11-2021

Efficient server-aided secure two-party computation in heterogeneous mobile cloud computing

Yulin WU

Xuan WANG

Willy SUSILO

Guomin YANG
Singapore Management University, gmyang@smu.edu.sg

Zoe L. JIANG

See next page for additional authors

Citation

WU, Yulin; WANG, Xuan; SUSILO, Willy; YANG, Guomin; JIANG, Zoe L.; CHEN, Qian; and XU, Peng. Efficient server-aided secure two-party computation in heterogeneous mobile cloud computing. (2021). *IEEE Transactions on Dependable and Secure Computing*. 18, (6), 2820-2834.
Available at: https://ink.library.smu.edu.sg/sis_research/7295

Efficient Server-Aided Secure Two-Party Computation in Heterogeneous Mobile Cloud Computing

Yulin Wu, Xuan Wang, *Member, IEEE*, Willy Susilo, *Senior Member, IEEE*, Guomin Yang, *Senior Member, IEEE*, Zoe L. Jiang, Qian Chen, and Peng Xu, *Member, IEEE*

Abstract—With the ubiquity of mobile devices and rapid development of cloud computing, mobile cloud computing (MCC) has been considered as an essential computation setting to support complicated, scalable and flexible mobile applications by overcoming the physical limitations of mobile devices with the aid of cloud. In the MCC setting, since many mobile applications (e.g., map apps) interacting with cloud server and application server need to perform computation with the private data of users, it is important to realize secure computation for MCC. In this article, we propose an efficient server-aided secure two-party computation (2PC) protocol for MCC. This is the first work that considers collusion between a malicious garbled circuit evaluator and a semi-honest server while ensuring privacy and correctness. Also, it can guarantee fairness when collusion does not exist. The security analysis shows that our protocol can securely compute any function f(x, y) against different types of adversaries in the malicious model. Also, the experimental performance analysis shows that this work outperforms the previous works for at least 10 times with the same security level.

Index Terms—Secure two-party computation, server-aided computation, mobile cloud computing, garbled circuit

1 Introduction

Mobile devices with the aid of wireless communication technologies have gained tremendous popularity. More and more useful but complicated applications (such as map apps, car-hailing apps, social apps, and banking apps) need to be implemented on mobile devices. However, most of the data that these apps process is directly linked to the privacy of mobile users. Thus, to provide privacy protection for mobile device users, the most effective way to securely implement these applications is to execute secure two-party computation between the mobile device and an application server. However, the limited capabilities of mobile devices for computation, storage and communication have been a bottleneck for mobile devices to efficiently take the corresponding task of secure two-party computation. Fortunately, the cloud computing technology can help to resolve this problem, as more powerful computation, storage and communication resources can be offered as an on-demand service to mobile devices. Integrating the advantages of cloud computing, mobile cloud computing (MCC) arises rapidly as a new computing paradigm in recent years. It enables mobile devices to overcome the hardware limits and provides the possibility to realize complicated secure computation between a mobile device and an application server with the aid of the cloud server.

Secure two/multi-party computation (2/MPC) is one of the central problems in modern cryptography. It enables a group of parties with their own private inputs to jointly compute a function without the necessity of revealing any information about their inputs except for the output of the function. MPC protocols can not only achieve various privacy-preserving data analysis with the private data of participant parties, but can also resist one single point attack by distributing secrets and computation. After nearly three decades of development, MPC has achieved many achievements based on different adversarial models. Loosely speaking, there are two most common adversarial models: (1) semi-honest model, where adversaries follow the protocol but try to learn more than is allowed by inspecting the protocol transcript; (2) malicious model, where adversaries can adopt any strategy to break the protocol. For the semi-honest model, programming tools [1], [2], [3] realize the application on benchmark applications like AES and PSI, and some other complicated applications with complex logic [4], [5] and large scale sensitive data [6], [7], [8]. Although MPC protocols for semi-honest model are efficient and can be applied to many fields, they cannot provide security guarantee in the presence

- *Y. Wu and Q. Chen are with the School of Computer Science and Technology, Harbin Institute of Technology, Shenzhen, Guangdong 518055, China. E-mail: yulinwu@cs.hitsz.edu.cn, qianchen@stu.hit.edu.cn.*
- *X. Wang and Z. L. Jiang are with the School of Computer Science and Technology, Harbin Institute of Technology, Shenzhen, Guangdong 518055, China, and also with Pengcheng Laboratory, Shenzhen, Guangdong 518055, China. E-mail: wangxuan@cs.hitsz.edu.cn, zoeljiang@hit.edu.cn.*
- *W. Susilo and G. Yang are with the Institute of Cybersecurity and Cryptology, School of Computing and Information Technology, University of Wollongong, Wollongong, NSW 2522, Australia. E-mail: {wsusilo, gyang}@uow.edu.au.*
- *P. Xu is with the National Engineering Research Center for Big Data Technology and System, Services Computing Technology and System Lab, Cluster and Grid Computing Lab, Big Data Security Engineering Research Center, School of Cyber Science and Engineering, Huazhong University of Science and Technology, Wuhan, Hubei 430074, China. E-mail: xupeng@mail.hust.edu.cn.*

Manuscript received 21 Mar. 2019; revised 28 Oct. 2019; accepted 3 Jan. 2020. Date of publication 14 Jan. 2020; date of current version 11 Nov. 2021.
(Corresponding author: Xuan Wang.)
Digital Object Identifier no. 10.1109/TDSC.2020.2966632

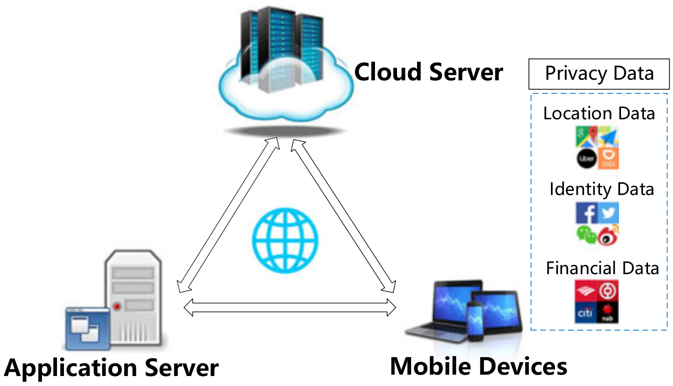

Fig. 1. Mobile cloud computing with applications.

of malicious adversaries. To achieve a higher security level, many works focused on the malicious model and made some progress. For the malicious model, recent works have shown that they can securely compute hundreds of thousands logical-gates per second [3], [9], [10], [11], [12], which provides the possibility for MPC to be further applied in practice.

However, all the above standard MPC protocols use the homogeneous computation model as the default setting, where all parties play symmetric (or similar) roles with symmetric (or similar) computation resources. Unfortunately, this setting is not very common in today heterogeneous computing paradigm, especially for the MCC setting. In the MCC, mobile devices usually take roles in collecting and storing the private data of users with the popular apps like map apps (such as Google map and Gaode map), car-hailing apps (such as Grab and Uber), social apps (such as Facebook and Twitter), etc. These applications all need to acquire external service from an application server or cloud server by executing some computation task with the private data of users, as shown in Fig. 1. Hence, privacy protection for this kind of computation is one of the significant concerns that would affect the widespread adoption of MCC. It is necessary to extend the standard secure computation protocol from the homogeneous computing setting to the heterogeneous setting.

In this work, we construct an efficient server-aided secure two-party computation (2PC) protocol for MCC. Our work extends the work [13] which is a new paradigm to obtain an extremely efficient secure 2PC protocol in malicious model. We adopt the heterogeneous computation setting by replacing the original homogeneous computing setting with two stronger devices, namely the cloud server and application server which have more powerful computation and storage resources, and one weak mobile device. Unlike the most similar work [14] as shown in Table 1, considering that the data privacy can be better protected by not uploading them to the application server, we set the role of the mobile device as the garbled circuit evaluator and application server as the garbled circuit generator which can complete garbled circuit generation without being provided with the data. This setting is also more suitable for practical situations where mobile devices store the data and call external services that the application server provides. The main contributions of our work are concluded as follows:

1) We propose an efficient server-aided secure two-party computation protocol for MCC based on the garbled circuit. This work provides the solution for efficient secure two-party computation in heterogeneous computing setting, especially for the MCC.
2) This is the first work that considers collusion between the garbled circuit evaluator and server with guaranteeing privacy and correctness in the malicious model. Also, it can guarantee fairness when collusion does not exist in the malicious model.
3) We implement our protocol and evaluate it on the benchmark AES circuit. The experimental performance analysis shows that this work outperforms all the previous works at least 10 times with the same security level.

The rest of this paper is organized as follows: In Section 2, we review the related work on server-aided 2/MPC protocols. In Section 3, we provide preliminaries for this work. In Section 4, we provide system model, threat model and security goals. In Section 5, we provide the efficient server-aided 2PC protocol construction. In Section 6, we provide security analysis. In Section 7, we provide performance evaluation. In Section 8, we conclude this work.

2 Related Work

Since the 1980s, MPC has experienced a development process from theory to practice. Especially for 2PC (the special case of MPC), it has been increasingly practical with recent advances. Over the past thirty years, there are tremendous efficiency improvements for 2PC based on Yao's protocol in

TABLE 1
Comparison of Related Work in the Literature

Work	Task of Server	Num of Server	Num of Client Parties	Security Model	Collusion	Fairness
[KMR11]([15])	Circuit Evaluation	1	2/n	Semi-honest & Malicious	×	×
[KMR12]([16])	Circuit Evaluation	1	n	Semi-honest & Malicious	×	✓
[CMTB13]([17])	Circuit Evaluation	1	2	Malicious	×	✓
[CLT14]([14])	Circuit Generation	1	2	Malicious	$P_1\&Server$	✓
[JNO14]([18])	Function Evaluation	m	n	Malicious	$Servers(\times)$	×
[CMTB16]([19])	Aid Circuit Evaluation	1	2	Malicious	$P_1\&Server$	×
[BB16]([20])	Circuit Evaluation	1	2	Malicious	×	✓
[MOR16]([21])	Aid Circuit Generation	1	2	Semi-honest & Malicious	×	✓
[BPPS17]([22])	Circuit Evaluation	1	n	Semi-honest	×	×
This Work	Circuit Evaluation	1	2	Malicious	$P_2\&Server$	✓

Note: P_1 refers to the garbled circuit generator, and P_2 refers to the garbled circuit evaluator.

the semi-honest model like works [23], [24], [25], [26], [27], [28], [29], [30]. However, achieving maliciously secure for 2PC is far more difficult. Based on the Yao's semi-honest secure protocol, researchers propose cut-and-choose technique [31], [32] which is the classic technique to lift Yao's garbled circuit to work efficiently in the malicious model. Traditional cut-and-choose approaches operate at the circuit level, whereas the Large Efficient Garbled-circuit Optimization (LEGO) [33], MiniLEGO [34] and TinyLEGO [35] approaches further improve asymptotic and concrete efficiency by operating cut-and-choose at the gate level. Recently, Wang *et al.* [13] proposed a new paradigm to obtain an extremely efficient maliciously secure 2PC based on the highly optimized TinyOT protocol. The main idea of this work is to use the information-theoretic MAC tags to enable each of the two parties to generate one part of the authenticated garbled circuit. This can not only prevent the selective failure attack, but also integrate the advantages of both original MPC paradigms: the constant round for garbled circuit based approaches and low communication for secret sharing based approaches.

However, all of these maliciously secure 2PC works either have the significant overhead or set high configuration demands for devices (like multicore CPUs and high thread counts). They cannot be applied directly to the mobile devices that do not have such considerable computation resources. To further reduce the overhead of two parties and efficiently achieve 2PC, a series of works outsource some work originally belonged to two parties to the third party server (e.g., cloud server). This also provides us with ideas to achieve secure computation in heterogeneous mobile cloud computing setting.

