EFFICIENT FAIR AND ROBUST SPDZ-LIKE MULTI-PARTY COMPUTATION

Chung-Li Wang
Alibaba Inc., Hangzhou City, Zhejiang Province, China

ABSTRACT

Effective multi-party computation protocols have been developed, but concerns regarding privacy and correctness persist. Classic results demonstrate that guaranteed output delivery can be achieved by assuming fairness and identifiable abort. However, if the majority is malicious, it is still challenging to design an efficient implementation that can deliver correct outputs while maintaining robustness and fairness. To address this issue, we have redesigned the secret-sharing mechanism and employed a semi-trusted third party (TTP) as the key manager to provide optimistic backup for output delivery. The verification and delivery procedures prevent the malicious parties from "stealing" the output, when there is at least one honest party. Furthermore, the TTP has no knowledge of output, so even if he is malicious and colluding, we only lose fairness. The decryption is needed only when misconduct is detected. Our scheme also enables identified abort for offline preprocessing, and the audit of the offline sub-protocols can be publicly performed, holding corrupted parties accountable before receiving private inputs. With fairness and identifiable abort, output delivery is guaranteed by excluding the cheaters.

KEYWORDS

Efficient Multi-Party Computation, Public Verifiability, Robustness, Fairness, Semi-Trusted Third Party

1. INTRODUCTION

In recent years, secure multi-party computation (MPC) has gained widespread attention from researchers due to its capability to securely aggregate data from multiple users and produce powerful results. SPDZ [1] and its variants [2 – 6] have played significant roles in the success of MPC. These protocols have two phases: an input-independent offline phase and an input-dependent online phase that is highly efficient. However, a single malicious party can cause the computation to fail by deviating from the protocol or providing false results. In this scenario, honest parties are unable to learn the outcome of the computation, while the malicious party may still gain access to it. To address this issue, numerous works have been developed to enhance the security of SPDZ by identifying malicious parties and imposing penalties for their actions.

Security Model. The client-server model is a widely used approach in which clients provide inputs to servers and receive outputs from them. The servers perform the requested computations without learning the clients' inputs. However, the clients in this model need to trust the servers to execute the desired function securely, and they typically have limited ability to guarantee that the servers deliver the correct results. To mitigate this issue, publicly identifiable abort is often utilized to ensure the correct execution of the protocol ([8 – 10]). This involves three steps: 1) verifying that the servers' outputs are correct, 2) aborting the protocol if the result is incorrect, and 3) identifying the misbehaving server that causes the abort, which must be held accountable by everyone including the clients, servers, and external parties known as auditors or verifiers. This allows clients to stay simple while providing strong incentives for the servers to follow the protocol honestly instead of cheating.

David C. Wyld et al. (Eds): ICDIPV, CBIoT, ICAIT, WIMO, NC, CRYPIS, ITCSE, NLCA, CAIML -2023
pp. 347-366, 2023. CS & IT - CSCP 2023 DOI: 10.5121/csit.2023.131327

Fairness and Robustness. The combination of public accountability and robustness, for which corrupted parties should not be able to prevent honest parties from receiving their output, is an emerging topic. This property, called "fairness," is investigated in [11] and proven to imply robustness assuming a broadcast channel. In general, this property can be achieved by using cryptographic tools such as [12 – 14]. However, even with optimistic models and semi-trusted third parties, for SPDZ-like protocols, these similar approaches were all considered to be unaffordable, as discussed in [5]. As a consequence, the protocols with identifiable abort for [5], [10], and [15] have to publicly reconstruct the secret before verifying the result, allowing corrupted parties to learn the output even though they cheated. The work in [16] instead uses threshold-t secret-sharing to support robustness. Despite of its better efficiency and usability, when the number of dishonest parties is not known, it is difficult to choose appropriate design parameters. For example, in its design only t server parties are needed to corrupt to obtain the secret without public opening. As a result, even with identifiable abort, if a malicious majority is assumed, having robustness and fairness is still an open problem.

One Honest Server. SPDZ-like protocols may lose the privacy and even security properties, when all parties are malicious. Taking the auditable SPDZ of [3] as an example, we find that it cannot detect cheating during the generation of multiplication triples when no server is honest. A fix to this issue would be worthless, because it might make the whole scheme inefficient. Therefore, we assume the presence of one honest server throughout our work.

Our Contribution. The proposed solution, called the "Multiplicative-Ciphered Secret-Sharing" (MUSS) scheme, implements a public auditable MPC with identifiable abort and fairness using an optimistic decryption phase. The framework is similar as SDPZ and has the offline preprocessing phase and online computation phase. Compared to the auditable SPDZ protocol in [3], our protocol has the similar online complexity with identifiable abort and fairness if no misconduct occurs. We also have much lower complexity than that of the protocol in [16], as our scheme does not need zero-knowledge proofs for the shares in the online phase.

Robust and Fair Online Phase. Parties that engage in manipulating the computation can be identified by examining their commitments and may be excluded from the protocol. A semi-trusted third party (TTP) manages a global key pair of public-key encryption (PKE) for the multiplicative ciphers, not the shares. The decryption is not necessary if all parties behave honestly for opening the shares. Our scheme provides better privacy than [17] as the secret aggregation or decryption is performed by the computing parties, not the TTP. The inability of adversaries to know the output until successful delivery enables the player-elimination technique proposed in [11] to achieve fairness and robustness. Besides, if the TTP is dishonest, we only lose fairness and robustness, and the protocol is still public auditable with identifiable abort.

Efficient Offline Phase. Our scheme improves upon previous approaches Overdrive's LowGear [6] by utilizing homomorphic addition to compute MUSS shares of random values and triples in a single, trackable routine. The correctness of the computation is guaranteed by verifying the double-sharing correlation using a zero-knowledge proof (ZKP). This creates a checkpoint specifically for the offline phase, adding an extra layer of security and auditability. In addition, our verification covers the generation of multiplicative triples and thus does not need an additional information-theoretic check as in [3].

Short Proof. It is necessary to ensure the correctness of each share, when correlated randomness is generated in the preprocessing phase. This can be guaranteed through the use of a short ZKP from each party, which enables a speedy start to the subsequent online phase. The approach, which is based on knowledge-of-exponent assumptions (KEA) [18] and short-pairing [19] allows for the verification of multiple shares generated during the offline phase as well as commitments in a batch.

Related Works. In the standard model, it is impossible to guarantee fairness with corrupted majority [20]. To overcome this impossibility, many protocols have been proposed to achieve fairness in non-standard models. In [12], the synchronization using a global ledger makes parties releasing their outputs in a timely manner. However, this technique compensates the honest party only with earnings (e.g., Bitcoins), so it is not really fair. Another approach involves employing semi-trusted third parties or physical assumptions [13]. It was recently shown that fair computation can be achieved by applying a multi-party fair exchange protocol in [14], [21], and [22], in which the exchange uses ciphertexts of output and requires a third party to generate a global key pair. As the encryption of shares demands excessive overhead in the online phase, this way is too expensive for SPDZ protocols. A robust MPC with identifiable abort is proposed in [16], where robustness is obtained by t-secure secret-sharing and needs ZKP for every share opening in the online phase. A cryptographic solution, described in [17], is suggested to achieve fairness, but it incurs high complexity and relies on an external party for decryption. Although our scheme also uses a TTP to provide encryption keys, it requires the ciphertexts of multiplicative ciphers in the offline phase, such that the online phase does not need to invoke cryptographic functionalities for the computation. The critical path of online phase stays simple when no misbehaviour happens. Besides, the robustness is implied by fairness and identifiable abort, for which the generation of ZKP is performed only once in the offline phase.

2. OVERVIEW

2.1. Security Model

Before designing the specific implementation, we must first establish the security model. Secure computation in the standalone model is defined through the real-ideal world paradigm. Throughout this paper, we will consider protocols that are executed over a synchronous network with static and rushing adversaries. In the real world, all parties communicate through the protocol Π, while in the ideal world, the parties send their inputs to an ideal functionality $\mathcal{F}$, also known as the trusted party, which computes the desired function $\mathcal{C}$ and returns the result to the parties. In informal terms, the protocol Π is considered to securely realize the functionality $\mathcal{F}$ if, for every real-world adversary $\mathcal{A}$, there exists an ideal-world adversary $\mathcal{S}$ (also known as the simulator) such that the joint output distribution of the honest parties and the adversary $\mathcal{A}$ in the real world is indistinguishable from the joint output distribution of the honest parties and $\mathcal{S}$ in the ideal world.

The security requirements of the protocol are defined through the concept of ideal functionality with Public Accountability with Output Fairness (PAOF). In this setup, there is a polynomial-time honest party P_A, that can retrieve all the output messages from the trusted party, assess their correctness, and output the correct result and/or a set of parties L, that are deemed responsible for any misbehavior. The output of the protocol is in the form of a 2-tuple $(y, \perp)$, (y, L), or $(\perp, L)$.

The Ideal Model with PAOF. Assume $\mathcal{P} = \{P_i\}_{i\in\{n\}}$ to be the set of computing server parties, $\mathcal{I} = \{I_k\}_{k\in\{m\}}$ the set of input client parties, $D \subset \mathcal{P}$ the set of corrupted computing parties, and $\mathcal{D}_I \subseteq \mathcal{I}$ the set of corrupted input parties. Before the execution, the non-adaptive adversary $\mathcal{A}$ decides $\mathcal{L}_I \subseteq \mathcal{D}_I$ and $\mathcal{L}_f, \mathcal{L}_p, \mathcal{L}_o, \mathcal{L}_k \subseteq \mathcal{D}$. Let $\mathcal{L}_I$ be the set of input parties hanging or giving ill-formed inputs, $\mathcal{L}_p$ be the set of computing parties manipulating the computation results, $\mathcal{L}_o$ be the set of computing parties cheating with the ciphertexts of MUSS ciphers, and $\mathcal{L}_k$ be the set of computing parties cheating with the plaintexts of MUSS ciphers. With $m \leq n$, the evaluation function $\mathcal{C}$ has m input and m output gates. The execution of ideal model with PAOF is denoted as $\mathcal{F}_{\text{Online}}$, which is briefly described as follows.

Inputs: The *i*-th party's input is denoted by x_i and $\boldsymbol{x} = (x_1 \dots, x_n)$. We assume that all valid inputs are defined in $\mathbb{F}$. The adversary receives an auxiliary input z.

Initialization: The trusted party informs the adversary $\mathcal{A}$ of the beginning of execution with the parameter set $(\mathcal{C}, \mathbb{F}, \mathbb{G})$. $\mathcal{A}$ sends the lists of malicious parties that corrupt the input and evaluation outcome to the trusted party. This decision is made by $\mathcal{A}$ and may depend on $(\mathcal{C}, \mathbb{F}, \mathbb{G})$ and the auxiliary input z. If misconduct is detected, the trusted party will catch $\mathcal{L}_f$ and abort the process.

Send Inputs to Trusted Party: Any honest party I_i sends its input x_i to the trusted party. The corrupted parties, controlled by $\mathcal{A}$, may either send their received input or send some other input to the trusted party. This decision is made by $\mathcal{A}$ and may depend on the input from the corrupted parties and the auxiliary input. If the invalid input is from $I_i \in \mathcal{L}_I$, the trusted party will catch I_i and abort the process.

Compute: For the i-th gate $f_i \in \mathcal{C}$, the trusted party computes $y_i = f_i(\boldsymbol{x})$ for computation gates, and for output gates it sets the outcome to $\perp$ with replying **Reject** if the computation or output is corrupted. Otherwise it replies **Accept**.

Trusted Party Answers Auditor: Upon the request by the auditor, the trusted party outputs $\mathcal{L}_p$ with **Reject** or replies **Accept** if no cheating is found.

Open: Upon the request by all parties, the trusted party outputs the result $\boldsymbol{y}$, if no misbehaviour occurs. Or it sends out $\mathcal{L}_o$ if the ciphertexts of MUSS ciphers fail the verification, and it sends $\mathcal{L}_k$ if the plaintexts of MUSS ciphers are incorrect. Then if the TTP P_T is honest, $\boldsymbol{y}$ will be delivered fairly to all parties. If P_T actively corrupts the decryption, the ideal model only loses fairness, and P_T will be identified.

The Real Model with PAOF. Let us consider the real model in which a real n-party protocol Π is executed with the set of n computing parties, m input parties, and trusted honest parties P_A and P_T. Let $\mathcal{D}$ and $\mathcal{D}_I$ denote the set of corrupted computing and input parties, controlled by an adversary $\mathcal{A}$. In this case, the adversary $\mathcal{A}$ sends all messages in place of corrupted parties, and may decide a polynomial-time strategy arbitrarily. In contrast, the honest parties follow the instructions of Π. Then the real execution of Π on inputs $\boldsymbol{x}$, auxiliary input z to $\mathcal{A}$, and security parameter λ, denoted by $\text{Real}_{\Pi,\mathcal{A}(z),\{\mathcal{D},\mathcal{D}_I\}}(\boldsymbol{x},\lambda)$, is defined as the output vector of the honest parties and the adversary $\mathcal{A}$ from the real execution of Π.

