SPDZ-BASED OPTIMISTIC FAIR MULTI-PARTY COMPUTATION

Chung-Li Wang
Alibaba Inc., Sunnyvale, California, USA

ABSTRACT

The fairness of multi-party computation has been investigated for long time. Classic results demonstrate that fair exchange can be achieved by utilizing cryptographic tools, as most of them are based on garbled circuits. For the secret-sharing schemes, such as SPDZ, it may incur significant overhead to simply apply a fair escrow scheme, since it encrypts all the shares of delivered results. To address this issue, we design a two-level secret-sharing mechanism. The escrow encryption is only for the first level of sharing and performed in preprocessing. The second level of sharing is used for computation and always handled by plaintexts, such that the online phase is still efficient. Our work also employs a semi-trusted third party (TTP) which provide optimistic escrow for output delivery. The verification and delivery procedures prevent the malicious parties from corrupting the outcome or aborting, 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 escrow decryption is needed only when misconduct is detected for opening the first-level shares.

KEYWORDS

Optimistic Multi-Party Computation, Secret-Sharing, Verifiability, 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 (e.g., [2]-[3]) have played significant roles in the success of MPC. These protocols have two phases: an input-independent offline phase for preprocessing 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. It is investigated in [7] and proven to imply robustness assuming a broadcast channel.

Fairness. The fairness is defined as the property that corrupted parties should not be able to prevent honest parties from receiving their output. In general, this property can be achieved by using cryptographic tools as in [8] and [9]. The escrow scheme [9] integrated garbled circuits and an optimistic escrow scheme with a semi-trusted third party. However, for SPDZ-like protocols, these similar approaches were all considered to be unaffordable, as discussed in [4]. As a consequence, the protocols with identifiable abort in [3] and [4][4] have to publicly reconstruct the secret before verifying the result, allowing corrupted parties to learn the output even though they cheated. The work in [12] uses threshold-t secret-sharing to support fairness and 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 and verifiability, if a malicious majority is assumed, having fairness is still an open problem for some of secret-sharing schemes.

Our Contribution. In the standard model, it is impossible to guarantee fairness with corrupted majority [17]. To overcome this impossibility, many protocols (e.g. [10]) have been proposed to achieve fairness in non-standard models using a trusted third party (TTP). The proposed solution is a public auditable MPC with identifiable abort and fairness using an optimistic escrow phase. Similarly as in SDPZ, the framework has the offline preprocessing phase and online computation phase. Our scheme uses a novel two-level secret sharing, the first-level is input-independent and randomly generated in the offline preprocessing, and the second-level is used in the online computation with correlated randomness.

Efficient Online Phase. Parties that engage in manipulating the computation can be identified by examining their commitments. A TTP manages the escrow scheme, which is not needed if all parties behave honestly for opening the shares. Our scheme provides better privacy than [13] as the TTP does not participate in the outcome delivery. Besides, if the TTP is dishonest, we only lose fairness and robustness, and the protocol is still public auditable with identifiable abort. The escrow scheme requires cryptographic tools, but our protocol takes advantage of two-level sharing and has the complexity overhead in the offline phase.

Verifiable Offline Phase. Our scheme improves upon the approaches in Overdrive's LowGear [5] to compute two-level shares of random values and triples in a single, trackable routine. The correctness and fairness of the computation is guaranteed by verifying the zero-knowledge proof (ZKP) of encrypted shares. This creates a checkpoint and adds 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].

Related Works. Typical approaches to have fairness involves employing semi-trusted third parties or other assumptions. [8] proposed an optimistic fair exchange using a TTP, and then it was recently shown that fair computation can be achieved by similar techniques in [9], [10], and [14], in which the output is encrypted and requires a third party to support a verifiable escrow scheme. As the encryption of shares demands excessive overhead in the online phase, this way is too expensive for SPDZ protocols. To save verification overhead, the trusted service is used in [14] but not assumed in this paper, but our work can be adapted in that case. An MPC with identifiable abort is proposed in [12], where fairness is obtained by t-secure secret-sharing when t or more servers are honest. A cryptographic solution, described in [13], is suggested to achieve fairness, but it incurs high complexity and relies on the TTP for decryption. We delay the in-depth comparisons of related works in the last chapter.

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 $m \le n$, 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*

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

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.

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 $z \in \mathbb{F}$:

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

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.

2.2. Important Blocks

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 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.

2.2.1 Novel Secret-Sharing

The online phase of the computation is conducted using the novel two-level sharing scheme, which is defined as below:

Definition 1 *(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 $[x] + [y] = \big((x_1 + y_1, \dots, x_n + y_n), (\tilde{x}_1 + \tilde{y}_1, \dots, \tilde{x}_n + \tilde{y}_n)\big)$, $e \cdot [x] = \big((e \cdot x_1, \dots, e \cdot x_n), (e \cdot \tilde{x}_1, \dots, e \cdot \tilde{x}_n)\big)$. 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 $[e + x] = [x] + (er^{-1}) \cdot [r]$.

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 [24], 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 1 *(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 Ψ $(\mathbf{D}, \mathbf{g}) \to \mathbf{D}\mathbf{g}^T = \boldsymbol{a} = (a_1 \dots, a_m)$* 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 **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.

2.2.2 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 [12]). 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 $\mathsf{E}_{(g,h)}([x], [r]) = \big((g^{x_1} h^{r_1}, \dots, g^{x_n} h^{r_n}), (g^{\tilde{x}_1} h^{\tilde{r}_1}, \dots, g^{\tilde{x}_n} h^{\tilde{r}_n})\big)$. When opening MUSS shares, we reconstruct the

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 $g_0, g_1, \dots, g_n, h \in \mathbb{G}$, and RO $\mathcal{Z}$ to verify ZKP. σ_f is the offline SIMD factor. Set $\boldsymbol{g}=\{g_i\}_{i=(0,\dots,n)}$.

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

1) The parties send $(\mathbf{Init}, \mathbb{F}, \mathbb{G}, \boldsymbol{g}, h)$ 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 $\nu_r \geq (2\nu_{\text{in}} + 4\nu_{\text{mul}})$, $\nu_{\text{trp}} \geq \nu_{\text{mul}}$ such that σ_f divides both ν_r and ν_{trp}. Then send $(\mathbf{Single}, \sigma_f, g_0, h)$ (ν_r/σ_f) times**,** $(\mathbf{Triple}, \sigma_f, g_0, h)$ $(\nu_{\text{trp}}/\sigma_f)$ times to $\mathcal{F}_{\text{Offline}}$.
3) Send $(\mathbf{Audit}, \sigma_f)$ $(\nu_r/\sigma_f + \nu_{\text{trp}}/\sigma_f)$ 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 $(\mathbf{Input}, I_i, id(u_i), u_i)$ from each $I_i \in \mathcal{I}$, $i \in \{m\}$ and $(\mathbf{Input}, I_i, id(u_i))$ 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 $(\mathbf{Compute}, \mathcal{C})$ 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 $\langle x_1\rangle, \dots \langle x_t\rangle$.
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).

