MVOC
A Lighter Multi-Client Verifiable Outsourced Computation for Malicious Lightweight Clients

Wang, Xingkai; Cao, Zhenfu; Liu, Zhen; Liang, Kaitai

DOI
10.1109/TDSC.2024.3449770

Publication date
2024

Document Version
Final published version

Published in
IEEE Transactions on Dependable and Secure Computing

MVOC: A Lighter Multi-Client Verifiable Outsourced Computation for Malicious Lightweight Clients

Xingkai Wang, Zhenfu Cao, *Senior Member, IEEE*, Zhen Liu, and Kaitai Liang, *Member, IEEE*

***Abstract*—Gordon et al. systematically studied the Universally Composable (UC) security of Multi-client Verifiable Computation (MVC), in which a set of computationally-weak clients delegate the computation of a general function to an untrusted server based on their private inputs, and proposed a UC-secure scheme ensuring that the protocol remains secure even when arbitrarily composed with other UC-secure instances. However, this scheme imposed a significant computational overhead on clients due to the utilization of fully homomorphic encryption, and the plaintext size scaled linearly with function input size. In this work, we present MVOC, a more efficient UC-secure MVC protocol, that significantly reduces the amortized overhead for clients in both semi-honest and malicious settings, by delegating a larger portion of the computation to the server. We enable clients to verify the garbled circuit before entering the online phase, ensuring security against malicious clients without incurring heavy overhead of compiling a semi-honest protocol into a malicious one. We present the detailed proof and analyze the theoretical complexity of MVOC. Furthermore, we implement our protocol and evaluate the performance, and the results demonstrate that the computation and communication overheads during the input phase can be decreased by at least 95.55% and 87.17%, respectively.**

***Index Terms*—Hybrid homomorphic encryption, outsourced computation, verifiable computation.**

I. Introduction

THE techniques of Verifiable Computation (VC) [1] and Multi-client Verifiable Computation (MVC) [2], [3] have been introduced to empower computationally weak clients by enabling them to delegate the computation of a function f on private inputs to a remote server, achieving privacy, soundness, and efficiency in both single-client and multi-client scenarios. In more detail, privacy necessitates the safeguarding of sensitive data, encompassing both input and output, in order to prevent any unauthorized disclosure. Soundness ensures the integrity and validity of the result produced by the server, ensuring they are correct and trustworthy. Efficiency guarantees the cost incurred by the client during outsourcing process remains significantly lower compared to the computational cost of performing the function independently.

Outsourced computing is a typical instance of verifiable computation. The landscape comprises a multitude of devices, ranging from smart utility systems and electric vehicles to smartphones and smartwatches. Despite their varying computational capacities, many of these devices rely on submitting their privacy-sensitive data to cloud servers for the computation of intricate models. The outcomes are then retrieved and utilized locally for enhanced functionality and convenience. As shown in Fig. 1, upon reaching consensus among clients regarding the function f to be executed, the server and each client generate key pairs, and exchange a limited number of verification messages to prepare for the online phase. These messages are represented by solid lines in the figure. Then, in each round of the online phase, the server computes ξ, the ciphertext representation of input x, through function F corresponding to f in its ciphertext form. The server then derives the ciphertext result ω and transmit it to the clients, enabling them to obtain the plaintext result y, using private decoding information τ. A more detailed definition of non-interactive multi-client verifiable computation will be shown in Section II, including the explanation of corresponding value of e, d, ρ.

Gennaro et al. [1] introduced the definition of verifiable computation, with the primary goal of achieving both privacy and soundness against a potentially malicious server, assuming that the client is honest. The protocol achieves efficiency *amortizedly*, by combining garbled circuit [4] with a fully homomorphic encryption (FHE) scheme [5]. More specifically, after generating a garbled circuit, the client may employ a FHE scheme to encrypt the circuit along with the encoded input labels. The IND-CPA security of the FHE scheme enables the client to reuse the same circuit without compromising soundness, thereby ensuring that the computational cost for the client remains bounded in an amortized sense. Choi et al. [2] later extended the concept of single-client VC to a multi-client setting, resulting in a MVC protocol, where a group of n clients collaborates to

Received 11 April 2024; revised 7 August 2024; accepted 13 August 2024. Date of publication 26 August 2024; date of current version 14 March 2025. This work was supported in part by the National Key Research and Development Program of China under Grant 2020YFA0712300), in part by the National Natural Science Foundation of China under Grant No. 62072305, Grant 62132013, Grant 62132005, and Grant 62172162, in part by European Union's Horizon Europe Research and Innovation Programme under Grant 101073920 (TENSOR), Grant 101070052 (TANGO), Grant 101070627 (REWIRE), and Grant 101092912 (MLSysOps), and in part by the Shanghai Technology Innovation Centre of Distributed Privacy-Preserving Artificial Intelligence. *(Corresponding author: Zhen Liu.)*

Xingkai Wang is with the Department of Computer Science and Engineering, Shanghai Jiao Tong University, Shanghai 200240, China (e-mail: starshine87@sjtu.edu.cn).

Zhenfu Cao is with the Shanghai Key Laboratory of Trustworthy Computing, East China Normal University, Shanghai 200262, China (e-mail: zfcao@sei.ecnu.edu.cn).

Zhen Liu is with the Department of Computer Science and Engineering, Shanghai Jiao Tong University, Shanghai 200240, China, and also with the Shanghai Qizhi Institute, Shanghai 200232, China (e-mail: liuzhen@sjtu.edu.cn).

Kaitai Liang is with the Cybersecurity Group, Delft University of Technology, 2628 Delft, CD, Netherlands (e-mail: kaitai.liang@tudelft.nl).

Digital Object Identifier 10.1109/TDSC.2024.3449770

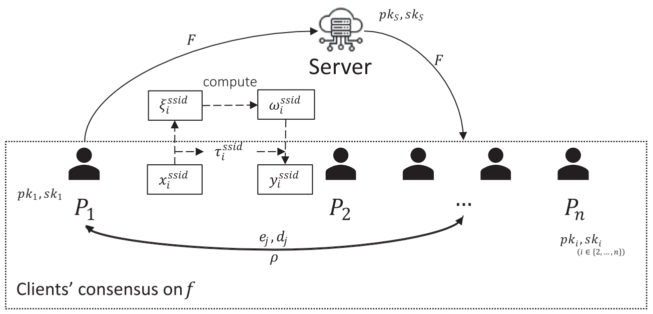

Fig. 1. Outsourcing.

compute a function f over a set of joint input $\{(x_1^{\mathsf{ssid}}, \dots, x_n^{\mathsf{ssid}})\}_{\mathsf{ssid}}$. In [2], Choi et al. introduced a new primitive *proxy oblivious transfer* (POT) constructed from a non-interactive key-exchange (NIKE) scheme, and serves the propose of keeping the clients' input confidential, from each other and from the server. However, the protocol cannot provide security guarantees in the presence of client-client corruption or malicious clients. Gordon et al. [3] systematically studied MVC in the universally composable (UC) model, which captures selective failure attack and adaptive soundness additionally, considering the participation of malicious clients, and consequently representing a stronger notion than the prior definitions. To achieve this "stronger" security, Gordon et al. proposed a protocol with a new primitive *attribute-hiding multi-sender attribute-based encryption* (ah-mABE), which can be constructed using a combination of a two-outcome ABE scheme with local encoding, an FHE scheme, and a POT protocol. Gordon et al. [3] also proposed a compiler that upgrades an MVC scheme secure against semi-honestly corrupted clients to one that can withstand maliciously corrupted clients, while maintaining non-interactivity. It is worth noting that Gordon et al. [3] pointed out the inherit impossibility of achieving security when there exists collusion between the server and an arbitrary client.

Recognized Limitations. In the constructions of [2] and [3], the clients remain using FHE for encrypting labels in proportion to their input size, where FHE is a bottleneck in efficiency, compared to other building blocks in their protocols. While outsourcability has been achieved in these FHE-based protocols, they still suffer from heavy client-side overheads. This is mainly because the messages requiring fully homomorphic encryption for the client scale proportionally with the size of function input. *Determining how to make this complexity independent of the input size remains an interesting long-lasting problem in the research line.*

Besides the FHE-related cost, a POT-based MVC scheme requires $O(n^2)$ instances of functionality during a single online phase of outsourced computation, where n signifies the number of clients. The overhead from POT rises substantially as the number of users increases, both in terms of communication and computation. Furthermore, the first client always carries the majority of the computation load, since it has to encrypt the input labels "twice" the actual length in a POT instance, leading to an imbalance in the overall overhead distribution. Thus, *a cost-effective and efficiency-balance protocol in multi-client context is worthy being considered, especially when n is sufficiently large.*

Additionally, although [3] achieves a secure MVC protocol against malicious corrupted clients, the implementation involves applying a compiler to a semi-honest secure construction. Gordon et al. only provide a proof of the compiler's existence, and its construction relies on the usage of zero-knowledge proofs, whose scale is related to the size of the delegated function and input-output length, which is never an efficient solution. *Designing a secure MVC protocol in the malicious setting while simultaneously preserving operational efficiency constitutes a matter of substantial practical significance.*

Contribution. We introduce MVOC, a novel multi-client verifiable computation scheme. In the conference version of this study [6], we primarily present an efficient solution in semi-honest scenario. In this full version, we further address this issue in the context of malicious model. The proposed scheme makes several key contributions, including reduced communication complexity and enhanced client efficiency, while ensuring robust security guarantees against both semi-honest and malicious adversaries. Our contributions can be summarized as follows:

– We revisit the definition of garbling scheme, and introduce enhancements, reducing the communication complexity from $O(n^2l)$ to $O(nl)$, where where n represents the number of clients and l is the input size. In the conference version [6], we presenting MOGC, addressing this challenge in the semi-honest model. Building upon MOGC, this paper introduces *Maliciously-Secure Multi-client Outsourced Garbled Circuit* (MS-MOGC), extending the solution to the malicious setting. This scheme utilizes a distributed approach for generating the encoding and decoding functions, allowing all clients to learn the garbled input wires and to require the final garbled result without the need for OT or POT. We further demonstrate that a secure MS-MOGC protocol implies a UC-secure one-time MVC protocol.

– We improve the outsourcability of MVC protocol, by reducing the FHE overhead from $O(l)$ to $O(\kappa)$, which is independent to the input size, where κ is the security parameter. Specifically, we incorporate the technique of hybrid homomorphic encryption into our MS-MOGC scheme, enable the construction of a UC-secure MVC protocol against malicious server or semi-honest client-client collusion.

– We implement the proposed MVOC using Yao's Garbled Circuit and the most efficient hybrid homomorphic encryption scheme, and conduct a comprehensive comparison on the clients' overhead in terms of both computation and communication. The experimental results clearly demonstrate the superior efficiency of our proposed scheme compared to other existing FHE-based works.

Technical Roadmap. In light of the proven impossibility of achieving input privacy in the presence of collusion between the server and any client [3], our discussion is confined to scenarios where collusion between the clients and server is not envisaged. Such assumptions are applicable, as discussed in [7], [8], [9], given that outsourcing server operators, driven by a long-term profitability prospective, typically prioritize their reputations and refrain from engaging in collusion with clients. We outline our roadmap consisting of the following two steps.

Multi-client Outsourced Garbled Circuit. In conventional garbling schemes, the generation of the garbled circuit is primarily delegated to a single party. This poses a challenge when the necessity arises to share the corresponding garbled labels with other parties within the protocol. For example, in Yao's secure two-party computation protocol, one party (referred to as Alice) is tasked with generating the garbled circuit, while the other party (referred to as Bob) has to obtain the relevant garbled labels through an OT process. The purpose of this OT is to facilitate the secure transfer of the necessary circuit randomness from Alice to Bob. A crucial point to emphasize is that Bob must acquire only the garbled labels pertinent to his input. This precaution is essential due to Bob's dual role as both the circuit executor and a protocol participant. Any access to extra garbled labels would provides him with an unfair advantage.