With the ideal-real model, the PAOF can be defined as follows:

Definition 1 *(PAOF): Let $\mathcal{C}$ be a circuit with inputs $\boldsymbol{x}$. A protocol Π is called publicly accountable with output fairness whenever one computing party, P_A, and P_T are honest, for every non-uniform probabilistic polynomial-time adversary $\mathcal{A}$ for the real model, there exist a non-uniform probabilistic polynomial-time adversary S for the ideal model $\mathcal{F}_{\text{Online}}$ such that for every $\mathcal{D} \subset \mathcal{P}, \mathcal{D}_I \subseteq \mathcal{I}$, every balanced vector $\boldsymbol{x} \in \mathbb{F}^m$, and every auxiliary input $z \in \mathbb{F}$:*

$$\text{Ideal}_{\mathcal{F}_{\text{Online}},S(z),\{\mathcal{D},\mathcal{D}_I\}}(\boldsymbol{x},\lambda) =^c \text{Real}_{\Pi,\mathcal{A}(z),\{\mathcal{D},\mathcal{D}_I\}}(\boldsymbol{x},\lambda)$$

Robustness. It should be noted that cheating input parties can only be detected if their inputs are invalid. Any corrupted inputs from the adversary in the ideal functionality correspond to true inputs. Our strategy is to iterate the protocol by excluding the parties which misbehave in the previous iteration. In this study, we assume that the adversary has a static strategy, with the set of malicious input parties determined at the beginning of the protocol execution and remaining unchanged. The set of malicious computing servers, on the other hand, is determined at the beginning of each iteration and may change in the next iteration. To accommodate robustness, the model first has to include the property that the output is fairly delivered as long as one computing party and the TTP are honest. Because the ideal model $\mathcal{F}_{\text{Online}}$ itself does not provide robustness in any sense, the following theorem will be proven in the full version by constructing a protocol in the $\mathcal{F}_{\text{Online}}$-hybrid model.

Theorem 1 *(Strong Robustness): Assume that the adversary $\mathcal{A}$ has a static strategy that $\mathcal{L}_f, \mathcal{L}_p, \mathcal{L}_o, \mathcal{L}_k$, and $\mathcal{L}_I$ are determined before the execution of every iteration. Let $\mathcal{C}$ be a circuit with predetermined inputs $\tilde{\boldsymbol{x}} \in \mathbb{F}^m$ and output $\boldsymbol{y} \in \mathbb{F}^m$. Let $\boldsymbol{x}^* = (x_1^* \dots, x_m^*)$ with $x_i^* = \tilde{x}_i$ if $x_i = \perp$ or $x_i^* = x_i$. If there is a protocol with PAOF, there exists a protocol with PAOF to output $\boldsymbol{y} = \mathcal{C}(\boldsymbol{x}^*)$ for overwhelming probability, with polynomially bounded environments E, whenever one computing party, P_A, and P_T are honest.*

2.2. Important Blocks

As noted in the introduction, our scheme utilizes a unique combination of secret-sharing and proof mechanisms, allowing for public auditing and fair delivery of the results. In this section, we will briefly outline our new secret-sharing technique and provide an overview of our MPC protocol.

Multiplicative-Ciphered Secret-Sharing. Combining the concepts of MAC and repetition, we devise an innovative scheme to share the secret. Assume that a distributive encryption and decryption are set up. Taking Alice and Bob as an instance, we show how to generate correlated randomness without the input.

Example 1 (MUSS): Assume the ciphertext is denoted as $[\![\cdot]\!]$ with homomorphic addition defined as $[\![a]\!] \oplus [\![b]\!] = [\![a + b]\!]$. Alice randomly samples γ_a and δ_a and broadcasts $[\![\gamma_a]\!]$ and $[\![\delta_a]\!]$; Bob samples γ_b and δ_b and broadcasts $[\![\gamma_b]\!]$ and $[\![\delta_b]\!]$. So Alice and Bob both have $[\![\gamma]\!]$ and $[\![\delta]\!]$ for $\gamma = \gamma_a + \gamma_b$ and $\delta = \delta_a + \delta_b$. The encryption is done by a distributive scheme of PKE with a key pair that everyone agrees on. Then Alice samples d and broadcasts $[\![\gamma]\!] \cdot d$; Bob samples c and f and broadcasts $[\![\delta]\!] \cdot c - f$. Publicly both have $[\![e]\!] \leftarrow [\![\gamma]\!] \cdot d + [\![\delta]\!] \cdot c - f$. The correlated randomness between plain and encoded shares ($a = e + f = \gamma d + \delta c$) for secret a is obtained by letting Alice call the distributive decryption of $[\![e]\!]$, so e is decrypted privately to her. As shown above, the multiplicative ciphers are hiding from both parties. When reconstructing a, semi-honest Alice and Bob announce *e* and *f*, respectively, or instead they can announce *d* and *c* and call for the distributive decryption of $[\![\gamma]\!]$ and $[\![\delta]\!]$.

In the scenario of a single malicious party, our scheme is able to detect deviations from the protocol by checking the correlation of double-shared randomness. This check is based on the correlation between the encoded and plain shares and can be verified using an information-theoretic method or ZKP which supports linear operations.

Fairness. The sharing scheme of SPDZ has two issues regarding fairness: the first is the opening of secrets prior to checking the MAC, and the second is the unfairness in output delivery. While many works have addressed the first issue, it is still a challenge to address both simultaneously. Our scheme adopts techniques based on information-theoretic security to ensure that the output is properly opened to all parties or not disclosed in the event of an abort otherwise.

Secret Opening. As suggested in [5] for SPDZ, a check to test the correlation of MUSS shares, e.g., $a = e + f = \gamma d + \delta c$, can serve as the first step of audit. As the MUSS sharing ciphers γ and δ are additively shared and committed by Alice or Bob, anyone cannot alter the shares without failing the check. Since if they could, they would be able to guess the MUSS ciphers. For privacy-preserving, the parties can only open the encoded shares d and c. Using the information-theoretic technique similar as the one used for MAC [2], we can check the correlation without opening the MUSS ciphers. The opening of γ and δ will not happen until the initialization and computation stages are finished without any error detected. For a practical implementation, we design each cipher γ to be kept by all parties in three versions: 1. committed additive shares in plaintexts, 2. ciphertexts by a distributive scheme of PKE (1. and 2. are already mentioned in the example), and additionally, 3. Ciphertext $[\![\gamma_a]\!]_{pko}$ and $[\![\gamma_b]\!]_{pko}$, which are encrypted by a PKE scheme using a global key pair (sko, pko). The global keys are managed by a TTP. Note that $[\![\cdot]\!]_{pko}$ and $[\![\cdot]\!]$ use different PKE key pairs. Once all parties agree to open the secret, Alice and Bob exchange $[\![\gamma_a]\!]_{pko}$,

$[\![\delta_a]\!]_{pko}$, $[\![\gamma_b]\!]_{pko}$, and $[\![\delta_b]\!]_{pko}$, and the relation of the three versions can be verified by running a ZKP scheme. If ciphertexts pass the check, the shares of MUSS ciphers and sko will be broadcasted from Alice, Bob, and the TTP, respectively. All parties can verify the plaintext by opening the commitment and catch the malicious party who cheats in the process.

Security and Privacy. Assuming the sharing of two secrets $a = \gamma d + \delta c$ and $a' = \gamma d' + \delta c'$, when opening the encoded shares (d, c) and (d', c'), the question is that, if there is any advantage of guessing the secret a and a'. The answer is no. Since $\begin{bmatrix} d & c \\ d' & c' \end{bmatrix}$ is a full-rank matrix with a probability close to 1 [29], a and a' are indistinguishable to two independent uniform random samples. Formally, we have the following theorem to show the security of opening encoded shares:

Theorem 2 *(Perfect Secrecy): Assume a cipher* $\mathcal{H} = (\mathbb{F}^m, \mathbb{F}^n, KG, \Phi, \Psi)$ *with message space* $\mathbb{F}^m$ *and key space* $\mathbb{F}^n$ *that a probabilistic PTTM* Φ: $\mathbb{F}^m \times \mathbb{F}^n \to \mathbb{F}^{mn}$ *and* Ψ: $\mathbb{F}^{mn} \times \mathbb{F}^n \to \mathbb{F}^m$ *with the definition* $\Psi(\mathbf{D}, \mathbf{g}) \to \mathbf{D}\mathbf{g}^T = \boldsymbol{a} = (a_1 \dots, a_m)$ with $\mathbf{D} = \{d_{i,j}\}_{i\in\{m\}, j\in\{n\}}$ *and* $\mathbf{g} = (g_1 \dots g_n)$ *for* $m \leq n$. *If* $\mathbf{D}$ *has full rank, and* $\mathbf{g}$ *is statistically indistinguishable from samples drawn from uniform random distribution in* $\mathbb{F}^n$, *the scheme* $\mathcal{H}$ *has perfect secrecy except a negligible probability.*

Proof: We can prove perfect secrecy by showing $P(\mathbf{g} \leftarrow KG: \Phi(\boldsymbol{a}, \mathbf{g}) = \mathbf{D} | \mathbf{D}, \boldsymbol{a}) = P(\mathbf{g} \leftarrow KG: \Phi(\boldsymbol{a}', \mathbf{g}) = \mathbf{D} | \mathbf{D}, \boldsymbol{a}')$, except a negligible probability. For every pair $\boldsymbol{a}$ and $\boldsymbol{a}'$ we always can find a vector $\mathbf{g}_1$ such that $\boldsymbol{a} = \mathbf{D}\,\mathbf{g}_1^T$ and $\mathbf{g}_2$ such that $\boldsymbol{a}' = \mathbf{D}\,\mathbf{g}_2^T$. The probability to have such $\mathbf{g}_1$ is $P(\mathbf{g} \leftarrow KG: \Phi(\boldsymbol{a}, \mathbf{g}) = \mathbf{D} | \mathbf{D}, \boldsymbol{a}) = \sum_{\mathbf{g}_1} P(\mathbf{g}_1, \boldsymbol{a} = \mathbf{D}\,\mathbf{g}_1^T | \mathbf{D}, \boldsymbol{a}) = \sum_{\boldsymbol{a}=\mathbf{D}\,\mathbf{g}_1^T} P(\mathbf{g}_1) = \sum_{\boldsymbol{a}=\mathbf{D}\,\mathbf{g}_1^T} 1/|\mathbb{F}|^n$, which is equal to that to have $\mathbf{g}_2$ such $\boldsymbol{a} = \mathbf{D}\,\mathbf{g}_2^T$. It leads to perfect secrecy of $\mathcal{H}$. □

By the security proofs in the full version of the paper it will be demonstrated that our protocol keeps the matrix $\mathbf{D}$ full ranked except a negligible probability.

The additive sharing with MAC in SPDZ is vulnerable to corruption by two collusive parties who lie about their shares without altering the sum. This renders Lemma 1 in [3] false, as the parties can deviate from the protocol and still pass the check. Despite this, the corruption can still be detected during the audit, and therefore, it does not undermine the security proof of [3]. However, the maliciously controlled share values can reduce the security level and lead to information leakage, which may give an advantage to an eavesdropper. Our work overcomes this issue by using random MUSS ciphers, resulting in a negligible success probability of such cheating.

3. THE PROTOCOL

Let $\mathbb{G}$ be some Abelian multiplicative subgroup of order q where the DLP is hard to solve (with respect to a given computational security parameter λ). The protocol will evaluate a circuit $\mathcal{C}$ over $\mathbb{F} = \mathbb{Z}_q$ whereas we use the group $\mathbb{G}$ to commit to the output. We let $g, h \in \mathbb{G}$ be two generators of the group $\mathbb{G}$ where g and h are chosen by some random oracle with a common reference string (CRS) as the input.

We assume a secure point-to-point network between all parties and a broadcast functionality. We also use the commitment functionality $\mathcal{F}_{\text{Com}}$, the random oracle $\mathcal{F}_{\text{Rnd}}$ for giving a random value over $\mathbb{F}$ to all parties, and the bulletin $\mathcal{F}_{\text{Blt}}$ to handle all communication, such that nothing in the bulletin can ever be changed or erased. These functionalities are outlined in Figure 1.

