Efficiently Making Secure Two-Party Computation Fair

Handan Kılınç[1] and Alptekin Küpçü[2]

[1] EPFL, Koç University
handan.kilinc@epfl.ch
[2] Koç University
akupcu@ku.edu.tr

Abstract. Secure two-party computation cannot be fair against malicious adversaries, unless a trusted third party (TTP) or a gradual-release type super-constant round protocol is employed. Existing optimistic fair two-party computation protocols with constant rounds are either too costly to arbitrate (e.g., the TTP may need to re-do almost the whole computation), or require the use of electronic payments. Furthermore, most of the existing solutions were proven secure and fair via a partial simulation, which, we show, may lead to insecurity overall. We propose a new framework for fair and secure two-party computation that can be applied on top of any secure two party computation protocol based on Yao's garbled circuits and zero-knowledge proofs. We show that our fairness overhead is minimal, compared to all known existing work. Furthermore, our protocol is fair even in terms of the work performed by Alice and Bob. We also prove our protocol is fair and secure simultaneously, through one simulator, which guarantees that our fairness extensions do not leak any private information. Lastly, we ensure that the TTP never learns the inputs or outputs of the computation. Therefore, even if the TTP becomes malicious and causes unfairness by colluding with one party, the security of the underlying protocol is still preserved.

1 Introduction

In two-party computation (2PC), Alice and Bob intend to evaluate a shared function with their private inputs. The computation is called secure when the parties do not learn anything beyond what is revealed by the output of the computation. Yao [38] introduced the concept of secure 2PC and gave an efficient protocol; but this protocol is not secure against malicious parties who try to learn extra information from the computation by *deviating* from the protocol. Many solutions [31, 37, 19, 26] are suggested to strengthen Yao's protocol against malicious adversaries.

When one considers malicious adversaries, fairness is an important problem. A fair computation should guarantee that Alice learns the output of the function *if and only if* Bob learns. This problem occurs since in the protocol one party learns the output earlier than the other party; therefore (s)he can abort the protocol after learning the output, before the other party learns it.

There are two main methods of achieving fairness in 2PC: using gradual release [33, 23, 34] or a trusted third party (TTP) [6, 25]. The ***gradual release*** based protocols [12, 5, 2] let the parties gradually (bit by bit, or piece by piece) and verifiably reveal the result. Malicious party will have one bit (or piece) advantage if the honest party starts to reveal the result first. Yet, if the malicious party has more computational power, he can abort the protocol earlier and learn the result via brute force, while the honest party cannot. In this case, fairness is not achieved. Another drawback is the necessity of many rounds.

The ***TTP approach*** employs a third party that is trusted by both Alice and Bob. A simple solution would be to give the inputs to the TTP, who computes the outputs and distributes fairly. In terms of efficiency and feasibility though, the TTP should be used in the ***optimistic*** model [1], where he gets involved in the protocol *only* when there is a dispute between Alice and Bob. It is very important to give the TTP the minimum possible workload because otherwise the system will have a bottleneck. Another important concern is ***privacy***. In an optimistic solution, if there is no dispute, the TTP should not even know a computation took place, and even with a dispute, the TTP should never learn the inputs or outputs, or even identities. **We achieve all these efficiency and privacy requirements on the TTP.**

Another problem regarding fairness in secure two-party computation is the **proof methodology**. In previous works [23, 6, 34], fairness and security (with abort) were proven *separately*, only partially simulating the protocol (*partial simulation*). However, it is important to simulate everything together to ensure that the fairness solution *does not leak* any information beyond the original secure two-party computation requirement. Therefore, as in the security of the secure two-party computation, there should be ideal/real world simulation (see Section 2) that covers **both fairness and security** (*full simulation*). In other words, **the simulator should learn the output in the real world only after it is guaranteed that both parties can learn the output in the real world** to achieve ideal and real world indistinguishability of the outputs.