We observe that the separation of roles between the circuit executor and the data providers offers an effective solution to the previously mentioned concern. According to the above observation, we first introduce a novel primitive named *Multi-client Outsourced Garbled Circuit* (MOGC) in the conference version of this study, giving a secure construction derived from Yao's Garbled Circuit protocol. Essentially, we introduce a (not necessarily trusted) third party, the server, designated for computation execution, while individual clients contribute their randomness for generating the garbled circuit. This setup eliminates the need for OT, as each client can autonomously generate its specific garbled labels using its own randomness. Consequently, this effectively addresses the challenge of transferring randomness from circuit generator to the data provider. In the current version, we enhance the primitive to *Maliciously-Secure MOGC* (MS-MOGC). In comparison to to MOGC, we further tackle the issue of transferring the randomness of data receivers, and introduce support for verification to achieve security against malicious clients. Notably, we establish a connection between a secure MS-MOGC protocol and a one-time maliciously-secure MVC protocol, the latter of which lays the foundation for further research and advancements in MVC solutions.

Multi-client Verifiable Outsourced Computation. We employ FHE to ensure circuit privacy, a concept akin to constructing VC from the intuitively implied one-time VC from Yao's Garbled Circuit [1]. To mitigate the substantial overhead associated with FHE, we adopt the philosophy of hybrid encryption: initially employing symmetric key encryption (SKE) scheme to encipher the message, followed by using FHE to encapsulate the symmetric key. This KEM-DEM-like technique efficiently shifts the computational burden of FHE to the server, thereby alleviating the heavy FHE overhead on the client side. More concretely, after the generator generates the garbled circuit F for a function f by MS-MOGC in the offline phase, each client P_i derives its respective garbled input X_i^{ssid} using its own encoding function share, corresponding to its private input x_i^{ssid}. Each client subsequently performs FHE setup to obtain the FHE key pair $(\mathsf{pk}_{\mathsf{FHE}}, \mathsf{sk}_{\mathsf{FHE}})$. It then generates a hybrid ciphertext comprising of an SKE ciphertext of garbled input, along with n FHE ciphertexts, each encrypting the secret key with each

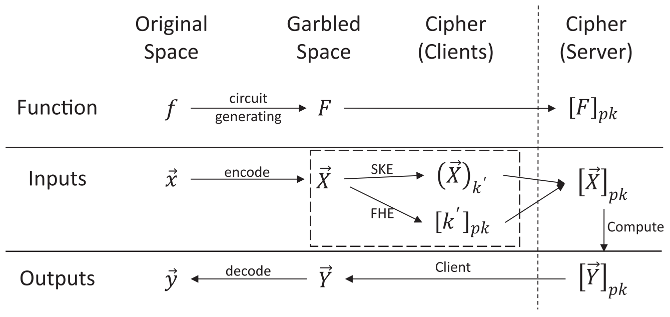

Fig. 2. Protocol Data Flow.

client's FHE public key. This hybrid ciphertext is transmitted to the server, who can calculate the FHE ciphertext of the garbled input wire using each client's hybrid ciphertext, on each client's FHE ciphertext space. The server then performs the computation on the garbled circuit as in MS-MOGC. Detailed data flow is shown in Fig. 2. The three rows in the figure represent the transformation process of data between function, inputs and outputs through different data space: data space, garbled space, client ciphertext space and server ciphertext space. The data to the left of the dashed line is held by the client, while the data to the right is managed and processed by the server. The process within the dashed box illustrates the online phase where each client encrypts the garbled inputs, demonstrating the efficiency improvements mentioned before. The correctness of FHE and the soundness of MS-MOGC ensures the result is verifiable. With the use of a secure channel for transmitting FHE ciphertext of the hybrid, the protocol also achieves privacy against the clients, where the UC-secure channel can be implemented by secure message transmitting [10], [11].

Security Against Malicious Clients. Expanding beyond the consideration of security against semi-honest corruption of clients, as discussed in the conference version, we further delve into security against malicious corruption of clients in this work. The extension is initially tackled by Gordon et al. [3], which presented a compiler capable of upgrading a protocol secure against semi-honest clients into one secure against malicious clients, under the condition of *perfect privacy*. However, such an approach has certain drawbacks. First, the compiler needs an additional trusted setup $\mathcal{F}_{\mathsf{CRS}}$, which introduces potential security assumptions. Second, it results in significant overhead due to the use of zero-knowledge proofs for clients, both during the offline and online phases.

We tackle these issues by having the clients reach a consensus on a valid garbled circuit. This approach is applicable since the circuit will never actually be executed by any client participant, aligning with our scenario. Specifically, during the initialization phase of MS-MOGC, the client responsible for generating the garbled circuit sends back the randomness used during the generation process. The server also forwards the circuit received from the generator for verification purposes. If any client rejects the circuit, the protocol is terminated. This method significantly enhances the efficiency of the protocol, rendering it more practical for real-world applications.

It is worth mentioning that a conference version of this paper was previously presented by Wang et al. [6]. Our earlier conference paper primarily focuses on security against semi-honest client corruptions. As previously discussed, to achieve malicious security, a SNARG-based compiler is required. In this paper, we confront this challenge by enhancing MOGC to MS-MOGC, achieving security against malicious client corruptions directly, eliminating the need for other cryptographic tools. Additionally, we make adjustments to the definitions of two ideal functionalities in the manuscript, adding corrupting behaviors for the adversary to align with malicious settings. Furthermore, we have conducted more comprehensive experiments to showcase the substantial efficiency improvement achieved by our work.

Related Work. Several studies have proposed different approaches to achieve verifiable computation since Gennaro et al.'s VC protocol [1]. One such approach involves the application of succinct functional encryption (FE) technique, initially introduced by Goldwasser et al. [12]. This technique has been further extended by Goldwasser et al. [13] to multi-input functional encryption (MIFE), which implies an efficient MVC protocol. However, neither ABE nor indistinguishable obfuscation (iO), especially the latter, is a cost-effective building block. In particular, iO introduces a strong assumption, which limits its practicality and makes the protocol less feasible in real-world scenarios.

Another method to achieve private verifiable computation is proving the correctness after computing, rather than proving with computing. Fiore et al. [14] proposed a protocol for verifiable delegation of computation on encrypted data by developing a novel homomorphic hashing technique that significantly reduces overhead. The core idea is to use homomorphic hashing to verify the correctness of computation on ciphertext, thereby addressing the challenge posed by FHE ciphertext expansion. Later, Fiore et al. [15] and Bois et al. [16] extended the protocol of [14]. The former supported public verifiability, and expanded the degree of the delegated function from two to any constant value through well-designed zk-SNARKs for polynomial rings, while the latter improved the efficiency of HE scheme by allowing flexible choices of the encryption parameters. Nonetheless, the incurred extra time cost on verification for client might be unsuitable for computation-restricted devices. However, the technique of homomorphic hashing is particularly efficient only for specific classes of functions, such as linear combinations, high-degree univariate polynomials, and multivariate quadratic polynomials, which limits its general applicability in other scenarios. Additionally, these schemes are generally designed for single-client scenarios and do not provide specific handling for multi-client environments.

Gennaro et al. [17] proposed a new primitive referred to as fully homomorphic message authenticator (HA), which enables the receiver to verify the computation result, constructed using FHE. However, this scheme suffers from slow verification efficiency, as it is directly proportional to the circuit size. Around the same time, Catalano et al. [18] presented a more efficient construction of HA using different building blocks, albeit at the expense of the maximum size of the delegated circuit, but it imposes an upper limit on the size of the supported functions, making it not a general-purpose solution. In the multi-client scenario, Fiore et al. [19] introduced various constructions of multi-key homomorphic authenticator. However, the time cost of verifying a result for the client is not lower than executing the computation itself. One idea to circumvent this overhead for the client is to outsource the verification function to the server. However, existing solutions either introduce extra communication complexity that breaks the non-interactive property, or rely on SNARG-like proof systems, which can be computationally expensive. Similar to previous works, these schemes also face challenges in efficiency and applicability, making them not general-purpose solutions.

Outline. In Section II, we show the syntax and security definitions of multi-client verifiable computation, including its ideal functionality. In Section III, we introduce the preliminaries on corresponding building blocks, comprising of garbling scheme and fully homomorphic encryption. In Section IV, we introduce a novel primitive MS-MOGC, and also present a construction which is secure against malicious adversaries. In Section V, we present our construction for MVC, and provide the proof for its UC-security defined in Section II. Later in Section VI, we show the efficiency of our construction both theoretically and in practice. The paper is concluded in Section VII.

II. Multi-Client Verifiable Computation

In this section, we first introduce the syntax of multi-client verifiable computation and then proceed to present its security definition in the UC framework by defining its ideal functionality.

A. Syntax

We first revise the notion of non-interactive multi-client verifiable computation (MVC) [3]. Let κ denote the security parameter. Suppose there are n clients $\mathsf{P}_1,\ldots,\mathsf{P}_n$ intending to delegate some computation tasks on an n-ary function $f : \mathcal{X}^n \to \mathcal{Y}^n$ to a remote server Serv for multiple times, and to require the validity of their answers. The length of input and output message space are polynomial in κ.

Briefly speaking, a MVC protocol can be divided into three phases, as shown in Fig. 1:

1) In setup phase, each participant is allowed to access an initial setup $\mathcal{G}$.
2) In offline phase, each client is allowed to send a fixed number of message to every other clients respectively, and is also required to send a fixed number of messages to the server Serv. Particularly, the aforementioned fixed number remains constant, regardless of the client number, the size of the delegated function and the data.
3) During online phase, there might be multiple subsessions in which clients are delegating some computations on the same function with different inputs. In a subsession, each client is allowed to send a single message to Serv, and to receive an output from Serv.

The detailed definition is given as follows.

Definition 1 (non-interactive Multi-client Verifiable Computation [3]). Let κ be the security parameter, n be the number

TABLE I
COMPARISON OF PRIVACY-PRESERVING VERIFIABLE COMPUTATION

Schemes	Gennaro et al. [1]	Choi et al. [2]	Gordon et al. [3]	MVOC (conf. ver.) [6]	MVOC (this ver.)
Model	Semi-honest	Semi-honest	Malicious	Malicious	Malicious
Collusion	✗	✗	✓	✓	✓
UC	✗	✗	✓	✓	✓
Multi-client	✗	✓	✓	✓	✓
Efficiency	✗	✗	✗	✗	✓
Comm.	-	$O(n^2l)$	$O(n^2l)$	$O(nl)$	$O(nl)$
Comp.	$O(dl\kappa)$	$O(dl\kappa)$	$O(dl\kappa)$	$O(d\kappa) + O(l)$	$O(d\kappa) + O(l)$

* We denote d as the expansion rate of complexity caused by FHE.
** The term *efficiency* represents whether a zero-knowledge compiler is used to build a maliciously secure protocol.
*** The terms *Comm.* and *Comp.* denote the communication and computation complexity, respectively, both exclusively pertaining to the constructions in the semi-honest setting.

of clients. A non-interactive multi-client verifiable computation comprises the following three phases:

Setup Phase: All participants have access to a setup $\mathcal{G}$, where each party P_i obtains $(\mathsf{pk}_i, \mathsf{sk}_i)$, and the server Serv obtains $(\mathsf{pk}_\mathsf{S}, \mathsf{sk}_\mathsf{S})$.

Offline Phase: After the delegated function f is chosen, each client P_i receives from each other client P_j the corresponding encoding mapping e_j and decoding mapping d_j. Then, one of the clients, namely P_1 for simplicity without loss of generality, generates the garbled version of f, noted as F, from the encoding and decoding mappings. P_1 then sends F to all the other participants, including the server Serv and all the other clients. All the other clients will receive the extra random value ρ used when generating the garbled circuit for verifying, and all clients have the right to abort the protocol at the end of this phase.

Online Phase: During a single subsession indexed by ssid, after input $(\mathsf{ssid}, x_i^{\mathsf{ssid}})$ provided by P_i is determined, the client computes $(\xi_i^{\mathsf{ssid}}, \tau_i^{\mathsf{ssid}})$. The first value will be sent to the server while the second one is kept private by P_i. After receiving information from all clients, the server Serv computes and sends the result $(\mathsf{ssid}, \omega_i^{\mathsf{ssid}})$ to each client P_i. Each client then decodes the encrypted result and obtains $y_i^{\mathsf{ssid}} \setminus \perp$, where $\perp$ indicates that the client is not convinced by the server's result, and will no longer continue executing the protocol unless restarting from Setup Phase.