3.1. Secret-Sharing Scheme

The online phase of the computation is conducted using the MUSS scheme, which is defined as below:

Definition 2 *(MUSS): Let x, y, e* $\in \mathbb{F}$, $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_n)$ *and then the Multiplicative-Ciphered Secret-Sharing of x is defined as* $[x]_{\boldsymbol{\alpha}} = \big((x_1, \ldots, x_n), (\tilde{x}_1, \ldots, \tilde{x}_n)\big)$ *, where the correlation* $x = \sum_{i=1}^{n} \alpha_i x_i = \sum_{i=1}^{n} \tilde{x}_i$ *holds. Since the keys* $\boldsymbol{\alpha}$ *are fixed for the whole session,* $[x]_{\boldsymbol{\alpha}}$ *can be denoted as* $[x]$ *without confusion. Each player* P_i *will hold its MUSS shares* x_i *and* $\tilde{x}_i$ *of* $[x]$*. The key* α_i *for* P_i *is additively shared by all players, such that every player has* α_{ij} *and* $\alpha_i = \sum_{i=1}^{n} \alpha_{ij}$*. Moreover, we define* $[x] + [y] = \big((x_1 + y_1, \ldots, x_n + y_n), (\tilde{x}_1 + \tilde{y}_1, \ldots, \tilde{x}_n + \tilde{y}_n)\big)$ *,* $e \cdot [x] = \big((e \cdot x_1, \ldots, e \cdot x_n), (e \cdot \tilde{x}_1, \ldots, e \cdot \tilde{x}_n)\big)$*. We say that* $[x] \triangleq [y]$ *if the shares of x, y in* $[x]$*,* $[y]$ *reconstruct to the same value.*

Obviously, MUSS is linear. If all parties agree to apply one of defined linear functions, then they can perform these on the MUSS shares without interaction. For the addition between the MUSS share and a public value e, one needs to open a random MUSS share (e.g. $[r]$) as a gadget, so $[e + x] = [x] + (er^{-1}) \cdot [r]$.

3.2. Commitment Scheme

The proposed protocol forces the result given by the computing parties to be bound by a public witness. First, the parties have to commit the input by sending commitment to the bulletin. Since the commitment scheme uses a one-way function with homomorphic property, the expected commitment of output can be derived by a public auditor. The ways to catch the cheater include checking if each share opens the commitments correctly (as in [3]), and letting the party provide ZKP to prove its ability to give the correct decommitment (as in [16]). Our commitment scheme has a similar format as in [3]: we carry both the MUSS share of secret $[x]$ as well as the MUSS share of randomness $[r]$ of the commitment throughout the whole computation. The commitment handle to a value x is a Pedersen commitment $\mathsf{E}_{(g,h)}(x, r) = g^x h^r$ with $\mathsf{E}_{(g,h)}([x], [r]) = \Big((g^{x_1} h^{r_1}, \ldots, g^{x_n} h^{r_n}), \big(g^{\tilde{x}_1} h^{\tilde{r}_1}, \ldots, g^{\tilde{x}_n} h^{\tilde{r}_n}\big)\Big)$. When opening MUSS shares, we reconstruct the secret through either x_i or $\tilde{x}_i$, and the randomness (r_i or $\tilde{r}_i$) is also revealed. For simplicity, since (g, h) is fixed within one session, $\mathsf{E}_{(g,h)}([x], [r])$ can be denoted as $\mathsf{E}([x], [r])$. As discussed in [5], the computation of commitments is excluded in the circuit evaluation and invoked only after the failure of information-theoretic checks. This "on-demand" scheme yields favorable saving, especially when the adversary cheats at a lower rate in a large circuit.

3.3. Online Phase

The online phase of our protocol uses $\mathcal{F}_{\text{Offline}}$ for offline preprocessing that is demonstrated in the full version of the paper. The commands of $\mathcal{F}_{\text{Offline}}$ support single-instruction multiple-data (SIMD) processing with factors σ_f. Taking m inputs, the circuit $\mathcal{C}$ over $\mathbb{F}$ has ν_{in} input gates, ν_{mul} multiplication gates, and m output gates, with $m \leq n$, the number of computing parties. The online phase is presented in Figure 2 and Figure 3, which evaluates the circuit $\mathcal{C}$ of m input gates and m output gates. The stages **Input** and **Compute** are executed for each input and function gate of $\mathcal{C}$, respectively, and **Initialization**, **Audit**, and **Open** are invoked only once per circuit.

Initialization. The ideal functionality of the offline phase $\mathcal{F}_{\text{Offline}}$ sets up the MUSS ciphers. The commitment scheme obtains the key from the random oracle $\mathcal{K}$. The public-key infrastructure (PKI) is given by $\mathcal{H}$ and will be elaborated in Sec. 3.5.1. The TTP publishes the global public key pko. Each computing party P_j privately keeps the additive shares $\alpha_{i,j}$ for $i \in \{n\}$, where we set $\sum_{i \in \{n\}} \alpha_{i,j} = \alpha_i$. With $\overline{\boldsymbol{\alpha}}_j = \{\alpha_{i,j}\}_{i \in \{n\}}$, P_j commits to $\overline{\boldsymbol{\alpha}}_j$ toward $\mathcal{F}_{\text{Blt}}$ by $d_j = \mathsf{E}^{(n)}_{(\boldsymbol{g},h)}(\overline{\boldsymbol{\alpha}}_j, \beta_j)$ and encrypts $(\overline{\boldsymbol{\alpha}}_j, \beta_j)$ to have $c_j = [\![(\overline{\boldsymbol{\alpha}}_j, \beta_j)]\!]_{pko}$ along with its ZKP ζ_j to show the same plaintexts of c_j and d_j. The generation and verification of ζ_j will be provided in Sec. 3.5.3. Finally, the protocol asks the functionality $\mathcal{F}_{\text{Offline}}$ to generate random values and multiplication triples. $\mathcal{F}_{\text{Offline}}$ has its own check and audit for the output to ensure each player to have the correct share values as they

$\mathcal{F}_{\text{Com}}$

Commit: On input $(\mathbf{commit}, v, r, i, j, id)$ by P_i, where both v and r are either in $\mathbb{F}$ or $\perp$, and id is a unique ID, if v and r are either in $\mathbb{F}$ or $\perp$, store (v, r, i, j, id) on a list and outputs (i, id) to P_j. Otherwise output $(\perp, P_i)$ to $\mathcal{A}$.
Open: On input $(\mathbf{open}, i, j, id)$ by P_i, output (v, r, i, j, id) to P_j. If $(\mathbf{no\_open}, i, j, id)$ is given by a dishonest $P_i \in \mathcal{P}$, output $(\perp, \perp, i, j, id)$ to P_j.

$\mathcal{F}_{\text{Rnd}}$

Let $\mathbb{F}$ be a field such that there exists a PPT TM to efficiently sample value $r \in \mathbb{F}$ uniformly at random.

Random sample: Upon receiving $(\mathbf{rand}, \mathbb{F})$ from all parties, sample a uniform $r \in \mathbb{F}$ and output $(\mathbf{rand}, r)$ to all parties.

$\mathcal{F}_{\text{Blt}}$

Store: On input $(\mathbf{store}, id, x)$ from $P_i \in \mathcal{P}$:
Case 1: If (id, i, y) is stored, reply **Reject**.
Case 2: If not, send (id, i, x) to $\mathcal{A}$ and store it. reply **Accept**.
Read: On input $(\mathbf{read}, j, id)$ from P_i:
Case 1: if (id, j, x) is stored for some P_j, reply x.
Case 2: if (id, j, x) is not stored, reply **Reject**.

Figure 1. Ideal functionalities for the commitment, random oracle, and public bulletin.

committed to. If the misconduct is detected in $\mathcal{F}_{\text{Offline}}$, the malicious parties will be identified as L_f, and the protocol will be aborted.

Input. Each input client party in I is allowed to submit a value to the computation, where two random values are secretly opened to it. The client can then check that the commitment is correct, and blinds its input using the opened values. Here the protocol can only detect the blatant cheating, such as hanging or ill-formed input, we cannot prevent the malicious input client from giving an incorrect blinded input.

Compute (Add and **Multiply).** The protocol uses the linearity of the MUSS shares to perform linear operations on the shared values, and multiplies two representations using the multiplication triples from the preprocessing using the circuit randomization technique [23]. The multiplication requires to reconstruct values, and this is done by only opening the plain shares to keep the ciphers private. We do not check the recovered values in this stage and defer the check to the output gate.

Compute (Out). First, we check all the multiplications in the circuit by $\Pi^{\sigma}_{\text{ChkPln}}$ (Figure 6 and cf. Sec. 3.3.2) for the opened plain shares, where checking $\langle \eta \rangle$, $\langle \rho \rangle$, and $\langle t \rangle$ takes random values $\langle \eta' \rangle$, $\langle \rho' \rangle$, and $\langle t' \rangle$ as additional input. Then the encoded shares of output are published, and the correlation is checked by using the protocol $\Pi^{\sigma}_{\text{ChkEnc}}$ (Figure 4 and cf. Sec. 3.3.1). If any of them fails, the auditor P_A will invoke **Audit**. If both checks output **Accept**, all parties will invoke **Open** to output the result.

Audit. There are two audit procedures in the online protocol, which will be invoked by P_A when the precedented information-theoretic checks fail. One is to check the plain shares opened in **Multiply**, and the other is to check the encode shares for output delivery. If the audit passes, it means that the encoded shares are correct, and we are still going to **Open**. Please be noted that we do not identify anyone regarding the misbehavior happening in $\Pi^{\sigma}_{\text{ChkEnc}}$ and $\Pi^{\sigma}_{\text{ChkPln}}$ since it eventually does not prevent opening.

Π_{Online}

The parties evaluate the circuit $\mathcal{C}$ over $\mathbb{F}$, which has ν_{in} input gates and ν_{mul} multiplication gates. Every party is given $\mathcal{F}_{\text{KGD}}$, RO $\mathcal{K}$ with the CRS as input to choose the generator $g_0, g_1, \ldots, g_n, h \in \mathbb{G}$, and RO $\mathcal{Z}$ to verify ZKP. σ_f is the offline SIMD factor. Set $\boldsymbol{g}=\{g_i\}_{i=(0,\ldots,n)}$.

Initialize: On input $(\textbf{Init}, \mathcal{C}, \mathbb{F}, \mathbb{G})$ from all parties.

1) The parties send $(\textbf{Init}, \mathbb{F}, \mathbb{G}, \boldsymbol{g}, h)$ to $\mathcal{F}_{\text{Offline}}$. If $\mathcal{F}_{\text{Offline}}$ replies **Accept,** P_T has the global key pair (pko, sko). Each P_j in $\mathcal{P}$ randomly generates and commits to $\boldsymbol{\alpha}_j$ with randomness β_j to get d_j.
2) The parties choose the smallest $\nu_{\text{r}} \geq (2\nu_{\text{in}} + 4\nu_{\text{mul}})$, $\nu_{\text{trp}} \geq \nu_{\text{mul}}$ such that σ_f divides both ν_{r} and ν_{trp}. Then send $(\textbf{Single}, \sigma_f, g_0, h)$ $(\nu_{\text{r}}/\sigma_f)$ times, $(\textbf{Triple}, \sigma_f, g_0, h)$ $(\nu_{\text{trp}}/\sigma_f)$ times to $\mathcal{F}_{\text{Offline}}$.
3) Send $(\textbf{Audit}, \sigma_f)$ $(\nu_{\text{r}}/\sigma_f + \nu_{\text{trp}}/\sigma_f)$ times to $\mathcal{F}_{\text{Offline}}$.
4) If $\mathcal{F}_{\text{Offline}}$ replies **Accept**, all parties have random values $\langle x \rangle$, $\langle y \rangle$ (for **Input**), $\langle t \rangle$ (for **Multiply**), $\langle \eta' \rangle$, $\langle \rho' \rangle$, and $\langle t' \rangle$ (for Π^t_{ChkPln}) and multiplication triple $(\langle a \rangle, \langle b \rangle, \langle c \rangle)$.
5) Otherwise if $\mathcal{F}_{\text{Offline}}$ replies **Reject** and $\mathcal{L}_f$, then the protocol is aborted with output $(\perp, \mathcal{L}_f)$.

Input: On input $(\textbf{Input}, I_i, id(u_i), u_i)$ from each $I_i \in \mathcal{I}$, $i \in \{m\}$ and $(\textbf{Input}, I_i, id(u_i))$ from each $P_i \in \mathcal{P}$, with $id(u_i)$ a new ID and $u_i \in \mathbb{F}$ using a new random value $\langle x \rangle = ([x], [p], \varepsilon_{\langle x \rangle})$ and $\langle y \rangle = ([y], [r], \varepsilon_{\langle y \rangle})$.

1) I_i privately receives $(\tilde{x}_j, \tilde{p}_j)$ and $(\tilde{y}_j, \tilde{r}_j)$ for each $j \in \{n\}$, and checks the commitment. It broadcasts r_i such that $u_i = r_i \cdot x + y$.
2) All players check if r_i is valid. If not, add I_i to $\mathcal{L}_I$, and then protocol is aborted by replying $(\perp, \mathcal{L}_I)$. Or get $\langle u_i \rangle = r_i \cdot \langle x \rangle + \langle y \rangle$ and reply **Accept**.

Compute: On the input $(\textbf{Compute}, \mathcal{C})$ from all parties. If **Initialize** has been executed and inputs for all input wires of $\mathcal{C}$ have been assigned, evaluate every $f \in \mathcal{C}$ as follows:

Add: For two values $\langle x \rangle$, $\langle y \rangle$ with $id(x)$ and $id(y)$.