Feige *et al.* [36] first added a trusted third party to propose a minimal extension of 2PC, where the communication pattern was minimal. Although the original motivation of [36] was not to reduce the clients' work at the expense of the server, the work did put forward new ideas for the follow-up works. Kamara *et al.* [15] first initiated the study of MPC in the server-aided setting, where the client parties can outsource some part of their work to the server. However, if the server colluded with a subset of the client parties, any generic server-aided MPC protocol can be reduced to a standard MPC protocol where the colluding party still did the linear size work of the circuit and remain parties did the sublinear work. However, the reduced standard MPC protocol can only be achieved by Fully Homomorphic Encryption (FHE) [37]. To minimize the computation of client parties and let the outsourced party's work to be sublinear or independent of the circuit size, Kamara *et al.* [15] introduced non-colluding adversaries and formalized the security definitions for server-aided MPC. Later, Kamara *et al.* [16] proposed two server-aided MPC protocols which were secure against covert and malicious adversaries, respectively. Both of these server-aided MPC protocols also achieved fairness without the assumption that majority of the parties are honest in standard MPC protocols [38]. This provided a new direction to achieve fairness for standard MPC protocols with a dishonest majority. Unfortunately, this work introduced the assumption that the client parties should have high bandwidth capability, which brought higher demands to today's mobile devices and wireless network technology. In addition, [16] required to execute a fair coin tossing protocol to share a secret key. To address the above problem, Carter *et al.* [17] introduced a new outsourced oblivious transfer primitive to construct a circuit evaluation outsourced server-aided 2PC protocol in malicious model. To further improve the security and efficiency of previous work, Carter *et al.* [14] designed a server-aided 2PC protocol with outsourcing the garbled circuits generation task. With changing the outsourcing task, they eliminated the most expensive public key cryptography operations and reduce the rounds of communication appeared in oblivious transfers. They also achieved the stronger security guarantee by allowing collusion between the server and original circuit generation party. However, since this protocol was based on the cut-and-choose approach, the efficiency of it can still be improved. It also considered fairness in all but one collusion scenario. Jakobsen *et al.* [18] designed server-aided MPC framework where a number of servers rather than a single one run the underlying standard MPC protocol for the client parties. However, it required the underlying standard MPC protocol to be reactive computation where private values can be opened in the protocol execution [19]. Blanton *et al.* [20] focused on genomic computation and proposed a server-aided 2PC outsourcing circuit evaluation scheme. It also provided the fairness property for secure computation and required non-collusion assumption like [16]. However, it introduced additional public key operations. Mohassel *et al.* [21] extended the [39] to the server-aided setting with non-collusion assumption. It divided the protocol execution into offline and online phases which makes mobile devices to execute the protocol flexibly with the changeable bandwidth. The server did some auxiliary computation for garbled circuit generation. However, the underlying 2PC scheme [39] was not efficient enough in today's computing setting, which made this protocol less efficient. Carter *et al.* [19] proposed a scheme to transform any secure 2PC protocol into server-aided 2PC protocol. It leveraged the non-collusion assumption to produce low-cost output consistency check. Although the computation and bandwidth required by the mobile device were reduced, the cost of the evaluation increased. Baldimtsi *et al.* [22] focused on the online social networks and designed a server-aided MPC protocol to utilize online social data of multiple parties. They designed a sub-protocol to transform inputs under different keys into ones under the same key. This enables the other $n-1$ parties do not need to be online all the time. It needed the non-collusion assumption among server and client parties and did not provide the fairness property.

In addition to the above garbled circuit based approaches, there are also some protocols designed with the homomorphic encryption. Loosely speaking, in these protocols all the client parties need to encrypt their data with the FHE scheme and upload the ciphertexts to the server, and then the server performs computation directly on these ciphertexts and returns the ciphertext results to the client parties. However, the main challenge for these FHE based protocols is that how different client parties can decrypt the result with different secret keys. Asharov *et al.* [40] addressed this by secret-sharing the secret key among all the participants. Lopez-Alt *et al.* [41] based on the multi-key FHE scheme designed the on-the-fly multiparty computation to enable the client parties to have their long-term public and secret key pairs. To further improve the efficiency, Peter *et al.* [42] based on the additively

α	β	γ
0	0	0
0	1	0
1	0	0
1	1	1

(a)

α	β	γ
$L_{\alpha,0}$	$L_{\beta,0}$	$L_{\gamma,0}$
$L_{\alpha,0}$	$L_{\beta,1}$	$L_{\gamma,0}$
$L_{\alpha,1}$	$L_{\beta,0}$	$L_{\gamma,0}$
$L_{\alpha,1}$	$L_{\beta,1}$	$L_{\gamma,1}$

(b)

α	β	γ
$L_{\alpha,0}$	$L_{\beta,0}$	$E_{L_{\alpha,0},L_{\beta,0}}(L_{\gamma,0})$
$L_{\alpha,0}$	$L_{\beta,1}$	$E_{L_{\alpha,0},L_{\beta,1}}(L_{\gamma,0})$
$L_{\alpha,1}$	$L_{\beta,0}$	$E_{L_{\alpha,1},L_{\beta,0}}(L_{\gamma,0})$
$L_{\alpha,1}$	$L_{\beta,1}$	$E_{L_{\alpha,1},L_{\beta,1}}(L_{\gamma,1})$

(c)

γ
$E_{L_{\alpha,0},L_{\beta,1}}(L_{\gamma,0})$
$E_{L_{\alpha,1},L_{\beta,1}}(L_{\gamma,1})$
$E_{L_{\alpha,1},L_{\beta,0}}(L_{\gamma,0})$
$E_{L_{\alpha,0},L_{\beta,0}}(L_{\gamma,0})$

(d)

$L_{\gamma,0/1}$	γ
$L_{\gamma,0}$	0
$L_{\gamma,0}$	0
$L_{\gamma,0}$	0
$L_{\gamma,1}$	1

(e)

Fig. 2. Garbled circuit: (a) AND gate truth table; (b) AND gate with labels; (c) garbled AND gate; (d) garbled table; (e) output mapping table.

homomorphic encryption proposed two-server-aided multi-party computation scheme. However, the efficiency of these works was based on the underlying FHE scheme. Thus, it is still an open problem for constructing practical and efficient FHE schemes nowadays.

We summarize the main difference between our work and the previous works in Table 1. We conclude that our work is *the first work* that considers collusion between the garbled circuit evaluator and server with guaranteeing privacy and correctness in the malicious model. Also, it can guarantee fairness when collusion does not exist in the malicious model.

3 Preliminaries

3.1 Garbled Circuit

Garbled Circuit (GC) is the key technology in constructing generic 2PC protocol. It permits two parties P_1 and P_2 with their private inputs x and y respectively to securely compute any function $f(x, y)$ represented as the boolean circuit C_f. At a high level the garbled circuit protocol works as follows:

1) Based on the boolean circuit C_f, P_1 constructs the corresponding garbled circuit GC_f: it selects two secret keys $L_{w,0}$ and $L_{w,1}$ for every wire w in the circuit C_f as two labels of wire w to replace the true value 0 and 1, respectively. We take an AND gate for example as shown in Fig. 3. P_1 replaces the value of truth table for the AND gate with the labels $L_{w,0}$ and $L_{w,1}$ it chooses, as shown in Figs. 2a and 2b. For the output wire γ it uses the double-key symmetric encryption $E_{k_1,k_2}(m)$ to encrypt the label $L_{\gamma,0}$ and $L_{\gamma,1}$with the labels $L_{\alpha,0/1}$ and $L_{\beta,0/1}$ of input wires α and β, and generates the new truth table as shown in Fig. 2c. Finally, it randomly permutes the table to avoid leaking the information from the row and generates the garbled table as shown in Fig. 2d for one AND gate. After this, it sends the garbled tables for all the gates in the circuit C_f and the label of its input $L_{\alpha,x}$to the P_2.
2) P_2 runs 1-out-of-2 Oblivious Transfer protocol with P_1 so that $P2$ can get the label $L_{\beta,y}$ of its input y without leaking y to the P_1. Also, this enables P_1 not to leak another label $L_{\beta,\bar{y}}$ of the input wire to P_2.
3) Based on the two input wire labels $(L_{\alpha,x}, L_{\beta,y})$ and garbled tables for all the gates of the circuit C_f, P_2 evaluates the circuit gate-by-gate by decrypting the correct row of every garbled table for every gate, and it ends this decryption operation when he gets the final output wire label $L_{O,0/1}$.
4) To recover the output, either P_2 sends this label $L_{O,0/1}$ to the P_1, and P_1 outputs the real output z to P_2 based on the mapping table shown as Fig. 2e. Or P_1 sends the mapping table to P_2, and let P_2 know the real output z and send z to P_1.

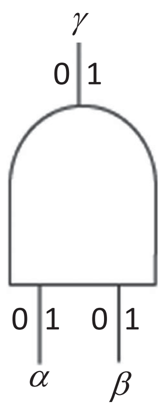

Fig. 3. AND gate.

3.2 Information-Theoretic MAC Tags

In the following, we would like to recall a brief summary on information-theoretic MAC tags, which is one of basic blocks for the protocol [13].

Let P_1 holds the random uniformly global key $\Delta_1 \in \{0,1\}^\kappa$ and a uniform key $K[s]$. At the same time P_2 holds the bit s and the Mac tag $M[s] := K[s] \oplus s\Delta_1$. Such that when P_2 sends pair $(s, M[s])$ to P_1, it can verify whether $M[s]$ equals to the $M[s]'$ generated with $(s, K[s], \Delta_1)$. If so, we denote $[s]_2$ as the authenticated bit s known to P_2 (i.e., P_2 holds $(s, M[s])$ and P_1 holds $K[s]$). Similarly, for the authenticated bit $[r]_1$, P_1 can verify its validity by sending the pair $(r, M[r])$ to P_2 who has the key $K[r]$, so that P_2 can verify the triple $(r, M[r], K[r])$. The important property of the above MAC tags is XOR-homomorphic. That is to say, if P_1 holds two authenticated pairs $(a, M[a])$ and $(b, M[b])$ while P_2 holds the corresponding keys $K[a]$ and $K[b]$, P_1 can generate the authenticated bit $[a \oplus b]_1$ by letting P_1 locally xor the pairs $(a \oplus b, M[a] \oplus M[b])$ and P_2 locally xor the key $K[a] \oplus K[b]$.

In this paper, the same as in the protocol [13], we utilize this information-theoretic MAC tags to authenticate the secret shared bits for every wire of the circuit. So that the malicious behaviors like values corruption and improper computation can be prevented by checking the MAC tags.

3.3 The Functionality F_{PRE}

Based on the above information-theoretic MAC tags, we use the functionality F_{PRE} of [13] as a critical component for our protocol and briefly recall it as follows. This functionality is used to set up the authenticated values on each wire of the circuit for P_1 and P_2. It is only executed with two parties P_1 and P_2, and there does not exist a third party to assist this work. It is an optimized version of the TinyOT protocol [39] based on the oblivious transfer technique.

There are three functions of the functionality F_{PRE} and we summarize them as follows:

1) Choose uniformly global key for both parties: both parties send $init$ to the F_{PRE} so that F_{PRE} returns global key Δ_1 to P_1 and Δ_2 to P_2.
2) Generate random authenticated bit shares: both parties send $random$ to the F_{PRE} so that F_{PRE} returns $(r, M[r], K[s])$ to P_1 and $(s, M[s], K[r])$ to P_2, where $\lambda = r \oplus s$.
3) Generate the authenticated secret shares of an AND gate: P_1 sends $(AND, (r_1, M[r_1], K[s_1]), (r_2, M[r_2], K[s_2]))$ to F_{PRE}, while P_2 sends $(AND, (s_1, M[s_1], K[r_1]), (s_2, M[s_2], K[r_2]))$ to F_{PRE}. The F_{PRE} first verifies $M[r_i] \stackrel{?}{=} K[r_i] \oplus r_i\Delta_1$ and $M[s_i] \stackrel{?}{=} K[s_i] \oplus s_i\Delta_2$ for $i \in \{1, 2\}$. If so, F_{PRE} continues; otherwise, it sends $cheat$ to both parties. If F_{PRE} continues, it defines $\lambda_3 = r_3 \oplus s_3$ and sets $r_3 = s_3 \oplus ((r_1 \oplus s_1) \wedge (r_2 \oplus s_2))$. Also, it sets $M[r_3] = K[r_3] \oplus r_3\Delta_2$ and $M[s_3] = K[s_3] \oplus s_3\Delta_1$. Finally, the F_{PRE} returns $(r_3, M[r_3], K[s_3])$ to P_1 and $(s_3, M[s_3], K[r_3])$ to P_2.

