a_{\mathsf{B},j}$) and $e_{\mathsf{A},j} := r_\beta \oplus b_{\mathsf{A},j}$ (respective, $e_{\mathsf{B},j} := s_\beta \oplus b_{\mathsf{B},j}$). P_{A} then publishes $d_{\mathsf{A},j}$ and $e_{\mathsf{A},j}$ while P_{B} publishes $d_{\mathsf{B},j}$ and $e_{\mathsf{B},j}$.

2. Each party locally computes $d_j := d_{\mathsf{A},j} \oplus d_{\mathsf{B},j}$ and $e_j := e_{\mathsf{A},j} \oplus e_{\mathsf{B},j}$.
3. P_{A} and P_{B} then can compute the authenticated values $\tilde{z}_{\mathsf{A},j}$ and $\tilde{z}_{\mathsf{B},j}$, respectively. P_{A} (respectively, P_{B}) then publishes $\tilde{z}_{\mathsf{A},j} \oplus r'_\gamma$ (respectively, $\tilde{z}_{\mathsf{B},j} \oplus s'_\gamma$). Then, any public party can check $\tilde{z}_{\mathsf{A},j} \oplus r'_\gamma \oplus \tilde{z}_{\mathsf{B},j} \oplus s'_\gamma$.

However, we are not dealing with checking a single AND gate. In our protocol, there are W such AND gates. To make such a check for all W AND gates, we generate L pairs of shared random AND triples. Then, we apply the cut-and-choose method [18] to check as follows:

- We select W random pairs from those L pairs for checking the AND gates.
- For the remaining $L - W$ from the L pairs, P_{A} and P_{B} simply reveal the corresponding authenticated triples for checking whether they are valid $L - W$ pairs of authenticated AND triples.

See Figure 7 for the realization of checking authenticated AND triples that we just discussed above.

Handling Opened Authenticated Shares at Output Wires of the Circuit. In the phase **Output Determination** in Figure 4, the parties need to reveal r_w for $w \in \mathcal{O}$ and the respective IT-MAC tags for checking authenticity. Since we follow the VOLEitH approach, IT-MAC tags should not be revealed after knowing $\{\Delta'_{\mathsf{A},j}\}_{j\in[\kappa]}$ and $\{\Delta'_{\mathsf{B},j}\}_{\jmath\in[\kappa]}$. Therefore, we enforce P_{A} to use a hiding and binding commitment scheme to commit to $(r_w, \{\mathsf{M}'_j[r_w]\}_{j\in[\kappa]})$ to obtain cmo_w for $w \in \mathcal{O}$. P_{A} then publishes cmo_w before knowing $\{\Delta'_{\mathsf{A},j}\}_{j\in[\kappa]}$ and $\{\Delta'_{\mathsf{B},j}\}_{\jmath\in[\kappa]}$ from verifier $\mathcal{V}$. By the binding property of the commitment scheme, P_{A} is unable to manipulate the committed messages behind the commitments. Therefore, after knowing $\{\Delta'_{\mathsf{A},j}\}_{j\in[\kappa]}$ and $\{\Delta'_{\mathsf{B},j}\}_{\jmath\in[\kappa]}$, P_{A} can open the commitments to obtain $(r_w, \{\mathsf{M}'_j[r_w]\}_{j\in[\kappa]})$. Hence, any public party, e.g., $\mathcal{V}$, can check the authenticity of $\{r_w\}_{w\in\mathcal{O}}$.

4 Publicly Auditable 2PC

In this section, we present the public auditable 2PC protocol following the technical overview presented in Section 3.2.

In Section 4.1, we introduce the functionality $\mathcal{F}_{\mathsf{sVOLE\text{-}2PC}}$ and its realization, namely, protocol $\Pi_{\mathsf{sVOLE\text{-}2PC}}$. Then, in Section 4.2, we present our protocols for publicly auditable 2PC.

4.1 Functionality $\mathcal{F}_{\mathsf{sVOLE\text{-}2PC}}$ and Protocol Π_{2PC}

We define the functionality $\mathcal{F}_{\mathsf{sVOLE\text{-}2PC}}$ in Figure 5 as a subroutine for generating suitable global values, IT-MAC tags, and corresponding keys. This functionality is suitable with SoftSpokenOT [26]. Hence, when realizing $\mathcal{F}_{\mathsf{sVOLE\text{-}2PC}}$ into protocol $\Pi_{\mathsf{sVOLE\text{-}2PC}}$ in Figure 5, we can be in $\mathcal{F}_{\mathsf{sVOLE}}$-hybrid model (c.f. Figure 2, discussed in Section 2.2)

Parties: P_{A} P_{B}, and a verifier $\mathcal{V}$.
Inputs: The sets $\mathcal{I}_{\mathsf{A}}, \mathcal{I}_{\mathsf{B}}$, and $\mathcal{W}$, parameters κ, τ, and L.

1. On inputs $(\mathsf{init}, \{r_w\}_{w\in\mathcal{I}_{\mathsf{A}}\cup\mathcal{I}_{\mathsf{B}}\cup\mathcal{W}}, \{r'_w\}_{w\in\mathcal{W}}, \{a^{(j)}_{\mathsf{A},i}, b^{(j)}_{\mathsf{A},i}, c^{(j)}_{\mathsf{A},i}\}_{j\in[\kappa],i\in[L]})$ from P_{A} and $(\mathsf{init}, \{s_w\}_{w\in\mathcal{I}_{\mathsf{A}}\cup\mathcal{I}_{\mathsf{B}}\cup\mathcal{W}}, \{s'_w\}_{w\in\mathcal{W}}, \{a^{(j)}_{\mathsf{B},i}, b^{(j)}_{\mathsf{B},i}, c^{(j)}_{\mathsf{B},i}\}_{j\in[\kappa],i\in[L]})$ from P_{B}, $\mathcal{F}_{\mathsf{sVOLE\text{-}2PC}}$ does:
   (a) Sample $\Delta'_{\mathsf{A},j}, \Delta'_{\mathsf{B},j} \xleftarrow{\$} \mathbb{F}_{2^\tau}$ $\forall j \in [\kappa]$.
   (b) If P_{A} is not corrupted, sample $\mathsf{M}'_j[r_w] \xleftarrow{\$} \mathbb{F}_{2^\tau}$ $\forall w \in \mathcal{I}_{\mathsf{A}} \cup \mathcal{I}_{\mathsf{B}} \cup \mathcal{W}$, $\mathsf{M}'_j[r'_w] \xleftarrow{\$} \mathbb{F}_{2^\tau}$ $\forall w \in \mathcal{W}$, and $\mathsf{M}'_j[a^{(j)}_{\mathsf{A},i}], \mathsf{M}'_j[b^{(j)}_{\mathsf{A},i}], \mathsf{M}'_j[c^{(j)}_{\mathsf{A},i}] \xleftarrow{\$} \mathbb{F}_{2^\tau}$ $\forall j \in [\kappa], \forall i \in [L]$. If P_{A} is corrupted, receive $\{\mathsf{M}'_j[r_w]\}_{w\in\mathcal{I}_{\mathsf{A}}\cup\mathcal{I}_{\mathsf{B}}\cup\mathcal{W}}$, $\{\mathsf{M}'_j[r'_w]\}_{w\in\mathcal{W}}$, and $\{\mathsf{M}'_j[a^{(j)}_{\mathsf{A},i}], \mathsf{M}'_j[b^{(j)}_{\mathsf{A},i}], \mathsf{M}'_j[c^{(j)}_{\mathsf{A},i}]\}_{j\in[\kappa],i\in[L]}$ from P_{A}. Then, compute $\{\mathsf{K}'_j[r_w]\}_{w\in\mathcal{I}_{\mathsf{A}}\cup\mathcal{I}_{\mathsf{B}}\cup\mathcal{W}}$, $\{\mathsf{K}'_j[r'_w]\}_{w\in\mathcal{W}}$, and $\{\mathsf{K}'_j[a^{(j)}_{\mathsf{A},i}], \mathsf{K}'_j[b^{(j)}_{\mathsf{A},i}], \mathsf{K}'_j[c^{(j)}_{\mathsf{A},i}]\}_{j\in[\kappa],i\in[L]}$ such that
$$\begin{cases} \mathsf{M}'_j[r_w] = \mathsf{K}'_j[r_w] \oplus r_w \cdot \Delta'_{\mathsf{A},j} & \forall w \in \mathcal{I}_{\mathsf{A}} \cup \mathcal{I}_{\mathsf{B}} \cup \mathcal{W}, \\ \mathsf{M}'_j[r'_w] = \mathsf{K}'_j[r'_w] \oplus r'_w \cdot \Delta'_{\mathsf{A},j} & \forall w \in \mathcal{W}, \\ \mathsf{M}'_j[a^{(j)}_{\mathsf{A},i}] = \mathsf{K}'_j[a^{(j)}_{\mathsf{A},i}] \oplus a^{(j)}_{\mathsf{A},i} \cdot \Delta'_{\mathsf{A},j} & \forall i \in [L], \\ \mathsf{M}'_j[b^{(j)}_{\mathsf{A},i}] = \mathsf{K}'_j[b^{(j)}_{\mathsf{A},i}] \oplus b^{(j)}_{\mathsf{A},i} \cdot \Delta'_{\mathsf{A},j} & \forall i \in [L], \\ \mathsf{M}'_j[c^{(j)}_{\mathsf{A},i}] = \mathsf{K}'_j[c^{(j)}_{\mathsf{A},i}] \oplus c^{(j)}_{\mathsf{A},i} \cdot \Delta'_{\mathsf{A},j} & \forall i \in [L]. \end{cases} \tag{5}$$
   (c) If P_{B} is not corrupted, sample $\mathsf{M}'_j[s_w] \xleftarrow{\$} \mathbb{F}_{2^\tau}$ $\forall w \in \mathcal{I}_{\mathsf{A}} \cup \mathcal{I}_{\mathsf{B}} \cup \mathcal{W}$, $\mathsf{M}'_j[s'_w] \xleftarrow{\$} \mathbb{F}_{2^\tau}$ $\forall w \in \mathcal{W}$, and $\mathsf{M}'_j[a^{(j)}_{\mathsf{B},i}], \mathsf{M}'_j[b^{(j)}_{\mathsf{B},i}], \mathsf{M}'_j[c^{(j)}_{\mathsf{B},i}] \xleftarrow{\$} \mathbb{F}_{2^\tau}$ $\forall j \in [\kappa], \forall i \in [L]$. If P_{A} is corrupted, receive $\{\mathsf{M}'_j[s_w]\}_{w\in\mathcal{I}_{\mathsf{A}}\cup\mathcal{I}_{\mathsf{B}}\cup\mathcal{W}}$, $\{\mathsf{M}'_j[s'_w]\}_{w\in\mathcal{W}}$, and $\{\mathsf{M}'_j[a^{(j)}_{\mathsf{B},i}], \mathsf{M}'_j[b^{(j)}_{\mathsf{B},i}], \mathsf{M}'_j[c^{(j)}_{\mathsf{B},i}]\}_{j\in[\kappa],i\in[L]}$ from P_{B}. Then, compute $\{\mathsf{K}'_j[s_w]\}_{w\in\mathcal{I}_{\mathsf{A}}\cup\mathcal{I}_{\mathsf{B}}\cup\mathcal{W}}$, $\{\mathsf{K}'_j[s'_w]\}_{w\in\mathcal{W}}$, and $\{\mathsf{K}'_j[a^{(j)}_{\mathsf{B},i}], \mathsf{K}'_j[b^{(j)}_{\mathsf{B},i}], \mathsf{K}'_j[c^{(j)}_{\mathsf{B},i}]\}_{j\in[\kappa],i\in[L]}$ such that
$$\begin{cases} \mathsf{M}'_j[s_w] = \mathsf{K}'_j[s_w] \oplus s_w \cdot \Delta'_{\mathsf{B},j} & \forall w \in \mathcal{I}_{\mathsf{A}} \cup \mathcal{I}_{\mathsf{B}} \cup \mathcal{W}, \\ \mathsf{M}'_j[s'_w] = \mathsf{K}'_j[s'_w] \oplus s'_w \cdot \Delta'_{\mathsf{B},j} & \forall w \in \mathcal{W}, \\ \mathsf{M}'_j[a^{(j)}_{\mathsf{B},i}] = \mathsf{K}'_j[a^{(j)}_{\mathsf{B},i}] \oplus a^{(j)}_{\mathsf{B},i} \cdot \Delta'_{\mathsf{B},j} & \forall i \in [L], \\ \mathsf{M}'_j[b^{(j)}_{\mathsf{B},i}] = \mathsf{K}'_j[b^{(j)}_{\mathsf{B},i}] \oplus b^{(j)}_{\mathsf{B},i} \cdot \Delta'_{\mathsf{B},j} & \forall i \in [L], \\ \mathsf{M}'_j[c^{(j)}_{\mathsf{B},i}] = \mathsf{K}'_j[c^{(j)}_{\mathsf{B},i}] \oplus c^{(j)}_{\mathsf{B},i} \cdot \Delta'_{\mathsf{B},j} & \forall i \in [L]. \end{cases} \tag{6}$$
   (d) If $\mathcal{V}$ is corrupted, receive $\{\mathsf{K}'_j[r_w]\}_{w\in\mathcal{I}_{\mathsf{A}}\cup\mathcal{I}_{\mathsf{B}}\cup\mathcal{W}}$, $\{\mathsf{K}'_j[r'_w]\}_{w\in\mathcal{W}}$, $\{\mathsf{K}'_j[a^{(j)}_{\mathsf{A},i}], \mathsf{K}'_j[b^{(j)}_{\mathsf{A},i}], \mathsf{K}'_j[c^{(j)}_{\mathsf{A},i}]\}_{j\in[\kappa],i\in[L]}$, $\{\mathsf{K}'_j[s_w]\}_{w\in\mathcal{I}_{\mathsf{A}}\cup\mathcal{I}_{\mathsf{B}}\cup\mathcal{W}}$, $\{\mathsf{K}'_j[s'_w]\}_{w\in\mathcal{W}}$, and $\{\mathsf{K}'_j[a^{(j)}_{\mathsf{B},i}], \mathsf{K}'_j[b^{(j)}_{\mathsf{B},i}], \mathsf{K}'_j[c^{(j)}_{\mathsf{B},i}]\}_{j\in[\kappa],i\in[L]}$, and compute $\{\mathsf{M}'_j[r_w]\}_{w\in\mathcal{I}_{\mathsf{A}}\cup\mathcal{I}_{\mathsf{B}}\cup\mathcal{W}}$, $\{\mathsf{M}'_j[r'_w]\}_{w\in\mathcal{W}}$, and $\{\mathsf{M}'_j[a^{(j)}_{\mathsf{A},i}], \mathsf{M}'_j[b^{(j)}_{\mathsf{A},i}], \mathsf{M}'_j[c^{(j)}_{\mathsf{A},i}]\}_{j\in[\kappa],i\in[L]}$, $\{\mathsf{M}'_j[s_w]\}_{w\in\mathcal{I}_{\mathsf{A}}\cup\mathcal{I}_{\mathsf{B}}\cup\mathcal{W}}$, $\{\mathsf{M}'_j[s'_w]\}_{w\in\mathcal{W}}$, and $\{\mathsf{M}'_j[a^{(j)}_{\mathsf{B},i}], \mathsf{M}'_j[b^{(j)}_{\mathsf{B},i}], \mathsf{M}'_j[c^{(j)}_{\mathsf{B},i}]\}_{j\in[\kappa],i\in[L]}$ such that both (5) and (6) hold.
   (e) Send $\{\mathsf{M}'_j[r_w]\}_{w\in\mathcal{I}_{\mathsf{A}}\cup\mathcal{I}_{\mathsf{B}}\cup\mathcal{W}}$, $\{\mathsf{M}'_j[r'_w]\}_{w\in\mathcal{W}}$, $\{\mathsf{M}'_j[a^{(j)}_{\mathsf{A},i}], \mathsf{M}'_j[b^{(j)}_{\mathsf{A},i}], \mathsf{M}'_j[c^{(j)}_{\mathsf{A},i}]\}_{j\in[\kappa],i\in[L]}$ to P_{A}.
   (f) Send $\{\mathsf{M}'_j[s_w]\}_{w\in\mathcal{I}_{\mathsf{A}}\cup\mathcal{I}_{\mathsf{B}}\cup\mathcal{W}}$, $\{\mathsf{M}'_j[s'_w]\}_{w\in\mathcal{W}}$, $\{\mathsf{M}'_j[a^{(j)}_{\mathsf{B},i}], \mathsf{M}'_j[b^{(j)}_{\mathsf{B},i}], \mathsf{M}'_j[c^{(j)}_{\mathsf{B},i}]\}_{j\in[\kappa],i\in[L]}$ to P_{B}.
2. On input (get) from P_{A}, P_{B}, and $\mathcal{V}$, $\mathcal{F}_{\mathsf{sVOLE\text{-}2PC}}$ sends $\{\Delta'_{\mathsf{A},j}\}_{j\in[\kappa]}$, $\{\mathsf{K}'_j[r_w]\}_{w\in\mathcal{I}_{\mathsf{A}}\cup\mathcal{I}_{\mathsf{B}}\cup\mathcal{W}}$, $\{\mathsf{K}'_j[r'_w]\}_{w\in\mathcal{W}}$, $\{\mathsf{K}'_j[a^{(j)}_{\mathsf{A},i}], \mathsf{K}'_j[b^{(j)}_{\mathsf{A},i}], \mathsf{K}'_j[c^{(j)}_{\mathsf{A},i}]\}_{j\in[\kappa],i\in[L]}$, $\{\Delta'_{\mathsf{B},j}\}_{j\in[\kappa]}$, $\{\mathsf{K}'_j[s_w]\}_{w\in\mathcal{I}_{\mathsf{A}}\cup\mathcal{I}_{\mathsf{B}}\cup\mathcal{W}}$, $\{\mathsf{K}'_j[s'_w]\}_{w\in\mathcal{W}}$, $\{\mathsf{K}'_j[a^{(j)}_{\mathsf{B},i}], \mathsf{K}'_j[b^{(j)}_{\mathsf{B},i}], \mathsf{K}'_j[c^{(j)}_{\mathsf{B},i}]\}_{j\in[\kappa],i\in[L]}$ to P_{A}, P_{B}, and $\mathcal{V}$.