1) All players locally compute $\langle z \rangle = \langle x \rangle + \langle y \rangle$. Assign a new $id(z)$.

Multiply: Multiply two values $\langle x \rangle$, $\langle y \rangle$ with $id(x)$ and $id(y)$ using a random value $\langle t \rangle$ and multiplication triple $\langle a \rangle$, $\langle b \rangle$, and $\langle c \rangle$. The output is $\langle z \rangle$ with a newly assigned $id(z)$.

1) The players calculate $\langle \eta \rangle = \langle x \rangle - \langle a \rangle$ and $\langle \rho \rangle = \langle y \rangle - \langle b \rangle$.
2) The players reconstruct $\langle \eta \rangle = ([\eta], [\chi], \varepsilon_{\langle \eta \rangle})$, $\langle \rho \rangle = ([\rho], [\mu], \varepsilon_{\langle \rho \rangle})$, and $\langle t \rangle = ([t], [s], \varepsilon_{\langle t \rangle})$ by only opening $(\tilde{\eta}_i, \tilde{\chi}_i)$, $(\tilde{\rho}_i, \tilde{\mu}_i)$, and $(\tilde{t}_i, \tilde{s}_i)$ for each $i \in \{n\}$. Open these results to $\mathcal{F}_{\text{BLT}}$.
3) Each player locally calculates $\langle z \rangle = \langle c \rangle + \rho \cdot \langle a \rangle + \eta \cdot \langle b \rangle + r \cdot \langle t \rangle$, such that $r \cdot t = \eta \cdot \rho$.

Output: The output is $\langle z_k \rangle$ with an already assigned $\text{id}(z_k)$ for each $k \in \{m\}$.

1) The parties open the shares of $\langle z_k \rangle$ toward $\mathcal{F}_{\text{BLT}}$.
2) Run Π^t_{ChkPln} for the previously opened $\langle x_1 \rangle, \ldots \langle x_t \rangle$.
3) Run Π^m_{ChkEnc} for $\{\langle z_k \rangle\}_{k \in \{m\}}$. If any check fails, reply **Reject**. Or reply **Accept**.

Figure 2. Π_{Online}: Protocol for the online phase (Part 1).

Open. Once all the parties agree that encoded shares are correct, each computing P_j will broadcast the ciphertext c_j, commitment d_j, and ZKP ζ_j to all the other parties, so all parties can verify the

Audit: On the input $(\mathbf{Audit}, \{id(z_k)\}_{k\in\{m\}})$ from P_A.

1) Run Π_{Audit} for $\langle x_1\rangle, \ldots \langle x_t\rangle$ if Π^t_{ChkPln} failed. Run Π_{Audit} for $\{\langle z_k\rangle\}_{k\in\{m\}}$ if Π^m_{ChkEnc} failed in **Output**.
2) If it passes, P_A replies **Accept**. Or it identifies cheaters and outputs $(\perp, \mathcal{L}_p)$. Stop.

Open: On the input $(\mathbf{Open})$ from all parties. Given an RO $\mathcal{Z}$ to verify ZKP. Set a flag $\text{Cheat} \leftarrow \perp$.

1) P_j broadcasts c_j and ZKP ζ_j for all other P_i and P_A to check, for $i \in \{n\}\backslash j$. If it passes, replies **Accept**. Or P_A identifies cheaters and outputs $(\perp, \mathcal{L}_p)$, stop.
2) P_T broadcasts sko, and P_j opens $\boldsymbol{\alpha_j}, \beta_j$, and c_j toward $\mathcal{F}_{\text{BLT}}$. All parties check if $\boldsymbol{\alpha_j}$ and β_j can correctly open d_j. If it fails, add P_j to $\mathcal{L}_k$.
3) All parties check if sko can decrypts c_j correctly. If it fails, P_A set $\text{Cheat} \leftarrow \top$.
4) The output depends on the following conditions:
   4.1. If $\text{Cheat} = \perp$, reply $(\boldsymbol{z}, \mathcal{L}_k)$ to all parties.
   4.2. If $\text{Cheat} = \top$ and $|L_k| > 0$, reply $(\perp, \{\mathcal{L}_k, P_T\})$ to all parties.
   4.3. If $\text{Cheat} = \top$ and $|L_k| = 0$, reply $(\boldsymbol{z}, P_T)$ to all parties.

Figure 3. Π_{Online}: Protocol for the online phase (Part 2).

correctness using an RO $\mathcal{Z}$ (cf. Sec 3.5.3). If the check fails, the process will be aborted here. If c_j is correct, P_T opens the global secret key, and all computing parties release the plaintext shares $\alpha_{i,j}$. If P_T gives the correct key, or the plaintexts are correct, the result will be known to everyone. Otherwise P_T or malicious parties that give the corrupted key, ciphertexts, or plaintexts will be identified.

The security of Π_{Online} is proven in the UC framework by the following theorem.

Theorem 3 *(Online Security): In the $(\mathcal{F}_{\text{Offline}}, \mathcal{F}_{\text{Blt}}, \mathcal{F}_{\text{Com}}, \mathcal{F}_{\text{KGD}})$-hybrid model with random oracles $\mathcal{K}$ and $\mathcal{Z}$, the protocol Π_{Online} implements $\mathcal{F}_{\text{Online}}$ with computational security against any static adversary corrupting all parties except one computing party and the auditor P_A if the DLP is hard in the group $\mathbb{G}$.*

Next, we introduce present how to check the correlation without opening the cipher, and how to check the opened plain shares used in **Multiply**.

3.3.1 Check and Audit for Encoded Shares

In the online phase, we use a purely information-theoretic check as the first step of verification. The advantage of checking the correlation before the audit is lower complexity for optimistic models. Moreover, since the correctness of shares is eventually verified by the audit, we will not identify the cheater that corrupts the correlation check. This keeps the design simple.

Correlation of Shares. MAC check in SPDZ guarantees that the correct secret can be recovered from the sum of opened shares. This is weaker in sense of security, because it does not guarantee the correctness of each share. However, MUSS provides a stronger security: the check using the correlation of MUSS shares, that is $\sum_{i=1}^{n} \alpha_i x_i = \sum_{i=1}^{n} \tilde{x}_i$, guarantees that all parties have the correct share values except probability $o(1/q)$. Let us formally define the following property.

Theorem 4 *(MUSS Correlation): Let $\mathbb{F}$ be a field of order p and $[x]$ be the MUSS share of x. If the shares $\tilde{x}_i$, x_i, and α_{ij} were opened correctly, the MUSS correlation will hold, which is $\sum_{i=1}^{n} \alpha_i x_i = \sum_{i=1}^{n} \tilde{x}_i$. Assume that at least one server is honest, if any server cheats on share values, the MUSS correlation will not hold except probability $o(1/q)$.*

$\Pi^{\sigma}_{\mathrm{ChkEnc}}$

Given $z_i^{(k)}$ and $r_i^{(k)}$ from $\mathcal{F}_{\mathrm{BLT}}$ for $i \in \{n\}$, $k \in \{\sigma\}$. Set $\mathrm{id}(\mathbf{z})=\{\mathrm{id}(z^{(k)})\}_{k\in\{\sigma\}}$.

Check Encoded Shares: On input (**ChkEnc**, $\mathrm{id}(\mathbf{z})$, σ) from all parties.

1) The parties use $\mathcal{F}_{\mathrm{Rnd}}$ to publicly sample a vector $\boldsymbol{w} \xleftarrow{\$} \mathbb{F}^{\sigma}$.
2) Each $P_j \in P$ publicly computes $z_i^{\boldsymbol{w}} = \sum_{k=1}^{\sigma} z_i^{(k)} \cdot w_k$ and $r_i^{\boldsymbol{w}} = \sum_{k=1}^{\sigma} r_i^{(k)} \cdot w_k$ for each $j \in \{n\}$ and sends toward $\mathcal{F}_{\mathrm{BLT}}$.
3) Each $P_j \in P$ privately computes $\tilde{z}_j^{\boldsymbol{w}} = \sum_{k=1}^{\sigma} \tilde{z}_j^{(k)} \cdot w_k$ and $\tilde{r}_j^{\boldsymbol{w}} = \sum_{k=1}^{\sigma} \tilde{r}_j^{(k)} \cdot w_k$. Then it computes and $\eta_i = \sum_{j=1}^{n} z_j^{\boldsymbol{w}} \cdot \alpha_{j,i} - \tilde{z}_j^{\boldsymbol{w}}$ and $\mu_i = \sum_{j=1}^{n} r_j^{\boldsymbol{w}} \cdot \alpha_{j,i} - \tilde{r}_j^{\boldsymbol{w}}$.
4) Each $P_i \in P$ uses $\mathcal{F}_{\mathrm{Com}}$ to commit to η_i and μ_i.
5) Each $P_i \in P$ uses $\mathcal{F}_{\mathrm{Com}}$ to open η_i and μ_i to all parties.
6) All parties compute and output $\eta = \sum_{i=1}^{n} \eta_i$ and $\mu = \sum_{i=1}^{n} \mu_i$.
7) If $\eta = \mu = 0$, all parties output **Accept**. Else output **Reject**.

Figure 4. $\Pi^{\sigma}_{\mathrm{ChkEnc}}$: Protocol for the correlation check of encoded shares.

Proof: The adversary has no information about α_i for all i. Consequently, the probability of successful cheating is $1/q$ such that the correlation is valid by guessing α_i and setting $\tilde{x}_i \leftarrow \tilde{x}_i'$ and then $x_i \leftarrow x_i+(\tilde{x}_i' - \tilde{x}_i)/\alpha_i$ with $\tilde{x}_i' \neq \tilde{x}_i$. By setting $x_i \leftarrow x_i'$ with $x_i \neq x_i'$ and then $\tilde{x}_i \leftarrow \tilde{x}_i+(x_i' - x_i) \cdot \alpha_i$, the probability of successful cheating is also $1/q$. □

The MUSS correlation can be used to verify the opened shares, and thus we call this "correlation check" and use it as the first step of the delivery, playing the same role of MAC in [5] as an effective way to verify the output. The correlation check protocol $\Pi^{\sigma}_{\mathrm{ChkEnc}}$ for the published encoded share is summarized in Figure 4 which keeps the share $x_i^{(k)}$ and cipher key α_i private. The protocol is designed to verify σ shares simultaneously by using a random vector $\boldsymbol{w}$. The correctness and soundness are stated as in following lemma.

Lemma 1 *(Correlation Check for Encoded Shares): The protocol $\Pi^{\sigma}_{\mathrm{ChkEnc}}$ is correct, i.e. it accepts if the encoded shares $\left(z_i^{(k)}, r_i^{(k)}\right)$ for all $i \in \{n\}$ and $k \in \{\sigma\}$ are correctly computed as defined in Def. 2. Moreover, it is sound, i.e. it rejects except with probability $o(1/q)$ in case at least one $\left(z_i^{(k)}, r_i^{(k)}\right)$ is not correctly computed, or any server deviates from the protocol.*

If $\Pi^{\sigma}_{\mathrm{ChkEnc}}$ passes, the encoded shares $\left(z_i^{(k)}, r_i^{(k)}\right)$ are verified, and $\boldsymbol{z}$ is ready to be recovered once the key $\boldsymbol{\alpha}$ is opened. If it returns **Reject**, we are not sure if the encoded shares are incorrect, some parties lied on the check outcome, or both happen, so in **Audit** P_{A} needs to verify the expected commitments to find out the cause. The audit protocol is demonstrated by Π_{Audit} in Figure 5. If the audit passes, the encoded shares are verified, the protocol still goes to output delivery. If both the check and audit fail, the encoded shares are considered incorrect, and the malicious parties that corrupt the output will be identified in the audit. The audit protocol can be accelerated by the technique in [3].

Not only the encoded shares but also the plain shares opened for multiplication need to go for the audit, if they fail the correlation check. In the online protocol, all shares that need audit are taken care of in one stage, such that batch processing can give additional efficiency improvement. The check of plain shares is more complicated than that of encoded shares and will be described in the next section.

Π_{Audit}

With published z_i and r_i from each P_i . Set $L \leftarrow \{\}$.