For the third function which is the key component of F_{PRE} is to generate the authenticated secret shared values $[x_1]_1$, $[x_2]_2$, $[y_1]_1$, $[y_2]_2$, $[z_1]_1$, and $[z_2]_2$ for an AND gate, such that $z_1 \oplus z_2 = (x_1 \oplus x_2) \wedge (y_1 \oplus y_2)$. To achieve this goal, it designs three sub-functions: (1) F_{abit}: Generate the authenticated random bit on the wire with the corresponding global key. This is the same as the above second function; (2) F_{HaAND}: Invoke the F_{abit} to generate the authenticated secret shared bit triple $[x_1]_1$ and $[x_2]_2$, and then generate the secret shares of $x_1y_2 \oplus x_2y_1$; (3) F_{LaAND}: Invoke the F_{abit} to generate the $[y_1]_1$ and $[z_1]_1$ for P_1 and $[y_2]_2$ for P_2, and invoke the F_{HaAND} to generate $[x_1]_1$ and $[x_2]_2$ and secret shares of $x_1y_2 \oplus x_2y_1$. Then, P_1 and P_2 locally computes x_1y_1 and x_2y_2 respectively. Finally, they can get the secret shares of $(x_1 \oplus x_2) \wedge (y_1 \oplus y_2)$ locally. For more details about how the functionality F_{PRE} is constructed with the oblivious transfer technique, we refer you to the work [13].

4 Problem Statement

4.1 System Model

Our system consists of three entities: the $Server$, the two client parties P_1 and P_2, as shown in Fig. 4. Their roles in our protocol are summarized as follows:

- $Server$: It is the third party in this protocol to assist the circuit evaluation work for client party P_2 for securely computing some function f. It has large-scale computation resources, such that it can be the cloud server in the MCC setting.
- P_1: It is one of the two client parties that takes one part of the garbled circuit generation task. It has less or equal computation resources compared with $Server$, namely, it can be the application server in the MCC setting.

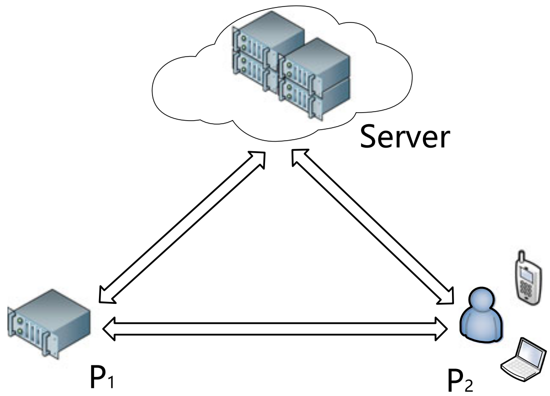

Fig. 4. System model.

- P_2: It is the other client party that takes the other part of garbled circuit generation task. Since it has limited computation resources, it employs $Server$ to assist the circuit evaluation task in standard 2PC. It can be the mobile device in the MCC setting.

4.2 Threat Model and Security Goals

4.2.1 Threat Model

In this paper, we can guarantee the security of protocol against the following adversary structure ADV:

1) Any one of the two client parties is malicious and cannot collude with the other semi-honest client party and semi-honest $Server$.
2) P_2 is malicious and can collude with semi-honest $Server$, while P_1 is honest.

$$ADV = \begin{cases} (P_1[m_{nc}], P_2[s], S[s]) \\ (P_1[s], P_2[m_{nc}], S[s]) \\ (P_1[h], P_2[m_c], S[s_c]) \end{cases},$$

where m_{nc} refers to malicious and non-collude, s refers to semi-honest, h refers to honest, m_c refers to malicious and collude, and s_c refers to semi-honest and collude.

4.2.2 Security Goals

1) *Privacy*: Any one of the two client parties cannot learn any information (including the private input of the other client party) from the protocol execution other than its computation output and what is inherently leaked from it. $Server$ cannot learn any information (including the private inputs of two client parties and the computation output) from the protocol execution.
2) *Correctness*: The two client parties should get the correct output of the computation for function f. $Server$ correctly executes computation operations.
3) *Fairness*: If any one of the two client parties gets the computation output, then the other client party does.

Note that we cannot guarantee fairness in $(P_1[h], P_2[m_c], S[s_c])$ setting. Since the $Server$ and P_2 can collude together in this setting, we cannot prevent P_2 from getting the computation output from $Server$ before P_1. Also, we cannot prevent P_2 from controlling $Server$ to abort the protocol before P_1 receiving the output. Actually, in this setting, our server-aided

2PC protocol can be reduced into the 2PC protocol which inherently cannot achieve fairness in malicious model [43].

To guarantee privacy: (1) For the two client parties, since the garbled circuit technique can withstand the malicious behavior of the circuit evaluation party (i.e., the combination of *Server* and $P2$), we mainly focus on preventing the selective failure attack launched by the malicious circuit generation party (i.e., P_1). This attack specifically refers to that a malicious circuit generation party can use inconsistent labels in garbled circuit generation and oblivious transfer, so that the private input of the circuit evaluation party would be leaked to the circuit generation party based on whether or not the protocol aborts. To prevent this attack, we utilize the authenticated garbled circuit secret sharing technique from Wang *et al.* [13] to let both client parties P_1 and P_2 generate one part of the garbled table. So that the input of P_2 is independent of the part of garbled table generated by P_1, and the malicious party P_1 cannot get the private input of P_2 by launching this attack. (2) For *Server*, we construct the protocol with the advantage of enabling all the values transferred to it are random and the result it computed is masked. Therefore, it cannot get any information from the protocol execution, either the private inputs of two client parties, or the computation result.

To guarantee correctness: (1) For the two client parties, we need to ensure that the malicious client party cannot replace the values for any of the internal wires in the circuit, which would lead to the incorrect result of the computation for function f. To address this, we use the information-theoretic MAC tags and the functionality F_{PRE} of [13], so that both client parties can verify whether or not the internal values are correctly constructed for their respective garbled tables. (2) For *Server*, we set *Server* to be semi-honest, which provides the correctness guarantee for its computation.

To guarantee fairness: we need to prevent any one of the malicious client parties from aborting the protocol once it receives the output so that the other client party cannot get the output. To cope with this, we let *Server* simultaneously release the masked output to both client parties based on whether the verification of values on the output wires succeeds or not. If so, the two client parties can decrypt the output directly; Otherwise, they cannot get the masked output, let alone the actual output. As mentioned above, we only provide fairness guarantee when *Server* does not collude with any client party.

5 Efficient Server-Aided 2PC Protocol

5.1 Overview

To complete the secure computation on function $f(x,y) \to z$, two client parties P_1 and P_2 with their private input x and y respectively, first reach a consensus on the boolean circuit C_f representing the evaluated function f. Then, both client parties execute the protocol of circuit preprocessing and input processing phase and provide values that *Server* needs for the circuit evaluation. Then, *Server* executes protocol of the circuit evaluation and outputs distribution phase. Finally, if the two-round verifications between the *Server* and two client parties are passed, the two client parties can get the actual output; Otherwise, the protocol aborts. The high-level idea of the four phases of the protocol is provided as follows:

1) *Circuit preprocessing phase:* The two client parties P_1 and P_2 first get their own MAC keys Δ_1 and Δ_2, respectively. Then, P_1 and P_2 preprocess the circuit C_f to generate the corresponding values for each wire of the circuit C_f based on the functionality F_{PRE}. Finally, they upload the corresponding values of all AND gates to *Server* for the subsequent circuit evaluation.
2) *Input processing phase:* The two client parties P_1 and P_2 both check whether each other provides the correct shared mask bits for input wires of the other party. If so, P_1 sends *Server* the masked inputs and the corresponding labels for its own input wires; P_2 sends *Server* the masked inputs and let P_1 send the corresponding labels for its input wires to *Server*.
3) *Circuit evaluation phase:* With the masked inputs and labels of both client parties P_1 and P_2, *Server* follows the logical topology of circuit C_f to compute the masked outputs and labels for the output wires of circuit C.
4) *Outputs distribution phase:* The two client parties P_1 and P_2 both check whether each other provides the correct shared mask bits for the output wries. If so, *Server* sends the masked outputs for all output wires of circuit C_f to P_1 and P_2. Finally, P_1 and P_2 locally recover the actual output.

5.2 Server-Aided 2PC Protocol

Inputs: The two client parties P_1 and P_2 agree on the circuit C_f which represents the evaluated function $f: \{0,1\}^{|I_1|} \times \{0,1\}^{|I_2|} \to \{0,1\}^{|O|}$. P_1 has its own private input $x \in \{0,1\}^{|I_1|}$ and P_2 has its own private input $y \in \{0,1\}^{|I_2|}$. We denote I_1 as the index set of the input wires for P_1, I_2 as the index set of the input wires for P_2, O as the index set of the output wires for circuit C_f, and W as the index set of the output wires for all AND gates in circuit C_f. Also, we define the computational security parameter as κ. As shown in Fig. 5, our protocol works as follows:

Phase 1: Circuit preprocessing

1) P_1 and P_2 both send $init$ to functionality F_{PRE} which respectively returns Δ_1 and Δ_2 to them.
2) For wire $w \in I_1 \cup I_2 \cup W$:
   P_1 and P_2 send $random$ to F_{PRE}. Then, F_{PRE} returns authenticated shared mask bit triple $(r_w, M[r_w], K[s_w])$ to P_1 and $(s_w, M[s_w], K[r_w])$ to P_2, where the mask bit $\lambda_w = s_w \oplus r_w$. P_1 then sets the label for bit 0 of wire w as $L_{w,0} \in \{0,1\}^{|\kappa|}$ and the label for bit 1 of wire w as $L_{w,1} := L_{w,0} \oplus \Delta_1$.
3) For each XOR gate $G = (\alpha, \beta, \gamma, \oplus)$ with two input wires α, β and one output wire γ:
   P_1 locally computes the authenticated shared mask bit triple $(r_\gamma, M[r_\gamma], K[s_\gamma]) := (r_\alpha \oplus r_\beta, M[r_\alpha] \oplus M[r_\beta], K[s_\alpha] \oplus K[s_\beta])$. P_2 locally computes the authenticated shared mask bit triple $(s_\gamma, M[s_\gamma], K[r_\gamma]) := (s_\alpha \oplus s_\beta, M[s_\alpha] \oplus M[s_\beta], K[r_\alpha] \oplus K[r_\beta])$. Then, P_1 sets the labels on that wire to be $L_{\gamma,0} := L_{\alpha,0} \oplus L_{\beta,0}$ and $L_{\gamma,1} := L_{\gamma,0} \oplus \Delta_1$. Define the mask bit $\lambda_\gamma = \lambda_\alpha \oplus \lambda_\beta$ for the output wire γ.
4) For each AND gate $G = (\alpha, \beta, \gamma, \wedge)$ with two input wires α, β and one output wire γ:
   (a) P_1 sends $(and, (r_\alpha, M[r_\alpha], K[s_\alpha]), (r_\beta, M[r_\beta], K[s_\beta]))$ to F_{PRE}, while P_2 sends $(and, (s_\alpha, M[s_\alpha], K[r_\alpha]),$