Remark 1. Compared with [3], our definition is different in two aspects. In offline phase, we allow each client to send several message to others. This does not increase communication complexity amortizedly, since offline phase would be executed only once before multiple computation queries being carried out. After receiving a failure result from Serv, our clients will no longer trust the server and abandon the present protocol. Clients may re-select another trusted outsourcer or rollback to the setup phase, in order to obtain a new trusted environment. Besides, a client may also abandon the protocol because the dishonest of P_1, and this abort will occur in offline phase. With theses features, our definition captures the soundness against malicious client adversaries.

Remark 2. Compared with the conference version of this paper [6], our definition is different in offline phase. Only one client generates the garbled function, not all clients. After the generation, it broadcasts the garbled function to all the other participants including clients, rather than only to the server. This modification ensures the privacy considering the existence of a malicious garbler, by allowing the non-garbler client verifying the correctness of the garbled circuit.

B. Security Definition

We follow the UC framework in [10]. We formally define the ideal functionality for MVC in Table II, which captures the correctness and privacy, while notably addressing adaptive soundness and selective failure attack. The server and clients are either semi-honest or malicious in our model. Due to the proven impossibility of achieving input privacy in the presence of collusion between the server and any client [3], it is ensured that no server-client collusion will occur in any circumstances within our scenario. This assumption is applicable and is also discussed in the subsequent works [7], [8], [9] which are based on [3].

Remark 3. We note our security definition is different from [3] in the behavior of server S. Since server-client collusion is prohibited, we claim that there is no difference between the behavior of a corrupted and an uncorrupted server S. This may be seen as a special case of the definition in [3], which does not show contradiction. Because the indices set of corrupted clients is always empty, no information will gained from the blackbox oracle where the simulation could query on function f for different inputs provided by corrupted clients.

Definition 2 (Universal Composability [10]). A protocol Π UC-realizes ideal functionality $\mathcal{F}$ if for any PPT adversary $\mathcal{A}$ there exists a PPT simulator $\mathcal{S}$ such that, for any PPT environment $\mathcal{E}$, the ensembles $\mathsf{EXEC}_{\Pi,\mathcal{A},\mathcal{E}}$ and $\mathsf{EXEC}_{\mathsf{IDEAL}_{\mathcal{F}},\mathcal{S},\mathcal{E}}$ are indistinguishable, where the ensembles denotes the set of random variables describing the outputs of the execution on different inputs, whose formal definition is defined in [10].

Definition 3 (UC-security of MVC *)* A protocol MVC is UC-secure if MVC UC-realizes $\mathcal{F}_{\mathsf{MVC}}$, against malicious server and clients, without client-server collusion.

Adaptive soundness against selective failure. There are multiple subsessions in our definition, which enables the functionality to capture adaptive soundness. We allow clients to report the output to environment and thus, the definition captures security against selective failure attacks.

Static malicious corruption. We assume a static corruption model, with malicious corrupted participants, same as the model

TABLE II
IDEAL FUNCTIONALITY OF MULTI-CLIENT VERIFIABLE COMPUTATION

Multi-client Verifiable Computation

The functionality $\mathcal{F}_{\mathsf{MVC}}$ is parameterized with an n-ary function $f\colon \mathcal{X}^n \to \mathcal{Y}^n$ along with the security parameter κ, and interacts with n clients $\mathsf{P}_1, ..., \mathsf{P}_n$, a remote server Serv, and a simulator S.

• *Initialization:*

Upon receiving $(\mathtt{Init}, \mathsf{P}_i)$ from client P_i for all $i \in [n]\backslash 1$, send $(\mathtt{Init}, \mathsf{P}_i)$ to the simulator S. After S responses $(\mathtt{Init}, \mathsf{P}_i)$, send $(\mathtt{Init}, \mathsf{P}_i)$ to client P_1. Upon receiving $(\mathtt{Init}, \mathsf{P}_1)$ from the client P_1, send $(\mathtt{Init}, \mathsf{P}_1)$ to the simulator S. After S responses $(\mathtt{Init})$, send $(\mathtt{Init}, \Phi(f))$ to the server, and send $(\mathtt{Init}, \mathsf{P}_1)$ to all the clients except P_1. Upon receiving $(\mathtt{Init}, \mathsf{Serv}, \mathsf{P}_i)$ from the server for all $i \in [n]\backslash 1$, send $(\mathtt{Init}, \mathsf{Serv}, \mathsf{P}_i)$ to the simulator S. Upon receiving $(\mathtt{Init}, \mathsf{Serv}, \mathsf{P}_i, \phi)$ from S, if $\phi = \mathtt{OK}$ then send $(\mathtt{Init}, \mathsf{Serv})$ to the client P_i; otherwise send $(\mathtt{Init}, \mathtt{FAILED})$ to the client P_i, and the ideal functionality halts after all clients are responded.

• *Outsourcing:*

Upon receiving $(\mathtt{Input}, \mathsf{ssid}, x_i)$ from client P_i, send $(\mathtt{Input}, \mathsf{ssid}, \mathsf{P}_i)$ to notify the simulator S. After S returns $(\mathsf{ssid}, \mathsf{P}_i)$, store (ssid, x_i) and send a notification $(\mathtt{Input}, \mathsf{ssid}, \mathsf{P}_i)$ to the server Serv. Upon receiving $(\mathtt{Input}, \mathsf{ssid})$ from Serv, retrieve x_i for all $i \in [n]$. If some x_i has not been stored yet, send $(\mathtt{Output}, \mathsf{ssid}, \mathtt{FAILED})$ to the server and all clients, and the ideal functionality halts. Otherwise, compute $(y_1, ..., y_n) \leftarrow f(x_1, ..., x_n)$, and send $(\mathtt{Output}, \mathsf{ssid})$ to S. Upon receiving $(\mathsf{ssid}, \mathsf{P}_i, \phi)$ from the simulator, if $\phi = \mathtt{OK}$ then send $(\mathtt{Output}, \mathsf{ssid}, y_i)$ to the client P_i; otherwise send $(\mathtt{Output}, \mathsf{ssid}.\mathtt{FAILED})$ to the client P_i.

• *Corruption:*

The simulator S may input $(\mathtt{Corrupt}, \mathsf{Serv})$ or input several messages organized as $(\mathtt{Corrupt}, \mathsf{P}_i)$ where $i \in [n]$, to corrupt either the server or an arbitrary subset of clients. Any corrupted party may input $(\mathtt{ABORT})$ at any time. Upon receiving $(\mathtt{ABORT})$, send $(\mathtt{ABORT})$ to all the other participants, and halt at the end of the phase. During the outsourcing phase, any corrupted client P_i may input $(\mathtt{Cheat}, \mathsf{ssid}, \mathsf{P}_i)$ instead of $(\mathtt{Input}, \mathsf{ssid}, \mathsf{P}_i)$. If at least one client does so, the ideal functionality sends $(\mathtt{Output}, \mathsf{ssid}, \mathtt{Cheat})$ to honest clients as output when the response ϕ from S is $\mathtt{OK}$, and the ideal functionality halts after this phase.

Besides, if Serv is corrupted and does not abort: (1) During the initialization phase, it may input $(\mathtt{Cheat}, \mathsf{Serv}, \mathsf{P}_i)$ instead of $(\mathtt{Init}, \mathsf{Serv}, \mathsf{P}_i)$. In this case, for each specific P_i, the ideal functionality responds with $(\mathtt{Init}, \mathtt{Cheat})$ to the client P_i instead of $(\mathtt{Init}, \mathsf{P}_i)$, and responds with $(\mathtt{ABORT})$ to each of the other client, when the response ϕ from S is $\mathtt{OK}$. (2) During the outsourcing phase, it may input $(\mathtt{Cheat}, \mathsf{ssid}, \mathtt{Serv})$ instead of $(\mathsf{ssid}, \mathtt{Input})$. In this case, Serv will further input $(\mathtt{Cheat}, \mathsf{ssid}, \mathsf{Serv}, \mathcal{I}^*)$, where $\mathcal{I}^*$ is a subset of the index set $\mathcal{I}$. The ideal functionality responds with $(\mathtt{Output}, \mathsf{ssid}, \mathtt{Cheat})$ to the client P_i instead of $(\mathtt{Output}, \mathsf{ssid}, y_i)$, where $i \in \mathcal{I}^*$, and responds with $(\mathtt{ABORT})$ to the client P_j, where $j \in \mathcal{I}\backslash\mathcal{I}^*$. The ideal functionality halts after this phase.

If P_1 is corrupted and does not abort, during the initialization phase, it may input $(\mathtt{Init}, \mathtt{Cheat})$ instead of $(\mathtt{Init}, \mathsf{P}_1)$. In this case, the ideal functionality responds with $(\mathtt{Init}, \mathtt{Cheat})$ to the client P_i instead of $(\mathtt{Init}, \mathsf{Serv})$ when the response ϕ from S is $\mathtt{OK}$, and then halts after this phase.

in [3]. In such a model, the adversary can only corrupt parties at the beginning of protocol execution, instead of corrupting any party once the protocol has been executed. A malicious corrupted party may arbitrarily deviate the original protocol.

Communication model. We assume that all of the communication channels among participants are secure. We can implement such channels with the ideal functionality $\mathcal{F}_{\mathsf{SMT}}$ [10]. All of the protocols are described assuming in $\mathcal{F}_{\mathsf{SMT}}$-hybrid world.

Outsourceability. The *outsourceability* defines the improvements of client efficiency brought by outsourcing tasks to servers (i.e., how much computational and storage costs could be offloaded to servers). It is articulated to guarantee the overheads of clients in the online phase should be less that the costs incurred by a client-side self execution. We later will focus on discussing online efficiency for the clients in terms of time and communication costs. For the former, as FHE is an expensive component in MVC, we should require that the plaintext of FHE is independent of the input size in an outsourceable MVC protocol. As for the latter, we may require the communication size to be constraint to at most proportional to the input size, particularly in the presence of a substantial number of clients.

III. BUILDING BLOCKS

A. Garbling Scheme

The technique of garbled circuits was first proposed by Yao [4]. We follow the well-designed definition in [20] culled out by Bellare et al. The garbled circuit is generated by a single party named *garbler*.

Definition 4 (Garbling Scheme [20]). A garbling scheme for a family of functions $\mathcal{F}$ whose arbitrary element f is a mapping that can be efficiently computed, comprises five algorithms $\mathsf{GS} = \mathsf{GS}.\{\mathsf{Gb}, \mathsf{En}, \mathsf{De}, \mathsf{Ev}, \mathsf{ev}\}$. The first algorithm is probabilistic and the others are deterministic. Specifically,

- $(F, e, d) \leftarrow \mathsf{Gb}(1^\kappa, f)$. Taking as input the security parameter κ and a object function f, output the garbled circuit F, encoding function e, and decoding function d.
- $X = \mathsf{Ev}(e, x)$. Taking as input the encoding function e and input x, output garbled input X.
- $y = \mathsf{De}(d, Y)$. Taking as input the decoding function d and garbled output Y, obtain the final output y.
- $Y = \mathsf{Ev}(F, X)$. Taking as input a garbled circuit F and garbled input X, obtain the garbled output Y.
- $y = \mathsf{ev}(f, x)$. Taking as input the delegated function f and input x, obtain the plaintext output y.

We require a garbling scheme satisfying the following properties. The *correctness* ensures that the final output decoded from the result of garbled circuit is the exact function value, i.e. $f = e \circ F \circ d$. The *obliviousness* ensures that a party acquiring (F, X), but not d, should not learn anything about f, x, or y. The *authenticity* means that a party acquiring (F, X) should not be able to produce a valid garbled output $Y' \neq F(X)$ such that $De(d, Y') \neq \perp$. The formal definition of authenticity is shown as follows.