Fig. 5. Functionality $\mathcal{F}_{\mathsf{sVOLE\text{-}2PC}}$.

Communication Cost of $\Pi_{\mathsf{sVOLE\text{-}2PC}}$ (c.f. Figure 6). We first notice that we need to run in total 2κ times the functionality $\mathcal{F}^{\mathcal{P},\mathcal{V},\tau,N}_{\mathsf{sVOLE}}$. When realizing this functionality by SoftSpokenOT, it requires $2 \cdot \kappa$ vector commitments and openings which subsequently determine the keys of authenticated values. We denote by $\mathsf{cost}_{\mathsf{vc\text{-}total}}$ to be the size of each vector commitment and its respective $N-1$ openings. Hence, in total, it requires $2 \cdot \kappa \cdot (\mathsf{cost}_{\mathsf{vc\text{-}total}}+)$ for this task.

Using protocol Π_{fix} costs $2 \cdot \kappa \cdot (|\mathcal{I}_{\mathsf{A}}| + |\mathcal{I}_{\mathsf{B}}| + 2 \cdot |\mathcal{W}|)$ bits for publishing $\hat{r}_{w,j}$, $\hat{s}_{w,j}$ (for $w \in \mathcal{I}_{\mathsf{A}} \cup \mathcal{I}_{\mathsf{B}} \cup \mathcal{W}$ and $j \in \kappa$), $\hat{r}'_{w,j}$ and $\hat{s}'_{w,j}$ (for $w \in \mathcal{W}$ and $j \in \kappa$).

Parties: P_{A} P_{B}, and a verifier $\mathcal{V}$.
Inputs: The sets $\mathcal{I}_{\mathsf{A}}, \mathcal{I}_{\mathsf{B}}$, and $\mathcal{W}$, parameters κ, τ, L, and $N = |\mathcal{I}_{\mathsf{A}} \cup \mathcal{I}_{\mathsf{B}} \cup \mathcal{W}| + |\mathcal{W}| + 3L$.
Execution:

1. On inputs $(\mathsf{init}, \{r_w\}_{w\in\mathcal{I}_{\mathsf{A}}\cup\mathcal{I}_{\mathsf{B}}\cup\mathcal{W}}, \{r'_w\}_{w\in\mathcal{W}}, \{a^{(j)}_{\mathsf{A},i}, b^{(j)}_{A,i}, c^{(j)}_{\mathsf{A},i}\}_{j\in[\kappa],i\in[L]})$ from P_{A} and $(\mathsf{init}, \{s_w\}_{w\in\mathcal{I}_{\mathsf{A}}\cup\mathcal{I}_{\mathsf{B}}\cup\mathcal{W}}, \{s'_w\}_{w\in\mathcal{W}}, \{a^{(j)}_{\mathsf{B},i}, b^{(j)}_{\mathsf{B},i}, c^{(j)}_{\mathsf{B},i}\}_{j\in[\kappa],i\in[L]})$ form P_{B}, $\Pi_{\mathsf{sVOLE\text{-}2PC}}$ works as follows:
   (a) P_{A} and $\mathcal{V}$ send init to κ instances of $\mathcal{F}^{P_{\mathsf{A}},\mathcal{V},\tau,N}_{\mathsf{sVOLE}}$. Besides, P_{B} and $\mathcal{V}$ send init to κ instances of $\mathcal{F}^{P_{\mathsf{B}},\mathcal{V},\tau,N}_{\mathsf{sVOLE}}$.
   (b) P_{A} receives outputs
   $$\mathbf{u}_j = ((\overline{r}_{w,j})_{w\in\mathcal{I}_{\mathsf{A}}\cup\mathcal{I}_{\mathsf{B}}\cup\mathcal{W},j\in[\kappa]} \| (\overline{r}'_{w,j})_{w\in\mathcal{W},j\in[\kappa]} \| (\overline{a}^{(i)}_{\mathsf{A},j}, \overline{b}^{(i)}_{\mathsf{A},j}, \overline{c}^{(i)}_{\mathsf{A},j})_{i\in[L]}) \quad j\in[\kappa],$$
   $$\mathbf{v}_j = ((\mathsf{M}'_j[\overline{r}_{w,j}])_{w\in\mathcal{I}_{\mathsf{A}}\cup\mathcal{I}_{\mathsf{B}}\cup\mathcal{W},j\in[\kappa]} \| (\mathsf{M}'_j[\overline{r}'_{w,j}])_{w\in\mathcal{W},j\in[\kappa]} \| (\mathsf{M}'_j[\overline{a}^{(i)}_{\mathsf{A},j}], \mathsf{M}'_j[\overline{b}^{(i)}_{\mathsf{A},j}], \mathsf{M}'_j[\overline{c}^{(i)}_{\mathsf{A},j}])_{i\in[L]}) \quad j\in[\kappa]$$
   from κ instances of $\mathcal{F}^{P_{\mathsf{A}},\mathcal{V},\tau,N}_{\mathsf{sVOLE}}$ which can be parsed as $\{(\overline{r}_{w,j}, \mathsf{M}'_j[\overline{r}_{w,j}])\}_{w\in\mathcal{I}_{\mathsf{A}}\cup\mathcal{I}_{\mathsf{B}}\cup\mathcal{W},j\in[\kappa]}$, $\{(\overline{r}'_{w,j}, \mathsf{M}'_j[\overline{r}'_{w,j}])\}_{w\in\mathcal{W},j\in[\kappa]}$, and $\{(\overline{a}^{(i)}_{\mathsf{A},j}, \mathsf{M}'_j[\overline{a}^{(i)}_{\mathsf{A},j}], \overline{b}^{(i)}_{\mathsf{A},j}, \mathsf{M}'_j[\overline{b}^{(i)}_{\mathsf{A},j}], \overline{c}^{(i)}_{\mathsf{A},j}, \mathsf{M}'_j[\overline{c}^{(i)}_{\mathsf{A},j}])\}_{i\in[L],j\in[\kappa]}$. Similarly, receives outputs from κ instances of $\mathcal{F}^{P_{\mathsf{B}},\mathcal{V},\tau,N}_{\mathsf{sVOLE}}$ which can be parsed as $\{(\overline{s}_{w,j}, \mathsf{M}'_j[\overline{s}_{w,j}])\}_{w\in\mathcal{I}_{\mathsf{A}}\cup\mathcal{I}_{\mathsf{B}}\cup\mathcal{W},j\in[\kappa]}$, $\{(\overline{s}'_{w,j}, \mathsf{M}'_j[\overline{s}'_{w,j}])\}_{w\in\mathcal{W},j\in[\kappa]}$, and $\{(\overline{a}^{(i)}_{\mathsf{B},j}, \mathsf{M}'_j[\overline{a}^{(i)}_{\mathsf{B},j}], \overline{b}^{(i)}_{\mathsf{B},j}, \mathsf{M}'_j[\overline{b}^{(i)}_{\mathsf{B},j}], \overline{c}^{(i)}_{\mathsf{B},j}, \mathsf{M}'_j[\overline{c}^{(i)}_{\mathsf{B},j}])\}_{i\in[L],j\in[\kappa]}$.
   (c) For $j\in[\kappa]$ and $w\in\mathcal{I}_{\mathsf{A}}\cup\mathcal{I}_{\mathsf{B}}\cup\mathcal{W}$, P_{A} and P_{B} use protocol Π_{fix} to respectively obtain $(r_w, \mathsf{M}'_j[r_w])$, from inputs r_w and $(\overline{r}_{w,j}, \mathsf{M}'_j[\overline{r}_{w,j}])$, and $(s_w, \mathsf{M}'_j[s_w])$, from inputs s_w and $(\overline{s}_{w,j}, \mathsf{M}'_j[\overline{s}_{w,j}])$. Doing this requires each $\hat{r}_{w,j} := \overline{r}_{w,j} \oplus r_w$ and $\hat{s}_{w,j} := \overline{s}_{w,j} \oplus s_w$ to be published.
   (d) For $j\in[\kappa]$ and $w\in\mathcal{W}$, P_{A} and P_{B} use protocol Π_{fix} to respectively obtain $(r'_w, \mathsf{M}'_j[r'_w])$, from inputs r'_w and $(\overline{r}'_{w,j}, \mathsf{M}'_j[\overline{r}'_{w,j}])$, and $(s'_w, \mathsf{M}'_j[s'_w])$, from inputs s'_w and $(\overline{s}'_{w,j}, \mathsf{M}'_j[\overline{s}'_{w,j}])$. Doing this requires each $\hat{r}'_{w,j} := \overline{r}_{w,j} \oplus r_w$ and $\hat{s}'_{w,j} := \overline{s}_{w,j} \oplus s_w$ to be published.
   (e) For $j\in[\kappa]$ and $i\in[L]$, P_{A} and B use Π_{fix} to achieve $\mathsf{M}'_j[a^{(i)}_{\mathsf{A},j}], \mathsf{M}'_j[b^{(i)}_{\mathsf{A},j}], \mathsf{M}'_j[c^{(i)}_{\mathsf{A},j}], \mathsf{M}'_j[a^{(i)}_{\mathsf{B},j}], \mathsf{M}'_j[b^{(i)}_{\mathsf{B},j}], \mathsf{M}'_j[c^{(i)}_{\mathsf{B},j}]$. Doing this requires $\hat{a}^{(i)}_{p,j} := \overline{a}^{(i)}_{p,j} \oplus a^{(i)}_{p,j}$, $\hat{b}^{(i)}_{p,j} := \overline{b}^{(i)}_{p,j} \oplus b^{(i)}_{p,j}$, $\hat{c}^{(i)}_{p,j} := \overline{c}^{(i)}_{p,j} \oplus c^{(i)}_{p,j}$ for $p\in\{\mathsf{A},\mathsf{B}\}$ to be published.
2. On input (get) from P_{A}, P_{B}, and $\mathcal{V}$, $\Pi_{\mathsf{sVOLE\text{-}2PC}}$ works as follows:
   (a) P_{A} and $\mathcal{V}$ send (get) to κ instances of $\mathcal{F}^{P_{\mathsf{A}},\mathcal{V},\tau,N}_{\mathsf{sVOLE}}$ to receive outputs which can be parsed as $\{\Delta'_{\mathsf{A},j}\}_{j\in[\kappa]}$ $\{\mathsf{K}'_j[\overline{r}_{w,j}]\}_{w\in\mathcal{I}_{\mathsf{A}}\cup\mathcal{I}_{\mathsf{B}}\cup\mathcal{W},j\in[\kappa]}$, $\{\mathsf{K}'_j[\overline{r}'_{w,j}]\}_{w\in\mathcal{W},j\in[\kappa]}$, $\{\mathsf{K}'_j[\overline{a}^{(i)}_{\mathsf{A},j}], \mathsf{K}'_j[\overline{b}^{(i)}_{\mathsf{A},j}], \mathsf{K}'_j[\overline{c}^{(i)}_{\mathsf{A},j}]\}_{i\in[L],j\in[\kappa]}$.
   Similarly, P_{A} and $\mathcal{V}$ send (get) to κ instances of $\mathcal{F}^{P_{\mathsf{B}},\mathcal{V},\tau,N}_{\mathsf{sVOLE}}$ to receive outputs which can be parsed as $\{\Delta'_{\mathsf{B},j}\}_{j\in[\kappa]}$, $\{\mathsf{K}'_j[\overline{s}_{w,j}]\}_{w\in\mathcal{I}_{\mathsf{A}}\cup\mathcal{I}_{\mathsf{B}}\cup\mathcal{W},j\in[\kappa]}$, $\{\mathsf{K}'_j[\overline{s}'_{w,j}]\}_{w\in\mathcal{W},j\in[\kappa]}$, and $\{\mathsf{K}'_j[\overline{a}^{(i)}_{\mathsf{B},j}], \mathsf{K}'_j[\overline{b}^{(i)}_{\mathsf{B},j}], \mathsf{K}'_j[\overline{c}^{(i)}_{\mathsf{B},j}]\}_{i\in[L],j\in[\kappa]}$.
   (b) Any party can compute locally, by applying protocol Π_{fix} for all $j\in[\kappa]$, as follows:
   $\forall w\in\mathcal{I}_{\mathsf{A}}\cup\mathcal{I}_{\mathsf{B}}\cup\mathcal{W}, \mathsf{K}'_j[r_{w,j}] := \mathsf{K}'_j[\overline{r}_{w,j}] \oplus \hat{r}_{w,j}\cdot\Delta'_{\mathsf{B},j}, \mathsf{K}'_j[s_{w,j}] := \mathsf{K}'_j[\overline{s}_{w,j}] \oplus \hat{s}_{w,j}\cdot\Delta'_{\mathsf{A},j}$;
   $\forall w\in\mathcal{W}, \mathsf{K}'_j[r'_{w,j}] := \mathsf{K}'_j[\overline{r}'_{w,j}] \oplus \hat{r}'_{w,j}\cdot\Delta'_{\mathsf{B},j}, \mathsf{K}'_j[s'_{w,j}] := \mathsf{K}'_j[\overline{s}'_{w,j}] \oplus \hat{s}'_{w,j}\cdot\Delta'_{\mathsf{A},j}$;
   $\forall i\in[L], \mathsf{K}'_j[a^{(i)}_{\mathsf{A},j}] := \mathsf{K}'_j[\overline{a}^{(i)}_{\mathsf{A},j}] \oplus \hat{a}^{(i)}_{\mathsf{A},j}\cdot\Delta'_{\mathsf{B},j}, \mathsf{K}'_j[a^{(i)}_{\mathsf{B},j}] := \mathsf{K}'_j[\overline{a}^{(i)}_{\mathsf{B},j}] \oplus \hat{a}^{(i)}_{\mathsf{B},j}\cdot\Delta'_{\mathsf{A},j}$;
   $\forall i\in[L], \mathsf{K}'_j[b^{(i)}_{\mathsf{A},j}] := \mathsf{K}'_j[\overline{b}^{(i)}_{\mathsf{A},j}] \oplus \hat{b}^{(i)}_{\mathsf{A},j}\cdot\Delta'_{\mathsf{B},j}, \mathsf{K}'_j[b^{(i)}_{\mathsf{B},j}] := \mathsf{K}'_j[\overline{b}^{(i)}_{\mathsf{B},j}] \oplus \hat{b}^{(i)}_{\mathsf{B},j}\cdot\Delta'_{\mathsf{A},j}$;
   $\forall i\in[L], \mathsf{K}'_j[c^{(i)}_{\mathsf{A},j}] := \mathsf{K}'_j[\overline{c}^{(i)}_{\mathsf{A},j}] \oplus \hat{c}^{(i)}_{\mathsf{A},j}\cdot\Delta'_{\mathsf{B},j}, \mathsf{K}'_j[c^{(i)}_{\mathsf{B},j}] := \mathsf{K}'_j[\overline{c}^{(i)}_{\mathsf{B},j}] \oplus \hat{c}^{(i)}_{\mathsf{B},j}\cdot\Delta'_{\mathsf{A},j}$.

Fig. 6. Protocol $\Pi_{\mathsf{sVOLE\text{-}2PC}}$.

Hence, in total, public transcript of $\Pi_{\mathsf{sVOLE\text{-}2PC}}$ costs

$$2\cdot\kappa\cdot(\mathsf{cost}_{\mathsf{vc\text{-}total}} + |\mathcal{I}_{\mathsf{A}}| + |\mathcal{I}_{\mathsf{B}}| + 2\cdot|\mathcal{W}|)$$

bits of communication.

4.2 Public Auditability 2PC

In this section, we first present the protocol $\Pi_{\mathsf{check\text{-}AND}}$, as discussed in Section 3.2, in Figure 7 for checking satisfaction of authenticated shared AND triples. Then, we present the protocols $\Pi_{\mathsf{pa\text{-}2PC\text{-}pre}}$ (for preprocessing) and $\Pi_{\mathsf{pa\text{-}2PC\text{-}eval}}$ (for evaluating) in Figures 8 and 9, respectively. These protocols employ $\Pi_{\mathsf{check\text{-}AND}}$ as a subroutine for executing and in the $\mathcal{F}_{\mathsf{sVOLE\text{-}2PC}}$-hybrid model. We denote by $\Pi_{\mathsf{pa\text{-}2PC}}$ the concatenation of $\Pi_{\mathsf{pa\text{-}2PC\text{-}pre}}$ and $\Pi_{\mathsf{pa\text{-}2PC\text{-}eval}}$.