1) Compute Commitments**:**
   We follow the computation gates of the evaluated circuit $\mathcal{C}$ in the same order as they were computed. For any gate, with assigned inputs having well-formed commitments $\varepsilon_{\langle x \rangle}$ and $\varepsilon_{\langle y \rangle}$ from $\mathcal{F}_{\text{BLT}}$. The parties do the following:
   Input: For input m and preprocessed random $\langle t \rangle$, $\langle z \rangle = y \cdot \langle t \rangle$ with $y = m \cdot t^{-1}$, compute $\varepsilon_{\langle z \rangle} = \left(\varepsilon_{\langle t \rangle}\right)^{y}$.
   Add: For $\langle z \rangle = \langle x \rangle + \langle y \rangle$, compute the expectancy $\varepsilon_{\langle z \rangle} = \varepsilon_{\langle x \rangle} \times \varepsilon_{\langle y \rangle}$.
   Multiply: For $\langle z \rangle = \langle x \rangle \times \langle y \rangle$ with the preprocessed triple $(\langle a \rangle, \langle b \rangle, \langle c \rangle)$ and random $\langle t \rangle$. Derive η, ρ, and t from $\tilde{\eta}$, $\tilde{\rho}_i$, and $\tilde{t}_i$ stored in $\mathcal{F}_{\text{BLT}}$.
   a) Compute $\varepsilon_{\langle \eta \rangle} = \varepsilon_{\langle x \rangle} \times \left(\varepsilon_{\langle y \rangle}{}^{-1}\right)$ and $\varepsilon_{\langle \rho \rangle} = \varepsilon_{\langle y \rangle} \times \left(\varepsilon_{\langle b \rangle}{}^{-1}\right)$.
   b) Compute $\varepsilon_{\langle z \rangle} = \varepsilon_{\langle c \rangle} \times \left(\varepsilon_{\langle a \rangle}\right)^{\rho} \times \left(\varepsilon_{\langle b \rangle}\right)^{\eta} \times \left(\varepsilon_{\langle t \rangle}\right)^{\eta\rho\left(t^{-1}\right)}$.
2) P_{A} gets $E(z_i, r_i)$ from $\varepsilon_{\langle z \rangle}$ and checks if (z_i, r_i) can correctly open it. If not, identify cheating P_i and set $L \leftarrow P_i \cup L$.
3) If $L = \{\}$, P_A output **Accept**. Else output (**Reject**, L).

Figure 5. Π_{Audit}: Sub-protocol for the audit of encoded shares.

3.3.2 Check for Plain Shares

The correlation check protects the computations defined in Def. 2, not including the multiplication, because it uses three random values which are obtained from opening the plain shares of $\langle \eta \rangle$, $\langle \rho \rangle$, and $\langle t \rangle$. We need a correlation check for the opened plain shares, which is described as $\Pi^{\sigma}_{\text{ChkPln}}$ in Figure 6. The approach is similar except using random shares $\langle \boldsymbol{s} \rangle$ and secret value v_i for hiding the encoded shares $z_i^{(k)}$ and $r_i^{(k)}$. Its correctness and soundness are stated in following lemma. If $\Pi^{\sigma}_{\text{ChkPln}}$ fails, the auditor P_{A} will check the commitments in the audit stage.

Lemma 2 *(Correlation Check for Plain Shares): The protocol* $\Pi^{\sigma}_{\text{ChkPln}}$ *is correct, i.e. it accepts if the plain shares* $\left(\tilde{z}_i^{(k)}, \tilde{r}_i^{(k)}\right)$ *for all* $i \in \{n\}$ *and* $k \in \{\sigma\}$ *are correctly computed as defined in Def. 2. Moreover, it is sound, i.e. it rejects except with probability* $o(1/q)$ *in case at least one* $(\tilde{z}_i^{(k)}, \tilde{r}_i^{(k)})$ *is not correctly computed, or any server deviates from the protocol.*

3.4 Fairness and Robustness

We see that if the encoded shares of random values and triples are statistically indistinguishable from the samples from uniform distribution, and then those of the immediate values and final results have the same property. This implies fairness property.

Proposition 1 *(Fairness): The protocol* Π_{Online} *has public accountability and fairness, that is, the malicious parties know the result only if the honest ones know. If the TTP* P_T *is malicious and colluding with other adversarial parties,* Π_{Online} *still has public accountability in the hybrid model with* $\mathcal{F}_{\text{Offline}}$, $\mathcal{F}_{\text{Rnd}}$, $\mathcal{F}_{\text{Com}}$, $\mathcal{F}_{\text{Blt}}$, $\mathcal{Z}$, *and* $\mathcal{K}$, *if one computing party and* P_A *are honest.*

Remark 1*: Until* **Open** of Π_{Online}, P_T *has no information of the output result, since any set of* $\text{n} - 1$ *shares are indistinguishable to samples from uniform distribution. The adversary gains no advantage from the existence of malicious* P_T. *During the* **Open** stage of Π_{Online}, *by providing an incorrect key,* P_T *is only able to prevent the output delivery and cannot modify the output. If* P_T *is*

$\Pi^{\sigma}_{\mathrm{ChkPln}}$

$\tilde{z}_i^{(k)}$ and $\tilde{r}_i^{(k)}$ has been opened from each P_i and $k \in \{\sigma\}$. Set $L \leftarrow \{\}$

Check Plain Shares: On input $(\mathbf{ChkPln}, \mathrm{id}(\mathbf{z}), \sigma)$ from P_A.

1) Each $P_i \in P$ privately sample $v_i \xleftarrow{\$} \mathbb{F}$. The parties use a new random value $\langle \boldsymbol{s} \rangle = ([\boldsymbol{s}], [\boldsymbol{t}], \varepsilon_{\langle s \rangle})$ and use $\mathcal{F}_{\mathrm{Rnd}}$ to publicly sample a vector $\boldsymbol{w} \xleftarrow{\$} \mathbb{F}^{\sigma}$.
2) Each $P_i \in P$ opens $\left(z_i^{(k)} + s_i^{(k)}\right)$ and $\left(r_i^{(k)} + t_i^{(k)}\right)$ and computes $z^{(\boldsymbol{w})} = \sum_{k=1}^{\sigma} w_k \left(\sum_{j=1}^{n} \tilde{z}_j^{(k)}\right)$, $r^{(\boldsymbol{w})} = \sum_{k=1}^{\sigma} w_k \left(\sum_{j=1}^{n} \tilde{r}_j^{(k)}\right)$, $x_i^{(\boldsymbol{w})} = \sum_{k=1}^{\sigma} w_k \left(z_i^{(k)} + s_i^{(k)}\right)$, and $y_i^{(\boldsymbol{w})} = \sum_{k=1}^{\sigma} w_k \left(r_i^{(k)} + t_i^{(k)}\right)$ and sends them toward $\mathcal{F}_{\mathrm{BLT}}$.
3) Each $P_i \in P$ privately computes $\tilde{s}_j^{\boldsymbol{w}} = \sum_{k=1}^{\sigma} \tilde{s}_j^{(k)} w_k$ and $\tilde{t}_j^{\boldsymbol{w}} = \sum_{k=1}^{\sigma} \tilde{t}_j^{(k)} w_k$, $\eta_i = \sum_{j=1}^{n} x_j^{\boldsymbol{w}} \cdot \alpha_{j,i} - \tilde{s}_j^{\boldsymbol{w}} - v_i \cdot z^{(\boldsymbol{w})}$, and $\mu_i = \sum_{j=1}^{n} y_j^{\boldsymbol{w}} \cdot \alpha_{j,i} - \tilde{t}_j^{\boldsymbol{w}} - v_i \cdot r^{(\boldsymbol{w})}$.
4) Each $P_i \in P$ uses $\mathcal{F}_{\mathrm{Com}}$ to commit to v_i, η_i, and μ_i.
5) Each $P_i \in P$ uses $\mathcal{F}_{\mathrm{Com}}$ to open v_i, η_i, and μ_i to all parties.
6) Each party computes and outputs $\eta = \sum_{i=1}^{n} \eta_i$ and $\mu = \sum_{i=1}^{n} \mu_i$.
7) If $\eta = z^{(\boldsymbol{w})}(1 - \sum_{i=1}^{n} v_i)$, $\mu = r^{(\boldsymbol{w})}(1 - \sum_{i=1}^{n} v_i)$, all parties output **Accept**. Else output **Reject.**

Figure 6. $\Pi^{\sigma}_{\mathrm{ChkPln}}$: Protocol for the correlation check of plain shares.

colluding, the adversary will know the result before the honest parties but cannot force the protocol to output wrong results. Besides, if P_T can be assumed to be honest, we can modify the protocol and model such that the global key generation only needs to be invoked once. Furthermore, with honest P_T, Step 1 of the **Open** *stage can be done in the offline phase.*

Supporting the proof of Theorem 1, we can construct a protocol that securely computes the ideal functionality of online phase $\mathcal{F}_{\mathrm{Online}}$ with guaranteed output delivery in the $(\mathcal{F}_{\mathrm{Online}}, (\mathcal{C}, \mathbb{F}, \mathbb{G}))$-hybrid model. Recall for every party I_i, we assign a default input value $\tilde{x}_i$ and replace the secret input x_i if I_i is excluded. The rest of proof is trivial, and the protocol is briefly given below:

- Let $\mathcal{P}_1 = \mathcal{P} = \{1, \dots, n\}$, $\mathcal{I}_1 = \mathcal{I} = \{1, \dots, m\}$. Let $\mathcal{P}_t$ and $\mathcal{I}_t$ be the set of computing and input parties in the t-th iteration. $\mathcal{C}$ is the evaluation function.
- For $t = 1 \dots n + m$,
    - All parties in I send their inputs to the trusted party executing $\mathcal{F}_{\mathrm{Online}}$ with parameters $(\mathcal{C}, \mathbb{F}, \mathbb{G})$. The party with the lowest index in $\mathcal{I}_t$ simulates all parties in $\mathcal{I} \backslash \mathcal{I}_t$, using their predetermined default input values $\tilde{x}_i$. The party with the lowest index in $\mathcal{P}_t$ simulates all parties in $\mathcal{P} \backslash \mathcal{P}_t$.
    - The auditor P_A checks whether $\boldsymbol{y}$ is a valid output, if so P_A outputs $\boldsymbol{y}$ and halts. Otherwise, all parties receive $(\perp, \mathcal{L})$ as output, where $\mathcal{L}$ is an index set of corrupted parties. If there exists any $i^* \in \mathcal{L}$ and $i^* \in \mathcal{P} \backslash \mathcal{P}_t$ (or $i^* \in \mathcal{I} \backslash \mathcal{I}_t$), all parties delete i^* and add to $\mathcal{L}$ the party with the lowest index in $\mathcal{P}_t$ (or $\mathcal{I}_t$).
    - Set $\mathcal{P}_{t+1} \leftarrow \mathcal{P}_t \backslash \mathcal{L}$ if $\mathcal{L} \subseteq \mathcal{P}_t$ (or $\mathcal{I}_{t+1} \leftarrow \mathcal{I}_t \backslash \mathcal{L}$ if $\mathcal{L} \subseteq \mathcal{I}_t$).

3.5 Offline Phase

Π_{Offline}

Initialize: On input $(\textbf{Init}, \sigma, \mathbb{F}, \mathbb{G}, \boldsymbol{g}, h)$ from all players. This generates encryption keys and MUSS ciphers.

1) The parties use $\mathcal{F}_{\text{KGD}}$ to generate the key pair (pkd, skd), where skd is shared among parties.
2) The third party P_T use $\mathcal{F}_{\text{KGD}}$ to generate the global key pair (pko, sko) if it has none. Each P_j is given pko.
3) P_j samples $\alpha_{i,j} \xleftarrow{\$} \mathbb{F}$ for all $i \in \{n\}$. Set $\alpha_i = \sum_{j\in\{n\}} \alpha_{i,j}$.
4) Each P_j computes and broadcasts $[\![\mathbf{1} \cdot \alpha_{i,j}]\!]_{pkd} = \text{Enc}_{pkd}(\mathbf{1} \cdot \alpha_{i,j})$ with all-one vector $\mathbf{1} \in \mathbb{F}^{\sigma}$ to $\mathcal{F}_{\text{Blt}}$.
5) All parties compute $[\![\mathbf{1} \cdot \alpha_i]\!]_{pkd} = \bigoplus_{j\in\{n\}} [\![\mathbf{1} \cdot \alpha_{i,j}]\!]_{pkd}$ for all $i \in \{n\}$.
6) Each P_j commits $\boldsymbol{\alpha}_j = \{\alpha_{i,j}\}_{i\in\{n\}}$ with $\beta_j \in \mathbb{F}$ by $d_j = E^{(n)}_{(g,h)}(\boldsymbol{\alpha}_j, \beta_j)$ toward $\mathcal{F}_{\text{BLT}}$.
7) Each P_j computes $\mathsf{c}_j = \text{Enc}_{pko}(\{\boldsymbol{\alpha}_j, \beta_j\}, u_j)$ with $u_j \in \mathbb{F}$ and invokes $(\textbf{genZKP}, P_j)$ of $\Pi^{\sigma}_{\text{CPRZK}}$ to obtain ζ_j.

Single: On input $(\textbf{Single}, \sigma, g, h)$ from all players. This generates σ random values for the input.

1) Run $\{\langle r^{(k)}\rangle\}_{k\in\{\sigma\}} \leftarrow \Pi^{\sigma}_{\text{ComShr}}(\perp)$.
2) Output $\{\langle r^{(k)}\rangle\}_{k\in\{\sigma\}}$.

Triples: On input $(\textbf{Triple}, \sigma, g, h)$ from all players. This generates σ triples for the multiplication.