Definition 5 (Authenticity of Garbling Scheme [20]). For a garbling scheme $\mathsf{GS} = \mathsf{GS}.(\mathsf{Gb}, \mathsf{En}, \mathsf{De}, \mathsf{Ev}, \mathsf{ev})$, and for any PPT adversary $\mathcal{A}$, consider the following experiment:

$\mathbf{Exp}_{\mathcal{A}}^{\mathsf{Aut}}[\mathsf{GS}, \kappa]$:

$(F, e, d) \leftarrow \mathsf{Gb}(1^\kappa, f)$; $X \leftarrow \mathsf{En}(e, x)$; $Y' \leftarrow \mathcal{A}(F, X)$; $y' \leftarrow \mathsf{De}(Y')$;

If $y' \neq \perp$ and $Y' \neq \mathsf{Ev}(F, X)$, output 1, else 0.

The garbling scheme GS is authentic, if for any PPT adversary $\mathcal{A}$, there is a negligible function $\mathsf{negl}(\cdot)$ such that $\Pr[\mathbf{Exp}_{\mathcal{A}}^{\mathsf{Aut}}[\mathsf{GS}, \kappa]] \leq \mathsf{negl}(\kappa)$.

Side-information function. As pointed out in [20], it is unable to achieve absolute privacy for a garbling scheme. The information we expected to reveal is captured by a *side-information function* Φ, which is a deterministic mapping from a function f to a side-information set $\phi = \Phi(f)$.

Definition 6 (Obliviousness of Garbling Scheme [20]). For a garbling scheme $\mathsf{GS} = \mathsf{GS}.(\mathsf{Gb}, \mathsf{En}, \mathsf{De}, \mathsf{Ev}, \mathsf{ev})$, and for any PPT adversary $\mathcal{A}$, for any two functions f_0, f_1 such that $\Phi(f_0) = \Phi(f_1)$, and for any two valid input x_0, x_1, consider the following experiment:

$\mathbf{Exp}_{\mathcal{A}}^{\mathsf{Obv}}[\mathsf{GS}, \kappa]$:

$b \xleftarrow{\$} \{0, 1\}$; $(F, e, d) \leftarrow \mathsf{Gb}(1^\kappa, f)$; $X \leftarrow \mathsf{En}(e, x)$; $b' \leftarrow \mathcal{A}(F, X)$;

If $b' = b$, output 1, else 0.

The garbling scheme GS is oblivious, if for any PPT adversary $\mathcal{A}$, there is a negligible function $\mathsf{negl}(\cdot)$ such that $\Pr[\mathbf{Exp}_{\mathcal{A}}^{\mathsf{Obv}}[\mathsf{GS}, \kappa]] - \frac{1}{2} \leq \mathsf{negl}(\kappa)$.

Textbook Yao [4] and its optimizations. Yao's Garbled Circuit protocol is a two-party computation (2PC) protocol whose algorithms satisfy the definition of Garbling Scheme as mentioned above. There is a garbler and an evaluator in the protocol, and the garbler acquires the final result. Next, we will discuss existing optimizations to improve the efficiency of Yao's protocol. The typical techniques are free-XOR [21], which enables XOR gates to be computed without encryption, and half-gates [22], which reaches the lower bound of the GC construction per AND gate, while keeping compatible with free-XOR. These optimizations are secure under the assumption of circular correlation-robust hash function (circular crHF), and this function can be implemented from a fixed-key blockcipher [23].

B. Fully Homomorphic Encryption

The syntax and definition of the IND-CPA security of fully homomorphic encryption (FHE) is shown as follows.

Syntax. For a permitted circuit set $\mathcal{C}$, a fully homomorphic encryption scheme FHE comprises four PPT algorithms:

- $(pk, sk) \leftarrow \mathsf{Gen}(1^\kappa)$. The key generation algorithm outputs public-private key pair of FHE.
- $c \leftarrow \mathsf{Enc}(pk, m)$. The encryption algorithm takes message m as input and outputs ciphertext c.
- $m := \mathsf{Dec}(sk, c)$. The decryption algorithm takes ciphertext c as input and outputs plaintext message $m \in \mathcal{M}$.
- $c_{\mathsf{eval}} := \mathsf{Eval}(C, \{c_i\})$. The evaluation algorithm executes the circuit $C \in \mathcal{C}$ on ciphertext input collection $\{c_i\}$, and outputs ciphertext result c_{eval}.

Properties. The *correctness* of FHE requires that the key pair generated by Gen allows Dec to produce the same message m as the input ciphertext from Enc. The *homomorphic correctness* requires that the output of Eval decrypts to the result of applying C to plaintext inputs $\{m_i\}$. The *compactness* entails that the size of homomorphic ciphertext should be independent of the size, depth, or number of inputs to C, and less than $\mathsf{poly}(\kappa)$.

Definition 7 (IND-CPA Security of FHE [5]). For a fully homomorphic encryption scheme $\mathsf{FHE} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Eval})$, and for any PPT adversary $\mathcal{A}$, consider the following experiment:

$\mathbf{Exp}_{\mathcal{A}}^{\mathsf{CPA}}[\mathsf{FHE}, \kappa]$:

$(pk, sk) \leftarrow \mathsf{Gen}(1^\kappa)$; $(m_0, m_1, \tau) \leftarrow \mathcal{A}^{\mathsf{Enc}(pk,\cdot)}$;

$b \xleftarrow{\$} \{0, 1\}$; $c_b \leftarrow \mathsf{Enc}(pk, m_b)$; $\hat{b} \leftarrow \mathcal{A}(\tau, c_b)$;

If $\hat{b} = b$, output 1, else 0.

We define $\mathcal{A}$'s advantage in the experiment above as: $\mathbf{Adv}_{\mathcal{A}}^{\mathsf{CPA}}(\mathsf{FHE}, \kappa) = |2\Pr[\mathbf{Exp}_{\mathcal{A}}^{\mathsf{CPA}}[\mathsf{FHE}, \kappa] = 1] - 1|$.

The fully homomorphic encryption scheme FHE is CPA-secure, if for any PPT adversary $\mathcal{A}$, there is a negligible function $\mathsf{negl}(\cdot)$ such that: $\mathbf{Adv}_{\mathcal{A}}^{\mathsf{CPA}}(\mathsf{FHE}, \kappa) \leq \mathsf{negl}(\kappa)$.

C. Symmetric Key Encryption

The syntax and definition of the semantic security of symmetric key encryption (SKE) is shown as follows.

Syntax. A symmetric key encryption scheme SKE comprises three PPT algorithms:

- $k \leftarrow \mathsf{Gen}(1^\kappa)$. The key generation algorithm generates a valid symmetric key k.
- $c \leftarrow \mathsf{Enc}(k, m)$. The encryption algorithm takes a key k and a plaintext m, and outputs a ciphertext c.
- $m := \mathsf{Dec}(k, c)$. The decryption algorithm takes a key k and a ciphertext c, and outputs a plaintext $m \in \mathcal{M}$.

The semantic security of SKE requires that an adversary cannot derive any information about the plaintext from the ciphertext.

Definition 8 (Semantic Security of SKE [24]). For a symmetric key encryption $\mathsf{SKE} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$, and for any PPT adversary $\mathcal{M}$, consider the following experiment:

$\mathbf{Exp}_{\mathcal{A}}^{\mathsf{CPA}}[\mathsf{FHE}, \kappa]$:

$k \leftarrow \mathsf{Gen}(1^\kappa)$; $(m_0, m_1, \tau) \leftarrow \mathcal{A}^{\mathsf{Enc}(k,\cdot)}$;

$b \xleftarrow{\$} \{0, 1\}$; $c_b \leftarrow \mathsf{Enc}(k, m_b)$; $\hat{b} \leftarrow \mathcal{A}(\tau, c_b)$;

If $\hat{b} = b$, output 1, else 0.

We define $\mathcal{A}$'s advantage in the experiment above as: $\mathbf{Adv}_{\mathcal{A}}^{\mathsf{CPA}}(\mathsf{SKE}, \kappa) = |2\Pr[\mathbf{Exp}_{\mathcal{A}}^{\mathsf{CPA}}[\mathsf{SKE}, \kappa] = 1] - 1|$.

The symmetric key encryption scheme SKE is semantically secure, if for any PPT adversary $\mathcal{A}$, there is a negligible function $\mathsf{negl}(\cdot)$ such that: $\mathbf{Adv}_{\mathcal{A}}^{\mathsf{CPA}}(\mathsf{SKE}, \kappa) \leq \mathsf{negl}(\kappa)$.

IV. Maliciously Secure Multi-Client Outsourced Garbled Circuits

In this section, we will revisit the definition of Garbling Scheme [20], and introduce a novel primitive called *Maliciously Secure Multi-client Outsourced Garbled Circuits* (MS-MOGC). Subsequently, we will present a specific construction of MS-MOGC and prove its security even in the presence of *malicious* adversaries.

In the conventional Yao's garbled circuits protocol for two-party computation [4], there are two parties, Alice "the garbler" and Bob "the evaluator", aiming to jointly execute a computation of function $f(\cdot, \cdot)$ using their respective private inputs x_a and

x_b. Specifically, Alice generates a garbled circuit F based on the function f, and transmits it to Bob, along with her garbled version of the private input X_a. Then, through an OT protocol, Bob learns the garbled version of his private input X_b: (i) without revealing any information about b to Alice, and (ii) ensuring Bob not to obtain any information about other garbled inputs except X_b. Subsequently, Bob carries out the computation on F and obtains the garbled result Y. After receiving Y from Bob, Alice can extract the final result $y = f(x_a, x_b)$ from the garbled result. Note that Yao's GC protocol is a secure two-party computation against *semi-honest* adversaries. We intend to extend the previous definition to multi-client outsourced scenarios.

In an MS-MOGC protocol, the computation is carried out by a third party, not necessarily trusted, who does not provide any input of their own. Specifically, there is a *generator*, a *server* and at least one *collaborator*. The generator and the collaborator(s) jointly outsource the computation of a function $f : \mathcal{X}^n \rightarrow \mathcal{Y}^n$ on their input data collection vector $\vec{x}$. As a result, all the clients learn its corresponding output $f(\vec{x})[i]$, where i denotes the client index. Under the setting of outsourcing, it is required that the server does not to collude with any client, since security cannot be achieved in such circumstances.

In addition to security, we extra require the protocol to be efficient. Specifically, the protocol should be non-interactive, which indicates that each party could only send a single message to another party during one instance of the protocol. The confirmation message is an exception, as it carries no information regarding the delegated function, input or output. Consequently, OT technique cannot be employed straightforwardly, hence the randomness for the garbled circuits should be contributed by all data providers, rather than solely the circuit generator. We will first present the syntax of MS-MOGC, and then construct a secure MS-MOGC protocol based on Yao's garbled circuit. The security is proved in the presence of malicious adversaries.

A. Syntax of MS-MOGC

Definition 9 (Maliciously Secure Multi-client Outsourced Garbled Circuits). An MS-MOGC for a family of function $\mathcal{F}$ whose arbitrary element f is a mapping that can efficiently compute, comprises seven algorithms: $\mathsf{MOGC} = \mathsf{MOGC}.\{\mathsf{Ma}, \mathsf{Gb}, \mathsf{Ve}, \mathsf{En}, \mathsf{De}, \mathsf{Ev}, \mathsf{ev}\}$. The first two algorithms are probabilistic, while the remaining algorithms are deterministic.

- $(e_c, d_c) \leftarrow \mathsf{Ma}(1^\kappa, f, c)$. The client generates its corresponding part of encoding and decoding mappings, where c represents the client index.
- $(F, \rho) \leftarrow \mathsf{Gb}(1^\kappa, f, \vec{e}, \vec{d})$. The generator uses the encoding and decoding mappings to compute the garbled circuit, incorporating the randomness ρ that is employed during the circuit generation process.
- $\{0, 1\} \leftarrow \mathsf{Ve}(f, F, \rho)$. The collaborators verify the correctness of the garbled circuit by using all the randomness provided by the garbler, and then return the verification result.
- $X_c = \mathsf{En}(e_c, x_c)$. The client derives the garbled input from its private input using the corresponding encoding function.
- $Y = \mathsf{Ev}(F, \vec{X})$. The server performs the computation on garbled circuits and obtains the encoded output.
- $y \backslash \perp = \mathsf{De}(d, Y)$. The generator decodes the output obtained from the server using the decoding function, thus recovering the final output.
- $y = \mathsf{ev}(f, \vec{x})$. An auxiliary function is employed to execute the original function f.