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 [4], the computation of commitments is excluded in the circuit evaluation and invoked only after

Audit: On the input $(\textbf{Audit}, \{id(z_k)\}_{k\in\{m\}})$ 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 $(\textbf{Open})$ 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 sets $\text{Cheat} \leftarrow \top$.
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).

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. THE PROTOCOL

3.1. 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. 3.3.1. 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. **Error! Reference source not found.**. 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 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 [18]. 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 $\Pi^{\sigma}_{\mathsf{ChkPln}}$ (Figure 6 and cf. Sec. 3.1.2) 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 $\Pi^{\sigma}_{\mathsf{ChkEnc}}$ (Figure 4 and cf. Sec. 3.1.1). 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 $\Pi^{\sigma}_{\mathsf{ChkEnc}}$ and $\Pi^{\sigma}_{\mathsf{ChkPln}}$ since it eventually does not prevent opening.

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 correctness using an RO $\mathcal{Z}$ (cf. Sec. **Error! Reference source not found.**). 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 Sec. 4.1 by the following theorem.

Theorem 2 *(Online Security): In the ($\mathcal{F}_{\mathsf{Offline}}$,$\mathcal{F}_{\mathsf{Blt}}$,$\mathcal{F}_{\mathsf{Com}}$,$\mathcal{F}_{\mathsf{KGD}}$)-hybrid model with random oracles $\mathcal{K}$ and $\mathcal{Z}$, the protocol Π_{Online} implements $\mathcal{F}_{\mathsf{Online}}$ 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**.

3.1.1 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.
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 [4] as an effective way to verify the output. The correlation check protocol $\Pi^{\sigma}_{\mathsf{ChkEnc}}$ 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, and the proof is omitted due to the length limit.

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

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

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

1) The parties use $\mathcal{F}_{\mathsf{Rnd}}$ to publicly sample a vector $\boldsymbol{w} \overset{\$}{\leftarrow} \mathbb{F}^{\sigma}$.
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 $\mathcal{F}_{\mathsf{Blt}}$.
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 $\mathcal{F}_{\mathsf{Com}}$ to commit to η_i and μ_i.
5) Each $P_i \in P$ uses $\mathcal{F}_{\mathsf{Com}}$ 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. $\Pi^{\sigma}_{\mathsf{ChkEnc}}$: Protocol for the correlation check of encoded shares.

Lemma 1 *(Correlation Check for Encoded Shares): The protocol $\Pi^{\sigma}_{\mathsf{ChkEnc}}$ 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 $\Pi^{\sigma}_{\mathsf{ChkEnc}}$ 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.

3.1.2 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 $\Pi^{\sigma}_{\mathsf{ChkPln}}$ 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, and the proof is omitted due to the length limit. If $\Pi^{\sigma}_{\mathsf{ChkPln}}$ fails, the auditor P_{A} will check the commitments in the audit stage.

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 $\varepsilon_{\langle \eta\rangle} = \varepsilon_{\langle x\rangle} \times ({\varepsilon_{\langle y\rangle}}^{-1})$ and $\varepsilon_{\langle \rho\rangle} = \varepsilon_{\langle y\rangle} \times ({\varepsilon_{\langle b\rangle}}^{-1})$.
   b) Compute $\varepsilon_{\langle z\rangle} = \varepsilon_{\langle c\rangle} \times \left(\varepsilon_{\langle a\rangle}\right)^{\rho} \times \left(\varepsilon_{\langle b\rangle}\right)^{\eta} \times \left(\varepsilon_{\langle t\rangle}\right)^{\eta\rho(t^{-1})}$.
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.

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.*

3.2 Fairness

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* $n - 1$ *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 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.*

3.3 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

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

$\tilde{z}_i^{(k)}$ 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 $v_i \overset{\$}{\leftarrow} \mathbb{F}$. The parties use a new random value $\langle \boldsymbol{s} \rangle = ([\boldsymbol{s}], [\boldsymbol{t}], \varepsilon_{\langle \boldsymbol{s} \rangle})$ and use $\mathcal{F}_{\mathrm{Rnd}}$ to publicly sample a vector $\boldsymbol{w} \overset{\$}{\leftarrow} \mathbb{F}^{\sigma}$.
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. $\Pi^{\sigma}_{\mathsf{ChkPln}}$: Protocol for the correlation check of plain shares.

ciphers, **Single** uses the procedure $\Pi^{\sigma}_{\mathsf{ComShr}}$ which generates random MUSS shares, together with commitments to the values. For multiplication triples, $\Pi^{\sigma}_{\mathsf{GenTrp}}$ computes a product of the two random values and output them with commitments in **Triples**. These sub-protocols can be found 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, $\Pi^{\sigma}_{\mathsf{CPRZK}}$ (**Error! Reference source not found.**) and $\Pi^{\sigma}_{\mathsf{ChkZKP}}$ (**Error! Reference source not found.**) 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 $\Pi^{\sigma}_{\mathsf{CPRZK}}$ was executed such that the ciphertexts of MUSS ciphers were correctly encrypted from the plaintexts.
- The procedure $\Pi^{\sigma}_{\mathsf{ChkZKP}}$ 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.

The security of offline protocol is provided as below with the proof deferred to Sec. 4.2. The reader can refer to [26] for the details of $\Pi^{\sigma}_{\mathsf{ChkZKP}}$, which are omitted here.

Theorem 3 *(Offline Security): Let $\mathcal{H}$ be a semi-homomorphic cryptosystem. Then* Π_{Offline} *implements* $\mathcal{F}_{\mathsf{Offline}}$ *with computational security against any static adversary corrupting all parties but one honest computing party and auditor in the* ($\mathcal{F}_{\mathrm{KGD}}$, $\mathcal{F}_{\mathrm{Com}}$, $\mathcal{F}_{\mathrm{Blt}}$)*-hybrid model if the DLP is hard in* $\mathbb{G}$.

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.

