Lighter Is Better: A Lighter Multi-Client Verifiable Outsourced Computation with Hybrid Homomorphic Encryption

Xingkai Wang[1][0000−0002−7625−7932], Zhenfu Cao[2][0000−0002−5250−5030]✉, Zhen Liu[1,4][0000−0001−9268−702X], and Kaitai Liang[3][0000−0003−0262−7678]

[1] Shanghai Jiao Tong University, China
{starshine87, liuzhen}@sjtu.edu.cn
[2] East China Normal University, China
zfcao@sei.ecnu.edu.cn
[3] Delft University of Technology, Netherlands
kaitai.liang@tudelft.nl
[4] Shanghai Qizhi Institute, China

Abstract. Gordon et al. (TCC 2015) systematically studied the security of Multi-client Verifiable Computation (MVC), in which a set of computationally-weak clients outsource the computation of a general function f over their private inputs to an untrusted server. They introduced the universally composable (UC) security of MVC and proposed a scheme achieving UC-security, where the protocol remains secure after arbitrarily composed with other UC-secure instances. However, the clients in their scheme have to undertake the heavy computation overhead caused by fully homomorphic encryption (FHE) and further, the plaintext size is linear to the function input size.

In this work, we propose a more efficient UC-secure multi-client privacy-preserving verifiable computation protocol, called MVOC, that sharply reduces amortized overheads for clients, in both semi-honest and malicious settings. In particular, our protocol achieves stronger *outsourcability* by outsourcing more computation to the server, so that it may be more friendly to those lightweight clients. More specifically, we revisit the definition of garbling scheme, and propose a novel garbled circuit protocol whose circuit randomness is non-interactively provided by multiple parties. We also realize the idea of hybrid homomorphic encryption, which makes the FHE plaintext size independent of the input size. We present the detailed proof and analyze the theoretical complexity of MVOC. We further implement our protocol and evaluate the performance, and the results show that, after adopting our new techniques, the computation and communication overheads during input phase can be decreased by 55.15%-68.05% and 62.55%-75% respectively.

Keywords: Verifiable computation · Outsourced Computation · Hybrid homomorphic encryption.

1 Introduction

The technique of verifiable computation (VC) [12] and multi-client verifiable computation (MVC) [7, 17] are proposed to allow computationally weak clients to delegate the computation of a function f on private inputs to a remoted server, achieving privacy, soundness and efficiency in the single and multi-client context. Specifically, privacy requires the protocol not to disclose sensitive data including input and output, soundness ensures the validity of the result provided by the server, and efficiency guarantees that the cost of the client during outsourcing should be much lower than that of computing the function by itself.

Gennaro et al. [12] introduced the definition of verifiable computation (VC). Its primary goal is to achieve privacy and soundness against malicious server, assuming that the client is honest. Their protocol achieves efficiency amortizedly. They combined garbled circuit [20] with a fully homomorphic encryption (FHE) scheme [14]. More precisely, after generating a garbled circuits, the client is allowed to use FHE to encrypt the circuit along with encoded input labels. Due to IND-CPA security of the FHE scheme, the client can reuse the same circuit without loss of soundness and hence the computational cost for the client is bounded in amortized sense. Choi et al. [7] later extended single-client VC to the multiple clients to yield a MVC protocol. In such a setting, n clients intend to compute some function f over a series of joint input $\{(x_1^{(\mathsf{ssid})}, ..., x_n^{(\mathsf{ssid})})\}_{\mathsf{ssid}}$. They adopted a new primitive *proxy oblivious transfer* (POT) constructed from a non-interactive key-exchange (NIKE) scheme. This new primitive is able to keep clients' input private from each other and from the server. But their protocol cannot guarantee the security in the context of either client-client corruption or the existence of malicious clients. Gordon et al. [17] systematically studied MVC in the universally composable (UC) model. The UC security captures selective failure attack and adaptive soundness, and considers the participation of malicious clients, which may be seen as a stronger notion than the prior definitions. They also proposed a protocol that satisfies the "stronger" security, with a new primitive *attribute-hiding multi-sender attribute-based encryption* (ah-mABE), constructed by a two-outcome ABE scheme with local encoding, an FHE scheme and a POT protocol. They eventually pointed out the impossibility of achieving the security when client-server collusion exists.

There are many studies that have been proposed to achieve verifiable computation since Gennaro et al.'s VC protocol [12]. A similar approach is to apply succincct functional encryption (FE) technique, which was introduced by Goldwasser et al. [16]. Later, Goldwasser et al. [15] extended the former definition 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. Moreover, iO introduces a strong assumption, which makes the protocol less practical.

Another method to achieve private verifiable computation is to prove after computing, rather than to prove with computing. Fiore et al. [9] proposed a protocol on verifiable delegation of computation on encrypted data, by developing a novel homomorphic hashing technique that may significantly reduce overhead.

The core idea is to use a VC to prove the correctness of homomorphic evalutation on ciphertext, and the new technique solves the difficulty of dealing with FHE ciphertext expansion. Later, Fiore et al. [11] and Bois et al. [2] extended the protocol of [9]. The former supported public verifiability, and extended the degree of delegated function from two to any constant value, using well-designed zk-SNARKs for polynomial rings. While the latter increases the efficiency of HE scheme by allowing flexible choices of HE parameters. However, the incurred extra time cost on verification for client may not be applicable to computation-restricted devices.

Gennaro et al. [13] proposed a new primitive denoted as fully homomorphic message authenticators (HA), which allows the receiver to verify the computation result, constructed with FHE. Almost at the same time, Catalano et al. [5] proposed a new construction of HA by much more efficient building blocks, while sacrificing the maximum size of delegated circuit. In multi-client scenario, Fiore et al. [10] proposed several constructions of multi-key homomorphic authenticator. But the time cost of verifying a result for client is not less than executing the computation. An idea to avoid this overhead for client could be to outsource the verification function to the server. However, the implemented solutions either cause extra communication complexity which break the non-interactive property, or introduce SNARG-like proof system. None of the solutions so far is efficient enough for practical use.

Possible Limitations. In the constructions of both [7] and [17], the clients remain using FHE to encrypt labels proportional to their input size, where FHE is a bottleneck in efficiency, compared to other building blocks in their protocols.