Our Contributions: The main achievement of this work is an efficient framework for making secure 2PC protocols fair, such that it guarantees fairness and security together, and can work on top of secure two party computation protocols extending Yao's garbled circuits to the malicious setting via zero-knowledge proofs (e.g., [19, 6, 15]). Note that the state-of-the-art optimistic fairness solution [6] is also based on zero-knowledge proofs.

- We use a simple-to-understand ideal world definition to achieve fairness and security together, and **prove our protocol's security and fairness with full simulation** which means proving security and fairness together.
- We show that proving security and fairness separately via only partial simulation is not necessarily secure (see Section 5).
- Our framework employs a trusted third party (TTP) for fairness, in the *optimistic* model. The **TTP's load is very light**: verification of signatures and commitments, and decryption only. If there is no dispute, the TTP does *not* even know a computation took place, and even with a dispute, the TTP *never* learns the inputs, outputs, or even identities of Alice and Bob. So, a semi-honest TTP is enough in our construction to achieve fairness.

- If the **TTP becomes malicious** (e.g., colludes with one of the parties), it **does not violate the security** of the underlying 2PC protocol; only the fairness property of the protocol is contravened.
- Our framework is also fair about the work done by Alice and Bob, since both of them perform the same steps in the protocol.
- The principles for fairness in our framework can be adopted by any 2PC protocol based on Yao's garbled circuits, employing zero knowledge proofs for the malicious setting, thereby achieving fairness with little overhead.
- We compare our framework with related fair secure two-party computation work and show that we achieve better efficiency and security.

Related Works: Cachin and Camenisch [6] present a state-of-the-art fair two-party computation protocol in the optimistic model. The protocol consists of two intertwined verifiable secure function evaluations. In the case of an unfair situation, the honest party interacts with the TTP. **The job of the TTP** can be as bad as almost repeating the whole computation, **linear in the circuit size**, creating a bottleneck in the system. Lindell [25] constructs a framework that can be adopted by any two-party functionality with the property that either both parties receive the output, or one party receives the output while the other receives a digitally-signed check (i.e., monetary compensation). However, one may argue that one party obtaining the output and the other obtaining the money may not always be considered fair, since *we do not necessarily know how valuable the output would be before the evaluation.* Kılınç and Küpçü [20] construct a fair *multi-party* computation (MPC) protocol in the optimistic model. While 2PC can be a special case of MPC, our solutions are optimized for the two-party case and hence are more efficient compared to applying their work to the two-party setting (e.g., they increase input and output sizes).

A detailed analysis of more related works is in the full version of the paper [21].

2 Definitions and Preliminaries

Yao's Two-Party Computation Protocol: We informally review Yao's construction [38], which is secure in the presence of *semi-honest* adversaries. Such adversaries follow the instructions of the protocol, but try to learn more information. The main idea in Yao's protocol is to compute a circuit without revealing any information about the value of the wires, except the output wires.

The protocol starts by agreeing on a circuit that computes the desired functionality. One party, called the *constructor*, generates two keys for every wire except the output wires. One key represents the value 0, and the other represents the value 1. Next, the constructor prepares a table for each gate that includes four double-encryptions with the four possible input key pairs (i.e., representing $00, 01, 10, 11$). The encrypted value is another key that represents these two input keys' output (e.g., for an AND gate, if keys $k_{a,0}$ and $k_{b,1}$ representing 0 and 1 are used as inputs of Alice and Bob, respectively, the gate's output key k', which is encrypted under $k_{a,0}$ and $k_{b,1}$, represents the value $0 = 0\ AND\ 1$.). Output gates contain double-encryptions of the actual output bits (no more keys are necessary, since the output will be learned anyway). All tables together are called the *garbled circuit*.

The other party is the *evaluator*. The constructor and the evaluator perform oblivious transfer (OT), where the constructor is the sender and the evaluator is the receiver, who learns the keys that represent his own input bits. Afterward, the constructor sends his input keys to the evaluator. The evaluator evaluates the garbled circuit by decrypting the garbled tables in topological order, and learns the output bits. The evaluator can decrypt one row of each gate's table, since he just knows one key for each wire. Since all he learns for the intermediary values are random keys and only the constructor knows which values these keys represent, the evaluator learns nothing more than what he can infer from the output. The evaluator finally sends the output to the constructor, who also learns nothing more than the output, since the evaluator did not send any intermediary values and they used OT for the evaluator's input keys.