3.3.1 Distributed Encryption

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 $\mathcal{F}_{\mathrm{KGD}}$ to generate the key pair (pkd, skd), where skd is shared among parties.
2) The third party P_T use $\mathcal{F}_{\mathrm{KGD}}$ to generate the global key pair (pko, sko) if it has none. Each P_j is given pko.
3) P_j samples $\alpha_{i,j} \overset{\$}{\leftarrow} \mathbb{F}$ for all $i \in \{n\}$. Set $\alpha_i = \sum_{j\in\{n\}} \alpha_{i,j}$.
4) Each P_j computes and broadcasts $[\![\mathbf{1}\cdot\alpha_{i,j}]\!]_{pkd} = \mathsf{Enc}_{pkd}(\mathbf{1}\cdot\alpha_{i,j})$ with all-one vector $\mathbf{1} \in \mathbb{F}^{\sigma}$ to $\mathcal{F}_{\mathrm{Blt}}$.
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 $d_j = E^{(n)}_{(\boldsymbol{g},h)}(\boldsymbol{\alpha}_j, \beta_j)$ toward $\mathcal{F}_{\mathrm{Blt}}$.
7) Each P_j computes $\mathrm{c}_j = \mathsf{Enc}_{pko}(\{\boldsymbol{\alpha}_j, \beta_j\}, u_j)$ with $u_j \in \mathbb{F}$ and invokes $(\textbf{genZKP}, P_j)$ of $\Pi^{\sigma}_{\mathrm{CPRZK}}$ 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 $\{\langle r^{(k)}\rangle\}_{k\in\{\sigma\}} \leftarrow \Pi^{\sigma}_{\mathrm{ComShr}}(\perp)$.
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 $\{\langle a^{(k)}\rangle\}_{k\in\{\sigma\}} \leftarrow \Pi^{\sigma}_{\mathrm{ComShr}}(\perp)$ and $\{\langle b^{(k)}\rangle\}_{k\in\{\sigma\}} \leftarrow \Pi^{\sigma}_{\mathrm{ComShr}}(\perp)$.
1) Run $\{\langle c^{(k)}\rangle\}_{k\in\{\sigma\}} \leftarrow \Pi^{\sigma}_{\mathrm{GenTrp}}$. 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 $\Pi^{\sigma}_{\mathrm{ComShr}}(\top)$ for **Single** and $\Pi^{3\sigma}_{\mathrm{ComShr}}(\top)$ for **Triples**.
2) Run $\Pi^{\sigma}_{\mathrm{ChkZKP}}$ once for **Single** and $\Pi^{\sigma}_{\mathrm{ChkZKP}}$ three times for **Triples**. If any of two replies **Reject**, P_A requests each P_i to open toward $\mathcal{F}_{\mathrm{Blt}}$ 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 $\mathcal{F}_{\mathrm{Blt}}$ the share of $\langle r_k\rangle$ as well as the proofs of commitments and encryptions. P_A reads transcripts from $\mathcal{F}_{\mathrm{Blt}}$ 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. $\mathrm{P_A}$ reads transcripts from $\mathcal{F}_{\mathrm{Blt}}$ and verifies to identify malicious parties $\mathcal{M}_T$.
3) If any check fails, $\mathrm{P_A}$ outputs **Reject** and $\{\mathcal{M}_R, \mathcal{M}_T\}$. Or P_A outputs **Accept**.

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

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

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

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 $[\![\boldsymbol{\alpha}_j]\!] = [\![\boldsymbol{1} \cdot \alpha_j]\!]$ for parallel processing.

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

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 $\Pi^{\sigma}_{\mathsf{GenShr}}$ 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 $\mathcal{F}_{\mathsf{Blt}}$. If $\mathsf{flag} = \top$, only open $\mathsf{E}(\boldsymbol{u}_i, \boldsymbol{v}_i)$.

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

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 $\boldsymbol{u}_i, \tilde{\boldsymbol{u}}_j \overset{\$}{\leftarrow} \mathbb{F}^{\sigma}$ 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 $\tilde{\boldsymbol{u}}_1$.

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

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 $\boldsymbol{\kappa}_{ji}, \tilde{\boldsymbol{\kappa}}_{ji} \overset{\$}{\leftarrow} \mathbb{F}^{\sigma}$ 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
$$[\![\boldsymbol{c}_j]\!] = \left(\boldsymbol{b}_j \otimes \left(\oplus_{i\in\{n\}}[\![\boldsymbol{a}_i \cdot \alpha_i]\!]\right)\right) \oplus \left(\oplus_{i\in\{n\}\backslash j}[\![\boldsymbol{\kappa}_{ij} \cdot \alpha_i]\!]\right) \oplus \left(\oplus_{i\in\{n\}\backslash j}[\![-\boldsymbol{\kappa}_{ji} \cdot \alpha_i]\!]\right)$$
$$[\![\tilde{\boldsymbol{c}}_j]\!] = \left(\tilde{\boldsymbol{b}}_j \otimes \left(\oplus_{i\in\{n\}}[\![\tilde{\boldsymbol{a}}_i]\!]\right)\right) \oplus \left(\oplus_{i\in\{n\}\backslash j}[\![\tilde{\boldsymbol{\kappa}}_{ij}]\!]\right) \oplus \left(\textstyle\sum_{i\in\{n\}\backslash j} -\tilde{\boldsymbol{\kappa}}_{ji}\right).$$
4) All parties call $\left(\textbf{Decrypt}, pkd, [\![\boldsymbol{c}_j]\!], P_i\right)$ and $\left(\textbf{Decrypt}, pkd, [\![\tilde{\boldsymbol{c}}_i]\!], P_i\right)$ for each $P_i \in \mathcal{P}$.
5) Each $P_j \in \mathcal{P}$ samples $\boldsymbol{t}_j \overset{\$}{\leftarrow} \mathbb{F}^{\sigma}$. All parties run $\Pi^{\sigma}_{\mathsf{GenShr}}$ 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 $\mathcal{F}_{\mathsf{Blt}}$ for $\langle \boldsymbol{c} \rangle = \left\{\langle c^{(k)} \rangle\right\}_{k\in\{\sigma\}}$.