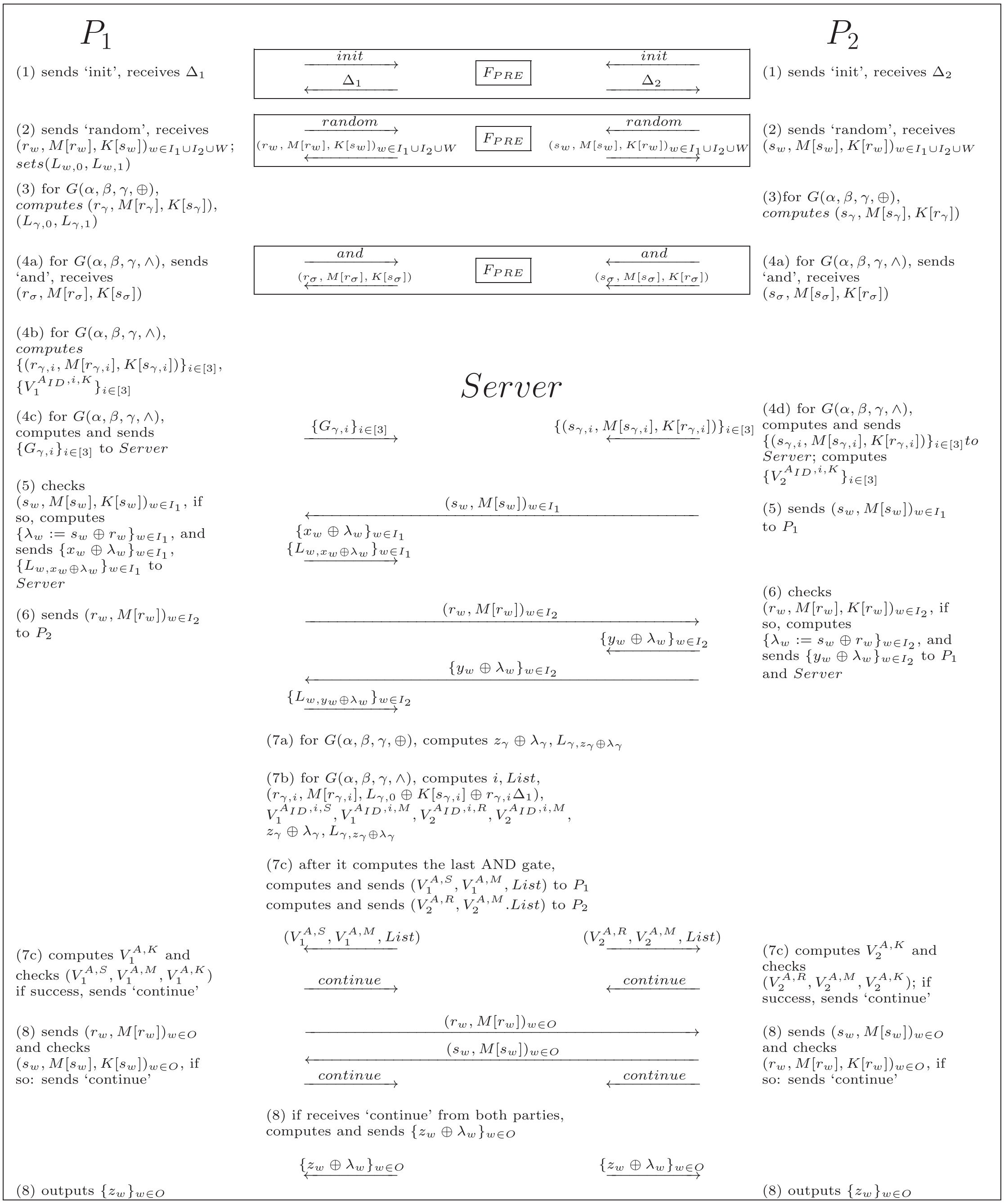

Fig. 5. The efficient server-aided 2PC protocol.

$(s_\beta, M[s_\beta], K[r_\beta]))$ to F_{PRE}. Then, P_1 receives $(r_\sigma, M[r_\sigma], K[s_\sigma])$ and P_2 receives $(s_\sigma, M[s_\sigma], K[r_\sigma])$ from F_{PRE}, where $r_\sigma \oplus s_\sigma = \lambda_\alpha \wedge \lambda_\beta$.

(b) P_1 locally computes the set $\{(r_{\gamma,i}, M[r_{\gamma,i}], K[s_{\gamma,i}])\}_{i\in[3]}$ as follows:

$$
\begin{aligned}
&r_{\gamma,0}, M[r_{\gamma,0}], K[s_{\gamma,0}]\\
&r_{\gamma,1}, M[r_{\gamma,1}], K[s_{\gamma,1}]\\
&r_{\gamma,2}, M[r_{\gamma,2}], K[s_{\gamma,2}]\\
&r_{\gamma,3}, M[r_{\gamma,3}], K[s_{\gamma,3}]
\end{aligned}
$$

where $r_{\gamma,i}$, $M[r_{\gamma,i}]$, and $K[s_{\gamma,i}]$ for $i \in [3]$ are respectively computed as follows:

$$
\begin{cases}
r_{\gamma,0} := r_\sigma \oplus r_\gamma\\
r_{\gamma,1} := r_\sigma \oplus r_\gamma \oplus r_\alpha\\
r_{\gamma,2} := r_\sigma \oplus r_\gamma \oplus r_\beta\\
r_{\gamma,3} := r_\sigma \oplus r_\gamma \oplus r_\alpha \oplus r_\beta
\end{cases}
$$

$$
\begin{cases}
M[r_{\gamma,0}] := M[r_\sigma] \oplus M[r_\gamma]\\
M[r_{\gamma,1}] := M[r_\sigma] \oplus M[r_\gamma] \oplus M[r_\alpha]\\
M[r_{\gamma,2}] := M[r_\sigma] \oplus M[r_\gamma] \oplus M[r_\beta]\\
M[r_{\gamma,3}] := M[r_\sigma] \oplus M[r_\gamma] \oplus M[r_\alpha] \oplus M[r_\beta]
\end{cases}
$$

$$
\begin{cases}
K[s_{\gamma,0}] := K[s_\sigma] \oplus K[s_\gamma]\\
K[s_{\gamma,1}] := K[s_\sigma] \oplus K[s_\gamma] \oplus K[s_\alpha]\\
K[s_{\gamma,2}] := K[s_\sigma] \oplus K[s_\gamma] \oplus K[s_\beta]\\
K[s_{\gamma,3}] := K[s_\sigma] \oplus K[s_\gamma] \oplus K[s_\alpha] \oplus K[s_\beta] \oplus \Delta_1
\end{cases}
$$

Then, P_1 computes $\{V_1^{A_{ID},i,K}\}_{i\in[3]} = \{K[s_{\gamma,i}]\}_{i\in[3]}$, where A_{ID} is the index of the AND gate in the circuit C.

(c) P_1 then computes the set $\{G_{\gamma,i}\}_{i\in[3]}$ as below and sends the set to $Server$:

$$G_{\gamma,0} := H(L_{\alpha,0}, L_{\beta,0}, \gamma, 0) \oplus (r_{\gamma,0}, M[r_{\gamma,0}], L_{\gamma,0} \oplus K[s_{\gamma,0}] \oplus r_{\gamma,0}\Delta_1)$$
$$G_{\gamma,1} := H(L_{\alpha,0}, L_{\beta,1}, \gamma, 1) \oplus (r_{\gamma,1}, M[r_{\gamma,1}], L_{\gamma,0} \oplus K[s_{\gamma,1}] \oplus r_{\gamma,1}\Delta_1)$$
$$G_{\gamma,2} := H(L_{\alpha,1}, L_{\beta,0}, \gamma, 2) \oplus (r_{\gamma,2}, M[r_{\gamma,2}], L_{\gamma,0} \oplus K[s_{\gamma,2}] \oplus r_{\gamma,2}\Delta_1)$$
$$G_{\gamma,3} := H(L_{\alpha,1}, L_{\beta,1}, \gamma, 3) \oplus (r_{\gamma,3}, M[r_{\gamma,3}], L_{\gamma,0} \oplus K[s_{\gamma,3}] \oplus r_{\gamma,3}\Delta_1)$$

(d) P_2 locally computes the set $\{(s_{\gamma,i}, M[s_{\gamma,i}], K[r_{\gamma,i}])\}_{i\in[3]}$ as below and sends the set to $Server$:

$$s_{\gamma,0}, M[s_{\gamma,0}], K[r_{\gamma,0}]$$
$$s_{\gamma,1}, M[s_{\gamma,1}], K[r_{\gamma,1}]$$
$$s_{\gamma,2}, M[s_{\gamma,2}], K[r_{\gamma,2}]$$
$$s_{\gamma,3}, M[s_{\gamma,3}], K[r_{\gamma,3}]$$

where $s_{\gamma,i}$, $M[s_{\gamma,i}]$, and $K[r_{\gamma,i}]$ for $i \in [3]$ are computed as follows:

$$\begin{cases} s_{\gamma,0} := s_\sigma \oplus s_\gamma \\ s_{\gamma,1} := s_\sigma \oplus s_\gamma \oplus s_\alpha \\ s_{\gamma,2} := s_\sigma \oplus s_\gamma \oplus s_\beta \\ s_{\gamma,3} := s_\sigma \oplus s_\gamma \oplus s_\alpha \oplus s_\beta \oplus 1 \end{cases}$$

$$\begin{cases} M[s_{\gamma,0}] := M[s_\sigma] \oplus M[s_\gamma] \\ M[s_{\gamma,1}] := M[s_\sigma] \oplus M[s_\gamma] \oplus M[s_\alpha] \\ M[s_{\gamma,2}] := M[s_\sigma] \oplus M[s_\gamma] \oplus M[s_\beta] \\ M[s_{\gamma,3}] := M[s_\sigma] \oplus M[s_\gamma] \oplus M[s_\alpha] \oplus M[s_\beta] \end{cases}$$

$$\begin{cases} K[r_{\gamma,0}] := K[r_\sigma] \oplus K[r_\gamma]) \\ K[r_{\gamma,1}] := K[r_\sigma] \oplus K[r_\gamma] \oplus K[r_\alpha]) \\ K[r_{\gamma,2}] := K[r_\sigma] \oplus K[r_\gamma] \oplus K[r_\beta]) \\ K[r_{\gamma,3}] := K[r_\sigma] \oplus K[r_\gamma] \oplus K[r_\alpha] \oplus K[r_\beta]) \end{cases}$$

Also, P_2 computes $\{V_2^{A_{ID},i,K}\}_{i\in[3]} = \{K[r_{\gamma,i}]\}_{i\in[3]}$.

Phase 2 : Inputs processing

5) For each wire $w \in I_1$:
P_2 sends $(s_w, M[s_w])$ to P_1 who checks whether the triple $(s_w, M[s_w], K[s_w])$ is valid. If so, P_1 recovers the mask bit $\lambda_w := s_w \oplus r_w$ and sends both the masked input $x_w \oplus \lambda_w$ and the label $L_{w,x_w\oplus\lambda_w}$ to $Server$.

6) For each wire $w \in I_2$:
P_1 sends $(r_w, M[r_w])$ to P_2 who checks whether the triple $(r_w, M[r_w], K[r_w])$ is valid. If so, P_2 recovers the mask bit $\lambda_w := s_w \oplus r_w$. Then, it sends the masked input $y_w \oplus \lambda_w$ to P_1 and $Server$. Finally, P_1 sends the corresponding label $L_{w,y_w\oplus\lambda_w}$ to $Server$.

Phase 3 : Circuit evaluation

7) $Server$ evaluates the circuit following the logical topology of circuit C_f. For each gate $G = (\alpha, \beta, \gamma, T)$ with two input wires α and β, and the output wire γ, $Server$ holds the two tuples $(z_\alpha \oplus \lambda_\alpha, L_{\alpha,z_\alpha\oplus\lambda_\alpha})$ and $(z_\beta \oplus \lambda_\beta, L_{\beta,z_\beta\oplus\lambda_\beta})$ received from P_1, where z_α and z_β are the actual values of the wires.

(a) If $T = \oplus$, for the output wire γ: $Server$ computes the masked output $z_\gamma \oplus \lambda_\gamma := (z_\alpha \oplus \lambda_\alpha) \oplus (z_\beta \oplus \lambda_\beta)$ and the corresponding label $L_{\gamma,z_\gamma\oplus\lambda_\gamma} := L_{\alpha,z_\alpha\oplus\lambda_\alpha} \oplus L_{\beta,z_\beta\oplus\lambda_\beta}$.

