Hindawi
Security and Communication Networks
Volume 2024, Article ID 9891407, 1 page
https://doi.org/10.1155/2024/9891407

Retraction

Retracted: Secure Two-Party Computation Based on Fast Cut-and-Choose Bilateral Oblivious Transfer

Security and Communication Networks

Received 8 January 2024; Accepted 8 January 2024; Published 9 January 2024

This article has been retracted by Hindawi, as publisher, following an investigation undertaken by the publisher [1]. This investigation has uncovered evidence of systematic manipulation of the publication and peer-review process. We cannot, therefore, vouch for the reliability or integrity of this article.

Please note that this notice is intended solely to alert readers that the peer-review process of this article has been compromised.

Wiley and Hindawi regret that the usual quality checks did not identify these issues before publication and have since put additional measures in place to safeguard research integrity.

We wish to credit our Research Integrity and Research Publishing teams and anonymous and named external researchers and research integrity experts for contributing to this investigation.

The corresponding author, as the representative of all authors, has been given the opportunity to register their agreement or disagreement to this retraction. We have kept a record of any response received.

References

[1] Y. Wang, K. Xiong, H. Tian, J. Zhang, and X. Yan, "Secure Two-Party Computation Based on Fast Cut-and-Choose Bilateral Oblivious Transfer," *Security and Communication Networks*, vol. 2022, Article ID 3880413, 10 pages, 2022.

Hindawi
Security and Communication Networks
Volume 2022, Article ID 3880413, 10 pages
https://doi.org/10.1155/2022/3880413

Research Article

Secure Two-Party Computation Based on Fast Cut-and-Choose Bilateral Oblivious Transfer

Yongjun Wang, Kun Xiong, He Tian, Jing Zhang, and Xixi Yan

School of Software, Henan Polytechnic University, Jiaozuo 454000, China

Correspondence should be addressed to Jing Zhang; zj_jsj@hpu.edu.cn

Received 22 February 2022; Revised 2 June 2022; Accepted 8 June 2022; Published 27 June 2022

Academic Editor: Zhiping Cai

In secure two-party computation, each party has its input and wants to jointly compute a function from which it obtains the output corresponding to its respective inputs. For achieving security against a malicious adversary, an effective approach is using cut-and-choose, which requires the circuit constructor P_1 to construct S copies of the circuit C (C is used to compute the function F). The circuit evaluator P_2 selects $S/2$ circuits to open for the check. If these $S/2$ circuits are correctly constructed, P_2 assumes that the remaining $S/2$ circuits are also correctly constructed and uses the remaining circuits to compute. However, this method introduces significant computational complexity and interactive rounds, mainly due to more circuits that must be used for security purposes and the need for multiple interactions to transmit the keys. In this paper, regarding the issue above, we present a novel secure two-party computation protocol, and it can achieve security against the malicious adversary. Concretely, we still use the idea of cut-and-choose but improve the cut-and-choose oblivious transfer (CCOT) of the usual secure two-party computation protocol into cut-and-choose bilateral oblivious transfer (CCBOT) and propose a variant of it that we call batch single-choice CCBOT, which makes our protocol only needs two rounds of interaction to complete the transmission of all keys and $28Sl$ of exponentiations. In addition, we use a check mechanism to prevent the case that p_1 cheats, but P_2 is powerless. Our proposed protocol with an error probability of 2^{-s} of P_1 significantly optimizes the communication rounds and computation overheads, solves the selective failure attack, and ensures the consistency of the input.

1. Introduction

1.1. Background. Secure two-party computation means that two mutually untrusted participants, each holding their input, collaborate to compute a function through a two-party computation protocol that satisfies multiple security properties and obtain the corresponding function output. These security properties mainly include privacy, correctness, input independence, guaranteed output delivery, and fairness [1]. Since professor Yao proposed secure two-party computation [2], it has become the subject of extensive research, with a focus on improving security and efficiency.

Yao's protocol based on oblivious transfer (OT) [3] and garbled circuit (GC) [2] is a well-known protocol of secure two-party computation. However, it only achieves security against the semihonest adversary. Because only one circuit is constructed, P_1 can easily cheat by constructing a wrong circuit. For improving the correctness of circuit computation and the security of the protocol, it is best to construct multiple circuits. Part of the circuits are used to check whether they are constructed correctly, and the rest circuits are used to compute the function. This method of dividing circuits to check and then for evaluation is called cut-and-choose [4]. It can solve the problem that Yao's protocol cannot achieve security against the malicious adversary. Its main idea is that P_1 constructs s identical circuits (only one circuit in Yao's protocol). Then P_2 selects some of them (usually half of the total) to check whether these selected circuits (called check circuits) are correct. If the check passes, P_2 can consider that the rest circuits (called computation circuits or evaluation circuits) are all correct. Finally, P_2 uses the evaluation circuits to compute and uses most of the same outputs as the final function output.

Although the protocols for cut-and-choose can combine the check of circuits and the oblivious transfer for transmitting keys to solve the problem of selective failure attacks [5], these protocols use CCOT [6–8] to transmit garbled keys. One significant feature of CCOT is that it can only transmit keys associated with P_2, s input wire. This feature to only transmit keys associated with one party will cause the protocol to generate many additional interactions for separately transmitting the relevant keys on P_1's input wire in check circuits and evaluation circuits, which seriously increases the number of interactions and round complexity. The CCOT protocol used in the famous Lindell and Pinkas protocol [9] requires 6 rounds of communication and cannot transmit all required keys. Additionally, it also requires additional interactions for zero-knowledge proofs, and there are 12 rounds of communication of the whole protocol. In addition to transmitting keys, additional interaction is needed to send the set $\mathcal{J}$ and verify its consistency of it in two interactions. The protocol [10] of Huang et al. uses a kind of multistage CCOT that still needs to send a cut-and-choose challenge and then uses additional interactions to send garbled input values. Similarly, more [11, 12] have the same problem. In addition, in most of Yao's garbled circuit protocols based on CCOT, the result takes most of the same values in all evaluation circuits. If there are a few evaluation circuits with different output values, then P_1 has cheated. However, P_2 will take most of the same value as the result according to the requirements of the protocol, even if it knows that P_1 is cheating. This method is likely to leak P_2's private input. P_1 can construct the wrong garbled circuits to cheat, but P_2 is powerless in this case.

With the intensive study of CCOT, CCBOT [13] has emerged. In CCBOT, the receiver R selects the received value according to its input, and the sender S can also actively choose other values to send to R. Compared with CCOT, CCBOT only needs one interaction to send all required keys. Almost all the secure two-party computation protocols using cut-and-choose technology complete the transmission of keys through CCOT. Currently, there has been a great deal of interest in CCBOT research, but there is still a lack of a complete and efficient secure two-party computation protocol that uses CCBOT to complete keys transfer. CCBOT makes the protocols for cut-and-choose have a new improvement. We believe that designing a secure and efficient two-party computing protocol based on CCBOT is research-intensive and challenging. Security here consists mainly of solving the input consistency problem and selective failure attack and achieving security against the malicious adversary. Efficiency is improved by using CCBOT to reduce the rounds of interaction and the complexity of communication between two parties. In addition, we are also committed to solving the problem that P_1 cheats, but P_2 is powerless.

In our paper, P_1 needs to construct many copies of the circuit, so we first introduce the cut-and-choose (introduce a new parameter s) into the CCBOT protocol of reference [14]. Then, for making P_2 input the same choice bit on each input wire, we improve CCBOT to the single-choice CCBOT. Since each input wire in each circuit must perform a single-choice CCBOT, it is necessary to perform the transfer key phase in batches. After the above improvements, we propose a batch single-choice CCBOT protocol, and see Section 4 for more details. Next, for the input consistency problem of P_1, we use a Diffie–Hellman pseudorandom synthesizer [15] to generate the keys $g^{a_i^0 \cdot r_j}$ and $g^{a_i^1 \cdot r_j}$ of its own input bits. This key structure makes it easy for P_1 to use the DDH assumption to prove the consistency of its inputs across all circuits. In addition, in our protocol, the inputs of both parties are done through only one interaction, which prevents the problem of inconsistent inputs between different interactions. And then, a selective failure attack is caused by the separation of circuit division from the oblivious transfer. Based on the features of the CCBOT protocol, we intertwine transmission keys with circuit checks. This method does not require the use of larger inputs and more circuits for the security of the protocol. Finally, for the problem that P_2 cannot abort the protocol when P_1 constructs the wrong circuits to cheat we have added an additional check mechanism. When the outputs of all evaluation circuits are inconsistent, P_2 will store a "proof" and then input this "proof" into the secure check protocol to obtain the private input of P_1 and compute $f(x, y)$ locally. The mechanism is detailed in reference [16].