Non-Interactivity and Public Verifiability. Notice that, although Δ_{A} and Δ_{B} are not revealed, as explained in Section 3.2, we can verify authenticity of all shared masks via $\{\Delta'_{\mathsf{A},j}, \Delta'_{\mathsf{B},j}\}_{j\in[\kappa]}$ by following the paradigm from [7]. Since $\{\Delta'_{\mathsf{A},j}, \Delta'_{\mathsf{B},j}\}_{j\in[\kappa]}$ are generated via $\mathcal{F}_{\mathsf{sVOLE}}$, as from [7], the protocol realizing $\mathcal{F}_{\mathsf{sVOLE}}$ can be public coin, i.e., $\{\Delta'_{\mathsf{A},j}, \Delta'_{\mathsf{B},j}\}_{j\in[\kappa]}$ can be generated uniformly. Hence, by applying the Fiat-Shamir paradigm [20] for generating $\{\mathcal{I}_j\}_{j\in[\kappa]}$ and $\{\Delta'_{\mathsf{A},j}, \Delta'_{\mathsf{B},j}\}_{j\in[\kappa]}$, we can transform the entire transcript of the concatenated protocol $\Pi_{\mathsf{pa\text{-}2PC}}$ obtained from concatenating $\Pi_{\mathsf{pa\text{-}2PC\text{-}pre}}$ and $\Pi_{\mathsf{pa\text{-}2PC\text{-}eval}}$ into a non-interactive ZKP, hence making it publicly auditable.

Theorem 2 (Security of $\Pi_{\mathsf{sVOLE\text{-}2PC}}$). *The protocol $\Pi_{\mathsf{sVOLE\text{-}2PC}}$ in Figure 6 securely realizes $\mathcal{F}_{\mathsf{sVOLE\text{-}2PC}}$ in Figure 5 against malicious adversaries with statistical security $2^{-\rho}$ in the $\mathcal{F}_{\mathsf{sVOLE}}$-hybrid model.*

Theorem 3 (Security of $\Pi_{\mathsf{pa\text{-}2PC}}$). *If H is modeled as a random oracle, the combined protocol from concatenating $\Pi_{\mathsf{pa\text{-}2PC\text{-}pre}}$ and $\Pi_{\mathsf{pa\text{-}2PC\text{-}eval}}$ in Figures 8 and 9, respectively, securely computes f against malicious adversaries with statistical security $2^{-\rho}$ in the $\mathcal{F}_{\mathsf{sVOLE\text{-}2PC}}$-hybrid model.*

Communication Cost of $\Pi_{\mathsf{check\text{-}AND}}$. In step 1, for any $j \in [\kappa]$ and any $i \in [W]$, the total cost is $6 \cdot (2^\tau + 1)$ for published bits and IT-MAC tags. Hence, the total cost in this phase is $\kappa \cdot W \cdot (2^\tau + 1)$. In step 2, there is nothing published. Hence, in total, this protocol costs

$$\kappa \cdot W \cdot (2^\tau + 1)$$

bits of communication.

Communication Cost of $\Pi_{\mathsf{pa\text{-}2PC}}$. We now analyze communication cost of $\Pi_{\mathsf{pa\text{-}2PC}}$ via $\Pi_{\mathsf{pa\text{-}2PC\text{-}pre}}$ and $\Pi_{\mathsf{pa\text{-}2PC\text{-}eval}}$.

For $\Pi_{\mathsf{pa\text{-}2PC\text{-}pre}}$ in Figure 8, the cost of this protocol mainly concentrates on Phases 3 and 4. In particular, each garbled table costs $4 \cdot (1 + (\kappa + 2) \cdot 2^\tau)$. On the other hand, each commitment from P_{B} costs $|\mathsf{cma}|$. Hence, in total, the cost for both phases is $4 \cdot |\mathcal{W}| \cdot (1 + (\kappa + 2) \cdot 2^\tau + |\mathsf{cma}|)$.

For $\Pi_{\mathsf{pa\text{-}2PC\text{-}eval}}$ in Figure 9, publishing for checking authenticated shared AND triples according to $j \in \kappa, i \in [L] \setminus \mathcal{I}_j$ costs in total $6 \cdot \kappa \cdot |\mathcal{W}| \cdot (1 + 2^\tau)$ bits. The cost for the phase **input processing** is $(|\mathcal{I}_{\mathsf{A}}| + |\mathcal{I}_{\mathsf{B}}|) \cdot (1 + (\kappa + 2) \cdot 2^\tau)$. In **circuit evaluation**, the cost is $|\mathcal{W}| \cdot 2 \cdot (1 + (\kappa + 2) \cdot 2^\tau)$. In **output determination**, the cost is $|\mathcal{O}| \cdot (1 + (\kappa + 1) \cdot 2^\tau)$. In **output determination**, the cost is $|\mathcal{O}| \cdot (1 + (\kappa + 1) \cdot 2^\tau)$. Hence, in total, the cost for the entire protocol is $6 \cdot \kappa \cdot |\mathcal{W}| \cdot (1 + 2^\tau) + (|\mathcal{I}_{\mathsf{A}}| + |\mathcal{I}_{\mathsf{B}}|) \cdot (1 + (\kappa + 2) \cdot 2^\tau) + 2 \cdot |\mathcal{W}| \cdot (1 + (\kappa + 2) \cdot 2^\tau) + |\mathcal{O}| \cdot (1 + (\kappa + 1) \cdot 2^\tau)$.

We assume that authenticated P_{A}'s values (respectively, P_{B}'s values) are with respect to $\Delta'_{\mathsf{B},j}$ (respectively, $\Delta'_{\mathsf{A},j}$) for $j \in [\kappa]$. E.g., $[\![u; \Delta'_{\mathsf{B},j}]\!]_{\mathsf{A}}$ is parsed as $\mathsf{M}'[u] = \mathsf{K}'[u] \oplus u \cdot \Delta'_{\mathsf{B},j}$.

Inputs: $\kappa, W \in \mathbb{N}$.

Execution:

1. On inputs

$$(\mathsf{init}, \{x^{(i)}_{\mathsf{A},j}, y^{(i)}_{\mathsf{A},j}, z^{(i)}_{\mathsf{A},j}, \mathsf{M}'_j[x^{(i)}_{\mathsf{A},j}], \mathsf{M}'_j[y^{(i)}_{\mathsf{A},j}], \mathsf{M}'_j[z^{(i)}_{\mathsf{A},j}]\}_{j\in[\kappa],i\in[W]},$$
$$\{a^{(i)}_{\mathsf{A},j}, b^{(i)}_{\mathsf{A},j}, c^{(i)}_{\mathsf{A},j}, \mathsf{M}'_j[a^{(i)}_{\mathsf{A},j}], \mathsf{M}'_j[b^{(i)}_{\mathsf{A},j}], \mathsf{M}'_j[c^{(i)}_{\mathsf{A},j}]\}_{j\in[\kappa],i\in[W]})$$

   from P_{A} and

$$(\mathsf{init}, \{x^{(i)}_{\mathsf{B},j}, y^{(i)}_{\mathsf{B},j}, z^{(i)}_{\mathsf{B},j}, \mathsf{M}'_j[x^{(i)}_{\mathsf{B},j}], \mathsf{M}'_j[y^{(i)}_{\mathsf{B},j}], \mathsf{M}'_j[z^{(i)}_{\mathsf{B},j}]\}_{j\in[\kappa],i\in[W]},$$
$$\{a^{(i)}_{\mathsf{B},j}, b^{(i)}_{\mathsf{B},j}, c^{(i)}_{\mathsf{B},j}, \mathsf{M}'_j[a^{(i)}_{\mathsf{B},j}], \mathsf{M}'_j[b^{(i)}_{\mathsf{B},j}], \mathsf{M}'_j[c^{(i)}_{\mathsf{B},j}]\}_{j\in[\kappa],i\in[W]})$$

   from P_{B}, for $j \in [\kappa]$, $i \in [W]$, it works as follows:
   (a) P_{A} locally computes and sends $d^{(i)}_{\mathsf{A},j} := x^{(i)}_{\mathsf{A},j} \oplus a^{(i)}_{\mathsf{A},j}$, $e^{(i)}_{\mathsf{A},j} := y^{(i)}_{\mathsf{A},j} \oplus b^{(i)}_{\mathsf{A},j}$, $\mathsf{M}'_j[d^{(i)}_{\mathsf{A},j}] := \mathsf{M}'_j[x^{(i)}_{\mathsf{A},j}] \oplus \mathsf{M}'_j[a^{(i)}_{\mathsf{A},j}]$, $\mathsf{M}'_j[e^{(i)}_{\mathsf{A},j}] := \mathsf{M}'_j[y^{(i)}_{\mathsf{A},j}] \oplus \mathsf{M}'_j[b^{(i)}_{\mathsf{A},j}]$ to P_{B} and $\mathcal{V}$.
   (b) P_{B} locally computes and sends $d^{(i)}_{\mathsf{B},j} := x^{(i)}_{\mathsf{B},j} \oplus a^{(i)}_{\mathsf{B},j}$, $e^{(i)}_{\mathsf{B},j} := y^{(i)}_{\mathsf{B},j} \oplus b^{(i)}_{\mathsf{B},j}$, $\mathsf{M}'_j[d^{(i)}_{\mathsf{B},j}] := \mathsf{M}'_j[x^{(i)}_{\mathsf{B},j}] \oplus \mathsf{M}'_j[a^{(i)}_{\mathsf{B},j}]$, $\mathsf{M}'_j[e^{(i)}_{\mathsf{B},j}] := \mathsf{M}'_j[y^{(i)}_{\mathsf{B},j}] \oplus \mathsf{M}'_j[b^{(i)}_{\mathsf{B},j}]$ to P_{A} and $\mathcal{V}$.
   (c) Each party (P_{A}, P_{B}, and $\mathcal{V}$) locally computes $d^{(i)}_j := d^{(i)}_{\mathsf{A},j} \oplus d^{(i)}_{\mathsf{B},j}$ and $e^{(i)}_j := e^{(i)}_{\mathsf{A},j} \oplus e^{(i)}_{\mathsf{B},j}$.
   (d) P_{A} locally computes and sends $\tilde{z}^{(i)}_{\mathsf{A},j} := z^{(i)}_{\mathsf{A},j} \oplus c^{(i)}_{\mathsf{A},j} \oplus d^{(i)}_j \cdot b^{(i)}_{\mathsf{A},j} \oplus e^{(i)}_j \cdot a^{(i)}_{\mathsf{A},j}$ and $\mathsf{M}'_j[\tilde{z}^{(i)}_{\mathsf{A},j}] := \mathsf{M}'_j[z^{(i)}_{\mathsf{A},j}] \oplus \mathsf{M}'_j[c^{(i)}_{\mathsf{A},j}] \oplus d^{(i)}_j \cdot \mathsf{M}'_j[b^{(i)}_{\mathsf{A},j}] \oplus e^{(i)}_j \cdot \mathsf{M}'_j[a^{(i)}_{\mathsf{A},j}]$ to P_{B} and $\mathcal{V}$.
   (e) P_{B} locally computes and sends $\tilde{z}^{(i)}_{\mathsf{B},j} := z^{(i)}_{\mathsf{B},j} \oplus c^{(i)}_{\mathsf{B},j} \oplus d^{(i)}_j \cdot b^{(i)}_{\mathsf{B},j} \oplus e^{(i)}_j \cdot a^{(i)}_{\mathsf{B},j}$ and $\mathsf{M}'_j[\tilde{z}^{(i)}_{\mathsf{B},j}] := \mathsf{M}'_j[z^{(i)}_{\mathsf{B},j}] \oplus \mathsf{M}'_j[c^{(i)}_{\mathsf{B},j}] \oplus d^{(i)}_j \cdot \mathsf{M}'_j[b^{(i)}_{\mathsf{B},j}] \oplus e^{(i)}_j \cdot \mathsf{M}'_j[a^{(i)}_{\mathsf{B},j}]$ to P_{A} and $\mathcal{V}$.
   (f) Each party (P_{A}, P_{B}, and $\mathcal{V}$) checks whether $\tilde{z}^{(i)}_{A,j} \oplus \tilde{z}^{(i)}_{B,j} \oplus d^{(i)}_j \cdot e^{(i)}_j = 0$.
2. On common inputs $\{\Delta'_{\mathsf{A},j}, \Delta'_{\mathsf{B},j}\}_{j\in[\kappa]}$ and inputs