(b) If $T = \wedge$, for the output wire γ: $Server$ first computes $i := 2(z_\alpha \oplus \lambda_\alpha) \oplus (z_\beta \oplus \lambda_\beta)$ and puts i into the index list $List := \{A_{ID}, i\}$. Then, it recovers $(r_{\gamma,i}, M[r_{\gamma,i}], L_{\gamma,0} \oplus K[s_{\gamma,i}] \oplus r_{\gamma,i}\Delta_1) := G_{\gamma,i} \oplus H(L_{\alpha,z_\alpha\oplus\lambda_\alpha}, L_{\beta,z_\beta\oplus\lambda_\beta}, \gamma, i)$. Then, it sets $V_1^{A_{ID},i,S} := s_{\gamma,i}$, $V_1^{A_{ID},i,M} := M[s_{\gamma,i}]$, $V_2^{A_{ID},i,R} := r_{\gamma,i}$ and $V_2^{A_{ID},i,M} := M[r_{\gamma,i}]$. Then, $Server$ computes the masked output $z_\gamma \oplus \lambda_\gamma := (s_{\gamma,i} \oplus r_{\gamma,i})$ and the corresponding label $L_{\gamma,z_\gamma\oplus\lambda_\gamma} := (L_{\gamma,0} \oplus K[s_{\gamma,i}] \oplus r_{\gamma,i}\Delta_1) \oplus M[s_{\gamma,i}]$.

(c) After $Server$ computes the last AND gate, it computes the following values: $V_1^{A,S} := \oplus_{A_{ID}\in A} V_1^{A_{ID},i,S}$, $V_1^{A,M} := \oplus_{A_{ID}\in A} V_1^{A_{ID},i,M}$, $V_2^{A,R} := \oplus_{A_{ID}\in A} V_2^{A_{ID},i,R}$, $V_2^{A,M} := \oplus_{A_{ID}\in A} V_2^{A_{ID},i,M}$. Then it sends $(V_1^{A,S}, V_1^{A,M}, List)$ to P_1 for check. P_1 first computes $V_1^{A,K} := \oplus_{A_{ID}\in A} V_1^{A_{ID},i,K}$ based on the $List$, and checks $(V_1^{A,S}, V_1^{A,M}, V_1^{A,K})$. If the verification successes, P_1 returns *continue* to $Server$; Otherwise, it returns *abort*. Similarly, $Server$ also sends $(V_2^{A,R}, V_2^{A,M}, List)$ to P_2 for check. P_2 first computes $V_2^{A,K} := \oplus_{A_{ID}\in A} V_2^{A_{ID},i,K}$ based on the $List$, and checks $(V_2^{A,R}, V_2^{A,M}, V_2^{A,K})$. If the verification successes, P_2 returns *continue* to $Server$; Otherwise, it returns *abort*. If $Server$ receives *continue* both from P_1 and P_2, it continues; Otherwise, it aborts.

Phase 4 : Outputs distribution

8) For each wire $w \in O$:
P_1 sends $(r_w, M[r_w])$ to P_2 and P_2 checks whether $(r_w, M[r_w], K[r_w])$ is valid. Similarly, P_2 sends $(s_w, M[s_w])$ to P_1 and P_1 checks whether $(s_w, M[s_w], K[s_w])$ is valid. If any one of the two verifications fails, the party sends *abort* to $Server$. Otherwise, $Server$ sends the masked output $z_w \oplus \lambda_w$ to both P_1 and P_2. Then P_1 and P_2 can recover the final actual output $z_w := (z_w \oplus \lambda_w) \oplus r_w \oplus s_w$

6 Security Analysis

6.1 Security Definition

We follow the security definition first formally provided by Kamara *et al.* [16] and first specified in two-party case by Carter *et al.* [17]. We summarize the definition here and suggest readers to the previous works for a more formal and complete definition.

In the real-model execution, the protocol is executed by three parties: two client parties P_1 and P_2, and one $Server$. P_1 and P_2 respectively provide the computation input x_i, auxiliary input z_i and random coins r_i, where $i \in \{1, 2\}$. $Server$ only provides the auxiliary input z_3 and random coins r_3. We assume that the $Server$ should not collude with party P_1 defined by [15]. There exists some subset of three independent malicious adveraries

$\{A_1, A_2, A_3\}$. Each adversary A_i of the subset can corrupt one participant party P_i. For honest party P_i, let OUT_i be the output of P_i. For the corrupted party P_i, let OUT_i be the view of the protocol for P_i. The ith partial output of a real-model execution is defined as follows:

$$REAL^{(i)}(k, x, r) = \{OUT_j : j \in H\} \cup OUT_i,$$

where k is the security parameter, $x = \{x_1, x_2\}$ is the set of computation inputs for all parties, $r = \{r_1, r_2, r_3\}$ is the set of random coins for all parties, and H is the set of honest parties.

In the ideal-model execution, there exist four parties: two client parties P_1 and P_2, one $Server$, and one trusted third party. The first three parties provide their inputs to the trusted third party. In particular, P_1 and P_2 respectively provides the computation input x_i, auxiliary input z_i and random coins r_i, where $i \in \{1, 2\}$; $Server$ provides the auxiliary input z_3 and random coins r_3. Once receiving these inputs, the trusted party evaluates the predefined function f and returns the output to P_1 and P_2. Note that $Server$ has no output because it does not provide the computation input of the function f to the trusted third party. If the party is honest or semi-honest, it provides the real input; While if the party is malicious, it provides the arbitrary input rather than the real one. If any party aborts early and refuses to send the input, the trusted third party will abort immediately and send no output. For honest party P_i, let OUT_i be the output of P_i from the trusted third party. For the corrupted party P_i, let OUT_i be the arbitrary value generated by P_i itself. The ith partial output of an ideal-model execution in the presence of independent malicious simulators $S = \{S_1, S_2, S_3\}$ is defined as follows:

$$IDEAL^{(i)}(k, x, r) = \{OUT_j : j \in H\} \cup OUT_i,$$

where the parameter k, x, r, H are the same as defined in the real-model execution.

Based on this real/ideal-model, the formal security definition is provided as follows:

Definition 1. *A server-aided protocol can securely compute the function f if there exists a set of probabilistic polynomial-time (PPT) simulators $\{Sim_i\}_{i\in[3]}$ such that all PPT adversaries $\{A_i\}_{i\in[3]}$, computation inputs x and auxiliary inputs z, for all $i \in [3]$:*

$$REAL^{(i)}(k, x, r)_{k\in N} \overset{c}{\approx} IDEAL^{(i)}(k, x, r)_{k\in N}.$$

Where $S = \{S_1, S_2, S_3\}$, $S_i = Sim_i(A_i)$ and r is chosen uniformly at random.

We also specialize the lemma in [15] for 3 parties that we will use for proofs as follows:

Lemma 1. *If a multi-party protocol among 3 parties $\{P_1, P_2, P_3\}$, securely computes function f, in presence of (1) semi-honest and independent parties and (2) a malicious party P_j and honest parties $\{P_i\}$ for $i \in \{1, 2, 3\}/\{j\}$, then the multi-party protocol is also secure in presence of a malicious party P_j with all the other semi-honest parties.*

6.2 Security Proofs

Based on the above security definition, we provide proofs for the following theorem.

Theorem 1. *The efficient server-aided two-party protocol securely computes a function $f(x, y) \to z$ to against the following adversary structure ADV: (1) Any one of the two client parties is malicious and cannot collude with the other semi-honest client party and semi-honest $Server$. (2) P_2 is malicious and can collude with semi-honest $Server$, while P_1 is honest.*

$$ADV = \begin{cases} (P_1[m_{nc}], P_2[s], S[s]) \\ (P_1[s], P_2[m_{nc}], S[s]) \\ (P_1[h], P_2[m_c], S[s_c]) \end{cases},$$

where m_{nc} refers to malicious and non-collude, s refers to semi-honest, h refers to honest, m_c refers to malicious and collude, and s_c refers to semi-honest and collude.

As the security definition in the Section 6.1 is based on the real/ideal model paradigm, we need to provide the security proofs to prove that the joint output distribution of the adversary and honest parties of the protocol for real-model execution and that for the ideal-model execution are indistinguishable. Since the security goals of the protocol are guaranteed in the ideal-model execution with a trusted third party, the indistinguishable result of the real-model and ideal-model executions indicates that the protocol for real-model execution can provide the same security guarantees as ideal-model execution.

To prove the first setting in Theorem 1: First, we prove the condition (1) of Lemma 1 which involves two cases: $(P_1[s], P_2[h], S[h])$ and $(P_1[h], P_2[s], S[h])$; Second, we prove the condition (2) of Lemma 1 respectively, namely $(P_1[m_{nc}], P_2[h], S[h])$ and $(P_1[h], P_2[m_{nc}], S[h])$; Finally, with the above proofs and Lemma 1, we reach the conclusion that our protocol securely computes function f in the first setting of Theorem 1, namely $(P_1[m_{nc}], P_2[s], S[s])$ and $(P_1[s], P_2[m_{nc}], S[s])$.

To prove the second setting in Theorem 1: we reduce it to the based maliciously secure 2PC protocol and provide the corresponding description.

6.2.1 Semi-Honest Party P_1 or P_2

(1) Semi-honest party P_1 $(P_1[s], P_2[h], S[h])$:

In this setting, P_1 is semi-honest and follows the protocol while P_2 and $Server$ are both honest. We construct simulator S_1 that runs A_1 as a subroutine and plays the role of P_1 interacting with the third trusted party. S_1 receives the input of P_1 and sends it to the trusted third party. Then, the third party computes the output of function f: $z := f(x, y)$ and returns z to P_1 and P_2. Then, S_1 plays the role of P_2 interacting with the semi-honest adversary A_1 to collect all the random bits $\{r_w, s_w\}_{w\in O}$. Hence, it can compute the mask bits $\{\lambda_w\}_{w\in O}$ and recover the actual output bits $\{z_w\}_{w\in O}$. It is obvious that the views of the semi-honest adversary A_1 are indistinguishable in both real and ideal model.

$$REAL^{(1)}(k, x, r)_{k\in N} \overset{c}{\approx} IDEAL^{(1)}(k, x, r)_{k\in N}.$$

(2) Semi-honest party P_2 $(P_1[h], P_2[s], S[h])$:

In this setting, P_2 is semi-honest and follows the protocol while P_1 and $Server$ are both honest. Since this setting is very similar to the above, we only provide the conclusion as bellow and omit the proof here.

$$REAL^{(2)}(k,x,r)_{k\in N} \overset{c}{\approx} IDEAL^{(2)}(k,x,r)_{k\in N}.$$

Based on the above we conclude that our protocol securely computes function f in presence of cases $(P_1[s], P_2[h], S[h])$ and $(P_1[s], P_2[h], S[h])$, which satisfies the condition (1) of Lemma 1.

6.2.2 Malicious Party P_1 $(P_1[m_{nc}], P_2[s], S[s])$

In this setting, P_1 can maliciously adopt any strategy to deviate from the protocol while P_2 and $Server$ are both semi-honest.

First, we prove $(P_1[m_{nc}], P_2[h], S[h])$ which satisfies the condition (2) of Lemma 1. Second, combined with the condition (1) of Lemma 1 proved at above, we can reach the conclusion that our protocol securely computes function f against $(P_1[m_{nc}], P_2[s], S[s])$.

For $(P_1[m_{nc}], P_2[h], S[h])$, we construct a simulator S_1 that runs A_1 as a subroutine and plays the role of P_1 interacting with the third trusted party. In particular, S_1 is defined as bellow in the ideal world setting:

1) For step 1 to 4, S_1 takes role of an honest P_2 and $Server$ and interacts with A_1. It records all the values that would have been sent to P_2 and $Server$. S_1 also takes the role fo F_{PRE} and records all the value sent to and received from P_1.
2) For step 5, for each wire $w \in I_1$, based on the $\hat{x}_w$ received from P_1 and r_w, s_w sent to P_1, S_1 computes $x_w = \hat{x}_w \oplus r_w \oplus s_w$. Then S_1 sends $x = \{x_w\}_{w\in I_1}$ to the third trusted party who returns $\{z_w\}_{w\in O} = z = f(x,y)$.
3) For step 6 to 7, S_1 takes role of an honest P_2 and interacts with A_1. It provides the 0-string as the honest P_2's input y. If P_2 aborts, S_1 sends $abort$ to the third trusted party; Otherwise, S_1 sends $continue$.
4) For step 8, for each wire $w \in O$, if $z'_w = z_w$, S_1 takes the role of an honest P_2 and sends $(s_w, M[s_w])$ to A_1; Otherwise, S_1 sends $(s_w \oplus 1, M[s_w] \oplus \Delta_1)$ to A_1. Then, S_1 outputs whatever A_1 outputs.