Although outsourcability has been initially achieved, most of existing FHE-based protocols still cannot avoid heavy client-side overheads. This is so because the messages to be fully homomorphic encrypted for client is proportion to the size of function input. *How to make such complexity independent to the input size is an interesting long-lasting problem in the research line.*

Besides the cost of FHE, a POT-based MVC scheme needs $O(n^2)$ instances of functionality during a single online phase of outsourced computation, where n is the number of clients. The overhead caused by POT rises substantially as the number of users increases in both communication and computation. Furthermore, the first client always needs to take over most of the computation, since it has to encrypt "twice" the actual length of input labels in an instance of POT, which may lead to imbalance in the overall overheads. Thus, *a cost-effective and efficiency-balance protocol in multi-client context is worthy being considered, especially when n is sufficiently large.*

Contribution. We propose a new multi-client verifiable computation scheme MVOC. The proposed scheme has lower communication complexity and is more efficient for clients than existing works, while still satisfying strong security guarantees in both semi-honest and malicious model. Our contributions is summarized as follows:

– We revisit the definition of garbling scheme, and design a new primitive *Multi-client Outsourced Garbled Circuit* (MOGC). Its encoding function is generated

by all clients in a distributed way. Clients, who do not generate circuits, can learn the garbled input wires without using OT or proxy OT, so that we can significantly reduce the communication overhead from $O(n^2l)$ to $O(nl)$. We further show that a secure MOGC protocol implies a UC-secure one-time MVC.
− We adopt the technique of hybrid homomorphic encryption to the MOGC to construct a UC-secure MVC protocol against malicious server or semi-honest client-client collusion . We further 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. We also show the possibility to construct a secure MVC against the corruption of malicious client by adopting a zero-knowledge compiler.
− We implement the proposed MVOC with Yao's Garbled Circuit and the most efficient hybrid homomorphic encryption scheme, and make a comparison on clients' overhead in both computation and communication. The experimental results show that the proposed scheme provides a significant advantage in efficiency over other existing FHE-based works.

Schemes	Security model	Collusion	UC	Multi-client	Communication	Computation
Gennaro et al. [12]	semi-honest	✗	✗	✗	-	$O(dl\kappa)$
Choi et al. [7]	semi-honest	✗	✗	✓	$O(n^2l)$	$O(dl\kappa)$
Gordon et al. [17]	malicious	✓	✓	✓	$O(n^2l)$	$O(dl\kappa)$
MVOC	malicious	✓	✓	✓	$O(nl)$	$O(d\kappa) + O(l)$

* We denote d as the expansion rate of complexity caused by FHE.

Table 1: Comparison of Privacy-preserving Verifiable Computation

Technical Roadmap. Since it is impossible to achieve input privacy when the server colludes with any client [17], we may assume the client-server collusion does not exist. We divide our technical roadmap into two steps.

Multi-client Outsourced Garbled Circuit. The garbled circuit in a garbling scheme is mostly generated by a single party. This feature causes the inconvenience of providing the corresponding garbled labels for other parties. Specifically, in Yao's secure two-party computation protocol, after Alice generates the garbled circuit, the second participant Bob obtains the garbled labels by oblivious transfer (OT). Essentially, OT transfers the randomness from Alice to Bob. Bob is not allowed to learn the garbled labels that is not related to his input, since Bob also plays the role of circuit executor, and he may benefit from this extra information.

We notice that, if we separate the role of circuit executor from data provider, the above concern will be tackled. Briefly, we introduce a (not necessarily trusted) third party, the server, who dedicates to executing the computation, and the client provides his own randomness for garbled circuit. In this case, OT may be no longer needed, since client could generate its corresponding garbled labels using its own randomness.

According to the above observation, we propose a new primitive named *Multi-client Outsourced Garbled Circuit* (MOGC), and construct a secure MOGC protocol from Yao's Garbled Circuit. We also claim that a secure MOGC protocol implies a one-time MVC protocol. The proposed MOGC protocol solves the problem of transferring randomness from circuit generator to data provider, plays the role of OT in secure 2PC and the role of POT in MVC, where POT is the crux of unbalance problem.

Multi-client Verifiable Outsourced Computation. Similar to the approach of constructing VC from the one-time VC intuitively implied from Yao's Garbled Circuit, we use FHE to provide circuit privacy. In order to avoid heavy overhead caused by FHE, we adopt the philosophy of hybrid encryption: firstly using symmetric key encryption (SKE) scheme to encipher message, and then FHE to encapsulate the symmetric key. This KEM-DEM-like technique is able to offload the heavy FHE overhead to the server. More concretely, after the generator generates the garbled circuit F for a function f by MOGC in offline phase, each client P_i uses its own encoding function share to obtain garbled input X_i^{ssid} corresponding to its private input x_i^{ssid}. After that, the generator executes FHE setup to acquire FHE key pair $(\mathsf{pk}_{\mathsf{FHE}}, \mathsf{sk}_{\mathsf{FHE}})$. Each client computes the hybrid ciphertext, which comprises of an FHE and a SKE ciphertext. The server can recover the FHE ciphertext of garbled input wire from the hybrid ciphertext sent by each client, and computes on the garbled circuit as in MOGC. The correctness of FHE and MOGC ensures the result is verifiable. As long as there is a secure channel for transmitting FHE ciphertext of the hybrid, the protocol also achieves privacy against clients.

2 Multi-Client Verifiable Computation

2.1 Syntax

We first revise the notion of non-interactive multi-client verifiable computation (MVC). Let κ denote the security parameter. Suppose there are n clients P_1, ..., P_n intending to delegate some computation 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:
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 message to every other clients respectively, and also needs to send a message to the server Serv.
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). *Let κ be the security parameter, n be the number of clients. A non-interactive multi-client verifiable computation comprises the following three phases:*

Setup Phase: *All participants have access to a setup $\mathcal{G}$. 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 $e_{j,i}$, and sends a garbled version of f to Serv, noted as F_i.*
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 [17], our definition is different in two aspects. In offline phase, we allow each client to send a 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.

2.2 Security Definition

We follow the UC framework in [3]. We formally define the ideal functionality for MVC in Table 2, which captures the correctness and privacy, but also adaptive soundness and selective failure attack. The server and clients are either semi-honest or malicious in our model. In any circumstances, the server is not allowed to collude with any client; otherwise input privacy will be never guaranteed [17].

Remark 2. We note our security definition is different from [17] in the behavior of server S. Since we do not allow client-server collusion, 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 origin definition in [17], 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 [3]). *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.*

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.*

Multi-client Verifiable Computation

The functionality $\mathcal{F}_{\mathsf{MVC}}$ is parameterized with an n-ary function $f : \mathcal{X}^n \to \mathcal{Y}^n$. The functionality $\mathcal{F}_{\mathsf{MVC}}$ interacts with n clients P_1, ..., P_n, a remote server Serv and a simulator S.