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

$(pk, c, x, r) \rightarrow \mathbf{Cor}\,(pk, c, x, r)$, that maps to 1 if $pk \overset{\$}{\leftarrow} \mathsf{KG}(1^{\lambda}), x \in \mathbb{F}, r \overset{\$}{\leftarrow} \chi$ and $c \leftarrow \mathsf{Enc}_{pk}(x, r)$, 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 $\mathcal{F}_{\mathsf{KGD}}$ that will be used for the preprocessing. The key pair can be securely generated by a key-generation protocol, where the secret key is

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

Initialize: On input $(\textbf{Init}, \mathcal{C}, \mathbb{F}, \mathbb{G})$ from all parties (where $\mathcal{C}$ is a circuit with m inputs and m outputs, consisting of addition and multiplication gates over $\mathbb{F}$).

1) Send $(\textbf{Init}, \mathcal{C}, \mathbb{F}, \mathbb{G})$ to $\mathcal{A}$ and wait until $\mathcal{A}$ sends $\mathcal{L}_f, \mathcal{L}_p, \mathcal{L}_c, \mathcal{L}_o, \mathcal{L}_k \subseteq \mathcal{D} \subset \mathcal{P}$ and $\mathcal{L}_I \subseteq \mathcal{D}_I \subseteq \mathcal{I}$.
2) If $|\mathcal{L}_f| = 0$, reply **Accept**. Or reply $(\perp, \mathcal{L}_f)$ to all parties and stop.

Input: On input $(\textbf{Input}, I_i, id(x_i), x_i)$ from each I_i and $(\textbf{Input}, I_i, id(x_i))$ from all computing parties, with $id(x)$ a new identifier and $x_i \in \mathbb{F}$. D_I is the set of corrupted input party. $\mathcal{L}_I$ is its subset sending $x_i = \perp$ to the trust party.

1) Get and override x_i for each party in $\mathcal{D}_I$ from $\mathcal{A}$. If $|\mathcal{L}_I| > 0$, reply $(\perp, \mathcal{L}_I)$ to all parties.
2) Set $\boldsymbol{x} = (x_1, \ldots x_m)$. Store $(id(\boldsymbol{x}), \boldsymbol{x})$. Reply **Accept**.

Compute: On input $(\textbf{Compute}, \mathcal{C})$ from all parties.

1) For every $f_i \in \mathcal{C}$, compute $y_i = f_i(\boldsymbol{x})$. Set $\boldsymbol{y} = \{y_i\}_{i \in \{m\}}$.
2) If $|\mathcal{L}_p| = 0$, set $y_i^* \leftarrow y_i$. Or $y_i^* \leftarrow \perp$.
3) If $|\mathcal{L}_p| = 0$ and $|L_c| = 0$, reply **Accept**. Or reply **Reject**.

Audit: On input $(\textbf{Audit}, id(\boldsymbol{x}))$ from P_A. If $|\mathcal{L}_p| = 0$, reply **Accept**. Or reply $(\perp, \mathcal{L}_p)$ to all parties and stop.

Open: On input $(\textbf{Open})$ from all parties.

1) Send y^* to P_T.
2) If $|\mathcal{L}_o| > 0$, reply $(\perp, \mathcal{L}_o)$ to all parties. Stop.
3) If $|\mathcal{L}_o| = 0$, send ok to P_T and wait for the response.
    3.1. If P_T is dishonest, $\mathcal{A}$ decides for him to send **Cheat** or $\perp$.
    3.2. If P_T is honest, he always sends back $\perp$.
4) If P_T replies $\perp$, reply $(y^*, \mathcal{L}_k)$ to all parties. Or if P_T replies **Cheat**, reply (y^*, P_T) if $|\mathcal{L}_k| = 0$ or $(\perp, \{\mathcal{L}_k, P_T\})$ to all parties.

Figure 9. $\mathcal{F}_{\text{Online}}$: Ideal functionality for the online phase.

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.

3.3.2 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 $\mathcal{F}_{\text{KGD}}$ 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 $\mathbf{1} = \{1\}_{i \in \{\ell\}}$, $\boldsymbol{r}_{i,j} \overset{\$}{\leftarrow} \mathbb{F}^{\ell}$, and $u_j \overset{\$}{\leftarrow} \mathbb{F}$ 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 $c_j = \mathsf{Enc}_{pko}(\{\overline{\boldsymbol{\alpha}}_j, \beta_j\}, u_j)$ for $\overline{\boldsymbol{\alpha}}_j = \{\alpha_{i,j}\}_{i \in \{n\}}$ 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: $R^{(n,\ell)}_{CPR,j} = \Big\{(\mathbf{s}, \boldsymbol{a}) | \mathbf{s} = \left(\{b_{i,j}\}_{i \in \{n\}}, c_j, d_j, pkd, pko\right), \boldsymbol{a} = (\boldsymbol{\alpha}_j, \beta_j, \boldsymbol{r}_{i,j}, u_j), \mathbf{Cor}(pkd, b_{i,j}, (\mathbf{1} \cdot$

$\mathcal{S}_{\mathsf{Online}}$

Initialize:
1) Set up $\mathcal{F}_{\mathsf{KGD}}, \mathcal{F}_{\mathrm{Blt}}, \mathcal{F}_{\mathsf{Com}}$ and RO's $\mathcal{K}$ and $\mathcal{Z}$. Start a local instance Π of Π_{Online} with which the dishonest parties will communicate.
2) Send $\mathcal{L}_f, \mathcal{L}_p, \mathcal{L}_c, \mathcal{L}_o, \mathcal{L}_k$ and $\mathcal{L}_I$ to $\mathcal{F}_{\mathsf{Online}}$. Then send $(\mathbf{Init}, \mathcal{C}, \mathbb{F}, \mathbb{G})$ for each $P_j \in \mathcal{D}$ to $\mathcal{F}_{\mathsf{Online}}$.
3) Run a copy of $\mathcal{F}_{\mathsf{Offline}}$, with which the dishonest parties and the simulated honest parties communicate. If $|\mathcal{L}_f| > 0$, stop.
4) Sample a generator $g \in \mathbb{G}$ at random, choose $t_1, \dots, t_n, s \overset{\$}{\leftarrow} \mathbb{Z}^*_{|\mathbb{G}|}$ and set $t_0 = 1$, $g_i = g^{t_k}, h = g^s$ for $k \in \{n\}$. Set $\boldsymbol{g}$=$\{g_i\}_{i=(0,\dots,n)}$ and the random oracle $\mathcal{K}$ to output $\boldsymbol{g}$ and h. Then run this protocol step as in Π_{Online}.
5) Send $(\mathbf{Init}, \mathbb{F}, \mathbb{G}, \boldsymbol{g}, h)$ for all simulated P_i. Record the $\{\alpha_{i,j}\}_{i,j\in\{n\}}$ used by $\mathcal{F}_{\mathsf{Offline}}$.
6) Send $(\mathbf{Input}, \mathrm{v}_{\mathrm{r}}, g_0, h)$ and $(\mathbf{Triple}, \mathrm{v}_{\mathrm{trp}}, g_0, h)$ as in Π_{Online} for all simulated P_i.