1.2. Our Contributions. We first improve the CCBOT protocol of reference [14] and formalize the improved protocol. Based on this, we present a novel secure two-party protocol. It can achieve security against the malicious adversary. Our contribution consists mainly of the following:

(1) We improve the original CCBOT protocol to a variant called batch single-choice CCBOT. It can complete the oblivious transfer tasks of each gate in all circuits in parallel and requires two rounds of communication and $15s\ell$ exponents. Since only two rounds are required to complete all key transmissions, no additional interaction is required to send the keys associated with their own inputs in check circuits and evaluation circuits.

(2) We apply our batch single-choice CCBOT on Yao's protocol and propose a new secure two-party computation protocol that can solve the problem of input consistency and selective failure attack. It achieves security against the malicious adversary. Moreover, P_1's error probability is the optimal 2^{-s} in our protocol.

(3) We use a check mechanism of work [16] to prevent the case that P_2 finds out that P_1 is cheating but P_2 is powerless. In general, when the outputs of all evaluation circuits are inconsistent, P_2 will find the wire in s copies of the circuit but output different results and then store the garbled values b_0 corresponding to 0 and b_1 corresponding to 1 for the output of this output wire in different circuits. Finally, P_2 inputs them into the check protocol to obtain the private input of P_1, so as to calculate $f(x, y)$ locally.

2. Related Work

Secure multiparty computation is an active research field of cryptography, and there is more and more extensive research on secure two-party computation. Its classical architectures often use garbled circuits [17] or GMW protocols [18], with the main research being on using garbled circuits to design an efficient secure computation protocol in different security models [19, 20] and optimizing the efficiency of various security protocols [21–23]. Particularly, the problem of efficiency has attracted a great deal of attention due to the large amount of bandwidth it requires. How to efficiently perform the computation in a malicious model is essential. Taking the running time of the entire protocol into account, the cut-and-choose approach is almost the best method to achieve security against the malicious adversary.

Pinkas et al. first introduced the cut-and-choose approach [4] into the garbled circuit in 2003, and Lindell proposed the first secure two-party protocol with complete security proof in 2007 [24]. The most classic protocol can be traced back to reference [9]. It divides 50% check circuits and 50% evaluation circuits to achieve security against the malicious adversary, and P_1's cheating probability is $2^{-0.311s}$. Therefore, an error probability of 2^{-40} can only be achieved when s is set to 132 (132 gates). This means the cost of achieving security against the malicious adversary is 132 times that of a semihonest adversary. In contrast, it divides 60% check circuits and 40% evaluation circuits to get an error probability of $2^{-0.32s}$ in reference [25]. Reference [26] proposed the idea of a symmetrical cut-and-choose. The idea is essentially that two parties involved work together to construct s copies of garbled circuits, which are then checked by someone else. The probability of an error is $2^{-k+O(\log k)}$. In reference [27], it applies the cut-and-choose technique to three-party computation. The cost of achieving security against a malicious adversary is comparatively small to other semihonest 3PC.

Using the cut-and-choose technique will bring some problems. In order to solve the input consistency problem, earlier work [4] used commitment schemes, but the communication overhead was significant. Shelat and Shen proposed a consistency detection method [25] in 2011 that reduces the communication overhead by transmitting a small number of parameters instead of zero-knowledge proofs. In 2014, Frederiksen constructed a circuit extension [28] that uses function f' instead of $f(x, y)$. With this modification, the calculation result of the new function f' is the same as the original $f(x, y)$, but it needs to come from the constructor P_1 and the evaluator P_2 additional random input bits. With statistical security, it can be verified that the inputs of both parties to all circuits are consistent. In addition, there are some schemes [29–31] that also give solutions to the problem of input consistency. For the selective failure attack problem, the evaluator in work [24] takes the XOR value of the real input and multiple random bits as the OT input so that the evaluator has nothing to do with the real input when it exits. The constructor cannot infer the evaluator input by selective failure attack. References [25, 32] solved this problem by a committing OT and a circuit extension, respectively.

Reference [16] uses a check mechanism to prevent P_2 from being powerless. If P_2 finds that the output values of all evaluation circuits are inconsistent, it stores a piece of evidence and then inputs this evidence in the check protocol to obtain the private input x of P_1 and finally calculates the function $f(x, y)$ locally. The work [33] proposed a method to encode trapdoors in the output wires, allowing the evaluator to recover the input of the constructor when multiple outputs are obtained.

In 2015, Zhao et al. first proposed the CCBOT primitive based on the CCOT protocol and constructed a complete CCBOT protocol based on homomorphic encryption. This CCBOT protocol achieved security against the malicious adversary. And then, an improved version of CCBOT based on the decisional Diffie–Hellman (DDH) assumption [14] was proposed in 2016. It greatly improved the efficiency of CCBOT but without using cut-and-choose. Ning and Wang proposed a novel CCBOT protocol [34] based on the computational Diffie–Hellman (CDH) assumption in 2020. It achieved security against the malicious adversary and the error probability of P_1 is 2^{-s}.

3. Preliminaries

We define ℓ, n, a, and s as the length of inputs, computational safety parameter, arbitrary strings, and statistical security parameter, respectively. In order to give a formal definition of security, we first introduce how to describe the indistinguishability of the probability ensemble. As we know, the cut-and-choose technique requires multiple circuits. Therefore, the probability ensemble is related to computational security parameter n, arbitrary strings a, and statistical security parameter s. In this paper, we use (n, s) – indistinguishability to describe the indistinguishability of the probability ensemble, and its formal definition is as follows:

3.1. (n, s) – Indistinguishability. There are two probability ensembles X and Y. They are in the form of $X = \{X(a, n, s)\}_{n,s\in\mathbb{N};a\in(0,1)^*}$ and $Y = \{Y(a, n, s)\}_{n,s\in\mathbb{N};a\in\{0,1\}^*}$ and satisfy that for any n and s, the value range of the two probability distributions is a string of length l, where l is represented as a polynomial of $n + s$. If for every nonuniform polynomial-time distinguisher D, $s \in \mathbb{N}$, polynomial $p(\cdot)$, $a \in \{0, 1\}^*$, and $n \in \mathbb{N}$, there exists a constant $-1 < c < 0$, and the following inequality satisfies

$$|\Pr[D(X(a, n, s), a, n, s) = 1] - \Pr[D(Y(a, n, s), a, n, s) = 1]| < \frac{1}{p(n)} + \frac{1}{2^{c\cdot s}}. \quad (1)$$

We can say X and Y are (n, s) – indistinguishability and denoted by $X^{n,s} \equiv Y$.

3.2. Ideal/Real Simulation Paradigm and Definition of Security. In this paradigm, there is an ideal world and a real world. In the real world, adversary $\mathcal{A}$ executes the secure computation protocol jointly with another honest party

(joint output is denoted as $\mathrm{IDEAL}_{f,S(z),i}(x, y, n, s)$), and in the ideal world, the simulator $\mathcal{S}$ executes the secure computation protocol jointly with another honest participant (joint output is denoted as $\mathrm{REAL}_{\pi,\mathcal{A}(z),i}(x, y, n, s)$). Below, we express the security of a secure two-party computation protocol:

$$\left\{\mathrm{IDEAL}_{f,S(z),i}(x, y, n, s)\right\}^{n,s} \equiv \left\{\mathrm{REAL}_{\pi,\mathcal{A}(z),i}(x, y, n, s)\right\}, \tag{2}$$

where $x, y, z \in \{0,1\}^*$, $|x| = |y|$, and $n, s \in \mathbb{N}$. Then, we can conclude protocol π securely computes function f.