• *Initialization:*

Upon receiving (**Init**) from client P_i, send (**Init**, P_i) to notify the simulator S. After S returns (**Init**, P_i), send (**Init**, $\mathsf{P}_i, \Phi(f_i)$) to the server Serv, and send (**Init**, P_i) to each other client P_j where $j \neq i$. After receiving all responces (**Init**, P_i) from client P_j for all $j \neq i$ and all responses (**Init**, $\mathsf{Serv}, \mathsf{P}_i$) for all $i \in [n]$ from server, send (**Init**, Serv) to notify the simulator S.

• *Outsourcing:*

Upon receiving (**Input**, sid, x_i) from client P_i, send ($\mathsf{ssid}, \mathsf{P}_i$) to notify S. After S returns ($\mathsf{ssid}, \mathsf{P}_i$), store ($\mathsf{ssid}, x_i$) and send a notification (**Input**, $\mathsf{ssid}, \mathsf{P}_i$) to Serv. Upon receiving (**Input**, ssid) from Serv, retrieve (ssid, x_i) for all $i \in [n]$. If some (ssid, x_i) has not been stored yet, send (**Output**, ssid, **FAIL**) to the server and all clients.

Compute $(y_1, ..., y_n) \leftarrow f(x_1, ..., x_n)$. Upon receiving ($\mathsf{ssid}, \mathsf{P}_i, \phi$) from the simulator S, if $\phi =$ **OK** then send (**Output**, ssid, y_i) to P_i; otherwise send (**Output**, ssid, **FAIL**) to P_i. Later when S returns ($\mathsf{ssid}, \mathsf{P}_i, \phi$), send (**Output**, ssid, y_i) to client P_i if $\phi =$ **OK**, else send (**Output**, ssid, **FAIL**) if $\phi =$ **FAIL** to client P_i.

Table 2: Ideal functionality of Multi-client Verifiable Computation

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. 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 clients are controlled by the adversary, while the channels between clients and server are secure. We can implement such channels with the ideal functionality $\mathcal{F}_{\mathsf{STP}}$ [3]. All of the protocols are described assuming in $\mathcal{F}_{\mathsf{STP}}$-hybrid world.

Outsourcability. The *oursourcability* 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 described in such 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 outsourcable MVC protocol. As for the latter, we may require the communication size to be constraint to at most proportional to the input size, in particular when there exists a relatively large client number.

3 Building Blocks

3.1 Garbling Scheme

The technique of garbled circuits was first proposed by Yao [20]. We first follow the well-designed definition in [1] culled out by Bellare et al. The garbled circuit is generated by a single party named *garbler*. We will later discuss a new case where the randomness of garbled circuit is provided by multiple parties.

Definition 4 (Garbling Schemes [1]). *A garbling scheme for a family of functions $\mathcal{F}$ whose arbitrary element f is a mapping that can efficiently compute, comprises five algorithms $\mathsf{Gb} = \mathsf{Gb}.\{\mathsf{Gb}, \mathsf{En}, \mathsf{De}, \mathsf{Ev}, \mathsf{ev}\}$. 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 origin function f and input x, obtain the plaintext output y.

We require a garbling scheme satisfying several 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 Schemes [1]). *For a garbling scheme $GS = Gb.(Gb, En, De, Ev, ev)$, and for any PPT adversary $\mathcal{A}$, consider the following experiment:* $\boldsymbol{Exp}^{\mathsf{Aut}}_{\mathcal{A}}[GS, \kappa]$:
$(F, e, d) \leftarrow Gb(1^\kappa, f)$; $X \leftarrow En(e, x)$; $Y' \leftarrow \mathcal{A}(F, X)$; $r \leftarrow De(Y')$;
If $r \neq \perp$ and $Y' \neq Ev(F, X)$, output 1, else 0.

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

Side-information function. 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 $\phi = \Phi(f)$. Specifically, in this paper, we allow the server to obtain the circuit size, including input and output size. In other words, we regard our protocol as a multi-client version of private function evaluation (PFE), rather than secure function evaluation (SFE). One could extend the definition to SFE version of MVC, by assigning $\Phi(f) = f$.

3.2 Fully Homomorphic Encryption

After Gentry [14] gave the first construction of FHE, Canetti et al. [4] proposed concrete implements of IND-CCA1-secure FHE. We don't need the security to be as strong as the one against CCA1 in our scheme. We show the syntax and define the security of IND-CPA as follows.

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

$-$ $(pk, sk) \leftarrow \mathsf{Gen}(1^\kappa)$. The key generation algorithm which outputs public-private key pair of FHE.

$-$ $c \leftarrow \mathsf{Enc}(pk, m)$. The encryption algorithm which takes message m as input and outputs ciphertext c.

$-$ $m \coloneqq \mathsf{Dec}(sk, c)$. The decryption algorithm which takes ciphertext c as input and outputs plaintext message $m \in \mathcal{M}$.

$-$ $c_{\mathsf{eval}} \coloneqq \mathsf{Eval}(C, \{c_i\})$. The evaluation algorithm which executes the circuit $C \in \mathcal{C}$ on ciphertext input collection $\{c_i\}$, and outputs ciphertext result c_{eval}.

Properties. The *Correctness* of FHE is defined as key pair generated by Gen allows the output of Dec which takes the ciphertext of Enc on some message m is identical to m. The *Homomorphic correctness* is defined as the output of Eval decrypts to the result of applying C on plaintext inputs $\{m_i\}$. The *Compactness* means that the size of homomorphic ciphertext should be independent of the size, depth value or number of inputs to C, and less than $\mathsf{poly}(\kappa)$.

Definition 6 (IND-CPA Security of FHE). *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:*

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

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

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

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

Note $\mathcal{A}$ *has access to* $\mathsf{Dec}(sk, \cdot)$ *as an oracle. We define its advantage in the experiment above as:* $\boldsymbol{Adv}_{\mathcal{A}}^{\mathsf{CPA}}(\mathsf{FHE}, \kappa) = \left| 2\Pr[\boldsymbol{Exp}_{\mathcal{A}}^{\mathsf{CPA}}[\mathsf{FHE}, \kappa] = 1] - 1 \right|$.

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

4 Multi-client Outsourced Garbled Circuits

In this section, we revisit the definition of Garbling Scheme [1], and introduce a new primitive called *Multi-client Outsourced Garbled circuits* (MOGC).