Secure Two-Party Computation (2PC): Alice and Bob want to compute a function $f : \{0,1\}^* \times \{0,1\}^* \to \{0,1\}^* \times \{0,1\}^*$. Alice has her private input x, and Bob has his private input y. In the end of computation of $f(x, y)$, Alice obtains the output $f_a(x, y)$ and Bob obtains the output $f_b(x, y)$. The computation is secure if the privacy, correctness, independence of inputs, guaranteed output delivery, fairness [27] are achieved by the computation.

Because of the impossibility result on the fairness property without honest majority [8], fairness in a secure computation is not considered in the 2PC literature. Security is formalized with the ideal/real simulation paradigm. For every real world adversary, there must exist an adversary in the ideal world such that the execution in the ideal and real worlds are indistinguishable (e.g., [14]).

Definition 1 (Ideal World). *It consists of the corrupted party C, the honest party H, and the universal trusted party $\mathfrak{U}$ (not the TTP). The ideal protocol is:*

1. *$\mathfrak{U}$ receives input x or the message* ABORT *from C, and y from H. If the inputs are invalid or C sends the message* ABORT*, then $\mathfrak{U}$ sends $\perp$ to both of the parties and halts.*
2. *Otherwise $\mathfrak{U}$ computes $f(x, y) = (f_c(x, y), f_h(x, y))$. Then, he sends $f_c(x, y)$ to C and $f_h(x, y)$ to H.*

The outputs of the parties in an ideal execution between the honest party H and an adversary $\mathcal{A}$ controlling C, where $\mathfrak{U}$ computes f, is denoted $\mathsf{IDEAL}_{f,\mathcal{A}(w)}(x, y, s)$ where x, y are the respective inputs of C and H, w is an auxiliary input of $\mathcal{A}$, and s is the security parameter.

The standard secure two-party ideal world definition [27, 16] lets the adversary $\mathcal{A}$ to ABORT *after* learning his output but *before* the honest party learns her output. Thus, proving protocols secure using the old definition would not meet the fairness requirements.

Definition 2 (Real World). *The real world consists of, besides the parties, an adversary $\mathcal{A}$ that controls one of the parties, and the* TTP *who is involved in the protocol when there is unfair behavior. The pair of outputs of the honest party* *and the adversary $\mathcal{A}$ in the real execution of the protocol π, possibly employing the* TTP*, is denoted* $\mathsf{REAL}_{\pi,\mathsf{TTP},\mathcal{A}(w)}(x,y,s)$*, where x, y, w and s are like above.*

Note that $\mathfrak{U}$ and TTP are *not* related to each other. TTP is part of the *real* protocol to solve the fairness problem when it is necessary, but $\mathfrak{U}$ is not real.

Definition 3 (Fair and Secure Two-Party Computation). *Let π be a probabilistic polynomial time (PPT) protocol and let f be a PPT two-party functionality. We say that π computes f* ***fairly and securely*** *if for every non-uniform PPT real world adversary $\mathcal{A}$ attacking π, there exists a non-uniform PPT ideal world adversary S so that for every $x, y, w \in \{0,1\}^*$, the ideal and real world outputs are computationally indistinguishable:*

$$\{\mathsf{IDEAL}_{f,S(w)}(x,y,s)\}_{s\in\mathbb{N}} \equiv_c \{\mathsf{REAL}_{\pi,\mathsf{TTP},\mathcal{A}(w)}(x,y,s)\}_{s\in\mathbb{N}}$$

For optimistic protocols, to simulate the complete view of the adversary, **the simulator also needs to simulate the behavior of the TTP** for the adversary. This simulation also needs to be indistinguishable.