Remark 4. Different from [20], our definition is specifically formulated for multi-party computation scenarios where $n \geq 3$. We have adapted the definition from [6], including supports to resist malicious adversaries, allowing participants to deviate from the protocol arbitrarily. A verification algorithm is introduced for collaborators to validate the legitimacy of the circuit provided by a potential malicious generator.

A secure MS-MOGC protocol should satisfy several properties. First, *correctness* ensures that the final output of the protocol matches the actual result of function evaluation. Second, *privacy* guarantees that each client's input remains confidential and is not disclosed to other client(s) or the server. This property protects the sensitive data of individual clients. Lastly, *authenticity* ensures that the server cannot provide an incorrect result that can be correctly decoded by client. This property prevents the server from manipulating the output to deceive the clients.

Before proceeding to the MS-MOGC construction, it is essential to analyze how Yao's garbled circuits can be utilized to achieve secure two-party computation (S2PC). In the context of S2PC, Alice and Bob each possess private inputs and wish to jointly compute a function on their inputs while preserving privacy. Alice acts as the circuit generator and creates a garbled circuit that obscures the function's logic and the input values, while Bob takes the role of the circuit evaluator and performs the actual computation using the garbled circuit. The privacy against Bob is guaranteed by the randomness introduced during circuit generation, which is provided by the generator Alice. Meanwhile, the privacy against Alice is protected through the use of an OT protocol, which delivers the necessary randomness from Alice to Bob.

An important observation is that the circuit executor should not simultaneously act as the circuit generator, since authenticity is protected by the randomness of encoded values, which is also the reason why using the same garbled circuit multiple times can compromise security. Fortunately, in MS-MOGC settings, the circuit execution is taken over by a third party "server", which is independent of the data providers. Hence there is no concern about such an authenticity crisis in the protocol.

Moreover, if the garbling scheme is *separable*, meaning that the randomness used in the encoding and decoding mappings can be considered as individually provided by each client, we may construct a protocol without adopting OT. More precisely, once the generator obtains the encoding and decoding mappings shares from each client, it can compute the garbled circuit using those encoding and decoding information. The definition of *separability* is provided as follows, and its proof will be presented shortly afterward.

Definition 10 (Separability of Garbling Scheme). We say that a garbling scheme is *separable* if its garbling algorithm Gb can be equivalently regarded as the following, where there are $|\mathcal{I}|$ input wires indexed by $[|\mathcal{I}|]$, and $|\mathcal{O}|$ output wires indexed by $[|\mathcal{O}|]$ with $\mathcal{I}$ being the set of all input wires and $\mathcal{O}$ being the set of all output wires, and two sub-algorithm ran and garb:

1) randomly select $|\mathcal{I}|$ randomness $r_{0,1},\ldots,r_{0,|\mathcal{I}|}$ and $|\mathcal{O}|$ randomness $r_{1,1},\ldots,r_{1,|\mathcal{O}|}$;
2) compute $e_i \leftarrow \mathsf{ran}(f, r_{0,i})$ for $i \in [|\mathcal{I}|]$, set $e = (e_1,\ldots,e_{|\mathcal{I}|})$;
3) compute $d_i \leftarrow \mathsf{ran}(f, r_{1,i})$ for $i \in [|\mathcal{O}|]$, set $d = (d_1,\ldots,d_{|\mathcal{O}|})$;
4) compute $F \leftarrow \mathsf{garb}(f, e, d)$;

Theorem 1. Assuming the existence of a correct, private and authentic garbling scheme with the property of separability, there exists a correct, private and authentic maliciously secure multi-client outsourced garbled circuit protocol, even in the presence of malicious adversaries.

Proof. We prove the theorem through a construction.

For Ma, given input f and the client index c (which determines the corresponding $|\mathcal{I}|$ and $|\mathcal{O}|$), each client executes Step 1, 2 and 3 on its corresponding input/output labels in the above definition. For Gb, given input f and all gathered encoding and decoding mapping shares $\vec{e}$ and $\vec{d}$, the generator executes Step 4in the above definition. In particular, ρ includes $\vec{e}$, $\vec{d}$, and all the randomness used during the generation of F. Taking textbook Yao as an example, this randomness refers to the garbled wire values of all intermediate parameters in the circuit.

For Ve, given the entire randomness ρ, a non-garbler client may check the correctness of garbled circuit F by verifying the validity of each gate in the circuit. If the check does not hold, the client aborts the protocol. The remaining algorithms are identical to the original garbling scheme, respectively.

The construction of MS-MOGC and a garbling scheme is merely different in the generation of encoding function e, decoding function d, and the verification procedure. The definitions of correctness, privacy (including obliviousness), and authenticity in semi-honest model are not affected by how e and d are generated. These properties can easily be inherited from those of a garbling scheme. Therefore, in the following, we will only discuss the security definitions in the presence of malicious adversaries.

For a malicious garbler, it may attempt to provide an invalid garbled circuit to obtain sensitive information from other clients. However, the verification algorithm Ve prevent this security crisis, and the probability of successful cheating is constrained by the security of the symmetric encryption scheme used in circuit generation. For a malicious server, it may attempt to provide a wrong garbled output as the result to deceive clients. However, an invalid garbled output that cannot be recognized by clients is negligible according to the definition of authenticity of the garbling scheme. □

B. Construction of MS-MOGC

We propose a protocol for MS-MOGC from conventional Yao's Garbled Circuits. In textbook Yao's circuits, each circuit gate is represented by a garbled gate. Let x_A and x_B be the input wires of a gate G, and x_C be the output wire. The generator Alice randomly chooses six values for each gate, denoted as w_t^b where $t \in \{A, B, C\}$ and $b \in \{0, 1\}$, representing the values 0 and 1 for the three wires, respectively. Each garbled gate consists of four ciphertexts $\gamma_G^{ij} = E_{w_A^i}(E_{w_B^j}(w_C)^{g(i,j)})$, where $i, j \in \{0, 1\}$, and E represents a well-designed symmetric encryption scheme.

It is evident that the textbook Yao's Garbled Circuits protocol is a separable garbling scheme. If we use $|\mathcal{I}|$ different input randomness, $r_{0,1},\ldots,r_{0,|\mathcal{I}|}$, along with $|\mathcal{O}|$ different output randomness, $r_{1,1},\ldots,r_{1,|\mathcal{O}|}$, to derive the encoded values, as opposed to generating $|\mathcal{I}|$ encoded input labels and $|\mathcal{O}|$ encoded output labels using the same randomness r, a PPT adversary is unable to distinguish between two sets of random values with merely different sources of randomness. Next, we partition the $|\mathcal{I}|$ input wires and $|\mathcal{O}|$ output wires into n buckets. When generating the encoded values for these input and output wires, we employ the same randomness within each bucket, while employing distinct randomnesses across different buckets. In this setup, a PPT adversary is still unable to distinguish the random values from the encoding values produced by Yao's protocol.

Based on the above observations, we give a secure MS-MOGC construction as follows. At first, each client C_i chooses his own randomness r_i. For each input or output wire that is relevant to the client C_i's input or output, we let C_i choose two random values of this wire using his own randomness r_i and then publish all the values to the generator, denoted as ρ_i. Then, after gathering all garbled input and output wires, the generator chooses random values of other intermediate wires, noted as $\bar{\rho}$, and computes the garbled circuit F using those values, and publishes F to all the other participants, while sending the entire randomness ρ to other clients, where $\rho = (\rho_1,\ldots,\rho_n,\bar{\rho})$. Each non-garbler client check the validity of F with ρ and choose to whether continue or abort the protocol. After acquiring F and all encoded inputs from clients, the server carries on the computation on F and obtains the encoded output, and handles each output label to its corresponding client it to the generator. The generator finally checks the output wire and recovers the final result. Since Yao's protocol is separable, our protocol is a secure MS-MOGC according to Theorem 1.

It is worth noting that our construction does not disrupt the adaptability of existing optimizations, as these optimizations do not introduce additional randomness. As for the reduced randomness, we can simply ignore it within the protocol to ensure compatibility.

V. Construction

We present our construction for MVC and give the proof for its UC-security in the malicious setting. We start the construction from designing a one-time MVC protocol from MS-MOGC. We define $\mathcal{F}_{\mathsf{OT\text{-}MVC}}$ as the ideal functionality of one-time MVC, and show that the proposed protocol UC-realizes the functionality. Then we construct an MVOC scheme that UC-realizes $\mathcal{F}_{\mathsf{MVC}}$. In the conference version of this paper, as discussed in [2], [3], we focus on a specific scenario in the subsequent construction: the case where only client P_1 can retrieve the output, without

TABLE III
IDEAL FUNCTIONALITY OF ONE-TIME MULTI-CLIENT VERIFIABLE COMPUTATION

One-Time Multi-client Verifiable Computation

The functionality $\mathcal{F}_{\mathsf{OT\text{-}MVC}}$ is parameterized with an n-ary function $f: \mathcal{X}^n \to \mathcal{Y}^n$ along with the security parameter κ, and interacts with n clients $\mathsf{P}_1, ..., \mathsf{P}_n$, a remote server Serv and a simulator S.

• *Initialization:*

Upon receiving $(\mathtt{Init}, \mathsf{P}_i)$ from client P_i for all $i \in [n] \backslash 1$, send $(\mathtt{Init}, \mathsf{P}_i)$ to the simulator S. After S responses $(\mathtt{Init}, \mathsf{P}_i)$, send $(\mathtt{Init}, \mathsf{P}_i)$ to client P_1. Upon receiving $(\mathtt{Init}, \mathsf{P}_1)$ from the client P_1, send $(\mathtt{Init}, \mathsf{P}_1)$ to the simulator S. After S responses $(\mathtt{Init})$, send $(\mathtt{Init}, \Phi(f))$ to the server, and send $(\mathtt{Init}, \mathsf{P}_1)$ to all the clients except P_1. Upon receiving $(\mathtt{Init}, \mathsf{Serv}, \mathsf{P}_i)$ from the server for all $i \in [n] \backslash 1$, send $(\mathtt{Init}, \mathsf{Serv}, \mathsf{P}_i)$ to the simulator S. Upon receiving $(\mathtt{Init}, \mathsf{Serv}, \mathsf{P}_i, \phi)$ from S, if $\phi = \mathtt{OK}$ then send $(\mathtt{Init}, \mathsf{Serv})$ to the client P_i; otherwise send $(\mathtt{Init}, \mathtt{FAILED})$ to the client P_i, and the ideal functionality halts after all clients are responded.

• *Outsourcing:*

Upon receiving $(\mathtt{Input}, x_i)$ from client P_i, send $(\mathtt{Input}, \mathsf{P}_i)$ to notify the simulator S. After S returns (P_i), store (x_i) and send a notification $(\mathtt{Input}, \mathsf{P}_i)$ to the server Serv. Upon receiving $(\mathtt{Input})$ from Serv, retrieve x_i for all $i \in [n]$. If some x_i has not been stored yet, send $(\mathtt{Output}, \mathtt{FAILED})$ to the server and all clients, and the ideal functionality halts. Otherwise, compute $(y_1, ..., y_n) \leftarrow f(x_1, ..., x_n)$, and send $(\mathtt{Output})$ to S. Upon receiving (P_i, ϕ) from the simulator, if $\phi = \mathtt{OK}$ then send $(\mathtt{Output}, y_i)$ to the client P_i; otherwise send $(\mathtt{Output}, \mathtt{FAILED})$ to the client P_i.