In conventional Yao's garbled circuits protocol for two-party computation [20], there are two parties, Alice and Bob, intending to execute a computation of function f over their private inputs x_a and x_b respectively. Specifically, Alice generates a garbled circuit F from the function f, and further sends it to Bob, along with her garbled version of private input X_a. Then Alice and Bob execute an OT protocol to enable Bob to learn the garbled version of his private input X_b without: (i) revealing information about b to Alice, and (ii) allowing Bob to know any garbled input except X_b. Bob then carries out the computation on F and obtains the garbled result Y. After receiving Y from Bob, Alice knows the final result $y = f(x_a, x_b)$.

We extend the previous definition to multi-client scenarios. In an MOGC protocol, the computation is carried out by a third party who does not provide any input. Specifically, there is a *generator*, a *server* and at least one *collaborator*. The generator and the collaborator(s) outsource the computation of a function f on their input data $\vec{x}$, and the generator learns the output $f(\vec{x})$. We require the protocol to be non-interactive, which indicates that each party could only send a single message to another party in an instance of the protocol. This means that we cannot trivially use the OT technique, and the randomness of the garbled circuits should be provided by all data providers.

We will first give the syntax of MOGC, construct a secure MOGC protocol from a Yao's protocol, and then shows that a MOGC protocol implies a one-time multi-client verifiable computation.

4.1 Syntax of MOGC

Definition 7 (Multi-client Outsourced Garbled Circuits). *An MOGC for a family of function $\mathcal{F}$ whose arbitrary element f is a mapping that can efficiently compute, comprises six algorithms $\mathsf{MOGC} = \mathsf{MOGC}.\{\mathsf{Gb_C}, \mathsf{Gb_G}, \mathsf{En}, \mathsf{De}, \mathsf{Ev}, \mathsf{ev}\}$. The first two algorithms are probabilistic and the rest are deterministic.*
$-$ $e_c \leftarrow \mathsf{Gb_C}(1^\kappa, f)$. The collaborator generates his corresponding part of encoding functions.
$-$ $(F, e_g, d) \leftarrow \mathsf{Gb_G}(1^\kappa, f, \vec{e_c})$. The generator generates his part of encoding and decoding function, computes the garbled circuits along with other parts of encoding functions.
$-$ $X_{c/g} = \mathsf{En}(e_{c/g}, x_{c/g})$. The generator or the collaborator obtains the garbled input from private input via the corresponding encoding function.
$-$ $Y = \mathsf{Ev}(F, \vec{X})$. The server carries out the computation on garbled circuits and obtains the encoded output.
$-$ $y = \mathsf{De}(d, Y)$. The generator recovers the final output using decoding function.
$-$ $y = \mathsf{ev}(f, \vec{x})$. An auxiliary function that carries out the original function f.

Our definition is defined in semi-honest model, so that all participants could gather information as much as they could without deviating from the original

protocol. We do not allow client-server corruption, but the corruptions between clients are permitted. We assume that there exists secure communication channels between client and server.

A secure MOGC protocol should satisfy several properties. The *Correctness* ensures the final output is identical to the result of function evaluation. The *Privacy* guarantees that each client's input is kept private from other client and the server. And the *Authenticity* requires that the server could not provide a wrong result that could be decoded by the generator.

Before proceeding to an MOGC construction, we analyze how Yao's garbled circuits can achieve secure two-party computation. Alice and Bob who have private inputs respectively want to compute a function on their inputs. Alice generates the garbled circuit while Bob carries out the exact computation. The privacy against Bob is guaranteed by the randomness of circuit generation, which is provided by the generator Alice. Meanwhile, the privacy against Alice is guaranteed by OT protocol, which delivers the randomness from Alice to Bob. A trivial observation is that the circuit executor should not be the circuit generator, since authenticity is protected by the randomness of encoded values, and that is the reason why the same garbled circuit should not be used twice. Fortunately, the circuit execution is taken over by a third party "server", which is independent of data providers. Hence there is no concern of such an authentic crisis. Moreover, if the encoding function is *separable*, which means that the randomness of the encoding function can be regarded as separately provided by each client correspondingly, we may construct a protocol without adopting OT. More precisely, after acquiring each client's encoding function share, the generator computes the garbled circuit with those encoding information. We give the definition of *separability* as follows.

Definition 8 (Separability of Garbling Scheme). *We say that a garbling scheme is separable if the garbling algorithm* Gb *can be equivalently regarded as the following, where there are* $|\mathcal{I}|$ *input wires indexed by* $[|\mathcal{I}|]$*, with* $\mathcal{I}$ *being the set of all input wires and two sub-algorithm* ran *and* garb*:*
1. randomly select $|\mathcal{I}|$ *randomness* $r_1, ..., r_{|\mathcal{I}|}$*;*
2. compute $e_i \leftarrow \mathsf{ran}(f, r_i)$ *for* $i \in [|\mathcal{I}|]$*, set* $e = (e_1, ..., e_{|\mathcal{I}|})$*;*
3. compute $d \leftarrow \mathsf{ran}(f, r_1)$*;*
4. compute $F \leftarrow \mathsf{garb}(f, e, d)$*;*

Theorem 1. *Assuming the existence of a separable garbling scheme, there exists a correct, private and authenticate multi-client outsourced garbled circuit protocol.*

Proof. We prove the theorem by construction. For Gb_C, on input f the collaborator executes Step 1 and 2 in the above definition. For Gb_G, on input f and all encoding function shares $\vec{e_c}$ from collaborators, the generator executes Step 2, 3 and 4 in the above definition. The remaining algorithms are identical to the original garbling scheme respectively.

The construction of MOGC and a garbling scheme is merely slightly different in the generation of encoding function e. The definitions of correctness, privacy

(including obliviousness), and authenticity are irrelevant to how e is generated. These properties can easily be inherited from those of a garbling scheme. □

4.2 Construction of MOGC

We define a protocol for MOGC from conventional Yao's Garbled Circuits. Yao's circuits include garbled gates corresponding to the circuit gates. Suppose x_a, x_b be the input wires of a gate, and x_c be the output wire. The generator Alice first randomly chooses six values for each gate, w_t^b where $t \in \{a, b, c\}$ and $b \in \{0, 1\}$, represents 0 and 1 values of three wires respectively. Each of garbled gates contains four ciphertexts $\gamma_{ij} = E_{w_a^i}(E_{w_b^j}(w_c)^{g(i,j)})$, where $i, j \in \{0, 1\}$ and E is a well-designed symmetric encryption scheme.