4. Cut-and-Choose Bilateral Obvious Transfer

The key transfer phase of our secure protocol uses the CCBOT protocol. A CCBOT protocol is a variant of CCOT with the additional property that the sender can actively send its own values to the receiver based on the subset selected by the receiver. The starting point for us to construct the CCBOT for Yao's protocol is the protocol of Wei et al. [14]. In the protocol of reference [14], b is the permutation bit of the sender S, and the value σ and τ are the choice-bit of the sender S and R, respectively. (x_0, x_1) are the garbled keys corresponding to 0 and 1 of S's input wires, and (r_0, r_1) are the garbled keys corresponding to 0 and 1 of R's input wires. Moreover, the receiver R has a choice-bit and indices set so that it can obtain the value of r corresponding to the choice-bit. Then it uses five tuples $(T_1, T_2, T_3, T_4, T_5)$ to transfer the values $(x_\sigma, x_{1-\sigma}, \sigma \oplus b, r_0)$ and r_1, respectively. After the transfer is complete, R either obtains the pairs (x_b, x_{1-b}), (r_0, r_1) or receives x_σ and r_τ. According to the value of j, R varies whether each of the five tuples is a DH tuple. If all the DH tuples have the same witness, S can obtain the corresponding values by the witness. To prevent the malicious R, here needs the zero-knowledge proof of the DH tuple to prove that the DH tuples generated by R are correct. We give the details of zero-knowledge proof of the DH tuple in the appendix.

It is worth noting that the CCBOT protocol constructed in reference [14] does not use the cut-and-choose technique. In order to apply the CCBOT protocol to Yao's protocol, the first step is to introduce the cut-and-choose technique into it, so a security parameter s is added. The essence of the CCBOT with the introduction of the cut-and-choose technique is to run s copies of the CCBOT protocol of reference [14]. Next, we will describe the functionality of CCBOT and improve it to single-choice CCBOT and batch single-choice CCBOT that we will use. As both single-choice CCBOT and batch single-choice CCBOT are based on the original CCBOT with certain restrictions and processing, these changes do not affect the security of the protocols, so we omit the security proofs for the following protocols, but give their functionality, detailed description, and approximate efficiency. In Figure 1, we formally define the functions of CCBOT, which are denoted by $\mathcal{F}_{\mathrm{ccbot}}$. In this protocol, there are $30s$ exponents and two rounds of communication.

For instantiating the CCBOT function, we describe its functionality in a secure protocol. Through $\mathcal{F}_{\mathrm{ccbot}}$, parts of these circuits are opened. P_2 obtains all the keys on the input wires of P_1 and P_2 in this part of the circuit. For the remaining circuit, P_2 receives the keys corresponding to its own input and the keys sent by P_1. For security, the order of the keys corresponding to P_2's input wires received by P_2 is replaced at random in check circuits. Obviously, P_2 can obtain the keys that are used to check circuits and evaluation circuits at one time, without the need for additional interactions to send additional keys and other proof of consistency.

CCBOT Functionality $\mathrm{F}_{\mathrm{ccbot}}$

Inputs:

— S inputs the vector of pairs $\{(x_0^j, x_1^j)\}_{i=1}^s, \{(r_0^j, r_1^j)\}_{i=1}^s$, permutation bit $b_1, \ldots, b_s \in \{0,1\}$, choice-bit $\sigma_1, \ldots, \sigma_s \in \{0,1\}$.

— R inputs choice-bit $\tau_1, \ldots, \tau_s \in \{0,1\}$ and the indices set $\mathrm{J} \subset [s]$.

Outputs:

— For every $j \in \mathrm{J}$, R receives the pairs $(x_{b_j}^j, x_{1-b_j}^j)$, (r_0^j, r_1^j), and the value of b_j.

— For every $j \notin \mathrm{J}$, R receives $x_{\sigma_j}^j$, $r_{\tau_j}^j$, and $\sigma_j \oplus b_j$.

Figure 1: The CCBOT functionality $\mathcal{F}_{\mathrm{ccbot}}$ between the sender and the receiver.

4.1. Single-Choice CCBOT. There will be a problem that the functionality $\mathcal{F}_{\mathrm{ccbot}}$ cannot guarantee that R inputs the identical choice-bit in the same input wire of all circuits. As we know, every bit of R's input needs an oblivious transfer. Therefore, it must be ensured that the inputs on the same input wire of R in all garbled circuits are identical. So we propose a variant of the original CCBOT protocol. Here, we use a single-choice CCBOT functionality $\mathcal{F}_{\mathrm{ccbot}}^S$ that is implemented by modifying the CCBOT protocol during the transmission phase for ensuring that R inputs the same choice-bit in each pair of tuples. Since R uses the same choice-bit in different tuples, then the key value associated with R's input wire only needs to be computed once so that this will reduce $s - 1$ exponents. As a result, in this protocol, there are $29s$ exponents and two rounds of communication.

In Figure 2, we formally define the protocol of single-choice CCBOT functionality $\mathcal{F}_{\mathrm{ccbot}}^S$, and a simple example is provided in Figure 3.

4.2. Batch Single-Choice CCBOT. Using the cut-and-choose approach requires the construction of s circuits. Single-choice CCBOT needs to be performed on all wires in every circuit. Hence, there we use an improvement of single-choice CCBOT called batch single-choice CCBOT. There are $15s\ell$ exponents and two rounds of communication in this protocol. In Figure 4, we formally define the protocol of batch single-choice CCBOT functionality $\mathcal{F}_{\mathrm{ccbot}}^{S,B}$, and a simple example is provided in Figure 5.

5. Secure Two-Party Computation Protocol

5.1. Protocol Description. The circuit constructor P_1 first locally constructs s garbled circuits as follows. P_1 chooses random value $a_1^0, a_1^1, \ldots, a_\ell^0, a_\ell^1$ and $r_1, \ldots, r_s$. Let the values $g^{a_i^0 \cdot r}$ and $g^{a_i^1 \cdot r}$ be the keys of P_1's input wire corresponding to 0 and 1 on the ith input wire in the jth circuit, respectively.

Single-Choice CCBOT Functionality $\mathrm{F}_{\mathrm{ccbot}}^{\mathrm{S}}$

Inputs:

— S inputs values $\{(x_0^j, x_1^j)\}_{j=1}^s$, $\{(r_0^j, r_1^j)\}_{j=1}^s$, permutation bit $b_1, \ldots, b_s \in \{0, 1\}$, and choice-bit $\sigma_1, \ldots, \sigma_s \in \{0,1\}$.

— R inputs choice-bit $\tau \in \{0,1\}$ and the indices set $\mathrm{J} \subset [s]$.

Outputs:

—For every $j \in \mathrm{J}$, R receives the pairs $(x_{b_j}^j, x_{1-b_j}^j)$, (r_0^j, r_1^j) and the value of b_j.

—For every $j \notin \mathrm{J}$, R receives $x_{\sigma_j}^j$, r_τ^j and $\sigma_j \oplus b_j$.

Figure 2: The single-choice CCBOT functionality $\mathcal{F}_{\mathrm{ccbot}}^{\mathrm{S}}$ between the sender and the receiver.

S's inputs are as follows, please note that it also input choice-bit σ_j and permutation bits b_j. R's inputs are τ and the indices set $\mathrm{J} \subset [s]$. R receives the values $(x_{b_j}^j, x_{1-b_j}^j)$ and (r_0^j, r_1^j) as well as the value b_j where $j \in \mathrm{J}$. For example, these are all pairs of S's inputs.

$$(x_0^1, x_1^1)(r_0^1, r_1^1) \quad (x_0^2, x_1^2)(r_0^2, r_1^2) \ldots (x_0^j, x_1^j)(r_0^j, r_1^j) \ldots (x_0^s, x_1^s)(r_0^s, r_1^s)$$

After the protocol, R receives the following values where $j \in \mathrm{J}$ and $\tau = 0$:

$$(x_{\sigma_1}^1)(r_0^1) \quad (x_{\sigma_1}^2)(r_0^2) \quad \ldots \quad (x_{b_j}^j, x_{1-b_j}^j)(r_0^j, r_1^j) \quad \ldots \quad (x_{\sigma_s}^s)(r_0^s)$$

Figure 3: A detailed description of $\mathcal{F}_{\mathrm{ccbot}}^{S}$ where $j \in \mathcal{J}$ and $\tau = 0$.