The closest such definition was given by Cachin and Camenisch [6]. Their definition's advantage is that it also considers misbehaving TTP, but their ideal world contacts the real world TTP, mixing both worlds. Thus, it does not fit the optimistic usage of TTP. We prefer to use the Definition 3, which is more intuitive and general (it can even include gradual release since it is not specific to only the protocols with TTP), to prove our proposed protocol in Section 4 because we use the TTP in the optimistic model and we assume that the TTP is semi-honest while proving the protocol.

Note that in our ideal world, the moment the adversary sends his input, $\mathfrak{U}$ computes the outputs and performs fair distribution. Thus, the adversary can either abort the protocol before any party learns anything useful, or cannot prevent fairness. This is represented in our proof with a **simulator who learns the output only when it is *guaranteed* that both parties can learn the output**. Also observe that, under this ideal world definition, **if the simulator learns the output in the ideal world but the adversary aborts in the real world, that simulation would be *distinguishable***.

Suppose that Alice is malicious and S simulates the behavior of honest Bob in the real world and the behavior of malicious Alice in the ideal world. Assume S learns the output of Alice from $\mathfrak{U}$ in order to simulate the real protocol *before* it is guaranteed that in a real protocol both of the parties could receive their outputs. Further suppose that the adversarial Alice then aborts the protocol so that S does not receive his output in the real world. Thus, in the real world the real Bob would have aborted, whereas the ideal Bob outputs the result of the computation. Clearly, the ideal and real worlds are *distinguishable* in this case. The proofs in [34, 23, 6] unfortunately fall into this pitfall.

Definition 4. *(Verifiable Escrow) An escrow is a ciphertext under the public key of the TTP. A verifiable escrow [1, 7] enables the recipient to verify, using only the public key of TTP, that the plaintext satisfies some relation. A public non-malleable label can be attached to a verifiable escrow [36].*

Communication Model: We do *not* need private and authenticated channels between the TTP and the parties. When there is dispute between the two parties, the TTP resolves the conflict *atomically*, which means the TTP interacts with either Alice or Bob at a given time, until that resolution is complete. We assume that the adversary cannot prevent the honest party from reaching the TTP eventually. We do not assume anything else about the communication model; our protocol's needs are minimal.

3 Our Solution

Failed Approaches and Major Issues: It looks like adding *fairness* to a 2PC protocol based on gabled circuits and zero knowledge using TTP does not need a lot of work. However, if we care efficiency of the protocol and resolution protocols with the TTP, it is challenging. Consider a very simple solution regarding constructor C, evaluator E, and the TTP. Assume that C constructs the circuit such that the output is not revealed directly, but instead the output of the circuit is an encrypted version of the real output, and C knows the key. Thus, after evaluation, E will learn this encrypted output, and C and E need to perform a fair exchange of this encrypted output and the key. This approach increases the circuit size, obviously. Besides, when a dispute occurs and E goes to the TTP for resolution, she cannot efficiently prove to the TTP that she evaluated C's garbled circuit correctly. Indeed, in the solution of Cachin and Camenisch [6], the *resolution* may require work proportional to the circuit size.

Alternatively, instead of encrypting the output, C constructs a garbled circuit where the outputs are encoded with some random values (like an encryption but without increasing the circuit size) in a secret table. So, in the end of the circuit evaluation, E learns some random values such that their corresponding bits are only known by C. Then, they can fairly exchange the table and the output. However, it can be hard to ensure that E sends the correct table and construct proper resolution protocols with TTP.

Because of these issues, we employ the dual-constructor methodology [29, 30], where both C and E construct circuits that output random numbers.