We then provide the following experiments to prove the security in the setting where P_1 is malicious, P_2 and $Server$ are honest.

$Hyb1(k,x,r)$: This is the hybrid-world protocol, where S_1 takes role of an honest P_2 and F_{PRE}. It uses the actual input of P_2.

$Hyb2(k,x,r)$: This experiment is the same as $Hyb1^{(1)}(k, x,r)$, except that: (1) For step 6, for each wire $w \in I_2$, based on the $\hat{x}_w$ received from P_1 and r_w, s_w sent to P_1, S_1 recovers $x_w = \hat{x}_w \oplus r_w \oplus s_w$. Then, S_1 sends x_w to the trusted third party to get $\{z_w\}_{w\in O} = f(x,y)$; (2) For step 8, S_1 calculates $s'_w := \hat{z}_w \oplus r_w \oplus z_w$ for each wire $w \in O$, and sends $(s'_w, K[s'_w] \oplus s'_w\Delta_1)$ to A_1.

Lemma 2.

$$Hyb1(k,x,r) \overset{c}{\approx} Hyb2(k,x,r).$$

Proof. Because these two experiments both use the inputs x and y to evaluate the function f, and the view of A_1 and S_1 is identical due to the z_w that A_1 calculates by $z_w = \hat{z}_w \oplus \lambda_w$ is the same as S_1 receives from the trusted third party. Furthermore, the output of P_2 does not change between these two experiments. □

$Hyb3(k,x,r)$: This experiment is the same as $Hyb2(k,x,r)$, except that for for step 1, S_1 randomly chooses $\{u_w\}_{w\in I_2}$ and send it to P_2 to replace the $\{s_w\}_{w\in I_2}$ used before, and set $s_w := u_w \oplus y_w$ for every $w \in I_2$.

Lemma 3.

$$Hyb2(k,x,r) \overset{c}{\approx} Hyb3(k,x,r).$$

Proof. Because $\{u_w\}_{w\in I_2}$ is randomly generated, so does $\{s_w\}_{w\in I_2}$. The view of S and adversary A_1 is identically distributed. The same for the output of P_2 in both experiments. □

$Hyb4(k,x,r)$: This experiment is the same as $Hyb3(k,x,r)$, except that for step 6, S_1 takes the role of an honest P_2 and interacts with A_1. It provides the 0-string as the input of the honest P_2.

Lemma 4.

$$Hyb3(k,x,r) \overset{c}{\approx} Hyb4(k,x,r).$$

Proof. Although the value of y is different in both experiments, but there exists $y_w \oplus \lambda_w = r_w \oplus u_w$ in both experiments. So the view of S and adversary A_1 is identically distributed. Also, If S_1 aborts, which means P_2 aborts, as to the P_2's abort based on y can take place by choosing which row of the garbled table to decrypt. This depends on the calculation on $\lambda_\alpha \oplus z_\alpha$ and $\lambda_\beta \oplus z_\beta$ which are distributed uniformly in these two experiments. If S_1 does not abort, the distribution on the output of P_2 in both experiments are identical. □

The experiment $Hyb4(k,x,r)$ is the ideal world execution described above. We conclude that based on above series of experiments, the following equation holds which proves Definition 1 when P_1 is malicious, P_2 and $Server$ are honest.

$$REAL(k,x,r)_{k\in N} \overset{c}{\approx} IDEAL(k,x,r)_{k\in N}.$$

Together the above proof with the proof for condition (1) in Lemma 1 and Lemma 1, we reach the conclusion that our server-aided protocol meets the case (1) in Theorem 1, namely $(P_1[m], P_2[s], S[s])$ where P_1 is malicious, P_2 and $Server$ are semi-honest.

6.2.3 Malicious Party P_2 $(P_1[s], P_2[m_{nc}], S[s])$

In this setting, P_2 can maliciously adopt any strategy to deviate from the protocol while P_1 and $Server$ are semi-honest.

Since the proof for this case is similar as above, thus we omit it here and provide the conclusion that our protocol can securely compute function f against $(P_1[s], P_2[m_{nc}], S[s])$.

6.2.4 Malicious Party P_2 and $Server$ $(P_1[h], P_2[m_c], S[s_c])$

In this setting, P_2 is malicious and can collude with semi-honest $Server$, while P_1 is honest. Thus, the security reduces to the original 2PC protocol [13] for the case where P_2 is

TABLE 2
Comparison of Computation Cost in the Literature

Work	$\mathbf{P_1}$	$\mathbf{P_2}$	**Server**
[KMR12]([16])	$\frac{2}{5}\sigma\|x\| + \frac{4}{5}\sigma\|z\| + \|C\| + 2(\|x\| + \|y\|)Hash$	$\frac{2}{5}\sigma\|y\| + \frac{4}{5}\sigma\|z\| + 2(\|x\| + \|y\|)Hash$	$\frac{4}{5}\sigma\|z\| + \sigma\|C\|$
[CMTB13]([17])	$2\sigma\|x\| + \sigma\|z\| + \sigma\|C\| + (\sigma\|x\| + 2\sigma\|y\|$ $+\sigma)Hash + tOT + 1CT$	$\frac{2}{5}\sigma\|x\| + \|z\| + \frac{2}{5}\sigma + tOT + 1CT$	$\frac{2}{5}\sigma\|z\| + \sigma\|C\| + \sigma Hash + \|z\|ZK$
[CLT14]([14])	$(1+\sigma)\|x\| + \frac{2}{5}\sigma\|z\|$	$\frac{3}{5}\sigma\|x\| + \frac{3}{5}\sigma\|y\| + \frac{3}{5}\sigma\|z\| + \frac{3}{5}\sigma\|c\| +$ $\frac{2}{5}\sigma\|C\| + \|z\|Hash + \|y\|OT + \sigma OT$	$5\sigma\|x\| + 5\sigma\|y\| + (2\sigma + 1)\|z\| +$ $3\sigma\|c\| + \sigma\|C\| + \|y\|OT + \sigma OT$
[CMTB16]([19])	$6\|x\| + 2\|z\| + 8\|x\|MAC$	$3\|x\|MAC + \|C\|(2PC - OP)$	$2\|x\| + \|z\| + 3\|x\|MAC + \|C\|(2PC - OP)$
[BB16]([20])	$2\|x\| + \|z\|Hash + 2\|x\|Com + \|x\|ZK$	$\|C\| + 2\|Z\|Hash + \|y\|OT +$ $(\|x\| + \|y\|)Com + \|y\|ZK$	$2\|x\| + 2\|y\| + \|C\| + \|y\|OT + (\|x\| + \|y\|)ZK$
[MOR16]([21])	$\|x\| + \|z\| + 18\|C\| + 2\|C\|Hash + 2\|C\|MAC$	$\|z\| + 34\|C\| + 2\|C\|Hash$	$6\|C\| + (\|x\| + \|y\| + 3\|C\|)MAC$
This Work	$2\|x\| + 3\|z\| + 46\|C\| + 8\|C\|Hash +$ $(2\|x\| + \|y\| + \|z\| + 2\|C\|)MAC$	$2\|y\| + 2\|z\| + 33\|C\| + 4\|C\|Hash +$ $(\|x\| + 2\|y\| + \|z\| + 2\|C\|)MAC$	$7\|C\| + \|C\|Hash$

Note: The computation cost is measured by the number of symmetric encryption operations or XOR operations, Hash operations ($Hash$), oblivious transfer operation (OT), coin tossing operation (CT), message authentication code operations (MAC) and zero-knowledge proofs operations (ZK). $|x|, |y|, |z|, |c|, |C|$ are the input size of P_1, input size of P_2, the output size, the special input size of $Server$ and circuit size respectively. σ is the number of circuits used in cut-and-choose approach.

malicious. Thus, we omit the proof here and suggest the reader to the [13] for more details. However, since P_2 and $Server$ collude together, the fairness property cannot be provided in this setting. Because P_2 can always get the final output earlier than P_1, and can decide whether to abort the protocol for preventing P_1 from receiving the output.

7 Performance Evaluation

7.1 Asymptotic Evaluation

We first provide the asymptotic evaluation by comparing the related previous works in the literature. Based on the Table 1, we select the works [14], [16], [17], [19], [20], [21] which achieve maliciously security. We omit the comparison for three works [15], [18], [22] and give reasons as follows: For the work [15], it is improved by the work [16], so it is a wise choice to compare with [16] directly. For the work [18], it works in a different framework from ours with multiple servers. Also, it neither permits the collusion between client parties and servers nor provides the fairness guarantee. For the work [22], it can only prevent semi-honest adversary, and neither considers collusion nor fairness.

As shown in Table 2, we conclude the asymptotic computation cost as follows: Since the protocols in [16], [17] and [14] are all based on the cut-and-choose technique to achieve security in the malicious model, the computation cost for each of the three participant parties related to the parameter σ which is the number of generated circuits used in the cut-and-choose based protocols. To ensure an adversary could succeed in cheating with probability at most 2^{-40}, the parameter σ should be 128 according to $2^{-40} = 2^{-0.32\times\sigma}$. This incurs significant computation overhead in these protocols. For the work [19], as it only provides the sever-aided 2PC framework where P_2 and $Server$ run some 2PC protocol as a black box for circuit evaluation, it is hard to estimate the computation cost for P_2 and $Server$. Therefore, we cannot tell whether the cost of their work is lower or higher than ours. For the work [20], the protocol utilizes oblivious transfer (OT), commitments and zero-knowledge proofs (ZK) to achieve input certification. However, these techniques are high overhead computation operations. Since the OT and ZK operations are at least 2-3 orders of magnitude slower than Pseudo Random Generator (PRG) utilized by information-theoretic MAC in this work against malicious adversary, these high overhead operations do affect the efficiency of work [20] compared with this work. However, it is not easy to get the result directly from the asymptotic computation cost comparison, as the operations calculation is complicated for this work. Therefore, we would provide the concrete experimental comparison for these two works in the next section. For the work [21], although the asymptotic computation cost shows that it has less overhead than ours, the experimental results provided in the next section show that its running time is 10 times less efficient than ours. The detailed reason is provided in the next section and omitted here.

As shown in Table 3, we conclude the asymptotic communication cost as follows: The first three works are all based on the cut-and-choose technique of which the parameter σ has to be at least 128 to achieve the security of 2^{-40}. Thus, the communication overhead of them is higher than our work. For the work [19], the communication cost of $P_1 \leftrightarrow Server$ and $P_1 \leftrightarrow P_2$ is lower than ours. However, as it provides the 2PC like a black box, the 2PC communication cost of $P_2 \leftrightarrow Server$ cannot be calculated. Therefore, it is hard to tell whether this protocol costs higher than ours or not in theory. For the work [20], the communication cost of these three types are all less than ours and the experimental data in next section also proves this. But this communication cost advantage does not benefit the execution time of it, the experimental data provided in the next section shows that this work is 81 times faster than work [20]. The detailed analysis for this is provided in the next section and omitted here. For the work [21], although communication cost of $P_2 \leftrightarrow Server$ and $P_1 \leftrightarrow P_2$ is lower than ours, this also does not benefit the running time of it as the communication latency will affect the execution time. The experimental data provided in the next section shows this work is 10 times faster than work [21]. The detailed analysis for this is provided in the next section and omitted here.