The keys associated with P_2's input wires are set to be $\vec{z}_1, \ldots, \vec{z}_\ell$. After two parties run the batch single-choice CCBOT protocol, P_2 has obtained all the keys ($\vec{z}_j$, $g^{a_i^{b_i} \cdot r_j}$, and $g^{a_i^{1-b_i} \cdot r_j}$ ($j \in \mathcal{J}$)) on the input wires of both parties in check circuits and the keys (z_{y_j} and $g^{a_i^{\sigma_i} \cdot r_j}$ ($j \notin \mathcal{J}$)) associated with the true inputs of both parties in evaluation circuits. Next, P_2 checks whether check circuits are correctly constructed and then decrypts evaluation circuits. If the output values of all evaluation circuits are inconsistent, P_2 will store the garbled values b_0 corresponding to 0 and b_1 corresponding to 1 for the output of this output wire of different circuits, respectively, and then P_1 and P_2 run a check protocol for preventing cheat. Essentially, P_1 and P_2 run the secure protocol of reference [9], and the circuits used in the secure protocol are for computing a bit-by-bit comparison function. If the check protocol is not aborted, P_2 obtains x and computes the function $f(x, y)$ locally.

Before giving a detailed protocol, it is worth noting that we have revised the garbled table (output translation table) commonly used in Yao's protocol. We respectively define k_i^0 as the garbled value corresponding to the bit 0, k_i^1 as the garbled value corresponding to the bit 1 of the wire i, and H as a hash function. Therefore, the garbled table on this wire is $[H(k_i^0), H(k_i^1)]$, where $k_i^0 \neq k_i^1$ and $H(k_i^0) \neq H(k_i^1)$. The complete protocol is shown in Figure 6.

5.2. Proof of Security. Here, we demonstrate the security of our protocol $\Pi_{2\mathrm{pc}}$ that is expressed as follows through the real/ideal simulation paradigm [35].

Theorem 1. *Assume that the batch single-choice CCBOT functionality in Section 4.2 is secure and the DDH assumption is hard. Then, the secure protocol in Figure 6 is secure for computing function $f(x, y)$ against the malicious adversary.*

Proof. We analyze our secure protocol $\Pi_{2\mathrm{pc}}$ in a hybrid model with a trusted party. The role of this trusted party is mainly to ensure the secure operation of batch single-choice CCBOT functionality of step 2 and the zero-knowledge proof of step 7. We prove the ability of our secure protocol against the malicious adversary in two parts: the first is the case of P_1 being corrupted, and the second is the case of P_2 being corrupted.

Batch Single-Choice CCBOT Functionality $\mathrm{F}_{\mathrm{ccbot}}^{\mathrm{S,B}}$

Inputs:

— S inputs ℓ vectors of pairs $\vec{x}_i$ and $\vec{r}_i$ of length s, for $i = 1, \ldots, \ell$

$\vec{x}_i = <(x_0^{i,1}, x_1^{i,1}), (x_0^{i,2}, x_1^{i,2}), \ldots, (x_0^{i,s}, x_1^{i,s})>$,

$\vec{r}_i = <(r_0^{i,1}, r_1^{i,1}), (r_0^{i,2}, r_1^{i,2}), \ldots, (r_0^{i,s}, r_1^{i,s})>$

S also inputs choice-bit σ_i and permutation bits $b_{i,j}$.

— R inputs $\tau_i \in \{0,1\}$ and the indices set $\mathrm{J} \subset [s]$, for $i = 1, \ldots, \ell$.

Outputs:

—For every $j \in \mathrm{J}$, R receives the jth pair of $\vec{x}_i$ and $\vec{r}_i$.(i.e., R receives $(x_{1-b_{i,j}}^{i,j}, x_{b_{i,j}}^{i,j})$, $(r_0^{i,j}, r_1^{i,j})$ and $b_{i,j}$)

—For every $j \notin \mathrm{J}$, R receives the value corresponding to σ_i in $\vec{x}_j$ and $\vec{r}_i$. (i.e., R receives $x_{\sigma_i}^{i,j}$, $r_{\tau_i}^{i,j}$ and $\sigma_i \oplus \tau_i$)

Figure 4: The batch single-choice CCBOT functionality $\mathcal{F}_{\mathrm{ccbot}}^{S,B}$ between the sender and the receiver.

Let the following matrix be the inputs of S.

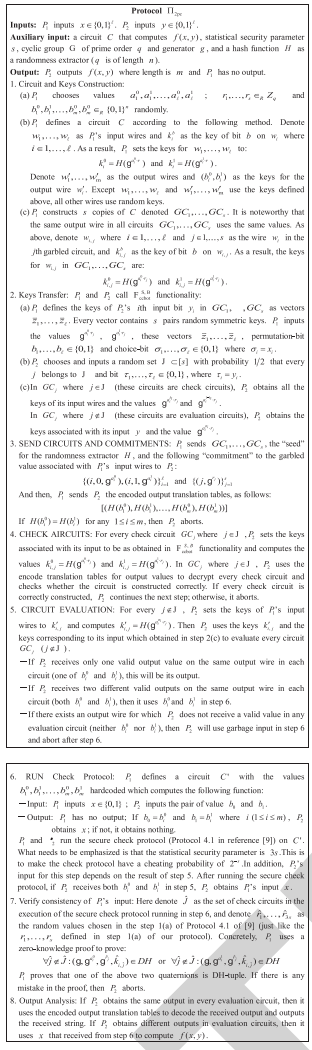
$$\begin{array}{cccc}
(x_0^{1,1}, x_1^{1,1})(r_0^{1,1}, r_1^{1,1}) & (x_0^{1,2}, x_1^{1,2})(r_0^{1,2}, r_1^{1,2}) & \cdots & (x_0^{1,s}, x_1^{1,s})(r_0^{1,s}, r_1^{1,s}) \\
(x_0^{2,1}, x_1^{2,1})(r_0^{2,1}, r_1^{2,1}) & (x_0^{2,2}, x_1^{2,2})(r_0^{2,2}, r_1^{2,2}) & \cdots & (x_0^{2,s}, x_1^{2,s})(r_0^{2,s}, r_1^{2,s}) \\
\vdots & \vdots & \vdots & \vdots \\
(x_0^{i,1}, x_1^{i,1})(r_0^{i,1}, r_1^{i,1}) & (x_0^{i,2}, x_1^{i,2})(r_0^{i,2}, r_1^{i,2}) & \cdots & (x_0^{i,s}, x_1^{i,s})(r_0^{i,s}, r_1^{i,s}) \\
\vdots & \vdots & \vdots & \vdots \\
(x_0^{\ell,1}, x_1^{\ell,1})(r_0^{\ell,1}, r_1^{\ell,1}) & (x_0^{\ell,2}, x_1^{\ell,2})(r_0^{\ell,2}, r_1^{\ell,2}) & \cdots & (x_0^{\ell,s}, x_1^{\ell,s})(r_0^{\ell,s}, r_1^{\ell,s})
\end{array}$$

For every $j \in \mathrm{J}$, R obtains both value of the vector $\vec{x}_i$ (the order of $(x_0^{i,j}, x_1^{i,j})$ depends on b_i) and $\vec{r}_i$. In addition, R receives one of the values of two vectors, depending on S's and R's inputs associated with that row. When $j \in \mathrm{J}$ and $\sigma_i = 0$. R obtains the following values.

$$\begin{array}{cccc}
(x_{\sigma_1}^{1,1})(r_{\tau_1}^{1,1}) & (x_{\sigma_1}^{1,2})(r_{\tau_1}^{1,2}) & \cdots & (x_{b_1}^{1,j}, x_{1-b_1}^{1,j})(r_0^{1,j}, r_1^{1,j}) \quad \cdots \quad (x_{\sigma_1}^{1,s})(r_{\tau_1}^{1,s}) \\
(x_{\sigma_2}^{2,1})(r_{\tau_2}^{2,1}) & (x_{\sigma_2}^{2,2})(r_{\tau_2}^{2,2}) & \cdots & (x_{b_2}^{2,j}, x_{1-b_2}^{2,j})(r_0^{2,j}, r_1^{2,j}) \quad \cdots \quad (x_{\sigma_2}^{2,s})(r_{\tau_2}^{2,s}) \\
\vdots & \vdots & \vdots & \vdots \\
(x_{\sigma_i}^{i,1})(r_{\tau_i}^{i,1}) & (x_{\sigma_i}^{i,2})(r_{\tau_i}^{i,2}) & \cdots & (x_{b_i}^{i,j}, x_{1-b_i}^{i,j})(r_0^{i,j}, r_1^{i,j}) \quad \cdots \quad (x_{\sigma_i}^{i,s})(r_{\tau_i}^{i,s}) \\
\vdots & \vdots & \vdots & \vdots \\
(x_{\sigma_\ell}^{\ell,1})(r_{\tau_\ell}^{\ell,1}) & (x_{\sigma_\ell}^{\ell,2})(r_{\tau_\ell}^{\ell,2}) & \cdots & (x_{\sigma_{iq}}^{\ell,j}, x_{1-b_\ell}^{\ell,j})(r_0^{\ell,j}, r_1^{\ell,j}) \quad \cdots \quad (x_{\sigma_\ell}^{\ell,s})(r_{\tau_\ell}^{\ell,s})
\end{array}$$

Figure 5: A detailed description of $\mathcal{F}_{\mathrm{ccbot}}^{S,B}$ where $j \in \mathcal{J}$ and $\sigma_i = 0$.