Our Solution: We show how to efficiently add fairness to any zero knowledge based secure 2PC protocol Γ using our framework. The key points are:

- Alice and Bob employ **dual garbling technique** [29], where Alice and Bob both act as the constructor and the evaluator, with almost equal responsibilities. **The circuit constructed by Alice *only* outputs Alice's output** and **the circuit constructed by Bob outputs Bob's output**.
  The garbled circuit is prepared as the underlying protocol Γ with minor differences in the construction of the input and output gates. The modification on the input gates allow us to **check input equality** between the two circuits. Modifications on the output gates are to **hide the actual output**.
- Alice and Bob exchange the garbled circuits and evaluate each others' circuits. In the end of evaluation, **Alice learns the output labels of Bob and Bob learns the output labels of Alice, both in a hidden way**. Therefore, they need to exchange the outputs fairly after this point.

- Before **fair exchange**, they execute **input equality test** protocol to see if both of them used the same inputs for the both circuits. It is ok to abort if the test fails, because they test for input equality, not output.
- If the equality test is successful, they **verifiably escrow the other party's output labels**. This is essentially a guarantee for the other party that if this party does not send the output labels later on, (s)he can contact the TTP to get them.
- Now, they exchange output labels so that each party can individually translate them back to the actual outputs, since they come from circuits that they themselves created. If there is a dispute about the fairness, they go to the TTP for the resolution.

Overview of the resolution protocols is the following:

Alice/Bob Resolve: We describe the resolution for Bob, though it is completely symmetric for Alice. Remember that Bob is equipped with a verifiable escrow. But, for the TTP to decrypt it for him, Bob must prove that he acted properly. He provides output labels of Alice, and proves that they are evaluated from Alice's garbled circuit. If so, the TTP provides the decryption for Bob, who can use it to translate back to his output bits.

Alice Abort: Alice may try to abort the protocol and block resolution attempts with the TTP, should she not receive Bob's verifiable escrow. When she contacts the TTP, if Bob has resolved before, she obtains her output labels from the TTP. Otherwise, the TTP marks the protocol as aborted, and would deny any resolution attempt by Alice or Bob.

Note that the **TTP only sees random output labels**, but not their translation tables. Furthermore, since each circuit only evaluates to one party's output, even if the TTP colludes with the malicious party and provides the other party's output labels, those are still meaningless without the corresponding bits. Thus, a **malicious TTP may only break fairness, but not security**.

Why Target Zero-Knowledge Proof based Garbled Circuit Protocols? We claimed that our framework can be applied on top of any zero-knowledge proof based garbled circuit protocols. There are two reasons for this:

1. As explained above, parties commit to output labels, for enabling efficient resolutions with the TTP (one of the major problems in previous work). They must prove to each other that they committed to the correct labels as in the garbled circuits. If the underlying protocol, for example, encrypts the garbled tables using AES, then such a proof cannot be efficiently done (without cut-and-choose), whereas if the underlying encryption scheme is number-theoretic (such as simplified Camenisch-Shoup [7, 19]), then using sigma protocols [10], the correctness proofs may be done very efficiently.
2. Item 1 above leaves out the cut-and-choose way of proving. The problem is that, if cut-and-choose is employed, then there will be multiple circuits, rather than one. In our solution, parties create verifiable escrows, and the TTP may need to decrypt them. Verifiable escrow is a primitive that inherently uses zero-knowledge proofs. It is unclear how to combine the verifiable

escrow idea with cut-and-choose, where multiple circuits exist, especially when the TTP needs to be able to verify and decrypt them.

In essence, one may think of our solution as a framework that can be applied on top of 2PC schemes that employ a single circuit, and use number-theoretic constructions (of encryption) for efficiency.

4 Making Secure 2PC Fair (Full Protocol)

Notation: Alice and Bob will evaluate a function $f(x,y) = (f_a(x,y), f_b(x,y))$, where Alice has an input x and gets an output $f_a(x,y)$, and Bob has an input y and gets an output $f_b(x,y)$, $f : \{0,1\}^\ell \times \{0,1\}^\ell \rightarrow \{0,1\}^\ell \times \{0,1\}^\ell$, where ℓ is a positive integer. For simplicity, we assume Alice and Bob have ℓ-bit inputs and outputs each. Alice's input bits are $x = \{x_1, x_2, ..., x_\ell\}$ and Bob's input bits are $y = \{y_1, y_2, ..., y_\ell\}$. They use a 2PC protocol Γ for the secure computation.