• *Corruption:*

The simulator S may input $(\mathtt{Corrupt}, \mathsf{Serv})$ or input several messages organized as $(\mathtt{Corrupt}, \mathsf{P}_i)$ where $i \in [n]$, to corrupt either the server or an arbitrary subset of clients. Any corrupted party may input $(\mathtt{ABORT})$ at any time. Upon receiving $(\mathtt{ABORT})$, send $(\mathtt{ABORT})$ to all the other participants. During the outsourcing phase, any corrupted client P_i may input $(\mathtt{Cheat}, \mathsf{P}_i)$ instead of $(\mathtt{Input}, \mathsf{P}_i)$. If at least one client does so, the ideal functionality sends $(\mathtt{Output}, \mathtt{Cheat})$ to honest clients as output when the response ϕ from S is $\mathtt{OK}$.

Besides, if Serv is corrupted and does not abort: (1) During the initialization phase, it may input $(\mathtt{Cheat}, \mathsf{Serv}, \mathsf{P}_i)$ instead of $(\mathtt{Init}, \mathsf{Serv}, \mathsf{P}_i)$. In this case, for each specific P_i, the ideal functionality responds with $(\mathtt{Init}, \mathtt{Cheat})$ to the client P_i instead of $(\mathtt{Init}, \mathsf{P}_i)$, and responds with $(\mathtt{ABORT})$ to each of the other client, both when the response ϕ from S is $\mathtt{OK}$. (2) During the outsourcing phase, it may input $(\mathtt{Cheat}, \mathtt{Serv})$ instead of $(\mathtt{Input})$. In this case, Serv will further input $(\mathtt{Cheat}, \mathsf{Serv}, \mathcal{I}^*)$, where $\mathcal{I}^*$ is a subset of the index set $\mathcal{I}$. The ideal functionality responds with $(\mathtt{Output}, \mathtt{Cheat})$ to the client P_i instead of $(\mathtt{Output}, y_i)$, where $i \in \mathcal{I}^*$, and responds with $(\mathtt{ABORT})$ to the client P_j, where $j \in \mathcal{I} \backslash \mathcal{I}^*$.

If P_1 is corrupted and does not abort, during the initialization phase, it may input $(\mathtt{Init}, \mathtt{Cheat})$ instead of $(\mathtt{Cheat}, \mathsf{P}_1)$. In this case, the ideal functionality responds with $(\mathtt{Init}, \mathtt{Cheat})$ to the client P_i instead of $(\mathtt{Init}, \mathsf{Serv})$ when the response ϕ from S is $\mathtt{OK}$, and then halts.

loss of generality. To cater to other clients, the protocol can be executed n times in parallel. Our distinctive approach, unlike prior schemes, permits each client to procure its individual output within a single execution of the online phase, which substantially minimizes overhead.

A. One-Time Multi-Client Verifiable Computation (OT-MVC)

There are n clients and a server participating in an OT-MVC protocol. All clients reach a consensus on the function $f : \mathcal{X}^n \to \mathcal{Y}^n$ to be computed. Each client P_i contributes its private input x_i. The primary objective is to facilitate each client P_i in acquiring $f(x_1, \ldots, x_n)[i]$, representing its component of the function result vector, while the server only knows the authorized side-information $\Phi(f)$. The server and all the clients are assumed to be malicious in this model, with the added condition that client-server collusion is not permitted. The ideal functionality $\mathcal{F}_{\mathsf{OT\text{-}MVC}}$ aligns with $\mathcal{F}_{\mathsf{MVC}}$, but with the distinction that the outsourcing phase occurs only once. The formal definition of the ideal functionality $\mathcal{F}_{\mathsf{OT\text{-}MVC}}$ is shown in Table III.

We give a construction of $\mathcal{F}_{\mathsf{OT\text{-}MVC}}$ from a secure MS-MOGC protocol. Let $f : (\{0,1\}^l)^n \to (\{0,1\}^l)^n$ be the outsourced function, P_1 be the generator, $\mathsf{P}_2, \ldots, \mathsf{P}_n$ be the collaborators, and Serv be the server. Assuming that all communication channels among the participants are secure and private, i.e. all communication is implemented by $\mathcal{F}_{\mathsf{SMT}}$. The parties work as follows:

– Each client P_i executes the algorithm Ma to generates its respective encoding and decoding mappings, denoted as e_i and d_i, on input of the delegated function f and its client index i. Then, all the collaborators send their mappings to the generator P_1.

– P_1 executes the algorithm Gb to generate the garbled circuit F and the aggregated circuit randomness ρ. Then P_1 sends F to the server and sends ρ to all the collaborators.

After receiving F, the server Serv forwards it to all the collaborators.

– Each collaborator P_i, where $i \in [n] \backslash \{1\}$, executes the algorithm Ve to verify the correctness of F. If the validity does not hold, the protocol is aborted.

– All clients execute the algorithm En on their respective private input $x_1, \ldots, x_n$ to obtain the garbled input $X_1, \ldots, X_n$, which are then sent to the server Serv.

– The server Serv executes the algorithm Ev on the garbled inputs, resulting in the garbled outputs $Y_1, \ldots, Y_n$, which are then transmitted back to their respective client.

– Each client P_i executes the algorithm De on the encoded output Y_i to recover the final result. If Y_i is a valid garbled output, the client accepts the final result as y_i. Otherwise, the client rejects it and outputs $\perp$ as the result.

Theorem 2. Suppose MOGC is a maliciously secure multi-client outsourced garbled circuit protocol, then the previously described protocol UC-realizes $\mathcal{F}_{\mathsf{OT\text{-}MVC}}$ against malicious corruption of the server, or any fixed subset of clients, in the $\mathcal{F}_{\mathsf{SMT}}$-hybrid model.

Proof. Let Π represent the above protocol. Our objective is to construct a simulator S, such that for any PPT adversary $\mathcal{A}$ capable of corrupting the server or a fixed subset of clients maliciously, for any PPT environment $\mathcal{E}$, the two ensembles $\mathsf{EXEC}_{\Pi,\mathcal{A},\mathcal{E}}$ and $\mathsf{EXEC}_{\mathcal{F}_{\mathsf{OT\text{-}MVC}},\mathcal{S},\mathcal{E}}$ are indistinguishable. For the simulator S: Upon receiving input from the environment $\mathcal{E}$, it writes this input on $\mathcal{A}$'s input tape. Upon obtaining an output value from the adversary $\mathcal{A}$, it writes this output on $\mathcal{E}$'s output tape.

Specifically, utilizing $\mathcal{F}_{\mathsf{SMT}}$ as the secure communication channel enables the simulator S to simulate the protocol with dummy messages when none of the clients or the server is corrupted. If any participant is corrupted, S can learn the actual message and obtain all the necessary information to accurately simulate the communication.

Case 1: Honest server and clients. Since we assume the channel is private, the simulator S can use the ciphertext of a random string to simulate the communication script. Because the server and all the clients are honest, the communication script is the only thing that the simulator S should simulate.

Case 2: Honest server and partially corrupted clients. The simulator S starts simulating internally the real world execution of the protocol, consisting of Serv, honest clients $\{\mathsf{P}_i\}_{i\in\mathcal{I}^*}$, and the adversary $\mathcal{A}$ that controls corrupted clients $\{\mathsf{P}_j\}_{j\in\mathcal{I}\setminus\mathcal{I}^*}$, where $\mathcal{I}$ is the honest client index set. For an honest client who causes halting of the Π or $\mathcal{F}_{\mathsf{OT\text{-}MVC}}$, we have the following:

(1) During the initialization phase, the probability that $\mathcal{F}_{\mathsf{OT\text{-}MVC}}$ halts, which is the probability of corrupted P_1 responding with $(\mathtt{Init}, \mathtt{Cheat})$, is the same as the probability of P_1 providing an invalid garbled circuit that cannot be verified, which is the probability of abortion by the honest client in the real world.

(2) During the outsourcing phase, the probability that $\mathcal{F}_{\mathsf{OT\text{-}MVC}}$ halts with $\mathtt{ABORT}$ is 0, since the server is honest. The probability that honest clients receives $(\mathtt{Output}, \mathtt{Cheat})$, which is the probability of any corrupted client P_j inputting $(\mathtt{Cheat}, \mathsf{P}_j)$, is the same as the probability of the corrupted client providing an invalid garbled input that cannot be verified, which is the probability of the result $\perp$ being received by an honest client in the real world.

Hence we just need to construct a simulator that can simulate the transcript and the view of all corrupted clients. The former can be easily simulated by using the ciphertext of a random string, since the communication channel is private. The latter contains the garbled circuit F and its randomness ρ, the encoded input X_i and output Y_i for each client P_i.

For generating the garbled circuit F, the simulator first randomly chooses two strings as the encoded wires for each intermediate wires in the circuit, and then uses the strings along with all encoding and decoding mappings to compute the garbled circuit $\bar{F}$. After that, it denote $\bar{\rho}$ as the combination of the intermediate randomness and all encoding (decoding) mappings. Then, the simulator S sets the encoded input (output) to the exact string corresponding to the input (output). Apparently, it is impossible to tell the difference between ρ and $\bar{\rho}$. And $\bar{F}$ is also indistinguishable from F because of the property of the garbling scheme used in MS-MOGC. It is also impossible to tell the difference between the encoded input (output) wires in the encoding (decoding) mappings and the newly generated random strings.

Case 3: Corrupted server and honest clients. The simulator S starts simulating internally the real world execution of the protocol, consisting of honest clients $\{\mathsf{P}_i\}_{i\in\mathcal{I}}$, and the adversary $\mathcal{A}$ that controls corrupted server Serv. For an honest client who causes halting of the Π or $\mathcal{F}_{\mathsf{OT\text{-}MVC}}$, we have the following:

1) During the initialization phase, the probability that $\mathcal{F}_{\mathsf{OT\text{-}MVC}}$ halts, which is the probability of corrupted Serv responding with $(\mathtt{Cheat}, \mathsf{Serv}, \mathsf{P}_i)$ for any $i \in \mathcal{I}$, is the same as the probability of Serv forwarding a wrong garbled circuit to some client, which is the probability of abortion by the honest client in the real world.

2) During the outsourcing phase, the probability that $\mathcal{F}_{\mathsf{OT\text{-}MVC}}$ halts with $\mathtt{ABORT}$, which is the probability of corrupted Serv inputting $(\mathtt{Cheat}, \mathsf{Serv}, \mathcal{I}^*)$, is the same as the probability of the corrupted server providing an invalid garbled output that cannot be verified, which is the probability of the result $\perp$ being received by some honest client in the real world, because of the *authenticity* of MOGC.

Specifically, besides the communication script, the simulator S needs to simulate the server Serv's view, including $(F, \{X\}_n)$. The simulator S randomly chooses the encoded input wires which are not chosen by clients, denoted as $\{\bar{X}\}_n$. Then S merges the two sets into the universal encoding function. Concretely speaking, encoded wires in $\{X\}_n$ and $\{\bar{X}\}_n$ are regarded as the encoded wires of 0's and 1's, respectively. After randomly choosing the encoded intermediate and output values, the simulator generates the garbled circuit F', and sets the decoding function to d'. If there exists a distinguisher that could distinguish $(F, \{X\}_n)$ from $(F', \{\bar{X}\}_n)$, then we can construct an adversary $\mathcal{B}$ that uses $(f, f', \{x\}_n, \{\vec{0}\}_n)$ as input to break the obliviousness of MS-MOGC.

In conclusion, the two ensembles $\mathsf{EXEC}_{\Pi,\mathcal{A},\mathcal{E}}$ and $\mathsf{EXEC}_{\mathcal{F}_{\mathsf{OT\text{-}MVC}},\mathcal{S},\mathcal{E}}$ are indistinguishable in all cases. □

B. Construction of MVOC

We give a construction that UC-realizes $\mathcal{F}_{\mathsf{MVC}}$ from a secure MS-MOGC scheme, a fully homomorphic encryption scheme FHE and a symmetric-key encryption scheme SKE. The protocol is in the $(\mathcal{F}_{\mathsf{SMT}}, \mathcal{G}_{\mathsf{FHE}})$-hybrid world, Specifically, $\mathcal{G}^{\mathsf{FHE}}$ serves as a self-registered PKI which allows any client to generate FHE key pair and register the public key, and it returns $\mathsf{pk}_{\mathsf{FHE}}$ when the server or any other party queries it. $\mathcal{F}_{\mathsf{SMT}}$ is the functionality of Secure Message Transmission [10], providing private channels between clients and server. The construction is described as follows. For simplicity, we omit the superscript ssid of the variables in online phase.