P_1 is corrupted: intuitively, one way for P_1 to cheat is by constructing the wrong circuit. In our secure protocol $\Pi_{2\mathrm{pc}}$, due to the check mechanism, if the output values of evaluation circuits are inconsistent, P_2 obtains the private input x of P_1 via the secure computation protocol of step 6. For causing the check mechanism to fail, the output values of all evaluation circuits must not differ. In that way, only when all check circuits are correctly constructed (check passes in step 4, and then P_2 decrypts evaluation circuits) and all evaluation circuits are not correctly constructed (because all evaluation circuits are wrong, no correct result output), P_1 can cheat. It is worth noting that after the batch single-choice CCBOT protocol of step 2 has been run and completed, whether a circuit is a check circuit or an evaluation circuit, whether it is correct, and whether it can be calculated has been determined. Of course, P_1 could also try to cheat by sending the incorrect values to the circuit input wires. However, these incorrect values do not pass the check of zero-knowledge proof, and the protocol is aborted. As a result, with the protocol in



Protocol Π_{2pc}

Inputs: P_1 inputs $x \in \{0,1\}^\ell$. P_2 inputs $y \in \{0,1\}^\ell$.

Auxiliary input: a circuit C that computes $f(x,y)$, statistical security parameter s, cyclic group G of prime order q and generator g, and a hash function H as a randomness extractor (q is of length n).

Output: P_1 outputs $f(x,y)$ where length is m and P_1 has no output.

1. Circuit and Keys Construction:

(a) P_1 chooses values $a_1^0, a_1^1, \ldots, a_\ell^0, a_\ell^1 : r_1, \ldots, r_\ell \in_R Z_q$ and $b_1^0, b_1^1, \ldots, b_m^0, b_m^1 \in_R \{0,1\}^n$ randomly.

(b) P_1 defines a circuit C according to the following method. Denote $w_1, \ldots, w_\ell$ as P_1's input wires and k_i^b as the key of bit b on w_i where $i = 1, \ldots, \ell$. As a result, P_1 sets the keys for $w_1, \ldots, w_\ell$ to:

$$k_i^b = H(g^{a_i^b}) \text{ and } k_i^b = H(g^{a_i^b}).$$

Denote $w_1', \ldots, w_m'$ as the output wires and (b_1^0, b_1^1) as the keys for the output wire w_i'. Except $w_1, \ldots, w_\ell$ and $w_1', \ldots, w_m'$ use the keys defined above, all other wires use random keys.

(c) P_1 constructs s copies of C denoted $GC_1, \ldots, GC_s$. It is noteworthy that the same output wire in all circuits $GC_1, \ldots, GC_s$ uses the same values. As above, denote $w_{i,j}$ where $i = 1, \ldots, \ell$ and $j = 1, \ldots, s$ as the wire w_i in the jth garbled circuit, and $k_{i,j}^b$ as the key of bit b on $w_{i,j}$. As a result, the keys for $w_{i,j}$ in $GC_1, \ldots, GC_s$ are:

$$k_{i,j}^b = H(g^{a_i^b \cdot r_j}) \text{ and } k_{i,j}^b = H(g^{a_i^b \cdot r_j}).$$

2. Keys Transfer: P_1 and P_2 call $F_{ABE}^{s,n}$ functionality:

(a) P_1 defines the keys of P_2's ith input bit y_i in $GC_1, \ldots, GC_s$ as vectors $\Xi_i, \ldots, \Xi_s$. Every vector contains s pairs random symmetric keys. P_2 inputs the values $g^{r_1}, \ldots, g^{r_s}$, these vectors $\Xi_1, \ldots, \Xi_s$, permutation-bit $b_1, \ldots, b_s \in \{0,1\}$ and choice-bit $\sigma_1, \ldots, \sigma_s \in \{0,1\}$ where $\sigma_i = y_i$.

(b) P_2 chooses and inputs a random set $J \subset [s]$ with probability 1/2 that every j belongs to J and bit $\tau_1, \ldots, \tau_s \in \{0,1\}$, where $\tau_i = y_i$.

(c) In GC_j where $j \in J$ (these circuits are check circuits), P_2 obtains all the keys of its input wires and the values g^{r_j} and $g^{a_i^{\tau_i} \cdot r_j}$.

In GC_j where $j \notin J$ (these circuits are evaluation circuits), P_2 obtains the keys associated with its input y and the value $g^{a_i^{\tau_i} \cdot r_j}$.

3. SEND CIRCUITS AND COMMITMENTS: P_1 sends $GC_1, \ldots, GC_s$, the "seed" for the randomness extractor H, and the following "commitment" to the garbled value associated with P_1's input wires to P_2:

$$\{(i, 0, g^{a_i^0}), (i, 1, g^{a_i^1})\}_{i=1}^{\ell} \text{ and } \{(j, g^{r_j})\}_{j=1}^{s}$$

And then, P_1 sends P_2 the encoded output translation tables, as follows:

$$\{(H(b_1^0), H(b_1^1)), \ldots, (H(b_m^0), H(b_m^1))\}$$

If $H(b_i^0) = H(b_i^1)$ for any $1 \le i \le m$, then P_2 aborts.

4. CHECK AIRCUITS: For every check circuit GC_j where $j \in J$, P_1 sets the keys associated with its input to be as obtained in $F_{ABE}^{s,n}$ functionality and computes the values $k_{i,j}^b = H(g^{a_i^b \cdot r_j})$ and $k_{i,j}^b = H(g^{a_i^b \cdot r_j})$. In GC_j where $j \in J$, P_2 uses the encode translation tables for output values to decrypt every check circuit and checks whether the circuit is constructed correctly. If every check circuit is correctly constructed, P_2 continues the next step; otherwise, it aborts.

5. CIRCUIT EVALUATION: For every $j \notin J$, P_2 sets the keys of P_1's input wires to $k_{i,j}^{\tau_i}$ and computes $k_{i,j}^{\tau_i} = H(g^{a_i^{\tau_i} \cdot r_j})$. Then P_2 uses the keys $k_{i,j}^{\tau_i}$ and the keys corresponding to its input which obtained in step 2(c) to evaluate every circuit GC_j ($j \notin J$).

- If P_2 receives only one valid output value on the same output wire in each circuit (one of b_i^0 and b_i^1), this will be its output.
- If P_2 receives two different valid outputs on the same output wire in each circuit (both b_i^0 and b_i^1), then it uses $b_i^0$and b_i^1 in step 6.
- If there exists an output wire for which P_2 does not receive a valid value in any evaluation circuit (neither b_i^0 nor b_i^1), then P_2 will use garbage input in step 6 and abort after step 6.

6. RUN Check Protocol: P_1 defines a circuit C' with the values $b_1^0, b_1^1, \ldots, b_m^0, b_m^1$ hardcoded which computes the following function:

- Input: P_1 inputs $x \in \{0,1\}$; P_2 inputs the pair of value b_0 and b_1.
- Output: P_1 has no output; If $b_0 = b_i^0$ and $b_1 = b_i^1$ where i ($1 \le i \le m$), P_2 obtains x; if not, it obtains nothing.

P_1 and P_2 run the secure check protocol (Protocol 4.1 in reference [9]) on C'. What needs to be emphasized is that the statistical security parameter is $3s$.This is to make the check protocol have a cheating probability of 2^{-s}.In addition, P_2's input for this step depends on the result of step 5. After running the secure check protocol, if P_2 receives both b_i^0 and b_i^1 in step 5, P_2 obtains P_1's input x.