We use $\mathfrak{C}$ to represent circuit. $\mathfrak{C}_a$ outputs the Alice's output and $\mathfrak{C}_b$ outputs Bob's output. Similarly, the garbled circuit that is generated by Alice is GC_a and the one generated by Bob is GC_b. We use apostrophe ($'$) for the values that are generated by Bob. When we say Alice's input wires, it means that Alice provides the input for these wires. Similarly, Alice's output wires correspond to Alice's output. Bob's input and output wires have the matching meaning. An *Input Gate* is a gate that has an input wire of Alice or Bob. Similarly, an *Output Gate* is a gate that has a wire of Alice's or Bob's output.

E_k shows an encryption with the key k. Therefore, $E_{k_1}E_{k_2}(m_1, m_2)$ means that m_1 and m_2 are both encrypted by the two keys k_1 and k_2.

Any commitments that have efficient zero knowledge proofs can be used in this framework. To exemplify the protocol we notate commitments as in Fujisaki-Okamoto commitments [13, 11] and Pedersen commitments [32].

Table 1. The review of the random numbers used for fairness in our framework.

Name	Form	Relation
Equality-Test Constants	$e = g^\rho$	There are four kinds of them, where each represents 0 or 1 and right or left.
Input-Gate Randoms	u	Each input gate has them. They are private; just known by the constructors.
Equality-Test Numbers	$m = e^u$	For each input-gate random u, there are four kinds of them, where each represents 0 or 1 and right or left according to e.
Output Labels	(δ, ε)	They are randomly chosen pairs, each representing a row of the garbled output gates.

We give a review of the random numbers that are used for fairness in Table 1. The protocol steps are described in detail below (and in Figure 1).

The TTP generates the group $\mathcal{G}_1$ that is used in Γ and picks generators $g, h \in \mathcal{G}_1$, secret and public key pair sk_{TTP}, pk_{TTP} for the verifiable escrow scheme. Additionally, he chooses a cyclic group $\mathcal{G}_2$ whose order is a large prime q and randomly selects its generators g_0, g_1, g_2 (for the equality test). He also picks a one-way function $\phi()$. Then, he announces his public key $PK_{\mathsf{TTP}} = [pk_{TTP}, (\mathcal{G}_1, g, h), (\mathcal{G}_2, q, g_0, g_1, g_2), \phi()]$.

Both Alice and Bob know PK_{TTP} and agree on a circuit $\mathfrak{C}$ that computes $f(x,y)$ and the protocol identifier id before the protocol begins.