$$(\mathsf{check}, \{\mathsf{K}'_j[x^{(i)}_{\mathsf{A},j}], \mathsf{K}'_j[y^{(i)}_{\mathsf{A},j}], \mathsf{K}'_j[z^{(i)}_{\mathsf{A},j}]\}_{j\in[\kappa],i\in[W]}, \{\mathsf{K}'_j[a^{(i)}_{\mathsf{A},j}], \mathsf{K}'_j[b^{(i)}_{\mathsf{A},j}], \mathsf{K}'_j[c^{(i)}_{\mathsf{A},j}]\}_{j\in[\kappa],i\in[W]})$$

   from P_{A} and

$$(\mathsf{check}, \{\mathsf{K}'_j[x^{(i)}_{\mathsf{B},j}], \mathsf{K}'_j[y^{(i)}_{\mathsf{B},j}], \mathsf{K}'_j[z^{(i)}_{\mathsf{B},j}]\}_{j\in[\kappa],i\in[W]}, \{\mathsf{K}'_j[a^{(i)}_{\mathsf{B},j}], \mathsf{K}'_j[b^{(i)}_{\mathsf{B},j}], \mathsf{K}'_j[c^{(i)}_{\mathsf{B},j}]\}_{j\in[\kappa],i\in[W]})$$

   from P_{B}, for $j \in [\kappa]$, $i \in [W]$, it works as follows:
   (a) P_{A} sends $\mathsf{K}'_j[x^{(i)}_{\mathsf{A},j}]$, $\mathsf{K}'_j[y^{(i)}_{\mathsf{A},j}]$, $\mathsf{K}'_j[z^{(i)}_{\mathsf{A},j}]$, $\{\mathsf{K}'_j[a^{(i)}_{\mathsf{A},j}]$, $\mathsf{K}'_j[b^{(i)}_{\mathsf{A},j}]$, and $\mathsf{K}'_j[c^{(i)}_{\mathsf{A},j}]$ to P_{B} and $\mathcal{V}$.
   (b) P_{B} sends $\mathsf{K}'_j[x^{(i)}_{\mathsf{B},j}]$, $\mathsf{K}'_j[y^{(i)}_{\mathsf{B},j}]$, $\mathsf{K}'_j[z^{(i)}_{\mathsf{B},j}]$, $\{\mathsf{K}'_j[a^{(i)}_{\mathsf{B},j}]$, $\mathsf{K}'_j[b^{(i)}_{\mathsf{B},j}]$, $\mathsf{K}'_j[c^{(i)}_{\mathsf{B},j}]$ to P_{A} and $\mathcal{V}$.
   (c) Each party (P_{A}, P_{B}, and $\mathcal{V}$) computes $\mathsf{K}'_j[d^{(i)}_{\mathsf{A},j}] := \mathsf{K}'_j[x^{(i)}_{\mathsf{A},j}] \oplus \mathsf{K}'_j[a^{(i)}_{\mathsf{A},j}]$, $\mathsf{K}'_j[e^{(i)}_{\mathsf{A},j}] := \mathsf{K}'_j[y^{(i)}_{\mathsf{A},j}] \oplus \mathsf{K}'_j[b^{(i)}_{\mathsf{A},j}]$ and checks whether $\mathsf{M}'_j[d^{(i)}_{\mathsf{A},j}] = \mathsf{K}'_j[d^{(i)}_{\mathsf{A},j}] \oplus d^{(i)}_{\mathsf{A},j} \cdot \Delta'_{\mathsf{B},j}$, $\mathsf{M}'_j[e^{(i)}_{\mathsf{A},j}] = \mathsf{K}'_j[e^{(i)}_{\mathsf{A},j}] \oplus e^{(i)}_{\mathsf{A},j} \cdot \Delta'_{\mathsf{B},j}$.
   (d) Each party (P_{A}, P_{B}, and $\mathcal{V}$) computes $\mathsf{K}'_j[d^{(i)}_{\mathsf{B},j}] := \mathsf{K}'_j[x^{(i)}_{\mathsf{B},j}] \oplus \mathsf{K}'_j[a^{(i)}_{\mathsf{B},j}]$, $\mathsf{K}'_j[e^{(i)}_{\mathsf{B},j}] := \mathsf{K}'_j[y^{(i)}_{\mathsf{B},j}] \oplus \mathsf{K}'_j[b^{(i)}_{\mathsf{B},j}]$ and checks whether $\mathsf{M}'_j[d^{(i)}_{\mathsf{B},j}] = \mathsf{K}'_j[d^{(i)}_{\mathsf{B},j}] \oplus d^{(i)}_{\mathsf{B},j} \cdot \Delta'_{\mathsf{A},j}$, $\mathsf{M}'_j[e^{(i)}_{\mathsf{B},j}] = \mathsf{K}'_j[e^{(i)}_{\mathsf{B},j}] \oplus e^{(i)}_{\mathsf{B},j} \cdot \Delta'_{\mathsf{A},j}$.
   (e) Each party (P_{A}, P_{B}, and $\mathcal{V}$) computes $\mathsf{K}'_j[\tilde{z}^{(i)}_{\mathsf{A},j}] := \mathsf{K}'_j[z^{(i)}_{\mathsf{A},j}] \oplus \mathsf{K}'_j[c^{(i)}_{\mathsf{A},j}] \oplus d^{(i)}_j \cdot \mathsf{K}'_j[b^{(i)}_{\mathsf{A},j}] \oplus e^{(i)}_j \cdot \mathsf{K}'_j[a^{(i)}_{\mathsf{A},j}]$ and checks whether $\mathsf{M}'_j[\tilde{z}^{(i)}_{\mathsf{A},j}] = \mathsf{K}'_j[\tilde{z}^{(i)}_{\mathsf{A},j}] \oplus \tilde{z}^{(i)}_{\mathsf{A},j} \cdot \Delta'_{\mathsf{B},j}$.
   (f) Each party (P_{A}, P_{B}, and $\mathcal{V}$) computes $\mathsf{K}'_j[\tilde{z}^{(i)}_{\mathsf{B},j}] := \mathsf{K}'_j[z^{(i)}_{\mathsf{B},j}] \oplus \mathsf{K}'_j[c^{(i)}_{\mathsf{B},j}] \oplus d^{(i)}_j \cdot \mathsf{K}'_j[b^{(i)}_{\mathsf{B},j}] \oplus e^{(i)}_j \cdot \mathsf{K}'_j[a^{(i)}_{\mathsf{B},j}]$ and checks whether $\mathsf{M}'_j[\tilde{z}^{(i)}_{\mathsf{B},j}] = \mathsf{K}'_j[\tilde{z}^{(i)}_{\mathsf{B},j}] \oplus \tilde{z}^{(i)}_{\mathsf{B},j} \cdot \Delta'_{\mathsf{A},j}$.

Fig. 7. Sub-Protocol $\Pi_{\mathsf{check\text{-}AND}}$.

Inputs: Both parties P_{A} and P_{B} agree on a common circuit for function $f : \{0,1\}^{|\mathcal{I}_{\mathsf{A}}|} \times \{0,1\}^{|\mathcal{I}_{\mathsf{B}}|} \to \{0,1\}^{|\mathcal{O}|}$. In the input-preprocessing phase, P_{A} and P_{B} holds $\mathbf{x} = (z_i)_{i \in \mathcal{I}_{\mathsf{A}}} \in \{0,1\}^{|\mathcal{I}_{\mathsf{A}}|}$ and $\mathbf{y} = (z_i)_{i \in \mathcal{I}_{\mathsf{B}}} \in \{0,1\}^{|\mathcal{I}_{\mathsf{B}}|}$, respectively. P_{A} and P_{B} also keeps common parameters κ and L. There is a verifier $\mathcal{V}$ verifying the computation of P_{A} and P_{B}.

Function-Independent Preprocessing:

1. P_{A} and P_{B} send init to $\mathcal{F}_{\mathsf{pre}}$ to receives Δ_{A} and Δ_{B}, respectively.
2. For each $w \in \mathcal{I}_{\mathsf{A}} \cup \mathcal{I}_{\mathsf{B}} \cup \mathcal{W}$, both P_{A} and P_{B} send random to $\mathcal{F}_{\mathsf{pre}}$. In return, $\mathcal{F}_{\mathsf{pre}}$ sends $(r_w, \mathsf{M}[r_w], \mathsf{K}[s_w])$ and $(s_w, \mathsf{M}[s_w], \mathsf{K}[r_w])$ to P_{A} and P_{B}, respectively. Define $\lambda_w := r_w \oplus s_w$. P_{A} picks a uniform κ-bit string $\mathsf{L}_{w,0}$ where κ denotes the computational security parameter.
3. Both P_{A} and P_{B} invokes $\mathcal{F}_{\mathsf{pre}}$ L times for L pairs of triples $(a^{(i)}_{\mathsf{A},j}, b^{(i)}_{\mathsf{A},j}, c^{(i)}_{\mathsf{A},j})$ and $(a^{(i)}_{\mathsf{B},j}, b^{(i)}_{\mathsf{B},j}, c^{(i)}_{\mathsf{B},j})$ satisfying $(a^{(i)}_{\mathsf{A},j} \oplus a^{(i)}_{\mathsf{B},j}) \cdot (b^{(i)}_{\mathsf{A},j} \oplus b^{(i)}_{\mathsf{B},j}) = c^{(i)}_{\mathsf{A},j} \oplus c^{(i)}_{\mathsf{B},j}$ for $i \in [L]$ and $j \in [\kappa]$.
4. P_{A} and P_{B} respectively send $(\mathsf{init}, \{r_w\}_{w \in \mathcal{I}_{\mathsf{A}} \cup \mathcal{I}_{\mathsf{B}} \cup \mathcal{W}}, \{r'_w\}_{w \in \mathcal{W}}, \{a^{(j)}_{\mathsf{A},i}, b^{(j)}_{\mathsf{A},i}, c^{(j)}_{\mathsf{A},i}\}_{j \in [\kappa], i \in [L]})$ and $(\mathsf{init}, \{s_w\}_{w \in \mathcal{I}_{\mathsf{A}} \cup \mathcal{I}_{\mathsf{B}} \cup \mathcal{W}}, \{s'_w\}_{w \in \mathcal{W}}, \{a^{(j)}_{\mathsf{B},i}, b^{(j)}_{\mathsf{B},i}, c^{(j)}_{\mathsf{B},i}\}_{j \in [\kappa], i \in [L]})$ to $\mathcal{F}_{\mathsf{sVOLE\text{-}2PC}}$. Then, P_{A} (respectively, P_{B}) receives $\{\mathsf{M}'_j[r_w]\}_{w \in \mathcal{I}_{\mathsf{A}} \cup \mathcal{I}_{\mathsf{B}} \cup \mathcal{W}}$, $\{\mathsf{M}'_j[r'_w]\}_{w \in \mathcal{W}}$, $\{\mathsf{M}'_j[a^{(j)}_{\mathsf{A},i}], \mathsf{M}'_j[b^{(j)}_{\mathsf{A},i}], \mathsf{M}'_j[c^{(j)}_{\mathsf{A},i}]\}_{j \in [\kappa], i \in [L]}$ (respectively, $\{\mathsf{M}'_j[s_w]\}_{w \in \mathcal{I}_{\mathsf{A}} \cup \mathcal{I}_{\mathsf{B}} \cup \mathcal{W}}$, $\{\mathsf{M}'_j[s'_w]\}_{w \in \mathcal{W}}$, $\{\mathsf{M}'_j[a^{(j)}_{\mathsf{B},i}], \mathsf{M}'_j[b^{(j)}_{\mathsf{B},i}], \mathsf{M}'_j[c^{(j)}_{\mathsf{B},i}]\}_{j \in [\kappa], i \in [L]}$).