Input: $I_i \in \mathcal{I}$ inputs a value u_i.
1) If I_i is honest, then follow Π_{Online} for a default input value $u_i \leftarrow \tilde{u}_i$.
2) If $I_i \in \mathcal{D}_I$ is dishonest, then extract the input value u_i from Π and send $(\mathbf{Input}, I_i, id(u_i), u_i)$ for I_i and $(\mathbf{Input}, I_i, id(u_i))$ for each dishonest $P_j \in \mathcal{D}$ to $\mathcal{F}_{\mathsf{Online}}$.

Compute: If **Initialize** has been executed and inputs for all m inputs of $\mathcal{C}$ have been provided, evaluate $\mathcal{C}$ gate per gate as follows:

Add and **Multiply**: Follow the steps in Π_{Online}.

Output: Send (**Compute**) to $\mathcal{F}_{\mathsf{Online}}$. Obtain the output y from $\mathcal{F}_{\mathsf{Online}}$. Simulate Π_{Online} as follows:

1) Generate correct shares for the simulated honest parties for Π:
   1.1. Let P_j be a simulated honest party and y' be the output of the simulated protocol. Set $\langle y' \rangle = ([y'], [r'], \varepsilon_{\langle y' \rangle})$.
   1.2. For P_j set new shares as follows: $y_i = y'_i + (y - y')/\alpha_i$ and $\tilde{y}_i = \tilde{y}'_i + (y - y')$, $r_i = r'_i + ((y' - y)/\alpha_i) \cdot s^{-1}$, and $\tilde{r}_i = \tilde{r}'_i + (y' - y) \cdot s^{-1}$.
   1.3. Follow the protocol Π_{Online} to run Π^t_{ChkPln} and Π^m_{ChkEnc} with $\mathcal{L}_c$ and $\mathcal{L}_k$.

Audit: Run **Audit** as in Π_{Online} to get $\mathcal{L}_p$. Then invoke **Audit** in $\mathcal{F}_{\mathsf{Online}}$.
Open: Run **Open** as in Π_{Online} to get $\mathcal{L}_o$ and $\mathcal{L}_k$. Then invoke **Open** in $\mathcal{F}_{\mathsf{Online}}$.

Figure 10. $\mathcal{S}_{\mathsf{Online}}$: Simulator for the protocol Π_{Online}.

$\alpha_{i,j}), r_{i,j}) = 1, \mathbf{Cor}(pko, c_j, \{\overline{\boldsymbol{\alpha}}_j, \beta_j\}, u_j) = 1, \{b_{i,j}\}_{i\in\{n\}} = \left\{[\![\mathbf{1} \cdot \alpha_{i,j}]\!]_{pkd}\right\}_{i\in\{n\}}, \mathrm{c}_j = \mathsf{Enc}_{pko}(\{\overline{\boldsymbol{\alpha}}_j, \beta_j\}, u_j), d_j = \mathsf{E}^{(n)}_{(\boldsymbol{g},h)}(\overline{\boldsymbol{\alpha}}_j, \beta_j)\Big\}.$

Based on [19], the ZKP protocol $\Pi^{(n,\ell)}_{\mathsf{CPRZK}}$ is described in the conference version of this paper [26].

4. PROOFS

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

Set $\mathcal{D}$ as the set of dishonest parties, and $\mathcal{H} = (\mathsf{KG}, \mathsf{Enc}, \mathsf{Dec}, \oplus, \otimes)$ as PKI.

Initialize: On input $(\textbf{Init}, \sigma, \mathbb{F}, \mathbb{G}, \boldsymbol{g}, h, \mathcal{K})$ from all players.

1) For all $P_j \in \mathcal{D}$ and $i \in \{n\}$, $\mathcal{A}$ input $\alpha_{i,j}, \beta_j, u_j \in \mathbb{F}$ and $\boldsymbol{r}_{i,j} \in \mathbb{F}^{\ell}$. For all $P_j \in \mathcal{P}\backslash\mathcal{D}$ and $i \in \{n\}$, the funtionality chooses $\alpha_{i,j}, \beta_j, u_j \overset{\$}{\leftarrow} \mathbb{F}$, $\boldsymbol{r}_{i,j} \overset{\$}{\leftarrow} \mathbb{F}^{\ell}$ uniformly at random.
2) Get $(pkd, skd) \leftarrow \mathcal{H}.\mathsf{KG}(1^{\lambda})$, $(pko, sko) \leftarrow \mathcal{H}.\mathsf{KG}(1^{\lambda})$ and send pkd and pko to $\mathcal{A}$. Send (pko, sko) to P_T.
3) Set $\mathbf{1} = \{1\}_{k\in\{\sigma\}}$. Obtain $[\![\mathbf{1}\cdot\alpha_{i,j}]\!]_{pkd}$ and send to $\mathcal{A}$. Set $\overline{\boldsymbol{\alpha}}_j = \{\alpha_{i,j}\}_{i\in\{n\}}$.
4) For each honest $P_i \in \mathcal{P}\backslash\mathcal{D}$ compute $\omega_i = \left(\left\{[\![\mathbf{1}\cdot\alpha_{i,j}]\!]_{pkd}\right\}_{i\in\{n\}}, c_j, d_j, \zeta_i\right)$ with $d_i = \mathsf{E}^{(n)}_{(\boldsymbol{g},h)}(\overline{\boldsymbol{\alpha}}_j, \beta_i)$, $c_i = [\![(\overline{\boldsymbol{\alpha}}_j, \beta_j)]\!]_{pko}$, and ZKP ζ_i.
5) For each dishonest $P_i \in \mathcal{D}$, $\mathcal{A}$ sends ω_i to P_i.

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

1) Set $\mathcal{M}_R \leftarrow \{\}$.
2) For each honest P_i, the functionality chooses, $\boldsymbol{u}_i, \boldsymbol{v}_i \overset{\$}{\leftarrow} \mathbb{F}^{\sigma}$, send these to P_i.
3) For each dishonest P_i, $\mathcal{A}$ inputs $\boldsymbol{u}_i, \boldsymbol{v}_i$. Add P_i to $\mathcal{M}_R$ if input is not in $\mathbb{F}^{\sigma}$
4) Run the sub-function to compute $(\langle\boldsymbol{u}\rangle, \mathcal{L}_u) \leftarrow \mathcal{F}^{\sigma}_{\text{GenShr}}(\boldsymbol{u}_1, \ldots, \boldsymbol{u}_{\text{n}}, \boldsymbol{v}_1, \ldots, \boldsymbol{v}_{\text{n}})$.
5) Add $\mathcal{L}_u$ to $\mathcal{M}_R$.