3) Run $\{\langle a^{(k)}\rangle\}_{k\in\{\sigma\}} \leftarrow \Pi^{\sigma}_{\text{ComShr}}(\perp)$ and $\{\langle b^{(k)}\rangle\}_{k\in\{\sigma\}} \leftarrow \Pi^{\sigma}_{\text{ComShr}}(\perp)$.
1) Run $\{\langle c^{(k)}\rangle\}_{k\in\{\sigma\}} \leftarrow \Pi^{\sigma}_{\text{GenTrp}}$. Set $t^{(k)} = (\langle a^{(k)}\rangle, \langle b^{(k)}\rangle, \langle c^{(k)}\rangle)$ with $c^{(k)} = a^{(k)} \cdot b^{(k)}$ for $k \in \{\sigma\}$.
2) Output $\{t^{(k)}\}_{k\in\{\sigma\}}$.

Audit: On input $(\textbf{Audit}, \sigma, g, h)$ from all parties. This verifies the output from **Initialize, Single** and **Triples.**

1) Run $\Pi^{\sigma}_{\text{ComShr}}(\top)$ for **Single** and $\Pi^{3\sigma}_{\text{ComShr}}(\top)$ for **Triples**.
2) Run $\Pi^{\sigma}_{\text{ChkZKP}}$ once for **Single** and $\Pi^{\sigma}_{\text{ChkZKP}}$ three times for **Triples**. If any of two replies **Reject**, P_A requests each P_i to open toward $\mathcal{F}_{\text{Blt}}$ the share of secret key skd_i and run the following steps.
   1.1. If the first fails, P_A requests each P_i to open toward $\mathcal{F}_{\text{Blt}}$ the share of $\langle r_k\rangle$ as well as the proofs of commitments and encryptions. P_A reads transcripts from $\mathcal{F}_{\text{Blt}}$ and verifies to identify malicious parties $\mathcal{M}_R$.
   1.2. If the second fails, P_A requests each P_i to open the random pads $\boldsymbol{\kappa}_{ij}$ and $\widetilde{\boldsymbol{\kappa}}_{ij}$, the share of t_k as well as the proofs of commitments and encryptions. P_A reads transcripts from $\mathcal{F}_{\text{Blt}}$ and verifies to identify malicious parties $\mathcal{M}_T$.
3) If any check fails, P_A outputs **Reject** and $\{\mathcal{M}_R, \mathcal{M}_T\}$. Or P_A outputs **Accept**.

Figure 7. Π_{Offline}: Protocol for the offline phase.

The protocol Π_{Offline} describes the full offline phase in Figure 7. Here we give a view to integrate all ideas that will be discussed later. During **Initialize** the parties will generate two key pairs to encrypt random MUSS ciphers α_i and the key (g, h) for the commitment scheme. With encrypted ciphers, **Single** uses the procedure $\Pi^{\sigma}_{\text{ComShr}}$ which generates random MUSS shares, together with commitments to the values. For multiplication triples, $\Pi^{\sigma}_{\text{GenTrp}}$ computes a product of the two random values and output them with commitments in **Triples**. These sub-protocols can be found

Set $[\![\cdot]\!] := [\![\cdot]\!]_{pkd}$, SIMD factor σ. $[\![\alpha_j]\!]$ as the encrypted cipher key. Define $\boldsymbol{u}_i = \left\{u_i^{(k)}\right\}_{k\in\{\sigma\}}$, $\boldsymbol{v}_i = \left\{v_i^{(k)}\right\}_{k\in\{\sigma\}}$, $\boldsymbol{a}_j \cdot \alpha_j = \left\{a_j^{(k)} \cdot \alpha_j\right\}_{k\in\{\sigma\}}$, $\mathsf{E}(\boldsymbol{u}_i, \boldsymbol{v}_i) = \left\{\mathsf{E}\left(u_i^{(k)}, v_i^{(k)}\right)\right\}_{k\in\{\sigma\}}$ and $[\![\boldsymbol{\mu}_i]\!] = \left\{[\![u_i^{(k)}]\!]\right\}_{k\in\{\sigma\}}$, $\boldsymbol{u}_j \otimes [\![\boldsymbol{\alpha}_j]\!] = \left\{[\![u_i^{(k)}\alpha_j]\!]\right\}_{k\in\{\sigma\}}$, $\boldsymbol{u}_j \otimes [\![\boldsymbol{a}_j]\!] = \left\{[\![u_i^{(k)}a_i^{(k)}]\!]\right\}_{k\in\{\sigma\}}$, and $[\![\boldsymbol{\alpha}_j]\!] = [\![\mathbf{1}\cdot\alpha_j]\!]$ for parallel processing.

$$\Pi^{\sigma}_{\mathrm{ComShr}}(\mathsf{flag})$$

With private share $\boldsymbol{u}_i$ and randomness $\boldsymbol{v}_i$ from each P_i. $\boldsymbol{u}_i = \left\{u_i^{(k)}\right\}_{k\in\{\sigma\}}$

1) Execute $\Pi^{\sigma}_{\mathrm{GenShr}}$ twice to obtain $\tilde{\boldsymbol{u}}_i$ and $\tilde{\boldsymbol{v}}_i$, respectively, for each P_i.
2) Each party P_i computes $\mathsf{E}(\boldsymbol{u}_i, \boldsymbol{v}_i)$ and $\mathsf{E}(\tilde{\boldsymbol{u}}_i, \tilde{\boldsymbol{v}}_i)$ If $\mathsf{flag} = \perp$, open both toward $\mathcal{F}_{\mathrm{Blt}}$. If $\mathsf{flag} = \top$, only open $\mathsf{E}(\boldsymbol{u}_i, \boldsymbol{v}_i)$.

$$\Pi^{\sigma}_{\mathrm{GenShr}}$$

With private share $\boldsymbol{u}_i$ and randomness $\boldsymbol{v}_i$ from each P_i.

1) Each $P_j \in \mathcal{P}\backslash P_1$ samples $\boldsymbol{u}_i, \tilde{\boldsymbol{u}}_j \xleftarrow{\$} \mathbb{F}^{\sigma}$ at random and opens $[\![\boldsymbol{\mu}_j]\!] = \boldsymbol{u}_j \otimes [\![\boldsymbol{\alpha}_j]\!] - \tilde{\boldsymbol{u}}_j$.
2) P_1 opens $[\![\boldsymbol{\mu}_1]\!] = \boldsymbol{u}_1 \otimes [\![\boldsymbol{\alpha}_1]\!]$. All parties compute $[\![\tilde{\boldsymbol{u}}_1]\!] = \oplus_{j\in\{n\}}[\![\boldsymbol{\mu}_j]\!]$.
3) All parties call (**Decrypt**, pk, $[\![\tilde{\boldsymbol{u}}_1]\!], P_1$) for P_1 to obtain $\tilde{\boldsymbol{u}}_1$.

$$\Pi^{\sigma}_{\mathrm{GenTrp}}$$

With private share $\langle\boldsymbol{a}\rangle$ and $\langle\boldsymbol{b}\rangle$.

1) Each $P_j \in \mathcal{P}$ opens $[\![\boldsymbol{a}_j \cdot \alpha_j]\!] = \boldsymbol{a}_j \otimes [\![\boldsymbol{\alpha}_j]\!]$ and $[\![\tilde{\boldsymbol{a}}_j]\!]$.
2) Each $P_j \in \mathcal{P}$ samples $\boldsymbol{\kappa}_{ji}, \tilde{\boldsymbol{\kappa}}_{ji} \xleftarrow{\$} \mathbb{F}^{\sigma}$ and sends $[\![\boldsymbol{\kappa}_{ji} \cdot \alpha_j]\!] = \boldsymbol{\kappa}_{ji} \otimes [\![\boldsymbol{\alpha}_j]\!]$ and $[\![\tilde{\boldsymbol{\kappa}}_{ji}]\!]$ toward each $P_i \in P\backslash P_j$.
3) Each $P_j \in \mathcal{P}$ computes
$$[\![\boldsymbol{c}_j]\!] = \left(\boldsymbol{b}_j \otimes \left(\oplus_{i\in\{n\}}[\![\boldsymbol{a}_i \cdot \alpha_i]\!]\right)\right) \oplus \left(\oplus_{i\in\{n\}\backslash j}[\![\boldsymbol{\kappa}_{ij} \cdot \alpha_i]\!]\right) \oplus \left(\oplus_{i\in\{n\}\backslash j}[\![-\boldsymbol{\kappa}_{ji} \cdot \alpha_i]\!]\right)$$
$$[\![\tilde{\boldsymbol{c}}_j]\!] = \left(\tilde{\boldsymbol{b}}_j \otimes \left(\oplus_{i\in\{n\}}[\![\tilde{\boldsymbol{a}}_i]\!]\right)\right) \oplus \left(\oplus_{i\in\{n\}\backslash j}[\![\tilde{\boldsymbol{\kappa}}_{ij}]\!]\right) \oplus \left(\sum_{i\in\{n\}\backslash j} -\tilde{\boldsymbol{\kappa}}_{ji}\right).$$
4) All parties call $\left(\mathbf{Decrypt}, pkd, [\![\boldsymbol{c}_j]\!], P_i\right)$ and $(\mathbf{Decrypt}, pkd, [\![\tilde{\boldsymbol{c}}_i]\!], P_i)$ for each $P_i \in \mathcal{P}$.
5) Each $P_j \in \mathcal{P}$ samples $\boldsymbol{t}_j \xleftarrow{\$} \mathbb{F}^{\sigma}$. All parties run $\Pi^{\sigma}_{\mathrm{GenShr}}$ for $[\boldsymbol{t}]$.
6) Each P_i computes and opens $\mathsf{E}(\boldsymbol{c}_i, \boldsymbol{t}_i)$ and $\mathsf{E}(\tilde{\boldsymbol{c}}_i, \tilde{\boldsymbol{t}}_i)$ toward $\mathcal{F}_{\mathrm{Blt}}$ for $\langle\boldsymbol{c}\rangle = \left\{\left\langle c^{(k)}\right\rangle\right\}_{k\in\{\sigma\}}$.

Figure 8. Sub-protocols for the generation of MUSS shares.

in Figure 8. If we assume the presence of at least one honest server and that the adversary has a static strategy to corrupt the servers, $\Pi^{\sigma}_{\mathrm{CPRZK}}$ (Figure 9) and $\Pi^{\sigma}_{\mathrm{ChkZKP}}$ (Figure 10) work as the audit to ensure that the following properties hold:

- All commitments of shares have ZKP's. All ciphertexts and commitments of MUSS ciphers have ZKP's, which are verified in the online phase.
- The procedure $\Pi^{\sigma}_{\mathrm{CPRZK}}$ was executed such that the ciphertexts of MUSS ciphers were correctly encrypted from the plaintexts.
- The procedure $\Pi^{\sigma}_{\mathrm{ChkZKP}}$ was executed such that the generation of shares followed the protocol, otherwise the malicious parties that cheat in **Single** and **Triples** of Π_{Offline} were identified.

$\Pi_{\text{CPRZK}}^{(n,\ell)}$

Given the RO $\mathcal{Z}$, the input is $\alpha_{i,j}, b_{i,j}, c_j, d_j$. For each $P_j \in \mathcal{P}$, $P_{j'} \in \{\mathcal{P}, P_A\}/P_j$.

Generate ZKP: On input $(\mathbf{genZKP}, P_j, P_{j'})$ from P_j for ZKP ζ_j to prove $R_{CPR,j}^{(n,\ell)}$.

1) For each $k \in \{\lambda\}, i \in \{n\}$, P_j samples $m_i^{(k)}, p^{(k)} \xleftarrow{\$} \mathbb{F}$ and $s_i^{(k)}, t^{(k)} \xleftarrow{\$} \chi$.
2) P_j computes $f_i^{(k)} \leftarrow \mathsf{Enc}_{pkd}\left(\mathbf{1} \cdot m_i^{(k)}, s_i^{(k)}\right)$, $g^{(k)} \leftarrow \mathsf{Enc}_{pko}\left(\left\{\left\{m_i^{(k)}\right\}_{i\in\{n\}}, p^{(k)}\right\}, t^{(k)}\right)$, and $v^{(k)} \leftarrow E_{(\boldsymbol{g},h)}^{(n)}\left(\left\{m_i^{(k)}\right\}_{i\in\{n\}}, p^{(k)}\right)$.
3) Set $\mathcal{T} = \left\{\left\{f_i^{(k)}\right\}_{i\in\{n\}} \| g^{(k)} \| v^{(k)}\right\}_{k\in\{\lambda\}}$, P_j computes $\mathbf{e} \leftarrow \mathcal{Z}\left(\mathcal{T} \left\| \{b_{i,j}\}_{i\in\{n\}} \| c_j \| d_j\right)\right.$ where $\mathbf{e} \in \{0,1\}^{\lambda}$.
4) P_j computes $\gamma_i^{(k)} = m_i^{(k)} + e^{(k)} \cdot \alpha_{i,j}$, $\delta^{(k)} = p^{(k)} + e^{(k)} \cdot \beta_j$, and $\rho_i^{(k)} = s_i^{(k)} + e^{(k)} \cdot r_{i,j}$ and $\sigma^{(k)} = t^{(k)} + e^{(k)} \cdot u_j$. Output $\zeta_j = \left\{\left\{f_i^{(k)}, \gamma_i^{(k)}, \rho_i^{(k)}\right\}_{i\in\{n\}}, g^{(k)}, \sigma^{(k)}, \delta^{(k)}, v^{(k)}\right\}_{k\in\{\lambda\}}$.