TABLE 3
Comparison of Communication Cost in the Literature

Work	$\mathbf{P_1 \longleftrightarrow Server}$	$\mathbf{P_2 \longleftrightarrow Server}$	$\mathbf{P_1 \longleftrightarrow P_2}$
[KMR12]([16])	$\frac{2}{5}\sigma\|x\| + 3\|z\| + \sigma\|C\| + 2\sigma + (\|x\| + \|y\|)Hash$	$\frac{2}{5}\sigma\|y\| + 3\|z\| + \frac{7}{5}\sigma + (\|x\| + \|y\|)Hash$	0
[CMTB13]([17])	$\frac{2}{5}\sigma\|x\| + \sigma\|y\| + (\frac{7}{5}\sigma + 1)\|z\| + \frac{4}{5}\sigma + \|z\|ZK$	$\frac{2}{5}\sigma\|x\| + (\frac{2}{5}\sigma + 1)\|z\| + \frac{2}{5}\sigma Hash + \|z\|ZK$	$\sigma\|z\| + 1CT$
[CLT14]([14])	$\sigma\|x\| + \|z\|$	$2\sigma\|x\| + (2\sigma + 1)\|y\| + (2\sigma + 1)\|z\| + \sigma\|C\| + (\|y\| + \sigma)OT$	$\sigma\|x\| + 2\|z\|$
[CMTB16]([19])	$\|z\| + 2MAC$	$2\|x\| + 2\|z\| + 4MAC + \|C\|(2PC - OP)$	$2\|x\| + \|z\| + 2MAC$
[BB16]([20])	$2\|z\| + 2\|x\|Com + \|x\|ZK$	$2\|z\| + \|C\| + \|y\|OT + \|x\|Com + \|y\|ZK$	$2\|x\| + \|y\| + 2\|z\|Hash$
[MOR16]([21])	$\|x\| + \|y\| + 3\|C\| + (\|x\| + \|y\| + 6\|C\|)MAC$	$\|x\| + \|y\| + 3\|C\| + (\|x\| + \|y\| + 6\|C\|)MAC$	$\|x\| + \|y\| + 2\|z\| + 4\|C\| + 2\|z\|MAC$
This Work	$2\|x\| + \|y\| + \|z\| + 4\|C\|Hash + 1MAC$	$\|y\| + \|z\| + 4\|C\| + (8\|C\| + 1)MAC$	$3\|x\| + 4\|y\| + 2\|z\| + 4\|C\| + (5\|x\| + 6\|y\| + 2\|z\| + 8\|C\|)MAC$

Note: The communication cost is measured by the number of symmetric encryption ciphertext, Hash strings ($Hash$), oblivious transfer strings (OT), coin tossing strings (CT), message authentication codes (MAC) and zero-knowledge proofs (ZK). $|x|, |y|, |z|, |C|$ are the input size of P_1, input size of P_2, the output size and circuit size respectively. σ is the number of circuits used in cut-and-choose approach.

7.2 Experimental Evaluation

7.2.1 Experiment Setup

Based on the implementation of [13], we expand it into the server-aided version for heterogeneous MCC. The corresponding setting and parameters for the implementation are shown as follows:

1) Computational security parameter $\kappa = 128$ and statistical security parameter $\rho = 40$.
2) Deployment platform: One single-core 3.1 GHz machine with Intel i5-7267U processor running Ubuntu Linux 14.04 LTS. Note that in real heterogeneous MCC setting, $Server$ is expected to have stronger computation resources such that it will have more cores and threads to run the code. Therefore, the execution time will be significantly reduced in practice.
3) Function f: We set the function f in our protocol to be AES, which is the standard benchmark for 2PC implementations and also tested widely in the server-aided 2PC implementations. One client party inputs the 128-bit text to be encrypted, and another party inputs the 128-bit key. Then, both client parties get the output for the function f, which is the ciphertext of the 128-bit text encrypted by AES. In our implementation, the corresponding parameter is $|I_1| = 128$, $|I_2| = 128$, $|O| = 128$ and $|C| = 6800$, where $|I_1|$ is the input size of P_1, $|I_2|$ is the input size of P_2, $|O|$ is the output size of function f, $|C|$ is the circuit size of the corresponding boolean circuit.

7.2.2 Experiment Data Analysis

We select two works [20] and [21] from previous server-aided 2PC works to be compared with for the following reasons: (1) With the experiment results that work [20] provided, it outperforms the work [16], [17], [14], and [19]. This makes it the representative work for the server-aided 2PC where server actually takes some task for client parties (i.e., circuit generation or circuit evaluation). Thus, we omit comparison with the other works; (2) Although [21] works in a different setting from ours and [20] (i.e., $Server$ does not take circuit generation or evaluation work for client parties), the execution time that it provided is even better than work [20]. We believe it is necessary to show that our work also outperforms it. We implement our protocol on the AES circuit as stated in Section 7.2.1. We then summarize AES evaluation results that work [20] and [21] provided and compare with our work, as shown in Table 5.

For work [20], as shown in Table 4, the execution time that the client party P_1 makes has been reduced to 266.36ms and 1.28ms for preprocessing and evaluation phase respectively, which makes the total execution time be reduced to 267.64ms that is 31 times less than work [20]. The execution time that the client party P_2 takes has been reduced to 272.39ms and 1.17ms for preprocessing and evaluation phase respectively, which makes the total execution time be reduced to 273.56ms that is 16 times less than work [20]. Even though we significantly decrease the execution time for both client parties compared with work [20], the execution time of $Server$ does not increase. It is 3446 times less than work [20] for the evaluation phase. Therefore, the total execution time of $Server$ is 3581 times less than work [20]. The reason for this kind of significant decrease is as follows: We replace the part of scheme built with OT, commitment and ZK in work [20] with the information-theoretic MAC technique to guarantee the correctness of our work in the malicious model. The work [20] requires hundreds of exponentiations per gate, while our

TABLE 4
Comparison With Work [20] on Execution Time

Work	Party	Circuit Preprocessing	Circuit Evaluation	Total
[BB16]([20])	P_1	8070ms	280ms	8350ms
	P_2	3990ms	400ms	4390ms
	$Server$	380ms	10200ms	10600ms
This Work	P_1	266.36ms	1.28ms	267.64ms
	P_2	272.39ms	1.17ms	273.56ms
	$Server$	–	2.96ms	2.96ms

Note: The execution time of this work is averaged by 5 executions. The offline phase includes steps 1-4 of the protocol, and the online phase includes steps 5-8 of the protocol.

TABLE 5
Comparison of the Selected Server-aided 2PC Works

Work	Execution Time			Communication Cost		
	Circuit Preprocessing	Circuit Evaluation	Total	$\mathbf{P_1 \longleftrightarrow Server}$	$\mathbf{P_2 \longleftrightarrow Server}$	$\mathbf{P_1 \longleftrightarrow P_2}$
[BB16]([20])	12440ms	10880ms	23320ms	149 KB	469 KB	4 KB
[MOR16]([21])	485ms	2500ms	2985ms	-	-	-
This Work	284.74ms	2.96ms	287.70ms	0.552 MB	1.065 MB	6.478 MB

Note: The communication cost is measured by the size of transfered data between the corresponding communication channel. The dash mark means that the number was not provided by the work.

work only requires hundreds of PRG and XOR operations per gate. This makes the computation overhead drops at least 2-3 orders of magnitude. However, as the work [20] does not provide the overall running time of the protocol, we respectively sum the time of three columns in Table 4, and set it as the whole execution time for the circuit preprocessing, circuit evaluation and total without considering the parallelism of three participant parties in the implementation as shown in Table 5. Compared with our work, our work is 44 times and 3676 times faster than it for circuit preprocessing and evaluation phase respectively, which makes our total time 81 times outperform it.

Since the communication time can be varied based on different networks, we provide the communication cost evaluation based on the transferred data size. As shown in Table 5, compared with work [20], the communication cost of our work for each of the three parties is a little bit higher than it. The most fundamental reason for this increment is that we need to do the authenticated work along with the circuit based on information-theoretic MAC technique, which enables the protocol to work in the malicious model without high computation cost operations like zero-knowledge proofs. It is the fact that the execution time decreases, but the communication cost increases. Especially for the cost of $P_1 \leftrightarrow P_2$, the increment is not only caused by the authenticated values for the inputs of two client parties but also the authenticated values of all the AND gates in the circuit. However, the work [20] does not permit collusion which makes the security level of [20] weaker than ours (note that we only permit collusion in the case where malicious P2 and semi-honest Server can collude, while P1 is semi-honest). Besides, the execution time (which involves the communication time) of our work is much less than [20], which means that these little higher communication costs have a negligible effect on the performance.

For work [21], as shown in Table 5, our work is 1.7 times and 845 times faster than it for circuit preprocessing and evaluation phase respectively, which makes our total cost 10 times outperform it. The reason for this improvement is shown as follows: (1) For the circuit preprocessing phase, the server of work [21] not only needs to generate the authenticated random bits shares for the inputs wires, but also the authenticated Beaver triple shares for the non-XOR gates in the circuit. However, in our work, the above high overhead work is amortized by two client parties, which reduces the execution time of preprocessing phase by nearly half; (2) For the circuit evaluation phase, work [21] follows the GMW paradigm where the evaluation for each AND gate of the circuit requires multiple local computations and interactions. This leads to the high overhead in the circuit evaluation phase, especially for the deep circuit. However, in our work, based on the Yao's garbled circuit paradigm, the evaluation for all AND gates of the circuit requires only one round interaction, which greatly improves the execution time of the evaluation phase. Furthermore, we provide the histogram to more directly show the performance improvement for this work compared with works [20] and [21], as shown in Fig. 6.

As the work [21] did not provide any concrete result on communication cost (i.e., the transferred data size) of the protocol, we omit this concrete communication comparison with it. Although the asymptotic communication comparison provided in Table 3 shows that the communication cost of $P_2 \leftrightarrow Server$ and $P_1 \leftrightarrow P_2$ is lower than ours, this does not benefit the total running time which contains the communication time as shown in Table 5. In fact, the communication time includes the latency which depends on the interaction time and data transferred time. As the interactions we need for circuit evaluation is quite lower than the work [21], which makes the communication time is lower and results in the lower execution time.

Based on the above experimental evaluation, we conclude that this work is the most efficient work compared with all the previous works. Besides, it can permit collusion between the garbled circuit evaluator and server with guaranteeing privacy and correctness in the malicious model. Also, it can guarantee fairness when collusion does not exist in the malicious model.

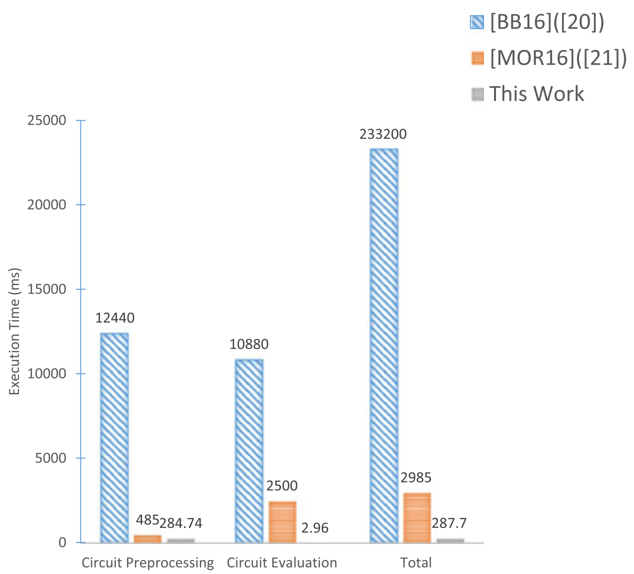

Fig. 6. Comparison for execution time.

8 Conclusion

Since the current computing setting is mainly based on heterogeneous computation model (e.g., MCC), the standard 2PC protocol focusing on traditional homogeneous computation model is not suited well to the current computing setting. To address this, we provide an efficient server-aided secure two-party computation protocol in heterogeneous mobile cloud computing. Compared with previous works, this work is the first work that considers collusion between the garbled circuit evaluator and server with guaranteeing privacy and correctness in the malicious model. Also, it can guarantee fairness when collusion does not exist in the malicious model. The security analysis shows that our protocol can securely compute a function $f(x, y)$ in the following two settings: (1) Any one of the two client parties is malicious and cannot collude with the other semi-honest client party and semi-honest $Server$; (2) P_2 is malicious and can collude with semi-honest $Server$, while P_1 is honest. The experimental performance analysis shows that this work outperforms all the previous work for at least 10 times with the same security level.