It is intuitive that Yao's Garbled Circuits protocol is a separable garbling scheme. Instead of generating $|\mathcal{I}|$ encoded values by the same randomness r, we use $|\mathcal{I}|$ different randomness $r_1, ..., r_{|\mathcal{I}|}$ to obtain the encoded values. A PPT adversary cannot distinguish between two sets of the random values with merely different randomness. Hence we achieve the separability.

Moreover, we separate the $|\mathcal{I}|$ input wires into n buckets. When generating the encoded values for input wires, we use the same randomness for inputs in the same bucket, and use variant randomnesses for inputs in different buckets. A PPT adversary also cannot distinguish the random values from the encoding values of Yao's protocol.

We define our protocol as follows. For each input wire that is relevant to the client C_i's input, we let C_i choose two random values of this wire using his own randomness r_i and then handle the values to the generator. Then, the generator chooses random values which are relevant to his own input and other non-input wires, and computes the garbled circuit F using those values. After acquiring F and all encoded inputs from clients, the server carries on the computation on F and obtains the encoded output, and handles 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 MOGC according to Theorem 1.

5 Construction

We present our construction for MVC and give the proof for its UC-security. We start the construction from designing one-time MVC protocol from MOGC. We define the ideal functionality of one-time MVC named $\mathcal{F}_{\mathsf{OT\text{-}MVC}}$, 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 $\mathcal{F}_{\mathsf{OT\text{-}MVC}}$-hybrid world.

Without loss of generality, we only consider one case in the following construction: when only client P_1 may obtain the output. We could simply executing the protocol in parallel to achieve output-retrieving for every client. Similar approaches also can be seen in [7, 17].

5.1 One-Time Multi-client Verifiable Computation (OT-MVC)

There are n clients and a server participating in an OT-MVC protocol. All clients are reached consensus on the function f to be computed. Each client P_i provides a private input x_i respectively. The goal is to enable the client P_1 to obtain the function result $f(x_1, ..., x_n)$, while the server only knows the legal side-information $\Phi(f)$. The client participants behave semi-honestly while the server behaves maliciously, and client-server collusion is not allowed in our model. The ideal functionality $\mathcal{F}_{\mathsf{OT\text{-}MVC}}$ is the same as $\mathcal{F}_{\mathsf{MVC}}$ except that the outsourcing phase is only executed once. The formal definition of the ideal functionality $\mathcal{F}_{\mathsf{OT\text{-}MVC}}$ is shown in Table 3.

One-Time Multi-client Verifiable Computation

The functionality $\mathcal{F}_{\mathsf{OT\text{-}MVC}}$ is parameterized with an n-ary function $f : \mathcal{X}^n \to \mathcal{Y}^n$, and interacts with n clients P_1, ..., P_n, a remote server Serv and a simulator S.
Upon receiving (**Init**) from client P_i, send $(\mathbf{Init}, \mathsf{P}_i)$ to notify the simulator S. After S returns $(\mathbf{Init}, \mathsf{P}_i)$, send $(\mathbf{Init}, \mathsf{P}_i, \Phi(f_i))$ to the server Serv, and send $(\mathbf{Init}, \mathsf{P}_i)$ to each client P_j where $j \neq i$. After receiving all responces $(\mathbf{Init}, \mathsf{P}_i)$ from client P_j for all $j \neq i$ and all responses $(\mathbf{Init}, \mathsf{Serv}, \mathsf{P}_i)$ for all $i \in [n]$ from server, send $(\mathbf{Init}, \mathsf{Serv})$ to notify the simulator S.
Upon receiving $(\mathbf{Input}, x_i)$ from client P_i, send (P_i) to notify S. After S returns (P_i), store (x_i) and send a notification $(\mathbf{Input}, \mathsf{P}_i)$ to Serv. Upon receiving (**Input**) from Serv, retrieve (x_i) for all $i \in [n]$. If some (x_i) has not been stored yet, send $(\mathbf{Output}, \mathbf{FAIL})$ to the server and all clients.
Compute $(y_1, ..., y_n) \leftarrow f(x_1, ..., x_n)$. Upon receiving (P_i, ϕ) from the simulator S, if $\phi = \mathbf{OK}$ then send $(\mathbf{Output}, y_i)$ to P_i; otherwise send $(\mathbf{Output}, \mathbf{FAIL})$ to P_i. Later when S returns (P_i, ϕ), send $(\mathbf{Output}, y_i)$ to client P_i if $\phi = \mathbf{OK}$, else send $(\mathbf{Output}, \mathbf{FAIL})$ if $\phi = \mathbf{FAIL}$ to client P_i.

Table 3: Ideal functionality of One-Time Multi-client Verifiable Computation

We give a construction of $\mathcal{F}_{\mathsf{OT\text{-}MVC}}$ from a secure MOGC protocol. Let $f : (\{0,1\}^l)^n \to \{0,1\}^l$ be the outsourced function, P_1 be the generator, $\mathsf{P}_2, ..., \mathsf{P}_n$ be the collaborators, and Serv be the server. The parties work as follows:
– Collaborators $\mathsf{P}_2, ..., \mathsf{P}_n$ execute the algorithm Gb_C to generate encoding function shares $e_2, ..., e_n$ respectively, and send the shares to P_1.
– The generator P_1 executes the algorithm Gb_G to generate his encoded function share e_1, the garbled circuit F and the decoding function d. Then P_1 sends F to the server.
– All clients $\mathsf{P}_1, ..., \mathsf{P}_n$ execute the algorithm En on their own private inputs $x_1, ..., x_n$ respectively to obtain the garbled input $X_1, ..., X_n$, and send the garbled inputs to the server.
– The server Serv executes the algorithm Ev on encoded inputs to obtain the

encoded output Y, and sends the result to the generator P_1.
− The generator P_1 executes the algorithm De on encoded output Y to recover the final result. If Y is a valid value, it accepts and outputs the final result y; otherwise rejects and outputs $\perp$.