Verify ZKP: On input $(\mathbf{verZKP}, P_j, \zeta_j)$ from $P_{j'}$ to verifies ζ_j to prove $R_{CPR,j}^{(n,\ell)}$.

1) $P_{j'}$ computes $f'^{(k)}_i = \mathsf{Enc}_{pkd}\left(\mathbf{1} \cdot \gamma_i^{(k)}, \rho_i^{(k)}\right)$ and $g'^{(k)} = \mathsf{Enc}_{pko}\left(\left\{\left\{\gamma_i^{(k)}\right\}_{i\in\{n\}}, \delta^{(k)}\right\}, \sigma^{(k)}\right)$. Set $\mathcal{T}' = \left\{\left\{f_i^{(k)}\right\}_{i\in\{n\}} \| g^{(k)} \| v^{(k)}\right\}_{k\in\{\lambda\}}$ and $\mathbf{e}' \leftarrow \mathcal{Z}\left(\mathcal{T}' \left\| \{b_{i,j}\}_{i\in\{n\}} \| c_j \| d_j\right)\right.$.
2) $P_{j'}$ checks if for each $k \in \{\lambda\}, i \in \{n\}$
   $f'^{(k)}_i = f_i^{(k)} \oplus \left(e'^{(k)} \cdot b_{i,j}\right), g'^{(k)} = g^{(k)} \oplus \left(e'^{(k)} \cdot c_j\right)$,
   $\mathbf{Cor}\left(pkd, f'^{(k)}_i, \mathbf{1} \cdot \gamma_i^{(k)}, \rho_i^{(k)}\right) = 1$,
   $\mathbf{Cor}\left(pko, g'^{(k)}, \left\{\left\{\gamma_i^{(k)}\right\}_{i\in\{n\}}, \delta^{(k)}\right\}, \sigma^{(k)}\right) = 1$,
   $E_{(\boldsymbol{g},h)}^{(n)}\left(\left\{\gamma_i^{(k)}\right\}_{i\in\{n\}}, \delta^{(k)}\right) = v^{(k)} \cdot \left(d_j\right)^{e'^{(k)}}$.
   If one fails then $P_{j'}$ output **Reject**, otherwise **Accept**.

Figure 9. $\Pi_{\text{CPRZK}}^{(n,\ell)}$: Non-interactive ZKP for the relation $R_{CPR,j}^{(n,\ell)}$.

The security proof of offline protocol is provided in the full version of the paper. While we do not consider guaranteed output delivery for the offline phase, we compose player-elimination on the online phase, that invokes a copy of the offline phase to achieve the robustness. Since information is revealed due to the failed audit, everything will need to be generated again for a newly setup copy in the next iteration.

3.5.1 Distributed Encryption

We have a semi-homomorphic encryption scheme $\mathcal{H} = (\mathsf{KG}, \mathsf{Enc}, \mathsf{Dec}, \oplus, \otimes)$ with a message space $\mathbb{F}$ and randomness distribution χ. The ciphertext encrypted by H is denoted as $[\![x]\!]_{pk} := \mathsf{Enc}_{pk}(x, r)$ with key pair (pk, sk). In addition, $\mathcal{H}$ has a predicate

$\mathbf{Cor}: \{0,1\}^{n(\lambda)} \times \{0,1\}^{n(\lambda)} \times \{0,1\}^{n(\lambda)} \times \{0,1\}^{n(\lambda)} \to \{0,1\}$

$(pk, c, x, r) \to \mathbf{Cor}\,(pk, c, x, r)$, that maps to 1 if $pk \xleftarrow{\$} \mathsf{KG}(1^{\lambda}), x \in \mathbb{F}, r \xleftarrow{\$} \chi$ and $\mathrm{c} \leftarrow \mathsf{Enc}_{pk}(x, r)$, but otherwise indicates that *at least one of these four conditions are not true*. The operator $\oplus$ then guarantees that $\mathsf{Dec}_{sk}([\![x + y]\!]_{pk}) = \mathsf{Dec}_{sk}([\![x]\!]_{pk} \oplus [\![y]\!]_{pk})$, whereas we do not

use homomorphic multiplication. The scalar multiplication $\otimes$ guarantees that $\mathsf{Dec}_{sk}(y \otimes [\![x]\!]_{pk}) = \mathsf{Dec}_{sk}([\![x \cdot y]\!]_{pk})$.

In addition, we require the interactive functionality $\mathcal{F}_{\mathrm{KGD}}$ that will be used for the preprocessing. The key pair can be securely generated by a key-generation protocol, where the secret key is additively shared by all parties. The ciphertext can be jointly decrypted by yielding the plaintext publicly from all parties, or providing it to a specific party privately.

3.5.2 Generation of Multiplicative Ciphers

The ciphers are jointly generated by the computing parties. The protocol has two key pairs (pkd, skd) and (pko, sko). The first one is obtained using $\mathcal{F}_{\mathrm{KGD}}$ invoked by all computing parties. The second one is given by the external TTP. Henceforth, each party encrypts his share twice with pkd and pko. With $\mathbf{1} = \{1\}_{i\in\{\ell\}}$, $\boldsymbol{r}_{i,j} \xleftarrow{\$} \mathbb{F}^{\ell}$, and $u_j \xleftarrow{\$} \mathbb{F}$ the ciphertext $b_{i,j} = [\![\mathbf{1} \cdot \alpha_{i,j}]\!]_{pkd} = \mathsf{Enc}_{pkd}(\mathbf{1} \cdot \alpha_{i,j}, \boldsymbol{r}_{i,j})$ is broadcasted for the generation of correlated randomness, and $\mathsf{c}_j = \mathsf{Enc}_{pko}(\{\overline{\boldsymbol{\alpha}}_j, \beta_j\}, u_j)$ for $\overline{\boldsymbol{\alpha}}_j = \{\alpha_{i,j}\}_{\mathrm{i}\in\{n\}}$ is always held private until the output delivery of online phase. The relation between two ciphertexts is built by committing $\alpha_{i,j}$ toward the bulletin. Therefore, we need ZKP to ensure that these encryptions are all derived from the same plaintext. For $\mathsf{E}^{(n)}_{(\boldsymbol{g},h)}(\overline{\boldsymbol{\alpha}}_j, \beta_j) = \prod_{i=1}^{n} \mathsf{E}_{(g_i,h)}(\alpha_{i,j}, \beta_j)$ and $\boldsymbol{\alpha}_j = \{\alpha_{i,j}\}_{i\in\{n\}}$, and the relation is formalized as:

$$R^{(n,\ell)}_{CPR,j} = \Big\{(\mathbf{s}, \boldsymbol{a}) | \mathbf{s} = \big(\{b_{i,j}\}_{i\in\{n\}}, c_j, d_j, pkd, pko\big), \boldsymbol{a} = \big(\boldsymbol{\alpha}_j, \beta_j, \boldsymbol{r}_{i,j}, u_j\big), \mathbf{Cor}\big(pkd, b_{i,j}, \big(\mathbf{1} \cdot \alpha_{i,j}\big), r_{i,j}\big) = 1, \mathbf{Cor}\big(pko, c_j, \{\overline{\boldsymbol{\alpha}}_j, \beta_j\}, u_j\big) = 1, \{b_{i,j}\}_{i\in\{n\}} = \big\{[\![\mathbf{1} \cdot \alpha_{i,j}]\!]_{pkd}\big\}_{i\in\{n\}}, \mathsf{c}_j = \mathsf{Enc}_{pko}\big(\{\overline{\boldsymbol{\alpha}}_j, \beta_j\}, u_j\big), d_j = \mathsf{E}^{(n)}_{(\boldsymbol{g},h)}(\overline{\boldsymbol{\alpha}}_j, \beta_j)\Big\}.$$

Based on [24], the protocol $\Pi^{(n,\ell)}_{\mathrm{CPRZK}}$ of ZKP for each pair P_j and $P_{j'}$ in $\mathcal{P}$ is described in Figure 9, and we propose that it satisfies the properties as follows.

The protocol $\Pi^{\ell}_{\mathrm{CPRZK}}$ is correct due to the homomorphic addition of $\mathcal{H}$ and the commitment scheme, because P_j outputs transcripts that can be verified by **Cor**. Soundness follows due to the standard soundness of Σ-protocols which allows us to extract a witness. Since $\mathbf{e}$ was chosen from $\{0, 1\}^{\lambda}$, there is negligible probability for a dishonest P_j to forge an accepting transcript. Zero-knowledge follows trivially by assuming the programmable random oracle model of $\mathcal{Z}$, and security of Enc_{pko} and $\mathsf{E}^{(n)}_{(\boldsymbol{g},h)}$.

Proposition 2 *(ZKP for MUSS ciphers): The protocol* $\Pi^{(\mathrm{n},\ell)}_{\mathrm{CPRZK}}$ *is a non-interactive zero-knowledge proof of knowledge for the relation* $R^{(n,\ell)}_{CPR,j}$ *using the programmable RO* $\mathcal{Z}$.

3.5.3 Correlation Check Using Global KEA-ZKP

Whenever plaintexts are encrypted for cipher keys, and ciphertexts are decrypted for shares, the adversary will be able to influence the outcome of the encryption and decryption processes, and we hence have to check the output for correctness. Unlike other SPDZ protocols, we will not simply prove the correctness of ciphertexts or the relation between the ciphertext and commitment. Instead we check the output: KEA-based proof is utilized to prove the existence of committed MUSS shares which have the MUSS correlation. It can be proven that given at least one honest party, the MUSS correlation holds if and only if the parties follow the sub-protocols $\Pi^{\sigma}_{\mathrm{ComShr}}$ and $\Pi^{\sigma}_{\mathrm{GenTrp}}$. Given in Figure 10, the protocol $\Pi^{\sigma}_{\mathrm{ChkZKP}}$ elaborates the check function to verify the commitment and will be used to prove the following lemma.

$\Pi^{\sigma}_{\mathrm{ChkZKP}}$

Given the RO $\mathcal{R}$ and $\mathcal{Q}$. The commitment scheme uses keys g and h. We want to verify $\langle \boldsymbol{z} \rangle = \left\{ \langle z^{(k)} \rangle \right\}_{k\in\{\sigma\}} = \left([\boldsymbol{z}], [\boldsymbol{r}], \boldsymbol{\varepsilon}_{\langle \boldsymbol{z} \rangle} = \mathsf{E}([\boldsymbol{z}], [\boldsymbol{r}]) \right)$. Let $\langle \boldsymbol{u} \rangle = \left([\boldsymbol{u}], [\boldsymbol{v}], \boldsymbol{\varepsilon}_{\langle \boldsymbol{u} \rangle} = \mathsf{E}([\boldsymbol{u}], [\boldsymbol{v}]) \right)$ and $E_{(g,h)}\left(\tilde{u}_i^{(k)}, \tilde{v}_i^{(k)}\right)$ be private. Each party $P_i \in \mathcal{P}$ has MUSS cipher shares $\{\alpha_{j,i}\}_{i\in\{n\}}$.

Check ZKP: On input $(\mathbf{ChkZKP}, \mathrm{id}(\mathbf{z}), \mathrm{id}(\mathbf{u}), g, h, \sigma)$ from all parties.