7. Verify consistency of P_1's input: Here denote $\hat{J}$ as the set of check circuits in the execution of the secure check protocol running in step 6, and denote $\hat{r}_1, \ldots, \hat{r}_{3s}$ as the random values chosen in the step 1(a) of Protocol 4.1 of [9] (just like the $r_1, \ldots, r_s$ defined in step 1(a) of our protocol). Concretely, P_1 uses a zero-knowledge proof to prove:

$$\forall j \in \hat{J} : (g, g^{r_j}, g^{\hat{r}_j}, \hat{k}_{i,j}^{\tau_i}) \in DH \text{ or } \forall j \notin \hat{J} : (g, g^{r_j}, g^{\hat{r}_j}, \hat{k}_{i,j}^{\tau_i}) \in DH$$

P_1 proves that one of the above two quaternions is DH-tuple. If there is any mistake in the proof, then P_2 aborts.

8. Output Analysis: If P_2 obtains the same output in every evaluation circuit, then it uses the encoded output translation tables to decode the received output and outputs the received string. If P_2 obtains different outputs in evaluation circuits, then it uses x that received from step 6to compute $f(x,y)$.

Figure 6: Protocol of computing $f(x,y)$.

Figure 6, P_2 always obtains the final result $f(x,y)$ except in the case where all check circuits are correctly constructed and all evaluation circuits are incorrectly constructed, either because all the evaluation circuits are correct and output the same result or because P_1 cheats so that P_2 learns P_1's input x. Through the above analysis, below, we provide the formal proof.

We define $\mathcal{A}$ as the adversary controlling P_1 during the execution of the secure protocol. In the ideal world, there is a simulator $\mathcal{S}$ and a trusted party f. Simulator $\mathcal{S}$ plays the role of the real-world adversary $\mathcal{A}$ and executes the secure protocol with the trusted party f and P_2. Here is how $\mathcal{S}$ would work in the ideal world:

(1) $\mathcal{S}$ obtains the inputs that the adversary $\mathcal{A}$ inputs to the trusted party for the batch single-choice CCBOT functionality. These inputs form a $l \times s$ matrix of pairs $\left\{(x_0^{i,j}, x_1^{i,j}), (z_0^{i,j}, z_1^{i,j})\right\}$, where $i = 1, \ldots, \ell$ and $j = 1, \ldots, s$. $\mathcal{S}$ play the honest P_2 with input $y = 0^\ell$.

(2) $\mathcal{S}$ receives s copies of the circuit $GC_1, \ldots, GC_s$ that send by $\mathcal{A}$ and the values $\left\{(i, 0, g^{a_i^0}), (i, 1, g^{a_i^1})\right\}_{i=1}^{\ell}$ and $\left\{(j, g^{r_j})\right\}_{i=1}^{s}$.

(3) Under the restriction that the probability that j belongs to $\mathcal{J}$ is one half, $\mathcal{S}$ randomly selects a subset $\mathcal{J} \subset [s]$ and sends $\left\{(x_0^{i,j}, x_1^{i,j}), (z_0^{i,j}, z_1^{i,j})\right\}$ to $\mathcal{A}$, where $j \in \mathcal{J}$.

(4) $\mathcal{S}$ obtains all the keys of P_1 and P_2's input wires when $j \in \mathcal{J}$. Then, $\mathcal{S}$ decrypts the circuit through these keys and judges whether it is constructed correctly. Otherwise, $\mathcal{S}$ send $\perp$ to the trusted party and simulates aborting.

(5) Let $x = x_1, \ldots, x_\ell$ be the witness of the zero-knowledge proof in step 7. $\mathcal{S}$ receives $x = x_1, \ldots, x_\ell$ from $\mathcal{A}$. If one of the $x_1, \ldots, x_\ell$ is invalid, simulates aborting. Otherwise, $\mathcal{S}$ obtains x and sends it to the trusted party.

There we denote our secure protocol by π. We can conclude that

$$\left\{IDEAL_{f,S}(x, y, z, n, s)\right\}_{x,y,z \in \{0,1\}^*; n,s \in \mathbb{N}} \overset{n,s}{\equiv} \left\{\mathrm{REAL}_{\pi,\mathcal{A}}(x, y, n, s)\right\}_{x,y,z \in \{0,1\}^*; n,s \in \mathbb{N}}, \quad (3)$$

where $x| = |y|$.

Now, we formally calculate the probability of successful cheating for P_1. Let badAll be the case where all evaluation circuits are wrongly constructed and noAbort be the case where P_2 is not aborted in step 4. □

Claim 1. For every $s \in \mathbb{N}$, it holds that

$$\Pr[\text{noAbort} \wedge \text{badAll}] = \left(\frac{1}{2}\right)^s = 2^{-s}. \quad (4)$$

For proof of security, the joint distribution must be indistinguishable, that is, the joint distribution consisting of the outputs of $\mathcal{A}$ and P_2 of the real execution is indistinguishable from the joint distribution consisting of the outputs of $\mathcal{S}$ and P_2 of the ideal execution. It should be noted that since the input of P_2 in the simulation is $y = 0^\ell$, which is different from the input to P_2 in the real world. This may lead to a different probability of $\mathcal{S}$ aborting in the simulation than in the reality of P_2. In addition, We also need to show that if P_2 does not abort in a real protocol execution, then the probability that it also outputs an error result is $2^{-s} + \varepsilon(\cdot)$, where $\varepsilon(\cdot)$ is a negligible function.

Here, we define good and bad circuits. From the whole protocol, we can know whether a circuit is good or bad has been determined in step 3, and it is important to note that a circuit is either good or bad. We claim that a circuit is bad if it cannot be decrypted using the keys $[H(g^{a_1^0 \cdot r_j}), H(g^{a_1^1 \cdot r_j}), \ldots, H(g^{a_\ell^0 \cdot r_j}), H(g^{a_\ell^0 \cdot r_j})]$ of P_1's input

and the keys corresponding to P_2's input ($[(z_0^{1,j}, z_1^{1,j}), \ldots, (z_0^{\ell,j}, z_1^{\ell,j})]$) to get the correct output. These key values are part of the P_1's inputs to the batch single-choice CCBOT.

We now prove the (n, s) – indistinguishability according to claim 1. In fact, as long as the event (noAbort∧badAll) does not occur, the views of the ideal and real world are identically distributed. Firstly, we discuss that if there is a check circuit that is **bad**, then P_2 will detect and abort the protocol. Next, if all check circuits are **good** and at least one **good** circuit exists in the remaining evaluation circuits, and P_2 is not aborted during the entire protocol execution, then P_2 has the same output distribution in the real and simulated execution. This is due to the fact that if one of the evaluation circuits is **good**, then P_2 must be able to get the correct output from this circuit. And if a **bad** circuit outputs a result different from the correct output $f(x, y)$, P_2 must receive two different valid outputs on the same output wire in different evaluation circuits and saves a "proof", then P_2 will obtain P_1's private input x in step 6. Because the secure computation protocol in [9] has an error probability of $2^{-0.311s}$, so we use $3s$ to guarantee an error probability of $2^{-s} + \varepsilon(\cdot)$ for step 6, As a result, P_2 either obtains x or aborts the protocol in step 6. Finally, there is a case where an output wire does not output a valid value in all the evaluation circuits; then P_2 aborts. This is because in this case all the evaluation circuits are wrong and P_2 will continue with step 6. If there is a **bad** check circuit in step 6, then P_2 aborts; conversely, though all check circuits are **good**, P_2 still aborts because no valid value is received. Here, it may differ from the simulation, as it depends on the input of P_2. We conclude that P_2 obtains $f(x, y)$ in real and ideal executions as long as no aborts occur, which is exactly the same as the view of $\mathcal{A}$ in the execution.

From the above analysis, we know that the adversary cannot know whether the output $f(x, y)$ of P_2 is due to all evaluation circuits being correct so the outputs are the same or due to P_2 detecting cheating and then learning P_1's private input x. In addition, the adversary also does not know whether P_2 aborts the protocol because P_1 find that the check circuit is **bad** or because all the evaluation circuits do not output the same result. As a result, if and only if all check circuits are **good** and all evaluation circuits are **bad**, the simulation deviates from the real output distribution. Since the probability of each circuit being **good** or **bad** is exactly 1/2, the probability of P_1 cheating successfully is 2^{-s}.

P_2 is corrupted: In this case, since P_1 is honest, all circuits are **good,** and a circuit is either a check circuit or an evaluation circuit. P_2 will only obtain the same output. In other words, P_2 can only obtain the same value on the same output wire of different evaluation circuits. The security here is guaranteed by the DDH assumption. However, a malicious P_2 may cheat and want to obtain the private input x of P_1 by the following method. P_2 declares that it has obtained two different values on the same output wire of different evaluation circuits and then forges these two different values b_0 and b_1 in step 6 in order to obtain the value of x. In this case, since the hardcoded circuit is random and the computation protocol is secure in step 6, the probability of successful forgery is negligible. Below, we provide the formal proof.