Function-Dependent Preprocessing: This is divided into two phases:

(i) Phase 1: Following the topological ordering of the circuit, P_{A} and P_{B} work as follows:

1. For each XOR gate $(\alpha, \beta, \gamma, \oplus)$, P_{A} computes $(r_\gamma, \mathsf{M}[r_\gamma], \mathsf{K}[s_\gamma]) := (r_\alpha \oplus r_\beta, \mathsf{M}[r_\alpha] \oplus \mathsf{M}[r_\beta], \mathsf{K}[s_\alpha] \oplus \mathsf{K}[s_\beta])$ and $\mathsf{L}_{\gamma,0} := \mathsf{L}_{\alpha,0} \oplus \mathsf{L}_{\beta,0}$. P_{B} computes $(s_\gamma, \mathsf{M}[s_\gamma], \mathsf{K}[r_\gamma]) := (s_\alpha \oplus s_\beta, \mathsf{M}[s_\alpha] \oplus \mathsf{M}[s_\beta], \mathsf{K}[r_\alpha] \oplus \mathsf{K}[r_\beta])$.
2. For each AND gate $(\alpha, \beta, \gamma, \cdot)$:
   - P_{A} (respectively, P_{B}) sends $(\mathsf{and}, (r_\alpha, \mathsf{M}[r_\alpha], \mathsf{K}[s_\alpha]), (r_\beta, \mathsf{M}[r_\beta], \mathsf{K}[s_\beta]))$ (respectively, $(\mathsf{and}, (s_\alpha, \mathsf{M}[s_\alpha], \mathsf{K}[r_\alpha]), (s_\beta, \mathsf{M}[s_\beta], \mathsf{K}[r_\beta]))$) to $\mathcal{F}_{\mathsf{pre}}$. In return, $\mathcal{F}_{\mathsf{pre}}$ sends $(r'_\gamma, \mathsf{M}[r'_\gamma], \mathsf{K}[s'_\gamma])$ and $(s'_\gamma, \mathsf{M}[s'_\gamma], \mathsf{K}[r'_\gamma])$ respectively to P_{A} and P_{B}. Here, $r'_\gamma \oplus s'_\gamma = \lambda_\alpha \cdot \lambda_\beta$.
   - $\forall k \in [0,3]$, let $(k_0, k_1) := \mathsf{bin}(k) \in \{0,1\}^2$, i.e., $k = k_0 + 2 \cdot k_1$, and P_{A} computes
   $$r_{\gamma,k} := r'_\gamma \oplus r_\gamma \oplus k_0 \cdot r_\alpha \oplus k_1 \cdot r_\beta,$$
   $$\mathsf{M}[r_{\gamma,k}] := \mathsf{M}[r'_\gamma] \oplus \mathsf{M}[r_\gamma] \oplus k_0 \cdot \mathsf{M}[r_\beta] \oplus k_1 \cdot \mathsf{M}[r_\alpha],$$
   $$\mathsf{K}[r_{\gamma,k}] := \mathsf{K}[r'_\gamma] \oplus \mathsf{K}[r_\gamma] \oplus k_0 \cdot \mathsf{K}[r_\beta] \oplus k_1 \cdot \mathsf{K}[r_\alpha] \oplus k_0 \cdot k_1 \cdot \Delta_{\mathsf{A}}.$$
   - $\forall k \in [0,3]$, let $(k_0, k_1) := \mathsf{bin}(k) \in \{0,1\}^2$, i.e., $k = k_0 + 2 \cdot k_1$, and P_{B} computes
   $$s_{\gamma,k} := s'_\gamma \oplus s_\gamma \oplus k_0 \cdot s_\beta \oplus k_1 \cdot s_\alpha \oplus k_0 \cdot k_1,$$
   $$\mathsf{M}[s_{\gamma,k}] := \mathsf{M}[s'_\gamma] \oplus \mathsf{M}[s_\gamma] \oplus k_0 \cdot \mathsf{M}[s_\beta] \oplus k_1 \cdot \mathsf{M}[s_\alpha],$$
   $$\mathsf{K}[s_{\gamma,k}] := \mathsf{K}[s'_\gamma] \oplus \mathsf{K}[s_\gamma] \oplus k_0 \cdot \mathsf{K}[s_\beta] \oplus k_1 \cdot \mathsf{K}[s_\alpha].$$

(ii) Phase 2 (for Making VOLEitH Proof and Garbled Tables): For $w \in \mathcal{W}$, P_{A} and P_{B} use protocol Π_{fix} to respectively obtain $(r'_w, \mathsf{M}'[r'_w])$ and $(s'_w, \mathsf{M}[s'_w])$ where r'_w and s'_w are already achieved in Phase 1. Doing this requires each $\overline{r}'_w \oplus r'_w$ and $\overline{s}'_w \oplus s'_w$ to be published. Then, both parties run as follows:

- For each XOR gate $(\alpha, \beta, \gamma, \oplus)$, for $j \in [\kappa]$, P_{A} (respectively, P_{B}) computes $\mathsf{M}'_j[r_\gamma] := \mathsf{M}'_j[r_\alpha] \oplus \mathsf{M}'_j[r_\beta]$ (respectively, $\mathsf{M}'_j[s_\gamma] := \mathsf{M}'_j[s_\alpha] \oplus \mathsf{M}'_j[s_\beta]$).
- For each AND gate $(\alpha, \beta, \gamma, \cdot)$:
  - $\forall k \in [0,3]$, $\forall j \in [\kappa]$, let $(k_0, k_1) := \mathsf{bin}(k) \in \{0,1\}^2$, i.e., $k = k_0 + 2 \cdot k_1$, and P_{A} computes $\mathsf{M}'_j[r_{\gamma,k}] := \mathsf{M}'_j[r'_\gamma] \oplus \mathsf{M}'_j[r_\gamma] \oplus k_0 \cdot \mathsf{M}'_j[r_\beta] \oplus k_1 \cdot \mathsf{M}'_j[r_\alpha]$.
  - $\forall k \in [0,3]$, $\forall j \in [\kappa]$, let $(k_0, k_1) := \mathsf{bin}(k) \in \{0,1\}^2$, i.e., $k = k_0 + 2 \cdot k_1$, and P_{B} computes $\mathsf{M}'_j[s_{\gamma,k}] := \mathsf{M}'_j[s'_\gamma] \oplus \mathsf{M}'_j[s_\gamma] \oplus k_0 \cdot \mathsf{M}'_j[s_\beta] \oplus k_1 \cdot \mathsf{M}'_j[s_\alpha]$.

(iii) Phase 3 (for Encrypting and Committing): For each AND gate $(\alpha, \beta, \gamma, \cdot)$,

- P_{A} computes $\mathsf{L}_{\alpha,1} := \mathsf{L}_{\alpha,0} \oplus \Delta_{\mathsf{A}}$ and $\mathsf{L}_{\beta,1} := \mathsf{L}_{\beta,0} \oplus \Delta_{\mathsf{A}}$. P_{A} publishes the following, for all $k \in [0,3]$ with $(k_0, k_1) := \mathsf{bin}(k) \in \{0,1\}^2$:
  $$G_{\gamma,k} := H(\mathsf{L}_{\alpha,k_0}, \mathsf{L}_{\beta,k_1}, \gamma, k) \oplus (r_{\gamma,k}, \mathsf{M}[r_{\gamma,k}], \{\mathsf{M}'_j[r_{\gamma,k}]\}_{j \in [\kappa]}, \mathsf{L}_{\gamma,k} \oplus \mathsf{K}[s_{\gamma,k}] \oplus r_{\gamma,k} \cdot \Delta_{\mathsf{A}}).$$
- For $k \in [0,3]$, P_{B} commits to $(r_{\gamma,k}, \{\mathsf{M}'_j[r_{\gamma,k}]\}_{j \in [\kappa]})$ to obtain and publish $\mathsf{cma}_{\mathsf{B},\gamma,k}$.

(iv) Phase 4: For $w \in \mathcal{O}$, P_{A} commits to $(r_w, \{\mathsf{M}'_j[r_w]\}_{j \in [\kappa]})$ to obtain and publish $\mathsf{cmo}_{\mathsf{A},w}$.

Fig. 8. Protocol $\Pi_{\mathsf{pa\text{-}2PC\text{-}pre}}$ for Preprocessing for Publicly Auditable 2PC.

Proof Prepraration:

1. $\mathcal{V}$ samples and sends $\{\mathcal{I}_j\}_{j\in[\kappa]}$, where $\mathcal{I}_j \subseteq [L]$ and $|\mathcal{I}_j| = |\mathcal{W}|$ for $j \in [\kappa]$.
2. P_{A} publishes $\{a^{(j)}_{\mathsf{A},i}, b^{(j)}_{\mathsf{A},i}, c^{(j)}_{\mathsf{A},i}, \mathsf{M}'_j[a^{(j)}_{\mathsf{A},i}], \mathsf{M}'_j[b^{(j)}_{\mathsf{A},i}], \mathsf{M}'_j[c^{(j)}_{\mathsf{A},i}]\}_{j\in[\kappa], i\in[L]\setminus\mathcal{I}_j}$.
3. P_{B} publishes $\{a^{(j)}_{\mathsf{B},i}, b^{(j)}_{\mathsf{B},i}, c^{(j)}_{\mathsf{B},i}, \mathsf{M}'_j[a^{(j)}_{\mathsf{B},i}], \mathsf{M}'_j[b^{(j)}_{\mathsf{B},i}], \mathsf{M}'_j[c^{(j)}_{\mathsf{B},i}]\}_{j\in[\kappa], i\in[L]\setminus\mathcal{I}_j}$.
4. For each $j \in [\kappa]$ and each $i \in [L] \setminus \mathcal{I}_j$, each party ($P_{\mathsf{A}}$, P_{B}, and $\mathcal{V}$) locally checks whether
$$(a^{(i)}_{\mathsf{A},j} \oplus a^{(i)}_{\mathsf{B},j}) \cdot (b^{(i)}_{\mathsf{A},j} \oplus b^{(i)}_{\mathsf{B},j}) = c^{(i)}_{\mathsf{A},j} \oplus c^{(i)}_{\mathsf{B},j}.$$
5. P_{A} and P_{B} respectively send
$$(\mathsf{init}, \{r_\alpha, r_\beta, r'_\gamma, \mathsf{M}'_j[r_\alpha], \mathsf{M}'_j[r_\beta], \mathsf{M}'_j[r'_\gamma]\}_{j\in[\kappa],\ \text{all } (\alpha,\beta,\gamma,\cdot)},$$
$$\{a^{(i)}_{\mathsf{A},j}, b^{(i)}_{\mathsf{A},j}, c^{(i)}_{\mathsf{A},j}, \mathsf{M}'_j[a^{(i)}_{\mathsf{A},j}], \mathsf{M}'_j[b^{(i)}_{\mathsf{A},j}], \mathsf{M}'_j[c^{(i)}_{\mathsf{A},j}]\}_{j\in[\kappa], i\in\mathcal{I}_j}) \text{ and}$$
$$(\mathsf{init}, \{s_\alpha, s_\beta, s'_\gamma, \mathsf{M}'_j[s_\alpha], \mathsf{M}'_j[s_\beta], \mathsf{M}'_j[s'_\gamma]\}_{j\in[\kappa],\ \text{all } (\alpha,\beta,\gamma,\cdot)},$$
$$\{a^{(i)}_{\mathsf{B},j}, b^{(i)}_{\mathsf{B},j}, c^{(i)}_{\mathsf{B},j}, \mathsf{M}'_j[a^{(i)}_{\mathsf{B},j}], \mathsf{M}'_j[b^{(i)}_{\mathsf{B},j}], \mathsf{M}'_j[c^{(i)}_{\mathsf{B},j}]\}_{j\in[\kappa], i\in\mathcal{I}_j})$$
to protocol $\Pi_{\mathsf{check\text{-}AND}}$.
6. P_{A}, P_{B}, and $\mathcal{V}$ send (get) to $\mathcal{F}_{\mathsf{sVOLE\text{-}2PC}}$ to obtain $\{\Delta'_{\mathsf{A},j}\}_{j\in[\kappa]}$, $\{\mathsf{K}'_j[r_w]\}_{w\in\mathcal{I}_{\mathsf{A}}\cup\mathcal{I}_{\mathsf{B}}\cup\mathcal{W}}$, $\{\mathsf{K}'_j[r'_w]\}_{w\in\mathcal{W}}$, $\{\mathsf{K}'_j[a^{(j)}_{0,i}], \mathsf{K}'_j[b^{(j)}_{0,i}], \mathsf{K}'_j[c^{(j)}_{0,i}]\}_{j\in[\kappa],i\in[L]}$, $\{\Delta'_{\mathsf{B},j}\}_{j\in[\kappa]}$, $\{\mathsf{K}'_j[s_w]\}_{w\in\mathcal{I}_{\mathsf{A}}\cup\mathcal{I}_{\mathsf{B}}\cup\mathcal{W}}$, $\{\mathsf{K}'_j[s'_w]\}_{w\in\mathcal{W}}$, $\{\mathsf{K}'_j[a^{(j)}_{1,i}], \mathsf{K}'_j[b^{(j)}_{1,i}], \mathsf{K}'_j[c^{(j)}_{1,i}]\}_{j\in[\kappa],i\in[L]}$.
7. For each $j \in [\kappa]$, each $i \in [L] \setminus \mathcal{I}_j$, each $p \in \{\mathsf{A}, \mathsf{B}\}$, each party ($P_{\mathsf{A}}$, P_{B}, and $\mathcal{V}$) checks whether $\mathsf{M}'_j[a^{(i)}_{p,j}] = \mathsf{K}'_j[a^{(i)}_{p,j}] \oplus a^{(i)}_{p,j} \cdot \Delta'_{p',j}$, $\mathsf{M}'_j[b^{(i)}_{p,j}] = \mathsf{K}'_j[b^{(i)}_{p,j}] \oplus b^{(i)}_{p,j} \cdot \Delta'_{p',j}$ and $\mathsf{M}'_j[c^{(i)}_{p,j}] = \mathsf{K}'_j[c^{(i)}_{p,j}] \oplus c^{(i)}_{p,j} \cdot \Delta'_{p',j}$ where $p' = \mathsf{B}$, if $p = \mathsf{A}$, and $p' = \mathsf{A}$, otherwise.
8. The parties work as follows:
   (a) For each XOR gate $(\alpha, \beta, \gamma, \oplus)$, $\forall j \in [\kappa]$, each party (P_{A}, P_{B}, and $\mathcal{V}$) computes $\mathsf{K}'_j[r_\gamma] := \mathsf{K}'_j[r_\alpha] \oplus \mathsf{K}'_j[r_\beta]$ (respectively, $\mathsf{K}'_j[s_\gamma] := \mathsf{K}'_j[s_\alpha] \oplus \mathsf{K}'_j[s_\beta]$).
   (b) For each AND gate $(\alpha, \beta, \gamma, \cdot)$, $\forall k \in [0, 3]$, let $(k_0, k_1) := \mathsf{bin}(k) \in \{0, 1\}^2$, i.e., $k = k_0 + 2 \cdot k_1$, $\forall j \in [\kappa]$, each party (P_{A}, P_{B}, and $\mathcal{V}$) computes
$$\mathsf{K}'_j[r_{\gamma,k}] := \mathsf{K}'_j[r'_\gamma] \oplus \mathsf{K}'_j[r_\gamma] \oplus k_0 \cdot \mathsf{M}'_j[r_\alpha] \oplus k_1 \cdot \mathsf{K}'_j[r_\beta],$$
$$\mathsf{K}'_j[s_{\gamma,k}] := \mathsf{K}'_j[s'_\gamma] \oplus \mathsf{K}'_j[s_\gamma] \oplus k_0 \cdot \mathsf{M}'_j[s_\alpha] \oplus k_1 \cdot \mathsf{K}'_j[s_\beta].$$

Input Processing:

1. For $w \in \mathcal{I}_{\mathsf{A}}$, P_{A} publishes $(r_w, \mathsf{M}[r_w], (\mathsf{M}'_j[r_w])_{j\in[\kappa]})$ to P_{B}. P_{B} checks whether $\mathsf{M}[r_w] = \mathsf{K}[r_w] \oplus r_w \cdot \Delta_{\mathsf{B}}$ and $\mathsf{M}'_j[r_w] = \mathsf{K}'_j[r_w] \oplus r_w \cdot \Delta'_{\mathsf{B},j}\ \forall j \in [\kappa]$. P_{B} then publishes $\hat{z}_w = z_w \oplus \lambda_w = z_w \oplus r_w \oplus s_w$ to P_{A}. Finally, P_{A} publishes $\mathsf{L}_{w,\hat{z}_w}$.
2. For $w \in \mathcal{I}_{\mathsf{B}}$, P_{B} publishes $(s_w, \mathsf{M}[s_w], (\mathsf{M}'_j[s_w])_{j\in[\kappa]})$ to P_{A}. P_{A} checks whether $\mathsf{M}[s_w] = \mathsf{K}[s_w] \oplus s_w \cdot \Delta_{\mathsf{A}}$ and $\mathsf{M}'_j[s_w] = \mathsf{K}'_j[s_w] \oplus s_w \cdot \Delta'_{\mathsf{A},j}\ \forall j \in [\kappa]$. P_{A} then publishes $\hat{z}_w = z_w \oplus \lambda_w = z_w \oplus r_w \oplus s_w$ and $\mathsf{L}_{w,\hat{z}_w}$.

Circuit Evaluation:

- P_{B} evaluates the circuit following a topological order. For each gate $(\alpha, \beta, \gamma, \mathsf{op})$ where $\mathsf{op} \in \{\cdot, \oplus\}$, P_{B} holds $(\hat{z}_\alpha, \mathsf{L}_{\alpha,\hat{z}_\alpha})$ and $(\hat{z}_\beta, \mathsf{L}_{\beta,\hat{z}_\beta})$ where $\hat{z}_\alpha = z_\alpha \oplus \lambda_\alpha$ and $\hat{z}_\beta = z_\beta \oplus \lambda_\beta$.
  - If $\mathsf{op} = \oplus$, P_{B} computes $\hat{z}_\gamma := \hat{z}_\alpha \oplus \hat{z}_\beta$ and $\mathsf{L}_{\gamma,\hat{z}_\gamma} := \mathsf{L}_{\alpha,\hat{z}_\alpha} \oplus \mathsf{L}_{\beta,\hat{z}_\beta}$.
  - If $\mathsf{op} = \cdot$, P_{B} computes $k := 2\hat{z}_\alpha + \hat{z}_\beta$. Then, P_{B} computes $(r_{\gamma,k}, \mathsf{M}[r_{\gamma,k}], \{\mathsf{M}'[r_{\gamma,k}]\}_{j\in[\kappa]}, \mathsf{L}_{\gamma,0} \oplus \mathsf{K}[s_{\gamma,k}] \oplus r_{\gamma,k} \cdot \Delta_{\mathsf{A}}) := G_{\gamma,k} \oplus H(\mathsf{L}_{\alpha,\hat{z}_\alpha}, \mathsf{L}_{\beta,\hat{z}_\beta}, \gamma, k)$. Then, P_{B} checks whether $\mathsf{M}'_j[r_{\gamma,k}] = \mathsf{K}'_j[r_{\gamma,k}] \oplus r_{\gamma,k} \cdot \Delta'_{\mathsf{B},j}\ \forall j \in [\kappa]$. If so, P_{B} computes and publishes $\hat{z}_\gamma = s_{\gamma,k} \oplus r_{\gamma,k}$ and $\mathsf{L}_{\gamma,\hat{z}_\gamma} := (\mathsf{L}_{\gamma,0} \oplus \mathsf{K}[s_{\gamma,k}] \oplus r_{\gamma,k} \cdot \Delta_{\mathsf{A}}) \oplus \mathsf{M}[s_{\gamma,k}]$. P_{B} opens commitment $\mathsf{cma}_{\mathsf{B},\gamma,k}$ to publish $(s_{\gamma,k}, \{\mathsf{M}'_j[s_{\gamma,k}]\}_{j\in[\kappa]})$. Then, P_{A} and $\mathcal{V}$ checks whether $\mathsf{M}'_j[s_{\gamma,k}] = \mathsf{K}'_j[s_{\gamma,k}] \oplus s_{\gamma,k} \cdot \Delta'_{\mathsf{B},j}\ \forall j \in [\kappa]$.

Output Determination:

- For $w \in \mathcal{O}$, P_{A} opens cmo_w to obtain $(r_w, \{\mathsf{M}'_j[r_w]\}_{j\in[\kappa]})$ and publishes $(r_w, \mathsf{M}[r_w], \{\mathsf{M}'_j[r_w]\}_{j\in[\kappa]})$. P_{B} and $\mathcal{V}$ check whether $\mathsf{M}[r_w] = \mathsf{L}[r_w] \oplus r_w \cdot \Delta_{\mathsf{B}}$ and $\mathsf{M}'_j[r_w] = \mathsf{L}'_j[r_w] \oplus r_w \cdot \Delta'_{\mathsf{B},j}\ \forall j \in [\kappa]$, and whether $\{\mathsf{M}'_j[r_w]\}_{j\in[\kappa]}$ is a valid opening of cmo_w. If so, P_{B} computes $z_w := \hat{z}_w \oplus r_w \oplus s_w$.
- P_{A} and P_{B} respectively send
$$(\mathsf{check}, \{\mathsf{K}'_j[r_\alpha], \mathsf{K}'_j[r_\beta], \mathsf{K}'_j[r'_\gamma]\}_{j\in[\kappa],\ \text{all } (\alpha,\beta,\gamma,\cdot)}, \{\mathsf{K}'_j[a^{(i)}_{\mathsf{A},j}], \mathsf{K}'_j[b^{(i)}_{\mathsf{A},j}], \mathsf{K}'_j[c^{(i)}_{\mathsf{A},j}]\}_{j\in[\kappa],i\in\mathcal{I}_j}) \text{ and}$$
$$(\mathsf{check}, \{\mathsf{K}'_j[s_\alpha], \mathsf{K}'_j[s_\beta], \mathsf{K}'_j[s'_\gamma]\}_{j\in[\kappa],\ \text{all } (\alpha,\beta,\gamma,\cdot)}, \{\mathsf{K}'_j[a^{(i)}_{\mathsf{B},j}], \mathsf{K}'_j[b^{(i)}_{\mathsf{B},j}], \mathsf{K}'_j[c^{(i)}_{\mathsf{B},j}]\}_{j\in[\kappa],i\in\mathcal{I}_j})$$
to $\Pi_{\mathsf{check\text{-}AND}}$ for verifying the validity of authenticated values used for proving AND gates.