Theorem 2. *The above protocol UC-realizes $\mathcal{F}_{\mathsf{OT\text{-}MVC}}$ against semi-honest corruption of any fixed subset of clients, or against malicious server corruption.*

Proof. Let Π represent the above protocol. We intend to construct a simulator S for any PPT environment $\mathcal{E}$, such that for any PPT adversary $\mathcal{A}$ who is allowed to corrupt the server maliciously and to corrupt a fixed subset of clients semi-honestly, the two ensembles $\mathsf{EXEC}_{\Pi,\mathcal{A},\mathcal{E}}$ and $\mathsf{EXEC}_{\mathcal{F}_{\mathbf{OT\text{-}MVC}},\mathcal{S},\mathcal{E}}$ are indistinguishable. Upon receiving an input from the environment $\mathcal{E}$, the simulator S writes the input on $\mathcal{A}$'s input tape. Upon obtaining an output value from the adversary $\mathcal{A}$, the simulator S writes the output on $\mathcal{E}$'s output tape.
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 (semi-honest) clients. Besides the communication script, the simulator S needs to simulate the view of all corrupted clients, which contains the encoded function share and the encoded input for each client. The simulator S randomly chooses two strings as the encoded input wires for a label, and sets the encoded input to the exact string corresponding to the input. The adversary cannot tell the difference between the encoded input wires in the encoding function and the newly generated random strings.
Case 3: Corrupted (malicious) server and honest clients. 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 MOGC.

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

5.2 Construction of MVOC

We give a construction that UC-realizes $\mathcal{F}_{\mathsf{MVC}}$ from a secure 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, where $\mathcal{G}^{\mathsf{FHE}}$ serves as a self-registered PKI which allows P_1 to generate FHE key pair and

register the public key, and $\mathcal{F}_{\mathsf{SMT}}$ is the functionality of Secure Message Transmission [3]. It returns $\mathsf{pk}_{\mathsf{FHE}}$ when the server or any other party queries it. The construction is described as follows. For simplicity, we omit the superscript ssid of the variables in online phase.

1. Collaborators $\mathsf{P}_2, ..., \mathsf{P}_n$ execute the algorithm $\mathsf{MOGC.Gb_C}$ to generate encoding function shares $e_2, ..., e_n$ respectively, and send the shares to P_1.
2. The generator P_1 executes the algorithm $\mathsf{MOGC.Gb_G}$ to generate his encoded function share e_1, the garbled circuit F and the decoding function d. Then P_1 sends F to the server.
3. The generator P_1 interacts with $\mathcal{G}_{\mathsf{FHE}}$ and acquires a new FHE key pair $(\mathsf{pk}_{\mathsf{FHE}}, \mathsf{sk}_{\mathsf{FHE}})$. Then P_1 obtains X_1 by executing the algorithm $\mathsf{MOGC.En}$ on its own private input x_1, randomly chooses a symmetric key $k_1 \leftarrow \mathcal{K}_{\mathsf{SKE}}$, and computes $m_{11} = \mathsf{SKE.Enc}(k_1, X_1)$ and $m_{12} = \mathsf{FHE.Enc}(\mathsf{pk}_{\mathsf{FHE}}, k_1)$. P_1 finally sends $m_1 = (m_{11}, m_{12})$ to the server Serv.
4. For $i \in [n] \backslash \{1\}$, the collaborator P_i interacts with $\mathcal{G}_{\mathsf{FHE}}$ and acquires the current FHE public key $\mathsf{pk}_{\mathsf{FHE}}$. Similarly, P_i obtains X_i by executing the algorithm $\mathsf{MOGC.En}$ on its own private input x_i, randomly chooses a symmetric key $k_i \leftarrow \mathcal{K}_{\mathsf{SKE}}$, and computes $m_{i1} = \mathsf{SKE.Enc}(k_i, X_i)$ and $m_{i2} = \mathsf{FHE.Enc}(\mathsf{pk}_{\mathsf{FHE}}, k_i)$. P_i finally sends $m_i = (m_{i1}, m_{i2})$ to the server Serv.
5. After parsing $m_i = (m_{i1}, m_{i2})$, Serv computes $\hat{X}_i = \mathsf{SKE.Dec}(m_{i1}, \mathsf{FHE.Enc}(m_{i2}))$, for $i \in [n]$. Then Serv executes the algorithm $\mathsf{MOGC.Ev}$ on $\{\hat{X}_i\}_n$, obtains a circuit result $\hat{Y}$, and sends it to the generator P_1.
6. The generator P_1 computes $Y = \mathsf{FHE.Dec}(\mathsf{sk}_{\mathsf{FHE}}, \hat{Y})$, and recovers the final result by executing the algorithm $\mathsf{MOGC.De}$. If Y is a valid value, then P_1 accepts and outputs the final result y. Otherwise, P_1 rejects and outputs $\perp$.

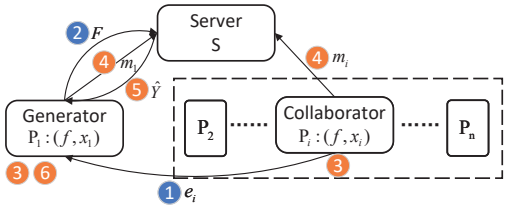

Fig. 1: Construction of MVOC

As shown in Fig. 1, Step 1 and 2 are in offline phase, and the rest are online. Step 3 and 6 are locally executed by each client and the generator respectively with no data transmission between participants. After Step 6 is executed without the result being $\perp$, the protocol comes back to Step 3; otherwise, it terminates. This abortion makes the advantage that the server gained from providing incorrect result cannot be carried over to the next sub-session.

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 secure multi-client outsourced garbled circuit protocol, then the aforementioned protocol UC-realizes* $\mathcal{F}_{\mathsf{MVC}}$ *against semi-honest corruption of any fixed subset of clients, or against malicious server corruption.*

Proof. Let Π represents the above protocol. We intend to construct a simulator S for any PPT environment $\mathcal{E}$, such that for any PPT adversary $\mathcal{A}$ who is allowed to corrupt the server maliciously and to corrupt a fixed subset of clients semi-honestly, the two ensembles $\mathsf{EXEC}_{\Pi,\mathcal{A},\mathcal{E}}$ and $\mathsf{EXEC}_{\mathcal{F}_{\mathsf{MVC}},\mathcal{S},\mathcal{E}}$ are indistinguishable. Upon receiving an input from the environment $\mathcal{E}$, the simulator S writes the input on $\mathcal{A}$'s input tape. Upon obtaining an output value from the adversary $\mathcal{A}$, the simulator S writes the output on $\mathcal{E}$'s output tape.