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

1) $\mathcal{A}$ sends a partial list of malicious parties M_T to the functionality.
2) For each honest P_i, the functionality chooses $\boldsymbol{a}_i, \boldsymbol{b}_i, \boldsymbol{r}_i, \boldsymbol{s}_i \overset{\$}{\leftarrow} \mathbb{F}^{\sigma}$and send to P_i.
3) For each dishonest P_i, $\mathcal{A}$ inputs $\boldsymbol{a}_i, \boldsymbol{b}_i, \boldsymbol{c}_i, \boldsymbol{r}_i, \boldsymbol{s}_i, \boldsymbol{t}_i$. Set $\boldsymbol{a} = \sum_{i\in\{n\}} \boldsymbol{a}_i \cdot \alpha_i$, $\boldsymbol{b} = \sum_{i\in\{n\}} \boldsymbol{b}_i \cdot \alpha_i$. Add P_i to $\mathcal{M}_T$ if input is not in $\mathbb{F}^{\sigma}$.
4) Randomly select one party P_i, the functionality samples $\boldsymbol{c}_j, \boldsymbol{t}_j \overset{\$}{\leftarrow} \mathbb{F}^{\sigma}$ for $P_j \in \mathcal{P}\backslash P_i$ and set $\boldsymbol{c}_i^* = \boldsymbol{a}\cdot\boldsymbol{b} - \sum_{j\in\{n\}/i} \boldsymbol{c}_j \cdot \alpha_j$ and $\boldsymbol{t}_i^* \overset{\$}{\leftarrow} \mathbb{F}^{\sigma}$ for P_i.
    - 4.1. If P_i is dishonest, send $\boldsymbol{c}_i^*$, $\boldsymbol{t}_i^*$ to $\mathcal{A}$ and wait for $\boldsymbol{c}_i, \boldsymbol{t}_i$ from $\mathcal{A}$. If $\boldsymbol{c}_i^* \neq \boldsymbol{c}_i$ or $\boldsymbol{t}_i^* \neq \boldsymbol{t}_i$, add P_i into $\mathcal{M}_T$.
    - 4.2. If P_i is honest, set $\boldsymbol{c}_i = \boldsymbol{c}_i^*$ and $\boldsymbol{t}_i = \boldsymbol{t}_i^*$.
5) Run the sub-protocols to compute
    $(\langle\boldsymbol{a}\rangle, \mathcal{L}_a) \leftarrow \mathcal{F}^{\sigma}_{\text{GenShr}}(\boldsymbol{a}_1, \ldots, \boldsymbol{a}_{\text{n}}, \boldsymbol{r}_1, \ldots, \boldsymbol{r}_{\text{n}})$,
    $(\langle\boldsymbol{b}\rangle, \mathcal{L}_b) \leftarrow \mathcal{F}^{\sigma}_{\text{GenShr}}(\boldsymbol{b}_1, \ldots, \boldsymbol{b}_{\text{n}}, \boldsymbol{s}_1, \ldots, \boldsymbol{s}_{\text{n}})$,
    $(\langle\boldsymbol{c}\rangle, \mathcal{L}_c) \leftarrow \mathcal{F}^{\sigma}_{\text{GenShr}}(\boldsymbol{c}_1, \ldots, \boldsymbol{c}_{\text{n}}, \boldsymbol{t}_1, \ldots, \boldsymbol{t}_{\text{n}})$. Add $\mathcal{L}_a$, $\mathcal{L}_b$, and $\mathcal{L}_c$ to $\mathcal{M}_T$.

Audit: On input $(\textbf{Audit}, \sigma)$ from all players. If $|\{\mathcal{M}_R, \mathcal{M}_T\}| > 0$. Reply **Reject** and $\{\mathcal{M}_R, \mathcal{M}_T\}$. Or reply **Accept**.

Figure 11. $\mathcal{F}_{\text{Offline}}$: Ideal functionality for the offline phase.

4.1 Security of the Online Phase

We will now prove security for the construction from Sec. 3.1 in the UC framework. which implies that Π_{Online} implements $\mathcal{F}_{\text{Online}}$ as in Figure 9 in a hybrid model as defined in Theorem 2.

Proof: The proof of the statement is provided by the simulator $\mathcal{S}_{\text{Online}}$ in Figure 10, which requires at least one honest party to run an instance of Π_{Online}. The simulator and the honest parties are controlled by $\mathcal{A}$. During **Initialize**, **Input**, **Add**, **Multiply**, $\mathcal{S}_{\text{Online}}$ performs the same steps as in

$\mathcal{S}_{\text{Offline}}$

Initialize:
1) Set up $\mathcal{F}_{\text{KGD}}, \mathcal{F}_{\text{Blt}}, \mathcal{F}_{\text{Com}}$, and the commitment key $(\boldsymbol{g}, h)$ and simulate honest parties in a local instance Π of Π_{Online} with the dishonest parties.
2) Send M_r and M_t to $\mathcal{F}_{\text{Offline}}$. Then send $(\textbf{Init}, \sigma, \mathbb{F}, \mathbb{G}, \boldsymbol{g}, h, H)$ for each $P_j \in \mathcal{D}$ to $\mathcal{F}_{\text{Offline}}$.
3) The simulator obtains (pkd, skd) and (pko, sko) from $\mathcal{F}_{\text{KGD}}$ in Π.