Fig. 9. Protocol $\Pi_{\mathsf{pa\text{-}2PC\text{-}eval}}$ for Evaluation for Publicly Auditable 2PC.

References

1. Aguilar-Melchor, C., Gama, N., Howe, J., Hülsing, A., Joseph, D., Yue, D.: The Return of the SDitH. In: Advances in Cryptology – EUROCRYPT 2023. Lecture Notes in Computer Science, vol. 14008, pp. 564–596. Springer Nature Switzerland (2023). https://doi.org/10.1007/978-3-031-30589-4_20
2. Aguilar-Melchor, C., Hülsing, A., Joseph, D., Majenz, C., Ronen, E., Yue, D.: SDitH in the QROM. In: Advances in Cryptology – ASIACRYPT 2023. Lecture Notes in Computer Science, vol. 14444, pp. 317–350. Springer Nature Singapore (2023). https://doi.org/10.1007/978-981-99-8739-9_11
3. Araki, T., Barak, A., Furukawa, J., Lichter, T., Lindell, Y., Nof, A., Ohara, K., Watzman, A., Weinstein, O.: Optimized Honest-Majority MPC for Malicious Adversaries — Breaking the 1 Billion-Gate Per Second Barrier. In: 2017 IEEE Symposium on Security and Privacy – S&P 2017. pp. 843–862. IEEE (2017). https://doi.org/10.1109/SP.2017.15
4. Baldimtsi, F., Kiayias, A., Zacharias, T., Zhang, B.: Crowd Verifiable Zero-Knowledge and End-to-End Verifiable Multiparty Computation. In: Advances in Cryptology – ASIACRYPT 2020. Lecture Notes in Computer Science, vol. 12493, pp. 717–748. Springer International Publishing (2020). https://doi.org/10.1007/978-3-030-64840-4_24
5. Baum, C., Braun, L., Munch-Hansen, A., Razet, B., Scholl, P.: Appenzeller to Brie: Efficient Zero-Knowledge Proofs for Mixed-Mode Arithmetic and Z2k. In: Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security – CCS 2021. p. 192–211. Association for Computing Machinery (2021). https://doi.org/10.1145/3460120.3484812
6. Baum, C., Braun, L., Munch-Hansen, A., Scholl, P.: Moz$\mathbb{Z}_{2^k}$arella: Efficient Vector-OLE and Zero-Knowledge Proofs over $\mathbb{Z}_{2^k}$. In: Advances in Cryptology – CRYPTO 2022. Lecture Notes in Computer Science, vol. 13510, pp. 329–358. Springer Nature Switzerland (2022). https://doi.org/10.1007/978-3-031-15985-5_12
7. Baum, C., Braun, L., de Saint Guilhem, C.D., Klooß, M., Orsini, E., Roy, L., Scholl, P.: Publicly Verifiable Zero-Knowledge and Post-Quantum Signatures from VOLE-in-the-Head. In: Advances in Cryptology – CRYPTO 2023. Lecture Notes in Computer Science, vol. 14085, pp. 581–615. Springer Nature Switzerland (2023). https://doi.org/10.1007/978-3-031-38554-4_19
8. Baum, C., Damgård, I., Orlandi, C.: Publicly Auditable Secure Multi-Party Computation. In: Security and Cryptography for Networks – SCN 2014. Lecture Notes in Computer Science, vol. 8642, pp. 175–196. Springer International Publishing (2014). https://doi.org/10.1007/978-3-319-10879-7_11
9. Baum, C., Malozemoff, A.J., Rosen, M.B., Scholl, P.: Mac′n′Cheese: Zero-knowledge proofs for boolean and arithmetic circuits with nested disjunctions. In: Advances in Cryptology – CRYPTO 2021. Lecture Notes in Computer Science, vol. 12828, pp. 92–122. Springer International Publishing (2021). https://doi.org/10.1007/978-3-030-84259-8_4
10. Baum, C., Nof, A.: Concretely-Efficient Zero-Knowledge Arguments for Arithmetic Circuits and Their Application to Lattice-Based Cryptography. In: Public-Key Cryptography – PKC 2020. Lecture Notes in Computer Science, vol. 12110, pp. 495–526. Springer International Publishing (2020). https://doi.org/10.1007/978-3-030-45374-9_17
11. Bui, D.: Shorter VOLEitH Signature from Multivariate Quadratic. Cryptology ePrint Archive, Paper 2024/465 (2024), https://eprint.iacr.org/2024/465

12. Carozza, E., Couteau, G., Joux, A.: Short Signatures from Regular Syndrome Decoding in the Head. In: Advances in Cryptology – EUROCRYPT 2023. Lecture Notes in Computer Science, vol. 14008, pp. 532–563. Springer Nature Switzerland (2023). https://doi.org/10.1007/978-3-031-30589-4_19
13. Chase, M., Derler, D., Goldfeder, S., Orlandi, C., Ramacher, S., Rechberger, C., Slamanig, D., Zaverucha, G.: Post-Quantum Zero-Knowledge and Signatures from Symmetric-Key Primitives. In: Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security – CCS 2017. p. 1825–1842. Association for Computing Machinery (2017). https://doi.org/10.1145/3133956.3133997
14. Cui, H., Liu, H., Yan, D., Yang, K., Yu, Y., Zhang, K.: ReSolveD: Shorter Signatures from Regular Syndrome Decoding and VOLE-in-the-Head. In: Public-Key Cryptography – PKC 2024. Lecture Notes in Computer Science, vol. 14601, pp. 229–258. Springer Nature Switzerland, Cham (2024). https://doi.org/10.1007/978-3-031-57718-5_8
15. Cui, H., Wang, X., Yang, K., Yu, Y.: Actively Secure Half-Gates with Minimum Overhead Under Duplex Networks. In: Advances in Cryptology – EUROCRYPT 2023. Lecture Notes in Computer Science, vol. 14005, pp. 35–67. Springer Nature Switzerland (2023). https://doi.org/10.1007/978-3-031-30617-4_2
16. Dittmer, S., Ishai, Y., Lu, S., Ostrovsky, R.: Improving Line-Point Zero Knowledge: Two Multiplications for the Price of One. In: Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security – CCS 2022. p. 829–841. Association for Computing Machinery (2022). https://doi.org/10.1145/3548606.3559385
17. Dittmer, S., Ishai, Y., Ostrovsky, R.: Line-Point Zero Knowledge and Its Applications. In: 2nd Conference on Information-Theoretic Cryptography – ITC 2021. Leibniz International Proceedings in Informatics (LIPIcs), vol. 199. Schloss Dagstuhl – Leibniz-Zentrum für Informatik (2021). https://doi.org/10.4230/LIPIcs.ITC.2021.5
18. Evans, D., Kolesnikov, V., Rosulek, M.: A Pragmatic Introduction to Secure Multi-Party Computation. Found. Trends Priv. Secur. **2**(2-3), 70–246 (2018). https://doi.org/10.1561/3300000019
19. Feneuil, T., Joux, A., Rivain, M.: Syndrome Decoding in the Head: Shorter Signatures from Zero-Knowledge Proofs. In: Advances in Cryptology – CRYPTO 2022. Lecture Notes in Computer Science, vol. 13508, pp. 541–572. Springer Nature Switzerland (2022). https://doi.org/10.1007/978-3-031-15979-4_19
20. Fiat, A., Shamir, A.: How To Prove Yourself: Practical Solutions to Identification and Signature Problems. In: Advances in Cryptology — CRYPTO 1986. Lecture Notes in Computer Science, vol. 263, pp. 186–194. Springer Berlin Heidelberg (1987). https://doi.org/10.1007/3-540-47721-7_12
21. Ishai, Y., Kushilevitz, E., Ostrovsky, R., Sahai, A.: Zero-knowledge from secure multiparty computation. In: Proceedings of the Thirty-Ninth Annual ACM Symposium on Theory of Computing – STOC 2007. p. 21–30. Association for Computing Machinery (2007). https://doi.org/10.1145/1250790.1250794
22. Jarecki, S., Shmatikov, V.: Efficient two-party secure computation on committed inputs. In: Advances in Cryptology - EUROCRYPT 2007. Lecture Notes in Computer Science, vol. 4515, pp. 97–114. Springer Berlin Heidelberg (2007). https://doi.org/10.1007/978-3-540-72540-4_6
23. Katz, J., Kolesnikov, V., Wang, X.: Improved Non-Interactive Zero Knowledge with Applications to Post-Quantum Signatures. In: Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security – CCS 2018.

p. 525–537. Association for Computing Machinery (2018). https://doi.org/10.1145/3243734.3243805
24. Katz, J., Ranellucci, S., Rosulek, M., Wang, X.: Optimizing authenticated garbling for faster secure two-party computation. In: Advances in Cryptology – CRYPTO 2018. Lecture Notes in Computer Science, vol. 10993, pp. 365–391. Springer International Publishing (2018). https://doi.org/10.1007/978-3-319-96878-0_13
25. Ozdemir, A., Boneh, D.: Experimenting with Collaborative zk-SNARKs: Zero-Knowledge Proofs for Distributed Secrets. In: 31st USENIX Security Symposium – USENIX Security 2022. pp. 4291–4308. USENIX Association (2022), https://www.usenix.org/conference/usenixsecurity22/presentation/ozdemir
26. Roy, L.: SoftSpokenOT: Quieter OT Extension from Small-Field Silent VOLE in the Minicrypt Model. In: Advances in Cryptology – CRYPTO 2022. Lecture Notes in Computer Science, vol. 13507, pp. 657–687. Springer Nature Switzerland (2022). https://doi.org/10.1007/978-3-031-15802-5_23
27. Delpech de Saint Guilhem, C., Orsini, E., Tanguy, T.: Limbo: Efficient Zero-knowledge MPCitH-based Arguments. In: Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security – CCS 2021. p. 3022–3036. Association for Computing Machinery (2021). https://doi.org/10.1145/3460120.3484595
28. Wang, X., Ranellucci, S., Katz, J.: Authenticated Garbling and Efficient Maliciously Secure Two-Party Computation. In: Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security – CCS 2017. p. 21–37. Association for Computing Machinery (2017). https://doi.org/10.1145/3133956.3134053
29. Weng, C., Yang, K., Katz, J., Wang, X.: Wolverine: Fast, Scalable, and Communication-Efficient Zero-Knowledge Proofs for Boolean and Arithmetic Circuits. In: 2021 IEEE Symposium on Security and Privacy – S&P 2021. pp. 1074–1091. IEEE (2021). https://doi.org/10.1109/SP40001.2021.00056
30. Weng, C., Yang, K., Xie, X., Katz, J., Wang, X.: Mystique: Efficient Conversions for Zero-Knowledge Proofs with Applications to Machine Learning. In: 30th USENIX Security Symposium – USENIX Security 2021. USENIX Association (2021), https://www.usenix.org/conference/usenixsecurity21/presentation/weng
31. Weng, C., Yang, K., Yang, Z., Xie, X., Wang, X.: AntMan: Interactive Zero-Knowledge Proofs with Sublinear Communication. In: Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security – CCS 2022. p. 2901–2914. Association for Computing Machinery (2022). https://doi.org/10.1145/3548606.3560667
32. Yang, K., Sarkar, P., Weng, C., Wang, X.: QuickSilver: Efficient and Affordable Zero-Knowledge Proofs for Circuits and Polynomials over Any Field. In: Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security – CCS 2021. p. 2986–3001. Association for Computing Machinery (2021). https://doi.org/10.1145/3460120.3484556