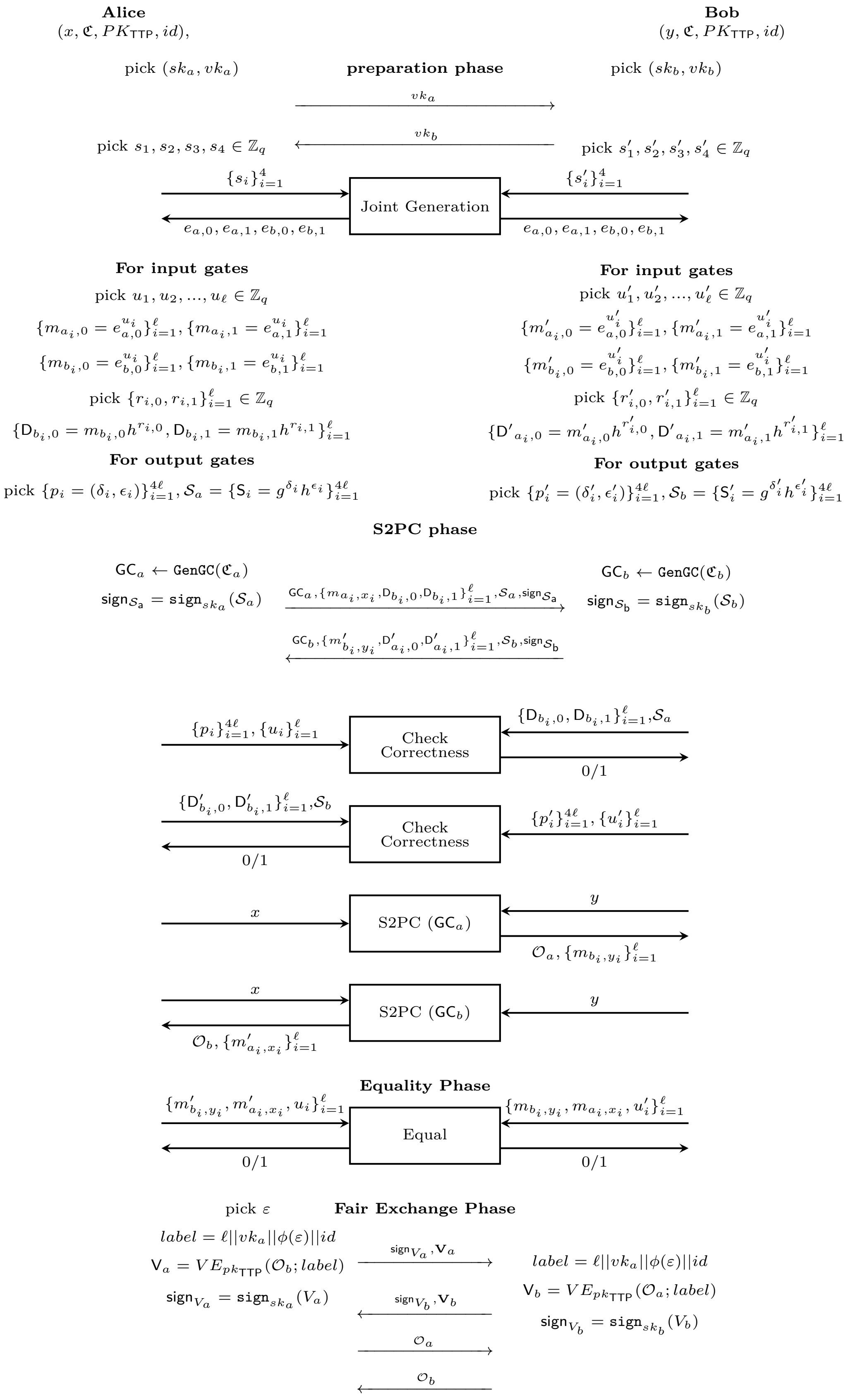
Alice
$(x, \mathfrak{C}, PK_{\mathsf{TTP}}, id)$,
Bob
$(y, \mathfrak{C}, PK_{\mathsf{TTP}}, id)$
pick (sk_a, vk_a)
preparation phase
pick (sk_b, vk_b)
vk_a
vk_b
pick $s_1, s_2, s_3, s_4 \in \mathbb{Z}_q$
pick $s'_1, s'_2, s'_3, s'_4 \in \mathbb{Z}_q$
$\{s_i\}_{i=1}^4$
Joint Generation
$\{s'_i\}_{i=1}^4$
$e_{a,0}, e_{a,1}, e_{b,0}, e_{b,1}$
$e_{a,0}, e_{a,1}, e_{b,0}, e_{b,1}$
For input gates
pick $u_1, u_2, ..., u_\ell \in \mathbb{Z}_q$
$\{m_{a_i,0} = e_{a,0}^{u_i}\}_{i=1}^\ell, \{m_{a_i,1} = e_{a,1}^{u_i}\}_{i=1}^\ell$
$\{m_{b_i,0} = e_{b,0}^{u_i}\}_{i=1}^\ell, \{m_{b_i,1} = e_{b,1}^{u_i}\}_{i=1}^\ell$
pick $\{r_{i,0}, r_{i,1}\}_{i=1}^\ell \in \mathbb{Z}_q$
$\{\mathsf{D}_{b_i,0} = m_{b_i,0}h^{r_{i,0}}, \mathsf{D}_{b_i,1} = m_{b_i,1}h^{r_{i,1}}\}_{i=1}^\ell$
For output gates
pick $\{p_i = (\delta_i, \epsilon_i)\}_{i=1}^{4\ell}, \mathcal{S}_a = \{\mathsf{S}_i = g^{\delta_i}h^{\epsilon_i}\}_{i=1}^{4\ell}$
For input gates
pick $u'_1, u'_2, ..., u'_\ell \in \mathbb{Z}_q$
$\{m'_{a_i,0} = e_{a,0}^{u'_i}\}_{i=1}^\ell, \{m'_{a_i,1} = e_{a,1}^{u'_i}\}_{i=1}^\ell$
$\{m'_{b_i,0} = e_{b,0}^{u'_i}\}_{i=1}^\ell, \{m'_{b_i,1} = e_{b,1}^{u'_i}\}_{i=1}^\ell$
pick $\{r'_{i,0}, r'_{i,1}\}_{i=1}^\ell \in \mathbb{Z}_q$
$\{\mathsf{D'}_{a_i,0} = m'_{a_i,0}h^{r'_{i,0}}, \mathsf{D'}_{a_i,1} = m'_{a_i,1}h^{r'_{i,1}}\}_{i=1}^\ell$