1) In the initialization phase, each client P_i executes the algorithm $\mathsf{MOGC.Ma}$ to generate the encoding and decoding mapping shares pair (e_i, d_i), for $i \in [n]$. Then, all non-garbler clients send their pairs to P_1.
2) After gathering all shares of encoding and decoding mappings, the generator P_1 executes the algorithm $\mathsf{MOGC.Gb}$ to obtain the garbled circuit F, along with the randomness ρ used during circuit generation. Then, P_1 sends F to the server and sends ρ to all the other clients.
3) After receiving F, the server Serv forwards it to all the collaborators.
4) All non-garbler clients execute the algorithm $\mathsf{MOGC.Ve}$ on the circuit and corresponding randomness. If the validation does not holds, the protocol is aborted.

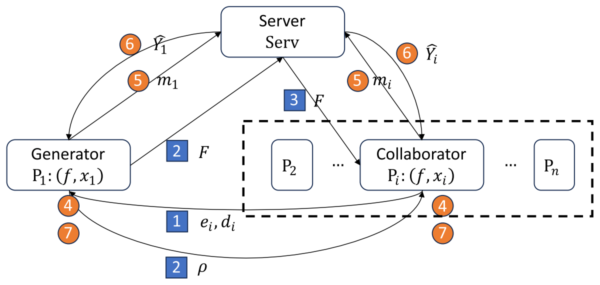

Fig. 3. Construction of MVOC.

5) In the outsourcing phase, each client P_i communicates with $\mathcal{G}_{\mathsf{FHE}}$ to obtain a current FHE key pair (pk_i, sk_i), for $i \in [n]$. After all the clients have registered the FHE public key, each client acquires all keys of other clients by calling $\mathcal{G}_{\mathsf{FHE}}$. Then, for $i \in [n]$, each client P_i computes its garbled input X_i by executing the algorithm $\mathsf{MOGC.En}$ on its own private input x_i, randomly chooses a symmetric key $k_i \leftarrow \mathsf{SKE.Gen}(1^\kappa)$, and computes $m_{i0} = \mathsf{SKE.Enc}(k_i, X_i)$ and $m_{ij} = \mathsf{FHE.Enc}(pk_j, k_i)$ for $j \in [n]$. P_i finally sends $m_i = (m_{i0}, \ldots, m_{in})$ to the server Serv.
6) After parsing $m_i = (m_{i0}, \ldots, m_{in})$, the server Serv computes $\hat{X}_{ij} = \mathsf{SKE.Dec}(m_{i0}, \mathsf{FHE.Enc}(m_{ij}))$, for $i, j \in [n]$. Then, Serv computes FHE ciphertexts of the garbled circuit, by computing $\hat{F}_j = \mathsf{FHE.Enc}(pk_j, F)$, for $j \in [n]$. Next, it executes the algorithm $\mathsf{MOGC.Ev}$ on $\{\hat{X}_{ij}\}_{i\in[n]}$ and obtains circuit result set $\{\hat{Y}_{ij}\}_{i\in[n]}$, and sends the output $\hat{Y}_{jj}$ to the corresponding output object P_j, for $j \in [n]$.
7) After receiving the encrypted result, each client P_i computes $Y_i = \mathsf{FHE.Dec}(sk_i, \hat{Y}_{ii})$, and recovers the final result by executing the algorithm $\mathsf{MOGC.De}$. If Y is a valid garbled output, then P_i accepts and outputs the final result y. Otherwise, P_i rejects it and output $\perp$ as the result, and the protocol is aborted.

As shown in Fig. 3, Step 1, 2 and 3 are in offline phase, and the rest are online. Step 4 and 7 are locally executed by each client and the generator respectively with no data transmission between participants. After Step 7 is executed without the result being $\perp$, the protocol comes back to Step 4; otherwise, it terminates. This abortion makes the advantage that the server gained from providing incorrect result cannot be carried over to the next subsession.

Theorem 3. Suppose FHE is an IND-CPA secure public-key fully homomorphic encryption scheme, SKE is a semantically secure symmetric-key encryption scheme, and MOGC is a maliciously secure multi-client outsourced garbled circuit protocol, then the aforementioned protocol UC-realizes $\mathcal{F}_{\mathsf{MVC}}$ against malicious corruption of any fixed subset of clients, or against malicious server corruption, in the $(\mathcal{F}_{\mathsf{SMT}}, \mathcal{G}^{\mathsf{FHE}})$-hybrid model.

Proof. Let Π represents the above protocol. Our objective is to construct a simulator S, such that for any PPT adversary $\mathcal{A}$ capable of corrupting the server or a fixed subset of clients maliciously, for any PPT environment $\mathcal{E}$, the two ensembles $\mathsf{EXEC}_{\Pi,\mathcal{A},\mathcal{E}}$ and $\mathsf{EXEC}_{\mathcal{F}_{\mathsf{MVC}},\mathcal{S},\mathcal{E}}$ are indistinguishable. For the simulator S: Upon receiving input from the environment $\mathcal{E}$, it writes this input on $\mathcal{A}$'s input tape. Upon obtaining an output value from the adversary $\mathcal{A}$, it writes this output on $\mathcal{E}$'s output tape. In this proof, we omit the corruption analysis on abortion, which is the same as the proof of Theorem 2.

Similar to the proof of Theorem 2, using $\mathcal{F}_{\mathsf{SMT}}$ as the secure communication channel allows the simulator S to simulate the protocol with dummy messages when no clients or the server are corrupted. If any participant is corrupted, S can learn the actual message and obtain all the necessary information to accurately simulate the communication.

Case 1: Honest server and client. Since we assume the channel is private, the simulator S could use the ciphertext of a random string to simulate the communication script. Because the server and all the clients are honest, the communication script is the only thing that the simulator S should simulate.

Case 2: Honest server and partially corrupted clients. Similar to the proof of Theorem 2, we just need to construct a simulator that can simulate the transcript and the view of all corrupted clients. The former can be easily simulated by using the ciphertext of a random string, since the communication channel is private. The latter contains a series of FHE public keys, circuit information, and the message sent to and received from the server for each corrupted client. Concretely speaking, the view of a client P_i in the real world contains FHE public keys $(pk_1, \ldots, pk_n)$, its own FHE secret key sk_i, the garbled circuit F along with its randomness ρ, the message sent to the server $(m_{i0}, \ldots, m_{in})$ and the message received from the server $\hat{Y}_{ii}$.

Apparently, FHE keys can be easily simulated by interacting with the self-registered PKI $\mathcal{G}_{\mathsf{FHE}}$. For the remaining part, the simulator S could just randomly choose two strings for each wire in the circuit as the encoded or decoded mapping shares, noted as ρ'. Specifically, we denote the encoding and decoding mapping of client P_i as e_i and d_i respectively. Then, S uses it to compute a garbled version of function f, noted as F'. Since the circuit labels are chosen randomly, the adversary cannot distinguish the difference of circuits with randomness between the real and ideal worlds. Next, the simulator S randomly choose a symmetric key $k'_i \leftarrow \mathsf{SKE.Gen}$, and uses the key to encrypt its garbled input $X'_i = e'_i(x_i)$ from input x_i, and obtains the ciphertext $\mathsf{SKE.Enc}(k'_i, X'_i)$, denoted as m'_{i0}. After that, S uses the public keys of all clients that are provided by $\mathcal{G}^{\mathsf{FHE}}$ to encrypt the symmetric key k'_i and obtains n encapsulated keys $\mathsf{FHE.Enc}(pk'_i, k'_i)$, denoted as $m'_{i1}, \ldots, m'_{in}$ respectively. The IND-CPA security of FHE and the semantic security of SKE ensures that a PPT adversary cannot distinguish the views between the ideal and real worlds.

Case 3: Corrupted malicious server and honest clients. Similar to the proof of Theorem 2, we just need to construct a simulator that can simulate the transcript and the view of S. The former can be easily simulated by using the ciphertext of a random string, since the communication channel is private. The latter contains a series of FHE public keys, the garbled circuit, and all the message sent to and received from the server. Concretely speaking, the view of S in the real world contains FHE public keys $(pk_1, \ldots, pk_n)$, the garbled circuit F, the message sent to the server $(m_{i0},\ldots,m_{in})$ and the message received from the server $\hat{Y}_{ii}$ for each client P_i, for $i \in [n]$.

Upon receiving $\Phi(f)$, the simulator randomly generates a circuit F' with the circuit structure information revealed by $\Phi(f)$. During online phase, the simulator interacts with $\mathcal{G}_{\mathsf{FHE}}$ to obtain all the FHE public key $(pk_1,\ldots,pk_n)$. Then, for each $i \in [n]$, the simulator chooses two random strings s_1 and s_2 of length k and w respectively, where k is the length of a valid SKE key and w is the length of a valid garbled wire. S computes $m'_{i0} := \mathsf{SKE.Enc}(s_1, s_2)$ and $m'_{ij} := \mathsf{FHE.Enc}(pk'_j, s_1)$, for $j \in [n]$. If there exists a distinguisher that could tell the views of the ideal and real worlds, it either distinguishes F from F', or distinguishes $(m_{i1},\ldots,m_{in})$ from $(m'_{i0},\ldots,m'_{in})$. If the former happens, then we can construct an adversary $\mathcal{B}$ using $(f, f', \{x\}_n, \{\vec{0}\}_n)$ as input to break the obliviousness of MS-MOGC. If the latter happens, there exists a $j \in [n]$ such that the adversary can distinguish the difference between m_{ij} and m'_{ij}. Hence, we can construct an adversary $\mathcal{C}$ that uses (m_{ij}, m'_{ij}) as input to break either the semantic security of SKE or the IND-CPA security of FHE, respectively.

Thus $\mathsf{EXEC}_{\Pi,\mathcal{A},\mathcal{E}}$ and $\mathsf{EXEC}_{\mathcal{F}_{\mathsf{MVC}},\mathcal{S},\mathcal{E}}$ are indistinguishable in all cases. □

VI. Evaluation

A. Efficiency Analysis

As discussed earlier, *outsourcability* is a concept synonymous with efficiency improvement for the client side. To assess the efficiency of our approach, we conduct a comparative analysis with Choi et al.'s work [2]. Their solution in multi-client settings is theoretically more efficient than the general UC-secure solution presented by Gordon et al. [3]. We break down the incurred costs in both offline and online phases. In contrast to the conference version of our work, we no longer need to execute the protocol n times within a single computing period. This is due to the fact that the final result is now distributed to each respective client by the server. Specifically,

• In offline phase, each client generates the randomness of its corresponding inputs and outputs labels. Besides, the generator client generates the garbled circuit using the randomnesses, while each of the non-generator client verifies the garbled circuit. In all, the total computational complexity of each client is $O(\Phi(f) + nl)$. For communication, each non-garbler client sends a message of size $O(l)$ to the generator, and receives messages of total size $O(\Phi(f) + nl)$ from the generator and the server. The generator client receives messages of total size $O(nl)$ from the collaborators, and sends a message of size $O(\Phi(f) + nl)$ to the server and each other client. The total verification cost for each collaborator is also $O(\Phi(f) + nl)$.

• In online phase, the generator runs an FHE key generation algorithm. Then each client executes a symmetric key encryption with size $O(l\kappa)$, and executes a fully homomorphic encryption with size $O(\kappa)$. During result recovering, each client executes a fully homomorphic decrypting algorithm with plaintext size $O(l\kappa)$. For communication, each client sends the ciphertext generated above.

TABLE IV
Overheads for Clients

Schemes	Offline phase		Input phase	
	Time(ms)	Comm.(KB)	Time(ms)	Comm.(KB)
Choi et al. [2]	1.158	300	609.071*	144
Wang et al. [6]	1.158	304	47.635	20.45
MVOC	1.158	404	27.079	18.45

* This time cost is underestimated, since it does not include the time consumed by POT, which is not a component in our implement.