Single: On input $(\textbf{Single}, \sigma, g, h)$, Π outputs random values $\{\langle r^{(k)}\rangle\}_{k\in\{\sigma\}}$.
1) Obtain the commitment $\varepsilon_i^{(k)}$ and $\tilde{\varepsilon}_i^{(k)}$ from Step 3 of $\mathcal{F}_{\text{GenShr}}^{\sigma}$ for honest parties. For each honest P_i of $i > 1$, sample $u_i^{(k)}, \tilde{u}_i^{(k)} \overset{\$}{\leftarrow} \mathbb{F}$ and set $v_i^{(k)}$ and $\tilde{v}_i^{(k)}$ such that $\varepsilon_i^{(k)} = \mathsf{E}\left(u_i^{(k)}, v_i^{(k)}\right)$ and $\tilde{\varepsilon}_i^{(k)} = \mathsf{E}\left(\tilde{u}_i^{(k)}, \tilde{v}_i^{(k)}\right)$. For the honest P_i, sample $u_i^{(k)} \overset{\$}{\leftarrow} \mathbb{F}$ and set $v_i^{(k)}$ such that $\varepsilon_i^{(k)} = \mathsf{E}\left(u_i^{(k)}, v_i^{(k)}\right)$.
2) For each dishonest P_i, decrypt $[\![\boldsymbol{\mu}_i]\!]$ (for $\boldsymbol{u}_i$ and $\boldsymbol{v}_i$) and $[\![\alpha_i]\!]$ using skd. Sample $u_i^{(k)}, v_i^{(k)} \overset{\$}{\leftarrow} \mathbb{F}$ and obtain $\tilde{u}_i^{(k)} = u_i^{(k)} \cdot \alpha_i - \mu_i^{(k)}$ and $\tilde{v}_i^{(k)}$ similarly. Send them to $\mathcal{F}_{\text{Offline}}$.

Triples: On input $(\textbf{Triple}, \sigma, g, h)$, Π outputs random values $\{t^{(k)}\}_{k\in\{\sigma\}}$.
1) Execute the same simulation of **Single** for $\{\langle a^{(k)}\rangle\}_{k\in\{\sigma\}}$ and $\{\langle b^{(k)}\rangle\}_{k\in\{\sigma\}}$.
2) Obtain the commitment $\varepsilon_i^{(k)}$ and $\tilde{\varepsilon}_i^{(k)}$ from Step 3 of $\mathcal{F}_{\text{GenShr}}^{\sigma}$ for honest parties. For each honest P_i, sample $u_i^{(k)}, \tilde{u}_i^{(k)} \overset{\$}{\leftarrow} \mathbb{F}$ and set $v_i^{(k)}$ and $\tilde{v}_i^{(k)}$ such that $\varepsilon_i^{(k)} = \mathsf{E}\left(u_i^{(k)}, v_i^{(k)}\right)$ and $\tilde{\varepsilon}_i^{(k)} = \mathsf{E}\left(\tilde{u}_i^{(k)}, \tilde{v}_i^{(k)}\right)$. For the honest P_i, sample $u_i^{(k)} \overset{\$}{\leftarrow} \mathbb{F}$ and set $v_i^{(k)}$ such that $\varepsilon_i^{(k)} = \mathsf{E}\left(u_i^{(k)}, v_i^{(k)}\right)$.
3) For each dishonest P_i, decrypt $[\![\boldsymbol{c}_i]\!]$ and $[\![\tilde{\boldsymbol{c}}_j]\!]$ to obtain $c_i^{(k)}$and $\tilde{c}_i^{(k)}$. Decrypt $[\![\boldsymbol{\mu}_i]\!]$ (for $t_i^{(k)}$). Sample $t_i^{(k)} \overset{\$}{\leftarrow} \mathbb{F}$ and obtain $\tilde{t}_i^{(k)} = t_i^{(k)}\alpha_i - \mu_i^{(k)}$. Send them to $\mathcal{F}_{\text{Offline}}$.

Audit: Run Step 1 and 2 of **Single** in the simulation except random chosen $\tilde{v}_i^{(k)} \overset{\$}{\leftarrow} \mathbb{F}$. Then run **Audit** as in Π_{Offline} to get $\{\mathcal{M}_R, \mathcal{M}_T\}$. Then invoke **Audit** in $\mathcal{F}_{\text{Offline}}$.

Figure 12. $\mathcal{S}_{\text{Offline}}$: Simulator for the protocol Π_{Offline}

Π_{Online} and obtains MUSS ciphers α_i from $\mathcal{F}_{\text{Offline}}$. It also sets up the Random Oracle (RO) for commitment keys and uses a fixed default input $\tilde{u}_i$ defined by $\mathcal{C}$ for the simulated honest parties during **Input**.

Every set of at most $n-1$ MUSS encoded and plain shares of a value is uniformly random and does not reveal any information about the shared secret, so it is indistinguishable from a real transcript. During **Output**, the shares of one simulated honest party are adjusted to match the correct output y from $\mathcal{F}_{\text{Online}}$. By obtaining shares from $\mathcal{F}_{\text{Com}}$, the simulator derived the result y' of the simulation, so it can adjust the encoded and plain shares of a simulated honest party. For each encoded share y_i, there exists only one r_i that opens the commitment $E(y_i, r_i)$ correctly, so it is indistinguishable. The same is true for the plain shares.

If the encoded shares generated by $\mathcal{F}_{\text{Offline}}$ follow uniform distribution, the property still holds after any linear operation and scalar multiplication. It follows from Theorem 1 of [24] that except negligible probability $\mathrm{o}(1/\mathrm{q})$ the matrix composed of opened encoded shares achieves full rank, leading to perfect secrecy as stated in Theorem 1. This ensures that no information about the output can be gained from every set of MUSS encoded shares of m results. In the ideal world, the simulator outputs **Reject** in **Output** if any of the values opened by the dishonest parties was inconsistent, while Π_{Online} does so if $\Pi^{t}_{\mathrm{ChkPln}}$ and $\Pi^{m}_{\mathrm{ChkEnc}}$ may pass. This occurs with a probability of $\mathrm{o}(1/\mathrm{p})$, which is negligible in λ.

At the beginning of execution, $\mathcal{A}$ decides to stop the execution or affect the outcome by corrupting dishonest parties, then $\mathcal{S}_{\mathrm{Online}}$ will forward the set of malicious parties to the ideal functionality. During the **Audit** and **Opening** stage, we also do exactly the same as in the protocol. The MUSS ciphers α_i given by $\mathcal{F}_{\mathrm{Offline}}$ are uniformly random and thus indistinguishable from the counterparts from $\mathcal{F}_{\mathrm{Online}}$. □

4.2 Security of the Offline Phase

We will now prove security for the construction from Π_{Offline} in Figure 7 in the UC framework, which implies that Π_{Offline} implements $\mathcal{F}_{\mathrm{Offline}}$ described in Figure 11 and Theorem 3.