We define $\mathcal{A}$ as the adversary controlling P_2 during the execution of the secure protocol Π_{2pc}. In the ideal world, there is a simulator $\mathcal{S}$ and a trusted party f. Simulator $\mathcal{S}$ plays the role of the real-world adversary $\mathcal{A}$ and executes the secure protocol with the trusted party f and $\mathcal{S}$. Here is how $\mathcal{S}$ would work in the ideal world:

(1) $\mathcal{S}$ obtains P_2's inputs that include $y = \sigma_1, \ldots, \sigma_\ell$ and a set $\mathcal{J} \subset [s]$ from $\mathcal{A}$.

(2) $\mathcal{S}$ sends $y = \sigma_1, \ldots, \sigma_\ell$ to the trusted party and obtains the output $z = f(x, y)$.

(3) $\mathcal{S}$ randomly selects a pair of values (b_0, b_1) and uses them as the input of P_2 in step 6.

(4) If $b_0 = b_i^0$ and $b_1 = b_i^1$, then simulation aborted. Otherwise, the simulation continues according to the protocol of step 6 and then ends.

We can conclude that

$$\left\{\text{IDEAL}_{f,S}(x, y, z, n, s)\right\}_{x,y,z\in\{0,1\}^*;n,s\in\mathbb{N}}{}^{n,s} \equiv \left\{\text{REAL}_{\pi,\mathcal{A}}(x, y, n, s)\right\}_{x,y,z\in\{0,1\}^*;n,s\in\mathbb{N}}, \tag{5}$$

where $x| = |y|$.

We know that the probability of P_2 successfully forging b_0 and b_1 is negligible. In the following, we need to prove that when the above situation does not exist, that is, when P_2 forgery fails, the output distribution of the real world and the simulated world are indistinguishable. In the ideal world, the simulation aborts if P_2 is successfully faked. Conversely, if the forgery fails and the simulation continues, then the secure computing protocol in step 6 has no output, that is, P_2 cannot obtain the private input of P_1 (P_2) only obtains $f(x, y)$ from the trusted party f. In the real world, P_2 obtains $f(x, y)$ by decrypting the circuit, so the output distribution of the ideal world and the real world are indistinguishable.

5.3. Efficiency. As shown in Table 1, we analyze the approximate efficiency of the entire protocol, including the exponentiations, symmetric encryptions, group elements sent, and communication rounds. According to the form, $x^a \cdot y^b$ only need 1.25 standard exponentiations; we calculate the exponentiations for each step: (1) $2s\ell$ for the input key choice and circuit preparation in step 1, (2) $15s\ell$ for the batch single-choice CCBOT of step 2, (3) $9s\ell$ for the secure check protocol of step 6, and (4) $2s\ell$ for verifying the consistency of P_1's input in step 7. In total, there are $28s\ell$ in our protocol.

For the symmetric encryptions, we calculate as follows: $8|C|$ for each circuit construction (s circuits, total $8|C|s$). After running step 2, there are roughly $s/2$ check circuits and

Table 1: Efficiency analysis of each step of our protocol.

Step	Exponentiations	Symmetric encryptions	Group elements sent	Communication rounds of $\mathcal{F}_{\text{ccbot}}^{S,B}$/ $\mathcal{F}_{\text{ccbot}}^{S,B}$ with zero-knowledge proof
1	$2s\ell$	$8\|C\|s$		
2	$15s\ell$		$11s\ell$	
3				
4		$s/2 \cdot 8\|C\|$		
5		$s/2 \cdot 2\|C\|$		2/6
6	$9s\ell$	$39s\ell$		
7	$2s\ell$		$21s\ell$	
8				
Total	$28s\ell$	$13\|C\|s + 39s\ell$	$32s\ell$	

Protocol $\Pi_{\text{ZKPK-DH tuples}}$

Joint statement: The value (G, g_0, g_1, u, v) are elements of a group G which has an order q, and a generator g.

Auxiliary input for the prover: A witness w such that $u = (g_0)^w$ and $v = (g_1)^w$.

The protocol:

1. P chooses a random $a \in \{1,\ldots,q\}$ and computes $\alpha = (g_0)^a$ and send α to V.
2. The verifier V choose random $s,t \in \{1,\ldots,q\}$, computes $c = (g_0)^s \cdot \alpha^t$ and sends c to P (this is a perfectly-hiding commitment to s).
3. P chooses a random $r \in \{1,\ldots,q\}$ and computes $A = (g_0)^r$ and $B = (g_1)^r$. It then sends (A, B) to V.
4. V sends s and t as above to P.
5. P verifies that $c = (g_0)^s \cdot \alpha^t$. If no, it aborts. Otherwise, it sends $z = s \cdot w + r$ to V. In addition, it sends a as chosen above.

V accepts if and only if $\alpha = (g_0)^a$, $A = (g_0)^z / u^s$ and $B = (g_1)^z / v^s$.

Figure 7: Protocol of zero-knowledge proof for DH tuples.

Protocol $\Pi_{\text{ZKPK-EDH tuples}}$

Common input: $(g_0, g_1, h_0, h_1, x_1, y_1, \ldots, x_n, y_n)$

Prover witness: for every i, there exist a constant a which satisfies either $h_0 = (g_0)^a$ and $x_i = (y_i)^a$ or $h_1 = (g_1)^a$ and $x_i = (y_i)^a$.

The protocol:

1. V picks $\mu_1,\ldots,\mu_n \in_R \{0,1\}^L$ where $2^L < q$, and sends $\mu_1,\ldots,\mu_n$ to P.
2. V and P locally compute $x = \Pi_{i=1}^n (x_i)^{\mu_i}$ and $y = \Pi_{i=1}^n (y_i)^{\mu_i}$
3. P uses protocol $\Pi_{\text{ZKPK-DH tuples}}$ to prove that either (g_0, h_0, x, y) or (g_1, h_1, x, y) is a DH tuple. V accepts if and only it accepts in $\Pi_{\text{ZKPK-DH tuples}}$.

Figure 8: Protocol of zero-knowledge proof for extended DH tuples.

$s/2$ evaluation circuits. A total of $s/2 \cdot 8|C| + s/2 \cdot 2|C| = 5|C|s$ symmetric encryptions are required for check and evaluation. In addition, the error probability of the security computation protocol [9] used by the check mechanism of step 6 is $2^{-0.311s}$. Therefore, $3s$ circuits are required so that keep the error probability at 2^{-s} and need $3 \times 13 \cdot s \cdot \ell = 39s\ell$. In total, there has $8|C|s + 5|C|s + 39s\ell = 13|C|s + 39s\ell$ symmetric encryptions.

6. Conclusion

In this paper, we propose an improvement of CCBOT that is called batch single-choice CCBOT. We formalize our batch single-choice CCBOT function and apply it to Yao's protocol and then present a new secure two-party computation protocol that achieves security against the malicious adversary. Our new protocol not only can reduce the number of interactive rounds of the protocol but also can solve the problem of input consistency and prevent selective failure attacks. In addition, a check mechanism is used to prevent P_1 from cheating but P_2 is powerless and the probability of error for P_2 is 2^{-s}.

In the future, we will continue to study common secure computation protocols, focusing on efficient constant-round secure multiparty based on garbled circuits in the malicious model.

Appendix

A. Zero-Knowledge Proof for Diffie–Hellman Tuples

We show the zero-knowledge proof for DH tuples in Figure 7. There are 12 exponentiations, including 8 of $x^a \cdot y^b$. Then, all the form $x^a \cdot y^b$ costs $8 \times 1.25 = 10$ exponentiations. Concretely, the zero-knowledge proof for DH tuples costs 11 exponentiations and 5 rounds of communication.

B. Zero-Knowledge Proof for Extended Diffie–Hellman Tuples

We show the zero-knowledge proof for extended DH tuples in Figure 8 that is used in protocol Π_{2pc}. According to the previous work [9], there are $n + 18$ exponentiations and five rounds of communication.

Data Availability

No data were used to support this study.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Acknowledgments

The authors would like to thank their supervisor for always helping and supporting them. This work was supported by the Support Plan of the Scientific and Technological Innovation Team in Universities of Henan Province (No.