B. Implementation and Evaluation

We conducted our protocol implementation and experiments on an Ubuntu 22.04 Server featuring an Intel i5-12600KF CPU (3.7 GHz) and 64 GB DDR5 4400 MHz RAM. Our focus was primarily on simulating the client-side execution. To implement the garbling scheme, we chose TinyGarble [25], known for its maturity and integration of recent optimizations in the Garbled Circuit (GC) protocol. We used the AES-128 encryption algorithm as the computation benchmark. For simplicity, we configured the number of clients to be 2, where one provided a 128-bit key, and the other used a 128-bit plaintext. The original circuit consisted of 38,031 gates, with 6,400 non-XOR gates. The total size of the garbled circuit amounted to 300 KB. There were 38,287 wires in the circuit, including 256 input wires and 128 output wires. It is worth noting that the experiments conducted in the conference version and this version were performed in different environments. Specifically, the CPUs used in each experiment had different core counts. Our code is optimized for multi-core processors, which contributed to the variants in the data. In terms of communication complexity, we recalculated the amortized communication overhead to ensure more accurate efficiency testing in the context of this paper. For the Fully Homomorphic Encryption (FHE) scheme, we employed TFHE [26], as it supports boolean value encryption and can be optimized for fast gate bootstrapping. For symmetric encryption scheme, we used DASTA [27], a 6-round scheme, chosen for its efficient encryption capability and compatibility with TFHE. We set security parameter κ to 128, the key-size of DASTA was chosen as 351. All the experiments were conducted in single-thread mode. Our implementation was compared against [2] and [6], and the results are presented in Table IV. As mentioned earlier, the scalability of our protocol is directly related to the number of clients and the input size. In cases where only two clients were involved, each with a minimal input size of 128, which is roughly equivalent to the ciphertext extension rate of FHE, our protocol showed the least advantage. Under these specific conditions, we achieved a communication efficiency improvement of at least 7.80 times and a time cost reduction of at least 22.49 times compared to [2]. However, as the data size increased, the efficiency improvement ratio gradually converged to a rate that was proportional to the throughput of the two schemes. In our evaluation, the SKE throughput was 1594.05 b/ms, while TFHE provided a rate of 107.6 b/ms. Consequently, with larger input sizes, we achieved efficiency improvements of 24 times and 29.63 times, respectively.

It is important to note that the evaluation of efficiency improvement is based on the worst-case scenario, where the number of non-server participants is 2. As indicated in Table I, greater values of n result in higher efficiency improvements.

VII. Conclusion and Future Work

We proposed a lighter multi-client privacy-preserving verifiable outsourced computation scheme, designed to operate even in the presence of malicious participants. To adapt garbled circuits for use in a malicious multi-client environment, we developed a novel primitive called MS-MOGC, which builds on the garbling scheme and improves upon the previous work on MOGC. In our approach, each participant generates its own randomness, allowing every client to independently verify the circuit. Furthermore, we established that a secure MS-MOGC protocol implies a one-time multi-client verifiable computation. To construct an efficient MVOC protocol, we leveraged the hybrid encryption technique to mitigate the costly overhead associated with Fully Homomorphic Encryption (FHE). Our protocol's security in Universal Composability (UC) Framework was rigorously proven. We conducted a comprehensive theoretical analysis of its efficiency and also implemented our scheme. The results clearly demonstrate the effectiveness of our protocol in enhancing the efficiency of the input phase, with improvements of 7.80 times in communication efficiency and 22.49 times in computation cost, even in the worst case.

In the context of improving the efficiency of outsourced computation, this study opens up several avenues for future research. First, within the same technical framework as this paper, an under-explored area is the overhead associated with the fully homomorphic encryption (FHE) decrypting process. Reducing the computational cost of decryption presents a promising research direction that could significantly enhance the overall efficiency of outsourced computation. Second, while there are various technical approaches to addressing the problem of outsourced computation. However, comparing the efficiency of these different approaches is relatively complex. Future research could focus on defining a standardized benchmark to systematically compare the implementations of different technical routes. This would help in identifying the most suitable approach for various application scenarios.

References

[1] R. Gennaro, C. Gentry, and B. Parno, "Non-interactive verifiable computing: Outsourcing computation to untrusted workers," in *Proc. 30th Annu. Cryptol. Conf.*, 2010, pp. 465–482.

[2] S. G. Choi, J. Katz, R. Kumaresan, and C. Cid, "Multi-client non-interactive verifiable computation," in *Proc. 10th Theory Cryptogr. Conf.*, Springer, 2013, pp. 499–518.

[3] S. D. Gordon, J. Katz, F.-H. Liu, E. Shi, and H.-S. Zhou, "Multi-client verifiable computation with stronger security guarantees," in *Proc. Theory Cryptogr. Conf.*, Springer, 2015, pp. 144–168.

[4] A. C.-C. Yao, "How to generate and exchange secrets," in *Proc. IEEE 27th Annu. Symp. Found. Comput. Sci.*, 1986, pp. 162–167.

[5] C. Gentry, "Fully homomorphic encryption using ideal lattices," in *Proc. 41st Annu. ACM Symp. Theory Comput.*, 2009, pp. 169–178.

[6] X. Wang, Z. Cao, Z. Liu, and K. Liang, "Lighter is better: A lighter multi-client verifiable outsourced computation with hybrid homomorphic encryption," in *Proc. 27th Eur. Symp. Res. Comput. Secur.*, Copenhagen, Denmark, Springer, 2022, pp. 105–125.

[7] K. Peng, X. Shen, L. Gao, B. Wang, and Y. Lu, "Communication-efficient and privacy-preserving verifiable aggregation for federated learning," *Entropy*, vol. 25, no. 8, 2023, Art. no. 1125.

[8] C. Hahn, H. Kim, M. Kim, and J. Hur, "VerSA: Verifiable secure aggregation for cross-device federated learning," *IEEE Trans. Dependable Secure Comput.*, vol. 20, no. 1, pp. 36–52, Jan./Feb. 2023.

[9] E. Zhang, F.-H. Liu, Q. Lai, G. Jin, and Y. Li, "Efficient multi-party private set intersection against malicious adversaries," in *Proc. ACM SIGSAC Conf. Cloud Comput. Secur. Workshop*, 2019, pp. 93–104.

[10] R. Canetti, "Universally composable security: A new paradigm for cryptographic protocols," in *Proc. IEEE Annu. Symp. Found. Comput. Sci.*, 2001, pp. 136–145.

[11] R. Canetti, P. Jain, M. Swanberg, and M. Varia, "Universally composable end-to-end secure messaging," in *Proc. Annu. Int. Cryptol. Conf.*, Springer, 2022, pp. 3–33.

[12] S. Goldwasser, Y. Kalai, R. A. Popa, V. Vaikuntanathan, and N. Zeldovich, "Reusable garbled circuits and succinct functional encryption," in *Proc. 41st Annu. ACM Symp. Theory Comput.*, 2013, pp. 555–564.

[13] S. Goldwasser et al., "Multi-input functional encryption," in *Proc. 33rd Annu. Int. Conf. Theory Appl. Cryptogr. Techn.*, Springer, 2014, pp. 578–602.

[14] D. Fiore, R. Gennaro, and V. Pastro, "Efficiently verifiable computation on encrypted data," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2014, pp. 844–855.

[15] D. Fiore, A. Nitulescu, and D. Pointcheval, "Boosting verifiable computation on encrypted data," in *Proc. 23rd IACR Int. Conf. Pract. Theory Public-Key Cryptogr.*, 2020, pp. 124–154.

[16] A. Bois, I. Cascudo, D. Fiore, and D. Kim, "Flexible and efficient verifiable computation on encrypted data," in *Proc. 24th IACR Int. Conf. Pract. Theory Public-Key Cryptogr.*, Springer, 2021, pp. 528–558.

[17] R. Gennaro and D. Wichs, "Fully homomorphic message authenticators," in *Proc. 19th Int. Conf. Theory Appl. Cryptol. Inf. Secur.*, Springer, 2013, pp. 301–320.

[18] D. Catalano and D. Fiore, "Practical homomorphic MACs for arithmetic circuits," in *Proc. 32nd Annu. Int. Conf. Theory Appl. Cryptogr. Techn.*, Springer, 2013, pp. 336–352.

[19] D. Fiore, A. Mitrokotsa, L. Nizzardo, and E. Pagnin, "Multi-key homomorphic authenticators," *IET Inf. Secur.*, vol. 13, no. 6, pp. 618–638, 2019.

[20] M. Bellare, V. T. Hoang, and P. Rogaway, "Foundations of garbled circuits," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2012, pp. 784–796.

[21] V. Kolesnikov and T. Schneider, "Improved garbled circuit: Free XOR gates and applications," in *Proc. Int. Colloq. Automata, Lang., Program.*, Springer, 2008, pp. 486–498.

[22] S. Zahur, M. Rosulek, and D. Evans, "Two halves make a whole," in *Proc. Annu. Int. Conf. Theory Appl. Cryptographic Techn.*, Springer, 2015, pp. 220–250.

[23] M. Bellare, V. T. Hoang, S. Keelveedhi, and P. Rogaway, "Efficient garbling from a fixed-key blockcipher," in *Proc. IEEE Symp. Secur. Privacy*, 2013, pp. 478–492.

[24] J. Katz and Y. Lindell, *Introduction to Modern Cryptography: Principles and Protocols*, London, U.K.: Chapman and Hall/CRC," 2007.

[25] E. M. Songhori, S. U. Hussain, A.-R. Sadeghi, T. Schneider, and F. Koushanfar, "Tinygarble: Highly compressed and scalable sequential garbled circuits," in *Proc. IEEE Symp. Secur. Privacy*, 2015, pp. 411–428.

[26] I. Chillotti, N. Gama, M. Georgieva, and M. Izabachène, "TFHE: Fast fully homomorphic encryption over the torus," *J. Cryptol.*, vol. 33, no. 1, pp. 34–91, 2020.

[27] P. Hebborn and G. Leander, "Dasta–alternative linear layer for rasta," *IACR Trans. Symmetric Cryptol.*, vol. 2020, no. 3, pp. 46–86, 2020.

Xingkai Wang received the BE degree in computer science from Shanghai Jiao Tong University, in 2015. He is currently working toward the PhD degree with the Department of Computer Science and Engineering, Shanghai Jiao Tong University. His research interests include verifiable computation, privacy-preserving computing, and applied cryptography.

Zhenfu Cao (Senior Member, IEEE) is currently a distinguished professor with East China Normal University, China. Since 1981, more than 500 academic papers have been published in journals or conferences. His research interests include number theory, cryptography, and information security. He serves as a member of the Expert Panel of the National Nature Science Foundation of China. He has received a number of awards and the Leader of Asia 3 Foresight Program (61161140320), and key project (61033014,61632012) of the National Natural Science Foundation of China.

Zhen Liu received the PhD degree in computer science from the City University of Hong Kong and Shanghai Jiao Tong University, in 2013. He is currently an associate professor with the Department of Computer Science and Engineering at Shanghai Jiao Tong University. His primary interest is applied cryptography, studying provable security and designing cryptographic primitives for the research problems motivated by practical applications. Currently, he is working on post-quantum cryptography, privacy-preserving computing, and cryptographic primitives for blockchain.

Kaitai Liang (Member, IEEE) is a tenured faculty member with the Delft University of Technology. His research, featured in international information security journals and conferences like USENIX Security, NDSS, Asiacrypt, ESORICS (with a best research paper award), *IEEE Transactions on Information Forensics and Security*, and *IEEE Transactions on Dependable and Secure Computing*, addresses cybersecurity challenges using information security and cryptographic tools. As a principal investigator in various EU funded projects, he has demonstrated real-world security impact through collaborations with academic and industrial partners. He has served TPC member, general chair, and steering committee for international security and privacy conferences, e.g., USENIX Security, IEEE Euro S&P, ESORICS, IEEE CSF, and PoPETs, and an associate editor for international journals. He has also contributed to ISO standards as a member of the standards committee 381027 “Cybersecurity & Privacy” at NEN.