Acknowledgments

This work was supported by the Basic Research Project of Shenzhen, China, (No. JCYJ20180507183624136), National Natural Science Foundation of China (No. 61872109), Guangdong Key R&D Program (No. 2019B010136001), National and Provincial Program Supporting Projects of Shenzhen, China (No. GJHS20170313113617970). The seventh author is partly supported with the National Natural Science Foundation of China under Grant No. 61872412.

References

[1] C. Liu, X. S. Wang, K. Nayak, Y. Huang, and E. Shi, "ObliVM: A programming framework for secure computation," in *Proc. IEEE Symp. Secur. Privacy*, 2015, pp. 359–376.

[2] 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.

[3] X. Wang, A. J. Malozemoff, and J. Katz, "Faster secure two-party computation in the single-execution setting," in *Proc. Annu. Int. Conf. Theory Appl. Cryptographic Techn.*, 2017, pp. 399–424.

[4] X. S. Wang, Y. Huang, T. H. Chan, A. Shelat, and E. Shi, "SCORAM: Oblivious ram for secure computation," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2014, pp. 191–202.

[5] S. Zahur *et al.*, "Revisiting square-root ORAM: Efficient random access in multi-party computation," in *Proc. IEEE Symp. Secur. Privacy*, 2016, pp. 218–234.

[6] J. Doerner, D. Evans, and A. Shelat, "Secure stable matching at scale," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2016, pp. 1602–1613.

[7] K. Nayak, X. S. Wang, S. Ioannidis, U. Weinsberg, N. Taft, and E. Shi, "Graphsc: Parallel secure computation made easy," in *Proc. IEEE Symp. Secur. Privacy*, 2015, pp. 377–394.

[8] X. S. Wang, Y. Huang, Y. Zhao, H. Tang, X. Wang, and D. Bu, "Efficient genome-wide, privacy-preserving similar patient query based on private edit distance," in *Proc. 22nd ACM SIGSAC Conf. Comput. Commun. Secur.*, 2015, pp. 492–503.

[9] I. Damgård, V. Pastro, N. Smart, and S. Zakarias, "Multiparty computation from somewhat homomorphic encryption," in *Proc. Advances Cryptol. Conf.*, 2012, pp. 643–662.

[10] Y. Lindell and B. Riva, "Blazing fast 2PC in the offline/online setting with security for malicious adversaries," in *Proc. 22nd ACM SIGSAC Conf. Comput. Commun. Secur.*, 2015, pp. 579–590.

[11] J. B. Nielsen, T. Schneider, and R. Trifiletti, "Constant round maliciously secure 2PC with function-independent preprocessing using LEGO," in *Proc. Netw. Distrib. System Secur. Symp.*, 2017, pp. 1–15.

[12] P. Rindal and M. Rosulek, "Faster malicious 2-party secure computation with online/offline dual execution," in *Proc. USENIX Secur. Symp.*, 2016, pp. 297–314.

[13] X. Wang, S. Ranellucci, and J. Katz, "Authenticated garbling and efficient maliciously secure two-party computation," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2017, pp. 21–37.

[14] H. Carter, C. Lever, and P. Traynor, "Whitewash: Outsourcing garbled circuit generation for mobile devices," in *Proc. 30th Annu. Comput. Secur. Appl. Conf.*, 2014, pp. 266–275.

[15] S. Kamara, P. Mohassel, and M. Raykova, "Outsourcing multi-party computation," *Cryptol. ePrint Archive*, Rep. 2011/272, 2011. [Online]. Available: https://eprint.iacr.org/2011/272

[16] S. Kamara, P. Mohassel, and B. Riva, "Salus: A system for server-aided secure function evaluation," in *Proc. ACM Conf. Comput. Commun. Secur.*, 2012, pp. 797–808.

[17] H. Carter, B. Mood, P. Traynor, and K. Butler, "Secure outsourced garbled circuit evaluation for mobile devices," in *Proc. USENIX Secur. Symp.*, 2013, pp. 289–304.

[18] T. P. Jakobsen, J. B. Nielsen, and C. Orlandi, "A framework for outsourcing of secure computation," in *Proc. 6th Ed. ACM Workshop Cloud Comput. Secur.*, 2014, pp. 81–92.

[19] H. Carter, B. Mood, P. Traynor, and K. Butler, "Outsourcing secure two-party computation as a black box," *Secur. Commun. Netw.*, vol. 9, no. 14, pp. 2261–2275, 2016.

[20] M. Blanton and F. Bayatbabolghani, "Efficient server-aided secure two-party function evaluation with applications to genomic computation," in *Proc. Privacy Enhancing Technol.*, 2016, pp. 144–164.

[21] P. Mohassel, O. Orobets, and B. Riva, "Efficient server-aided 2PC for mobile phones," *Proc. Privacy Enhancing Technol.*, pp. 82–99, 2016.

[22] F. Baldimtsi, D. Papadopoulos, S. Papadopoulos, A. Scafuro, and N. Triandopoulos, "Server-aided secure computation with off-line parties," in *Proc. Eur. Symp. Res. Comput. Secur.*, 2017, pp. 103–123.

[23] G. Asharov, Y. Lindell, T. Schneider, and M. Zohner, "More efficient oblivious transfer and extensions for faster secure computation," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2013, pp. 535–548.

[24] 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.

[25] Y. Huang, D. Evans, J. Katz, and L. Malka, "Faster secure two-party computation using garbled circuits," in *Proc. USENIX Secur. Symp.*, 2011, pp. 331–335.

[26] V. Kolesnikov, P. Mohassel, and M. Rosulek, "FleXOR: Flexible garbling for XOR gates that beats free-XOR," in *Proc. Int. Cryptol. Conf.*, 2014, pp. 440–457.

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

[28] D. Malkhi, N. Nisan, B. Pinkas, and Y. Sella, "Fairplay-secure two-party computation system," in *Proc. USENIX Secur. Symp.*, 2004, Art. no. 9.

[29] B. Pinkas, T. Schneider, N. P. Smart, and S. C. Williams, "Secure two-party computation is practical," in *Proc. Int. Conf. Theory Appl. Cryptol. Inf. Secur.*, 2009, pp. 250–267.

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

[31] Y. Lindell and B. Pinkas, "An efficient protocol for secure two-party computation in the presence of malicious adversaries," in *Proc. Annu. Int. Conf. Theory Appl. Cryptographic Techn.*, 2007, pp. 52–78.

[32] P. Mohassel and M. Franklin, "Efficiency tradeoffs for malicious two-party computation," in *Proc. Int. Workshop Public Key Cryptography*, 2006, pp. 458–473.

[33] J. B. Nielsen and C. Orlandi, "Lego for two-party secure computation," in *Proc. Theory Cryptography Conf*, 2009, pp. 368–386.

[34] T. K. Frederiksen, T. P. Jakobsen, J. B. Nielsen, P. S. Nordholt, and C. Orlandi, "Minilego: Efficient secure two-party computation from general assumptions," in *Proc. Annu. Int. Conf. Theory Appl. Cryptographic Techn.*, 2013, pp. 537–556.

[35] T. K. Frederiksen, T. P. Jakobsen, J. B. Nielsen, and R. Trifiletti, "TinyLEGO: An interactive garbling scheme for maliciously secure two-party computation," *Cryptol. ePrint Archive*, Rep. 2015/309, 2015. [Online]. Available: https://eprint.iacr.org/2015/309

[36] U. Feige, J. Killian, and M. Naor, "A minimal model for secure computation," in *Proc. 26th Annu. ACM Symp. Theory Comput.*, 1994, pp. 554–563.

[37] I. Damgård, S. Faust, and C. Hazay, "Secure two-party computation with low communication," in *Proc. Theory Cryptography Conf.*, 2012, pp. 54–74.

[38] R. Cleve, "Limits on the security of coin flips when half the processors are faulty," in *Proc. 18th Annu. ACM Symp. Theory Comput.*, 1986, pp. 364–369.

[39] J. B. Nielsen, P. S. Nordholt, C. Orlandi, and S. S. Burra, "A new approach to practical active-secure two-party computation," in *Proc. Advances Cryptol. Conf.*, 2012, pp. 681–700.

[40] G. Asharov, A. Jain, A. López-Alt, E. Tromer, V. Vaikuntanathan, and D. Wichs, "Multiparty computation with low communication, computation and interaction via threshold FHE," in *Proc. Annu. Int. Conf. Theory Appl. Cryptographic Techn.*, 2012, pp. 483–501.

[41] A. López-Alt, E. Tromer, and V. Vaikuntanathan, "On-the-fly multiparty computation on the cloud via multikey fully homomorphic encryption," in *Proc. 44th Annu. ACM Symp. Theory Comput.*, 2012, pp. 1219–1234.

[42] A. Peter, E. Tews, and S. Katzenbeisser, "Efficiently outsourcing multiparty computation under multiple keys," *IEEE Trans. Inf. Forensics Secur.*, vol. 8, no. 12, pp. 2046–2058, Dec. 2013.

[43] O. Goldreich, *Foundations Cryptography: Volume 2, Basic Applications.* Cambridge, U.K.: Cambridge Univ. Press, 2009.

Yulin Wu received the bachelor's degree in information security from Northeastern University, China, in 2016. Since 2016, she has been working toward the PhD degree in computer science at the Harbin Institute of Technology, Shenzhen, China. Her research interests include secure multi-party computation, secure outsourcing computation, and cloud security.

Xuan Wang (Member, IEEE) received the PhD degree in computer science from the Harbin Institute of Technology, in 1997. He is one of the inventors of Microsoft Pinyin, and once worked in Microsoft headquarter in Seattle due to his contribution to Microsoft Pinyin. He is currently the dean of the School of Computer Science and Technology, Harbin Institute of Technology, Shenzhen, China. His main research interests include cybersecurity, information game theory, and artificial intelligence.

Willy Susilo (Senior Member, IEEE) is a senior professor in the School of Computing and Information Technology, Faculty of Engineering and Information Sciences at the University of Wollongong (UOW), Australia. He is the director of Institute of Cybersecurity and Cryptology, School of Computing and Information Technology, UOW and the head of School of Computing and Information Technology at UOW (2015 till now). Prior to this role, he was awarded the prestigious Australian Research Council Future Fellowship in 2009. He has published more than 500 papers in journals and conference proceedings in cryptography and network security. In 2016, he was awarded the "Researcher of the Year at UOW, due to his research excellence and contributions. He is the editor-in-chief of the Elsevier's Computer Stanrdards and Interface and the Information journal. He is currently an associate editor of the *IEEE Transactions on Dependable and Secure Computing*. He has also served as the program committee member of several international conferences.

Guomin Yang (Senior Member, IEEE) received the PhD degree in computer science from the City University of Hong Kong, in 2009. He was a research scientist with the Temasek Laboratories, National University of Singapore, from 2009 to 2012. He is currently an associate professor with the School of Computing and Information Technology, University of Wollongong, Australia. His research interest include applied cryptography and network security. He received the Australian Research Council Discovery Early Career Researcher Award in 2015.

Zoe L. Jiang received the PhD degree from the University of Hong Kong, in 2010. She is currently an associate professor with the School of Computer Science and Technology, Harbin Institute of Technology, Shenzhen, China. Her research interests include secure multi-party computation, secure outsourcing computation and cloud security.

Qian Chen received the master's degree in computer science from the Harbin Institute of Technology, Shenzhen, China, in 2018. She is currently working toward the PhD degree in computer science at the Harbin Institute of Technology, Shenzhen, China. Her research interests include deep learning, data privacy protection, and network security.

Peng Xu (Member, IEEE) received the PhD degree in computer science from the Huazhong University of Science and Technology, Wuhan, China, in 2010. He worked as a post-doctor at the Huazhong University of Science and Technology, Wuhan, China, from 2010 to 2013, and as an associate research fellow at the University of Wollongong, Australia, from 2018 to 2019. Currently, he is an associate professor at the Huazhong University of Science and Technology. His research interest includes field of cryptography. He authored more than 30 research papers and two books. He was PI in eight grants, including three NSF fundings.