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 (semi-honest) clients. Besides the communication script, the simulator S needs to simulate the view of all corrupted clients, which contains an FHE public key, the encoded function share and the message sent to the server for each client. Concretely speaking, if the generator P_1 is not corrupted, the view in the real world contains $(\mathsf{pk}_{\mathsf{FHE}})$ and $(e_i, k_i, m_{i1}, m_{i2})$ for each client P_i. It is intuitive that all the above variables could be reproduced by the same way as in the protocol, since all the collaborators only send messages to others and do not receive any message from other parties. Specifically, the simulator S randomly chooses the encoded wires for simulating the encoding function share, noted as e_i'. Then in each online phase, after acquiring input x_i from the environment, S interacts with $\mathcal{G}_{\mathsf{FHE}}$ to obtain $\mathsf{pk}_{\mathsf{FHE}}$. Next, the simulator S randomly chooses a symmetric key $k_i' \leftarrow \mathcal{K}_{\mathsf{SKE}}$, and uses the key to encrypt its garbled input $X_i' = e_i'(x_i)$ corresponding to x_i and obtains the ciphertext $\mathsf{SKE.Enc}(k_i', X_i)$, denoted as m_{i2}'. After that, S uses $\mathsf{pk}_{\mathsf{FHE}}$ to encrypt the symmetric key k_i' and obtains the encapsulated key $\mathsf{FHE.Enc}(\mathsf{pk}_{\mathsf{FHE}}, k_i')$, denoted as m_{i1}'. Because of the semantic security of SKE and the IND-CPA security of FHE, a PPT adversary cannot distinguish the views between the ideal and real worlds. On the other hand, if the generator P_1 is corrupted, the view contains $(\mathsf{pk}_{\mathsf{FHE}}, \mathsf{sk}_{\mathsf{FHE}}, d, F)$ and $(e_i, k_i, m_{i1}, m_{i2})$ for each client P_i. The only difference from the above case is that the generator P_0 needs to simulate the circuit F along with its decoding function d. Actually he could reproduce these values by first randomly generating the decoding function d' and its encoding function share e_1', and then randomly choosing intermediate garbled wires, and computing a circuit F' using the garbled wires above. These new variables are chosen randomly, since the adversary still cannot distinguish the views between the two worlds.

Case 3: Corrupted (malicious) server and honest clients. Besides the communication script, S needs to simulate the server Serv's view, including $(F, \mathsf{pk}_{\mathsf{FHE}}, \hat{Y})$, and (m_{i1}, m_{i2}) for each $i \in [n]$. 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 the public key

$\mathsf{pk}_{\mathsf{FHE}}$. 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 $\mathsf{SKE.Enc}(s_1, s_2)$ and $\mathsf{FHE.Enc}(\mathsf{pk}_{\mathsf{FHE}}, s_1)$, denoted as m'_{i1} and m'_{i2} respectively. 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}, m_{i2}) from m'_{i1} and m'_{i2}. 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 MOGC. If the latter happens, we can construct an adversary $\mathcal{B}$ that uses (m_{i1}, m'_{i1}) and (m_{i2}, m'_{i2}) as input to break the semantic security of SKE and 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. □

5.3 From semi-honest clients to malicious clients

We inherit the approach in [17] to upgrade an MVC protocol that is secure against semi-honest clients to the one against malicious clients. The theorem in [17] indicates that, if an MVC protocol which is secure against semi-honest client corruptions satisfies *perfect privacy*, then there exists an MVC protocol which is secure against malicious client corruptions, in the ZK and self-registered PKI setup hybrid world. We will give the definition as follows.

Definition 9 (Perfect Privacy of MVC [17]). *An MVC protocol which is secure against semi-honest client corruptions is perfectly private if for all inputs $x_1, ..., x_n$, for an adversary $\mathcal{A}$ that semi-honestly corrupts some subset of the parties with index set $\mathcal{I}$ where $\mathcal{I} \subset [n]$, and for every random tape $r_{\mathcal{A}}$ belonging to $\mathcal{A}$, there exists a simulator S such that the two distributions $\mathsf{View}_{\Pi(x_1,...,x_n),\mathcal{A}}$ and $\mathsf{S}(\{x_i, y_i\}_{i\in\mathcal{I}}, r_{\mathcal{A}})$ are identical.*

We then show our previous MVOC construction satisfying perfect privacy.

Claim. If MOGC, FHE and SKE in the aforementioned MVOC construction are perfectly correct, then MOVC satisfies perfect privacy.

Proof. Since all the collaborators in our protocol do not receive any message, their view could be easily simulated by executed the protocol honestly. Hence we only need to simulate the view of the generator P_1 in the case of $1 \in \mathcal{I}$, without loss of generality. The view $\mathsf{View}_{\Pi(x_1,...,x_n),\mathcal{A}}$ for P_1 contains (F, e_1, d) and $\hat{Y}$ for each online phase. During the protocol P_1 uses randomness r to obtain e_1, d, and randomness r_{sid} to obtain k_1, m_{11}, m_{12}. S could use r_{sid} to compute the current encoded result by encrypting the encoded wire of y_1 using FHE, producing a view which is identical to $\hat{Y}$. Thus S can simulate a perfect identical view.

6 Evaluation

6.1 Efficiency Analysis

As we discussed before, *outsourcability* may be seen as an efficiency improvement for client side. We make a comparison between our scheme and Choi et al.'s [7]

in terms of efficiency performance, since [7] is a general solution in the multi-client settings, which is theoretically more efficient than the general UC-secure solution [17]. We analyze the cost in the offline and online phases separately. Because the final result could only be obtained by the generator, for average consideration, we aggregate a total complexity by executing the protocol for n times in a single computing period, during which each client acts as the generator once and as the collaborator $n-1$ times, where n is the number of clients.

• In offline phase, the clients together generate the whole garbled circuit. The total computational complexity is $O(\Phi(f))$. For communication, each collaborator sends a message of size $O(l)$ to the generator, and the generator sends an $O(\Phi(f))$-size circuit to server.
• 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, the generator executes a fully homomorphic decrypting algorithm with plaintext size $O(l\kappa)$. For communication, each client sends the ciphertext generated above.

6.2 Implementation and Evaluation

We implement our protocol, and run all the experiments on an Ubuntu Server with Intel i7-10700 CPU (2.9GHz) and 80 GB DDR4 2133MHz RAM. We focus on simulating the execution of clients. We adopt TinyGarble [19] as the implement of garbling scheme, which is mature and uses several most recent optimization on GC protocol. The computation benchmark we choose is the AES encryption algorithm. For simplicity, we set the number of clients be 2. One client provides a 128-bit key, and the other uses a 128-bit plaintext. There are 6,400 non-XOR gates in the circuit, and the total size of the garbled circuit is 2.1×10^6 bits. We use TFHE [6] as the FHE scheme, since it supports the encryption of boolean values and can be optimized for fast gate bootstrapping. For symmetric encryption scheme, we deploy two schemes, a 4-round algorithm AGRASTA [8] and a 6-round DASTA [18]. The former provides small key size while the latter is efficient in encrypting. Both schemes are well compatible with TFHE. We note that one may use more efficient SKE schemes (which could be less compatible with FHE) but relatively huge overhead may be incurred in the server side.