20IRTSTHN013), the Youth Talent Support Program of Henan Association for Science and Technology (No. 2021HYTP008), the Fundamental Research Funds for the Universities of Henan Province (No. NSFRF210312), and the PhD Foundation of Henan Polytechnic University (No. B2021-41).

References

[1] Y. Lindell, "Secure multiparty computation," *Communications of the ACM*, vol. 64, no. 1, pp. 86–96, 2021.

[2] A. C. Yao, "How to generate and exchange secrets," in *Proceedings of the 27th Annua Symposium on Foundations of Computer Science*, pp. 162–167, IEEE, Toronto, Canada, July1986.

[3] A. Shamir, "How to share a secret," *Communications of the ACM*, vol. 22, no. 11, pp. 612-613, 1979.

[4] B. Pinkas, "Fair secure two-party computation," in *Proceedings of the Int Conf on the Theory and Applications of Cryptographic Techniques*, pp. 87–105, Berlin, Heidelberg, May 2003.

[5] M. Kiraz and B. Schoenmakers, "A protocol issue for the malicious case of Yao's garbled circuit construction," *27th Symposium on Information Theory in the Benelux*, vol. 29, no. 4, pp. 283–290, 2006.

[6] V. Kolesnikov and R. Kumaresan, "On cut-and-choose oblivious transfer and its variants," in *Proceedings of the Advances in Cryptology ASIACRYPT 2015*, pp. 386–412, Berlin, Heidelberg, January 2015.

[7] X. Wei, L. Xu, M. Zhao, and H. Wang, "Secure extended wildcard pattern matching protocol from cut-and-choose oblivious transfer," *Information Sciences*, vol. 529, no. 11, 2020.

[8] X. Wei, L. Xu, H. Wang, and Z. Zheng, "Permutable Cut-And-Choose Oblivious Transfer and its Application," *IEEE Access*, vol. 8, pp. 17378–17389, 2020.

[9] Y. Lindell and B. Pinkas, "Secure Two-Party Computation via Cut-and-Choose Oblivious Transfer," *Theory of Cryptography*, vol. 25, no. 4, pp. 329–346, 2011.

[10] Y. Huang, J. Katz, V. Kolesnikov, R. Kumaresan, and A. J Malozemoff, "Amortizing garbled circuits," in *Proceedings of the Annual Cryptology Conferance*, pp. 458–475, Berlin, Heidelberg, May 2014.

[11] C. Hazay and Y. Lindell, *Efficient Secure Two-Party Protocols: Techniques and Constructions*, Information Security and Cryptography, Berlin, Heidelberg, pp. 3–254, 2010.

[12] Y. Lindell and B. Riva, *Cut-and-Choose Yao-Based Secure Computation in the Online/Offline and Batch Settings*, CRYPTO, Berlin, Heidelberg, pp. 476–494, 2014.

[13] C. Zhao, H. Jiang, X. Wei, Q. Xu, and M. Zhao, "Cut-and-choose bilateral oblivious transfer and its application," in *Proceedings of the 14th IEEE International Conference on Trust, Security and Privacy in Computing and Communications*, pp. 384–391, Helsinki. Finland, August 2015.

[14] X. Wei, H. Jiang, C. Zhao, M. Zhao, and Q. Xu, "Fast cut-and-choose bilateral oblivious transfer for malicious adversaries," in *Proceedings of the IEEE International Conference on Big Data Science and Engineering*, pp. 418–425, Tianjin, China, Februray 2016.

[15] M. Naor and O. Reingold, *Synthesizers and their application to the parallel construction of psuedo-random functions*, vol. 58, pp. 170–181, Journal of Computer and System Sciences, 1995.

[16] Y. Lindell, "Fast cut-and-choose-based protocols for malicious and covert adversaries," *Journal of Cryptology*, vol. 29, no. 2, pp. 456–490, 2016.

[17] Y. Huang, D. Evans, J. Katz, and L. Malka, "Faster Secure Two-Party Computation Using Garbled Circuits," in *Proceedings of the USENIX Security Symposium*, Boston, MA, USA, August 2011.

[18] O. Goldreich, S. Micali, and A. Wigderson, "How to play any mental game or A completeness theorem for protocols with honest majority," in *Proceedings of the nineteenth annual ACM symposium on Theory of computingThe 19th STOC*, pp. 218–229, New York, NY, USA, January 1987.

[19] A. Ben-Efraim, Y. Lindell, and E. Omri, "Efficient Scalable Constant-Round MPC via Garbled Circuits," in *Proceedings of the International Conference on the Theory and Application of Cryptology and Information Security*, pp. 471–498, ASIACRYPT, Cham, November 2017.

[20] P. Mohassel and B. Riva, "Garbled circuits checking garbled circuits: more efficient and secure two-party computation," in *Proceedings of the Annual Cryptology ConferenceCRYPTO*, pp. 36–53, Berlin, Heidelberg, July 2013.

[21] V. Kolesnikov and T. Schneider, *Improved Garbled Circuit: Free XOR Gates and Applications*, pp. 486–498, ICALP, Berlin, Heidelberg, 2008.

[22] N. Büscher, M. Franz, A. Holzer, H. Veith, and S. Katzenbeisser, "On compiling Boolean circuits optimized for secure multi-party computation," *Formal Methods in System Design*, vol. 51, no. 2, pp. 308–331, 2017.

[23] O. Biçer, "Efficiency Optimizations on Yao's Garbled Circuits and Their Practical Applications," 2017, https://arxiv.org/abs/1703.03473.

[24] Y. Lindell and B. Pinkas, "An efficient protocol for secure two-party computation in the presence of malicious adversaries," in *Proceedings of the Annual Int Conf on the Theory and Applications of Cryptographic Techniques*, pp. 52–78, Berlin, Heidelberg, May 2007.

[25] A. Shelat and C. H. Shen, "Two-output secure computation with malicious adversaries," in *Proceedings of the In Annual International Conference on the Theory and Applications of Cryptographic TechniquesEUROCRYPT*, pp. 386–405, Berlin, Heidelberg, 2011.

[26] Y. Huang, J. Katz, and D. Evans, "Efficient secure two-party computation using symmetric cut-and-choose," in *Proceedings of the Annual Cryptology ConferenceCRYPTO*, pp. 1–17, Berlin, Heidelberg, 2013.

[27] C. Chen, "Efficient Three-Party Computation: An Information-Theoretic Approach from Cut-And-Choose," 2019, https://arxiv.org/abs/1908.03718.

[28] T. K. Frederiksen and J. B. Nielsen, "Fast and Maliciously Secure Two-Party Computation Using the GPU," in *Proceedings of the International Conference on Applied Cryptography and Network Security*, Cham, January 2013.

[29] A. Afshar, P. Mohassel, B. Pinkas, and B. Riva, "Non-interactive secure computation based on cut-and-choose," in *Proceedings of the Annual International Conference on the Theory and Applications of Cryptographic Techniques*, Berlin, Heidelberg, 2014.

[30] L. T. A. N. Brandao, "Secure two-party computation with reusable bit-commitments, via a cut and-choose with forge-and-lose technique," in *Proceedings of the Int Conf on the Theory and Application of Cryptology and Information Security*, pp. 441–463, Berlin, Heidelberg, December 2013.

[31] B. Pinkas, T. Schneider, N. Smart, and S. Williams, "Secure two-party computation is practical," in *Proceedings of the*

International Conference on the Theory and Application of Cryptology and Information SecurityASIACRYPT, pp. 250–267, Berlin, Heidelberg, June 2009.

[32] Y. Lindell and B. Pinkas, "An efficient protocol for secure two-party computation in the presence of malicious adversaries," in *Proceedings of the 26th Annual International Conference on Advances in Cryptology, EUROCRYPT*, vol. 28, no. 2, pp. 52–78, April 2015.

[33] X. Wang, A. J. Malozemoff, and J. Katz, "Faster secure two-party computation in the single execution setting," in *Proceedings of the Annual Int Conf on the Theory and Applications of Cryptographic Techniques*, pp. 399–424, Cham, April 2017.

[34] L. Ning and J. Wang, "A novel bilateral oblivious transfer protocol based on cut-and-choose technique," in *Proceedings of the IEEE 14th International Conference on Big Data Science and Engineering*, Guangzhou, China, December 2020.

[35] R. Canetti, "Security and Composition of Multiparty Cryptographic Protocols," *Journal of Cryptology*, vol. 13, no. 1, pp. 143–202, 2000.