Proof: The simulator $\mathcal{S}_{\mathrm{Offline}}$ in Figure 12 will generate MUSS shares that are uniformly random, and use the decryption key to fit these shares to the commitments that $\mathcal{F}_{\mathrm{Offline}}$ outputs for the honest parties. Hence, the values of the dishonest parties are consistent with those values that the honest parties obtain and are indistinguishable. If $\mathcal{A}$ cheats during the decryption, then the simulator will always abort by running $\Pi^{\sigma}_{\mathrm{ChkZKP}}$, which guarantee the correctness and soundness, assuming the existence of an honest verifier. We do not directly decrypt the ciphertexts and check MUSS correlation, because it is impossible to differentiate the misbehavior in the proofs and that in the corresponding shares. Since there is at least one honest computing party, the opening will always be invoked if the MUSS correlation is violated. One can see that if the check fails, $\mathcal{S}$ makes $\mathcal{F}_{\mathrm{Offline}}$ abort which is consistent with the protocol. □

5. DISCUSSIONS AND CONCLUSIONS

Comparisons. We compare our proposed MPC with the other three works in [9], [12], and [13], as summarized in Table 1. Cachin and Camenisch [9] use encrypted circuits for computation where two parties exchange input tokens through verifiable oblivious transfer. Moreover, it uses a TTP to resolve misbehavior. The protocol in [12] uses Shamir's sharing to construct a public auditable MPC in a lack of the TTP. The fairness is achievable when there exist enough honest parties to recover the secret. Seo's work in [13] requires all party to provide verifiable encrypted shares to the TTP such that it can verify and decrypt the shares.

Result delivery: "Decrypt to open?" means if the opening procedure requires **decryption**, which could be done unconditionally (by "Yes, always") or in case of detected misbehavior (by "Yes, optimistic"). "Verify first" denotes the verification of results **before** revealing their plaintexts to the parties, and "Reveal first" indicates the verification **after** the revealing.

Fairness: [9], [13], and our work are fair if there exists at least one honest party and TTP to detect the misconduct of other malicious parties. However, [12] needs at least t honest party due to threshold- t sharing and the lack of TTP.

Privacy: [12] can protect privacy against at most $\mathrm{t}-1$ malicious and colluding computing parties

Works	TTP existence	Decrypt to open?	Verify or/Reveal First?	Guaranteed Fairness	Guaranteed Privacy	Online # messages	Offline # messages

[9]	Yes	Yes, optimistic.	Verify first	At least 1 honest	Yes to TTP. Up to 1 malicious	$O(\ell\lambda n^2\lvert\mathcal{C}\rvert)$	N/A
[12]	No	No.	Reveal first	At least t honest	Up to $\mathrm{t}-1$ malicious	$O(\ell\lambda n^2 + n^2\lvert\mathcal{C}\rvert)$	$O(\lambda n^2\lvert\mathcal{C}\rvert)$
[13]	Yes	Yes, always.	Verify first	At least 1 honest	No to TTP Up to n-1 malicious	$O(\ell\lambda n + n^2\lvert\mathcal{C}\rvert)$	$O(\lambda n^2\lvert\mathcal{C}\rvert)$
This work	Yes	Yes, optimistic	Verify first	At least 1 honest	Yes to TTP. Up to n-1 malicious	$O(n^2\lvert\mathcal{C}\rvert)$	$O(\ell\lambda n^2 + \lambda n^2\lvert\mathcal{C}\rvert)$

Table. 1: Comparisons with previous works in various properties.

due to threshold- t sharing. [13] and our work can do with at most $\mathrm{n}-1$ malicious ones. [9] is susceptible to input corruption attack, as mentioned in [14]. Furthermore, it should be noted that in the schemes of [9] and this paper TTP has no knowledge of secret, but the TTP in [13] has the access to opening share in plaintext.

Communication: The protocol in [9] needs to check non-interactive ZKP (NIZKP) for each encrypted gate in a garbled circuit for each pair of parties. As a consequence, the number of messages is $O(\ell\lambda \mathrm{n}^2\lvert\mathcal{C}\rvert)$. Its offline number of messages is N/A because of its lack of preprocessing. The delivery procedure in [12] broadcasts NIZKP for the commitment of each share to each party. Besides, using beaver triples needs to broadcast n messages for each multiplication gate in online computing. The computing parties in [13] have to sends NIZKP of encrypted shares to the TPP, so the factor is only n. However, our work only requires $O(\mathrm{n}^2\lvert\mathcal{C}\rvert)$ for broadcasting shares in plaintext for result delivery and multiplication. In addition, [12], [13], and ours all use $O(\lambda \mathrm{n}^2\lvert\mathcal{C}\rvert)$ messages to generate correlated randomness in offline preprocessing, and our work additionally sends $O(\ell\lambda n^2)$ to broadcast NIZKP for $\Pi^{\sigma}_{\mathrm{CPRZK}}$.

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

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

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

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

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

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

[10] H. Kılınç and A. Küpçü, "Optimally Efficient Multi-Party Fair Exchange and Fair Secure Multi-Party Computation," CT-RSA 2015, pp. 330-349, 2015.

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

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

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

[14] A. Herzberg and H. Shulman, "Oblivious and Fair Server-Aided Two-Party Computation," ARES 2012, pp. 75-84, 2012.

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

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

[17] R. Cleve, "Limits on the Security of Coin Flips When Half the Processors Are Faulty (extended abstract)," STOC, pages 364–369. ACM, 1986.

[18] D. Beaver, "Efficient Multiparty Protocols Using Circuit Randomization," CRYPTO '91, pp. 420-432, 1991.

[19] R. Cramer and I. Damgård, "On the Amortized Complexity of Zero- Knowledge Protocols," CRYPTO 2009, pp. 177-191, 2009.

[20] R. Canetti, "Universally Composable Security: A New Paradigm for Cryptographic Protocols," FOCS 2001, pp. 136-145, 2001.

[21] MP-SPDZ 2022 [online] Available: https://github.com/data61/.

[22] M. Keller, "MP-SPDZ: A Versatile Framework for Multi-Party Computation," CCS 2020, pp. 1575-1590, 2020.

[23] E. Orsini, "Efficient Actively Secure MPC with a Dishonest Majority: A Survey," WAIFI 2020, pp. 42-71, 2020.

[24] 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.

[25] I. Damgård, "Non-Interactive Circuit Based Proofs and Non-Interactive Perfect Zero-Knowledge with Preprocessing, " EUROCRYPT 1992, LNCS, vol. 658, pp. 341-355, 1992.

[26] C. Wang, " Efficient Fair and Robust SPDZ-Like Multi-Party Computation," CRYPIS 2023, Available: https://aircconline.com/csit/abstract/v13n13/csit131327.html.

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-control coding, and information theory.