We set security parameter κ be 128. Specifically, the key-size of AGRASTA and DASTA are 129 and 351 respectively. All the experiments are executed in single-thread mode. We compare our implement with [7], and the result is shown in Table 4. As we claimed previously, the more number of client we set, the larger scale of input will be, and thus the more outsourcability our protocol provides. Our implement is the least advantegeous in this situation, where there are only two clients and the input length is only 128 for each client, which is almost as little as the ciphertext extension rate of FHE. Under such circumstances, we increase the communication efficiency by at least 2.67x and the time cost by at least 2.24x, as compared to [7]. As the data size increases, the improvement ratio of efficiency converges to a ratio proportion to the throughput rate of the

two schemes. According to our evaluation, the slowest SKE throughput is 72.98 bit/ms by AGRASTA, while TFHE offers 46.57 bit/ms. Hence, with large input scale, our efficiency improvement comes to 4x and 3.13x respectively.

It is worth noting that the evaluation of the efficiency improvement is based on the worst case, in which the number of non-server participants is 2. As shown in Table 1, a larger n brings higher efficiency improvements.

Schemes	Offline phase		Input phase	
	Time(ms)	Communication(kbits)	Time(ms)	Communication(kbits)
Choi et al. [7]	73.756	2099.2	1055.434*	147.456
MVOC (DASTA)	73.756	2131.968	65.22	55.232
MVOC (AGRASTA)	73.756	2131.968	480.695	41.024

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

Table 4: Overheads for Clients

7 Conclusion

We proposed a lighter multi-client privacy-preserving verifiable outsourced computation scheme. To adopt garbled circuit in multi-client scenario, we developed a new primitive MOGC based on garbling scheme. We further showed that a secure MOGC protocol implies a one-time multi-client verifiable computation. To construct an efficient MVOC protocol, we used hybrid encryption technique to avoid expensive overheads from FHE. We proved the UC security of the proposed protocol. We made a theoretical analysis on efficiency and implemented our scheme. The results demonstrate that the proposed protocol can enhance the efficiency of input phase by 2.67-4x and 2.24-3x in communication and computation cost, respectively.

Acknowledgements This work was supported by the National Key Research and Development Program of China (Grant No. 2020YFA0712300), the National Natural Science Foundation of China (No. 62132013, 62072305, 62132005, 62172162), and European Union's Horizon 2020 research and innovation programme under grant agreement No. 952697 (ASSURED) and No. 101021727 (IRIS).

References

1. Bellare, M., Hoang, V.T., Rogaway, P.: Foundations of garbled circuits. In: ACM CCS '12. pp. 784–796

2. Bois, A., Cascudo, I., Fiore, D., Kim, D.: Flexible and efficient verifiable computation on encrypted data. In: PKC '21. pp. 528–558. Springer
3. Canetti, R.: Universally composable security: A new paradigm for cryptographic protocols. In: FOCS '01. pp. 136–145
4. Canetti, R., Raghuraman, S., Richelson, S., Vaikuntanathan, V.: Chosen-ciphertext secure fully homomorphic encryption. In: PKC '17. pp. 213–240. Springer
5. Catalano, D., Fiore, D.: Practical homomorphic macs for arithmetic circuits. In: EUROCRYPT '13. pp. 336–352. Springer
6. Chillotti, I., Gama, N., Georgieva, M., Izabachène, M.: Tfhe: fast fully homomorphic encryption over the torus. Journal of Cryptology **33**(1), 34–91 (2020)
7. Choi, S.G., Katz, J., Kumaresan, R., Cid, C.: Multi-client non-interactive verifiable computation. In: TCC '13. pp. 499–518. Springer
8. Dobraunig, C., Eichlseder, M., Grassi, L., Lallemand, V., Leander, G., List, E., Mendel, F., Rechberger, C.: Rasta: a cipher with low anddepth and few ands per bit. In: CRYPTO '18. pp. 662–692. Springer
9. Fiore, D., Gennaro, R., Pastro, V.: Efficiently verifiable computation on encrypted data. In: ACM CCS '14. pp. 844–855
10. Fiore, D., Mitrokotsa, A., Nizzardo, L., Pagnin, E.: Multi-key homomorphic authenticators. IET Information Security **13**(6), 618–638 (2019)
11. Fiore, D., Nitulescu, A., Pointcheval, D.: Boosting verifiable computation on encrypted data (2020)
12. Gennaro, R., Gentry, C., Parno, B.: Non-interactive verifiable computing: Outsourcing computation to untrusted workers. In: CRYPTO '10. pp. 465–482
13. Gennaro, R., Wichs, D.: Fully homomorphic message authenticators. In: ASIACRYPT '13. pp. 301–320. Springer
14. Gentry, C.: Fully homomorphic encryption using ideal lattices. In: STOC '09. pp. 169–178
15. Goldwasser, S., Gordon, S.D., Goyal, V., Jain, A., Katz, J., Liu, F.H., Sahai, A., Shi, E., Zhou, H.S.: Multi-input functional encryption. In: EUROCRYPT '14. pp. 578–602. Springer
16. Goldwasser, S., Kalai, Y., Popa, R.A., Vaikuntanathan, V., Zeldovich, N.: Reusable garbled circuits and succinct functional encryption. In: STOC '13. pp. 555–564
17. Gordon, S.D., Katz, J., Liu, F.H., Shi, E., Zhou, H.S.: Multi-client verifiable computation with stronger security guarantees. In: TCC '15. pp. 144–168. Springer
18. Hebborn, P., Leander, G.: Dasta–alternative linear layer for rasta. IACR Transactions on Symmetric Cryptology pp. 46–86 (2020)
19. Songhori, E.M., Hussain, S.U., Sadeghi, A.R., Schneider, T., Koushanfar, F.: Tinygarble: Highly compressed and scalable sequential garbled circuits. In: S&P '15. pp. 411–428
20. Yao, A.C.C.: How to generate and exchange secrets. In: FOCS '86. pp. 162–167