1) All parties use $\mathcal{F}_{\mathrm{Rnd}}$ to get $\boldsymbol{w} \xleftarrow{\$} \mathbb{F}^{\sigma}$.
2) Each party $P_i \in \mathcal{P}$ gets by $(g^{\beta}, h^{\beta}) \leftarrow \mathcal{Q}(\boldsymbol{\varepsilon}_{\langle \boldsymbol{z} \rangle}, \boldsymbol{\varepsilon}_{\langle \boldsymbol{u} \rangle}, g, h)$.
3) Each $P_i \in P$ privately sample $s_i \xleftarrow{\$} \mathbb{F}$.
4) Each party $P_i \in \mathcal{P}$ computes $z_i^{\boldsymbol{w}} = \sum_{k\in\{\sigma\}} z_i^{(k)} w^{(k)}$, $r_i^{\boldsymbol{w}} = \sum_{k\in\{\sigma\}} r_i^{(k)} w^{(k)}$, similarly $u_i^{\boldsymbol{w}}, v_i^{\boldsymbol{w}}, \tilde{u}_i^{\boldsymbol{w}}, \tilde{v}_i^{\boldsymbol{w}}, \tilde{z}_i^{\boldsymbol{w}}$, and $\tilde{r}_i^{\boldsymbol{w}}$. Furthermore, it derives and opens $a_i = \mathsf{E}_{(g,h)}(z_i^{\boldsymbol{w}}, r_i^{\boldsymbol{w}})$, $b_i = \mathsf{E}_{(g,h)}(u_i^{\boldsymbol{w}}, v_i^{\boldsymbol{w}})$, $c_i = \mathsf{E}_{(g,h)}(\tilde{z}_i^{\boldsymbol{w}}, \tilde{r}_i^{\boldsymbol{w}})$, $a_i' = \mathsf{E}_{(g^{\beta},h^{\beta})}(z_i^{\boldsymbol{w}}, r_i^{\boldsymbol{w}})$, $b_i' = \mathsf{E}_{(g^{\beta},h^{\beta})}(u_i^{\boldsymbol{w}}, v_i^{\boldsymbol{w}})$, and $c_i' = \mathsf{E}_{(g^{\beta},h^{\beta})}(\tilde{z}_i^{\boldsymbol{w}}, \tilde{r}_i^{\boldsymbol{w}})$.
5) P_A checks if $a_i = \prod_{k\in\{\sigma\}} \left(\mathsf{E}_{(g,h)}\left(z_i^{(k)}, r_i^{(k)}\right) \right)^{w^{(k)}}$, $b_i = \prod_{k\in\{\sigma\}} \left(\mathsf{E}_{(g,h)}\left(u_i^{(k)}, v_i^{(k)}\right) \right)^{w^{(k)}}$, $c_i = \prod_{k\in\{\sigma\}} \left(\mathsf{E}_{(g,h)}\left(\tilde{z}_i^{(k)}, \tilde{r}_i^{(k)}\right) \right)^{w^{(k)}}$. If the checks fail, P_A output **Reject**. Stop.
6) P_A computes $\beta \leftarrow \mathcal{R}(\boldsymbol{\varepsilon}_{\langle \boldsymbol{z} \rangle}, \boldsymbol{\varepsilon}_{\langle \boldsymbol{u} \rangle}, g, h)$ and check if $a_i' = (a_i)^{\beta}$, $b_i' = (b_i)^{\beta}$, and $c_i' = (c_i)^{\beta}$. If the checks fail, P_A output **Reject**. Stop.
7) Each party $P_i \in \mathcal{P}$ computes $d_i = \mathsf{E}_{(g,h)}(\tilde{u}_i^{\boldsymbol{w}}, \tilde{v}_i^{\boldsymbol{w}})$, $d_i' = \mathsf{E}_{(g^{\beta},h^{\beta})}(\tilde{u}_i^{\boldsymbol{w}}, \tilde{v}_i^{\boldsymbol{w}})$, and $\tau_i = \prod_{j\in\{n\}} (a_j \cdot b_j)^{\alpha_{j,i}} (c_j)^{s_i} / d_i$ and $\tau_i' = \prod_{j\in\{n\}} (a_j' \cdot b_j')^{\alpha_{j,i}} (c_j')^{s_i} / d_i'$
8) Each $P_i \in P$ uses $\mathcal{F}_{\mathrm{Com}}$ to commit to s_i, τ_i and τ_i'.
9) Each $P_i \in P$ uses $\mathcal{F}_{\mathrm{Com}}$ to open s_i, τ_i and τ_i'. to all parties. Check if $\tau_i' = (\tau_i)^{\beta}$. If the checks fail, P_A output **Reject** and stop.
10) Set $s = \sum_{i\in\{n\}} s_i$. All parties compute and output $\rho = \prod_{i\in\{n\}} \tau_i / (c_i)^{1+s}$. If $\rho = 1$, P_A outputs **Accept**. Or it outputs **Reject**.

Figure 10. $\Pi^{\sigma}_{\mathrm{ChkZKP}}$: Global proof of knowledge of MUSS shares.

Proposition 3 *(Correlation check using global ZKP): If DLP and KEA3 assumptions [18] are true, the protocol* $\Pi^{\sigma}_{\mathrm{ChkZKP}}$ *is a non-interactive ZKP of knowledge for the relation* R^{σ}_{MUSS} *using the programmable RO's* $\mathcal{R}$ and $\mathcal{Q}$.

$$R^{\sigma}_{MUSS} = \left\{ (\boldsymbol{\varepsilon}, \boldsymbol{w}) \,\middle|\, \boldsymbol{\varepsilon} = \left\{ \boldsymbol{\varepsilon}_i = \left(\varepsilon_i^{(k)}, \tilde{\varepsilon}_i^{(k)} \right) \right\}_{i=\{n\},k=\{\sigma\}} \right.$$
$$\wedge \boldsymbol{w} = \left\{ \left(z_i^{(k)}, \tilde{z}_i^{(k)} \right), \left(r_i^{(k)}, \tilde{r}_i^{(k)} \right) \right\}_{i=\{n\},k=\{\sigma\}}$$
$$\wedge \left\{ \varepsilon_i^{(k)} = \mathsf{E}_{(g,h)}\left(z_i^{(k)}, r_i^{(k)} \right), \tilde{\varepsilon}_i^{(k)} = \mathsf{E}_{(g,h)}\left(\tilde{z}_i^{(k)}, \tilde{r}_i^{(k)} \right) \right\}_{i=\{n\},k=\{\sigma\}}$$
$$\left. \wedge \left\{ \sum_{i=1}^{n} \alpha_i z_i^{(k)} = \sum_{i=1}^{n} \tilde{z}_i^{(k)} \right\}_{k=\{\sigma\}} \wedge \left\{ \sum_{i=1}^{n} \alpha_i r_i^{(k)} = \sum_{i=1}^{n} \tilde{r}_i^{(k)} \right\}_{k=\{\sigma\}} \right\}$$

Lemma 3 *(MUSS correlation of output): Assume that there exists at least one honest party. If all parties follow* $\Pi^{\sigma}_{\mathrm{ComShr}}$ *and* $\Pi^{\sigma}_{\mathrm{GenTrp}}$, $\Pi^{\sigma}_{\mathrm{ChkZKP}}$ *will accept the output. Otherwise, it will reject except a negligible probability.*

The protocol $\Pi^{\sigma}_{\mathrm{ChkZKP}}$ is correct due to the linearity of the commitment scheme and because P_i only outputs transcripts that satisfy the KEA and R^{σ}_{MUSS}. Soundness follows due to the KEA and

the binding of Pedersen commitment which allows us to extract a witness. Since $\alpha_{i,j}$, s_i, and $\boldsymbol{w}$ were chosen from a large enough space uniformly so it is computationally infeasible for P_i to forge an accepting transcript. Zero-knowledge follows the DLP assumption in the programmable random oracle model $\mathcal{R}$ and Q for the transcript. The simulator is given in the full version of the paper.

4. CONCLUSIONS

We described a proposed scheme to address the issues of privacy and correctness in multi-party computation protocols. The solution introduced a semi-trusted third party as the key manager and redesigns the secret-sharing mechanism. The design ensures that the malicious parties cannot know the output by causing an abort, and the output delivery is guaranteed by excluding cheaters and restarting the protocol. The offline sub-protocols can be audited publicly by verifying zero-knowledge proofs based on KEA, holding corrupted parties accountable. The security of the protocol can be proven in the universal composability framework.

REFERENCES

[1] I. Damgård, V. Pastro, N. P. Smart, and S. Zakarias, "Multiparty Computation from Somewhat Homomorphic Encryption," CRYPTO 2012, pp. 643-662, 2012.

[2] I. Damgard, M. Keller, E. Larraia, V. Pastro, P. Scholl, and N. P. Smart, "Practical Covertly Secure MPC for Dishonest Majority - Or: Breaking the SPDZ Limits," ESORICS 2013, pp. 1-18, 2013.

[3] C. Baum, I. Damgård, and C. Orlandi, "Publicly Auditable Secure Multi-Party Computation," SCN 2014, pp. 175-196, 2014.

[4] M. Keller, E. Orsini, and P. Scholl, "MASCOT: Faster Malicious Arithmetic Secure Computation with Oblivious Transfer", CCS 2016, pp. 830-842, 2016.

[5] G. Spini and S. Fehr, "Cheater Detection in SPDZ Multiparty Computation," ICITS 2016, pp. 151-176, 2016.

[6] M. Keller, V. Pastro, and D. Rotaru, "Overdrive: Making SPDZ Great Again," EUROCRYPT 2018, pp. 158-189, 2018.

[7] C. Baum, D. Cozzo, and N. P. Smart, "Using TopGear in Overdrive: A More Efficient ZKPoK for SPDZ," SAC 2019, pp. 274-302, 2019.

[8] C. Baum, E. Orsini, P. Scholl, and E. Soria-Vazquez, "Efficient Constant-Round MPC with Identifiable Abort and Public Verifiability," CRYPTO 2020, pp. 562-592, 2020.

[9] B. Schoenmakers and M. Veeningen, "Universally Verifiable Multiparty Computation from Threshold Homomorphic Cryptosystems," ACNS 2015, pp. 3-22, 2015.

[10] G. Asharov and C. Orlandi, "Calling Out Cheaters: Covert Security with Public Verifiability," ASIACRYPT 2012, pp. 681-698, 2012.

[11] R. Cohen and Y. Lindell, "Fairness versus Guaranteed Output Delivery in Secure Multiparty Computation," ASIACRYPT 2014, pp. 466-485, 2014.

[12] A. Kiayias, H. Zhou, and V. Zikas, "Fair and Robust Multi-party Computation Using a Global Transaction Ledger," EUROCRYPT 2016, pp. 705-734, 2016.

[13] N. Asokan, V Shoup, and M. Waidner, "Optimistic Fair Exchange of Digital Signatures," EUROCRYPT 1998, pp. 591-606, 1998.

[14] C. Cachin and J. Camenisch, "Optimistic Fair Secure Computation (Extended Abstract)," CRYPTO 2000, LNCS, vol. 1880, pp. 93-111, 2000.

[15] C. Baum, E. Orsini, and P. Scholl, "Efficient Secure Multiparty Computation with Identifiable Abort," TCC 2016-B, pp. 461-490, 2016.

[16] M. Rivinius, P. Reisert, D. Rausch, and R. Küsters, "Publicly Accountable Robust Multi-Party Computation, " IEEE S&P 2022, pp. 2430-2449, 2022.

[17] M. Seo, "Fair and Secure Multi-Party Computation with Cheater Detection," Cryptography, vol. 5, no. 3, pp. 19-39, 2021.

[18] M. Bellare and A. Palacio, "The Knowledge-of-Exponent Assumptions and 3-Round Zero-Knowledge Protocols, " CRYPTO 2004, vol. 3152, pp. 273-289, 2004.

[19] J. Groth, "Short Pairing-Based Non-interactive Zero-Knowledge Arguments, " ASIACRYPT 2010, col. 6477, pp. 321-340, 2020.

[20]R. Cleve, “Limits on the Security of Coin Flips When Half the Processors Are Faulty (extended abstract),” STOC, pages 364–369. ACM, 1986.
[21] H. Kılınç and A. Küpçü, "Optimally Efficient Multi-Party Fair Exchange and Fair Secure Multi-Party Computation," CT-RSA 2015, pp. 330-349, 2015.
[22] A. Herzberg and H. Shulman, “Oblivious and Fair Server-Aided Two-Party Computation,” ARES 2012, pp. 75-84, 2012.
[23] D. Beaver, "Efficient Multiparty Protocols Using Circuit Randomization," CRYPTO ’91, pp. 420-432, 1991.
[24] R. Cramer and I. Damgård, "On the Amortized Complexity of Zero- Knowledge Protocols," CRYPTO 2009, pp. 177-191, 2009.
[25] R. Canetti, "Universally Composable Security: A New Paradigm for Cryptographic Protocols," FOCS 2001, pp. 136-145, 2001.
[26] MP-SPDZ 2022 [online] Available: https://github.com/data61/.
[27] M. Keller, "MP-SPDZ: A Versatile Framework for Multi-Party Computation," CCS 2020, pp. 1575-1590, 2020.
[28] E. Orsini, "Efficient Actively Secure MPC with a Dishonest Majority: A Survey," WAIFI 2020, pp. 42-71, 2020.
[29] C. Cooper, "On the distribution of rank of a random matrix over a finite field," Random Structures and Algorithms, vol. 17, pp. 197-212, 2000.
[30] I. Damgård, "Non-Interactive Circuit Based Proofs and Non-Interactive Perfect Zero-Knowledge with Preprocessing, " EUROCRYPT 1992, LNCS, vol. 658, pp. 341-355, 1992.
[31] S. Hada and T. Tanaka, "On the Existence of 3-Round Zero-Knowledge Protocols, " ePrint Archive, Report 1999/009, 1999, Available: http://eprint.iacr.org/1999/009/.

AUTHORS

Chung-Li Wang He is a Ph. D. from University of California, Davis and now a staff engineer with Alibaba, Inc. His research topic includes secure computation, cryptography, error-controlcoding, and information theory.