For output gates
pick $\{p'_i = (\delta'_i, \epsilon'_i)\}_{i=1}^{4\ell}, \mathcal{S}_b = \{\mathsf{S}'_i = g^{\delta'_i}h^{\epsilon'_i}\}_{i=1}^{4\ell}$
S2PC phase
$\mathsf{GC}_a \leftarrow \mathtt{GenGC}(\mathfrak{C}_a)$
$\mathsf{sign}_{\mathcal{S}_\mathsf{a}} = \mathtt{sign}_{sk_a}(\mathcal{S}_a)$
$\mathsf{GC}_a, \{m_{a_i,x_i}, \mathsf{D}_{b_i,0}, \mathsf{D}_{b_i,1}\}_{i=1}^\ell, \mathcal{S}_a, \mathsf{sign}_{\mathcal{S}_\mathsf{a}}$
$\mathsf{GC}_b \leftarrow \mathtt{GenGC}(\mathfrak{C}_b)$
$\mathsf{sign}_{\mathcal{S}_\mathsf{b}} = \mathtt{sign}_{sk_b}(\mathcal{S}_b)$
$\mathsf{GC}_b, \{m'_{b_i,y_i}, \mathsf{D}'_{a_i,0}, \mathsf{D}'_{a_i,1}\}_{i=1}^\ell, \mathcal{S}_b, \mathsf{sign}_{\mathcal{S}_\mathsf{b}}$
$\{p_i\}_{i=1}^{4\ell}, \{u_i\}_{i=1}^\ell$
Check Correctness
$\{\mathsf{D}_{b_i,0}, \mathsf{D}_{b_i,1}\}_{i=1}^\ell, \mathcal{S}_a$
0/1
$\{\mathsf{D}'_{b_i,0}, \mathsf{D}'_{b_i,1}\}_{i=1}^\ell, \mathcal{S}_b$
Check Correctness
$\{p'_i\}_{i=1}^{4\ell}, \{u'_i\}_{i=1}^\ell$
0/1
x
S2PC (GC_a)
y
$\mathcal{O}_a, \{m_{b_i,y_i}\}_{i=1}^\ell$
x
S2PC (GC_b)
y
$\mathcal{O}_b, \{m'_{a_i,x_i}\}_{i=1}^\ell$
Equality Phase
$\{m'_{b_i,y_i}, m'_{a_i,x_i}, u_i\}_{i=1}^\ell$
Equal
$\{m_{b_i,y_i}, m_{a_i,x_i}, u'_i\}_{i=1}^\ell$
0/1
0/1
pick ε
Fair Exchange Phase
$label = \ell||vk_a||\phi(\varepsilon)||id$
$\mathsf{V}_a = VE_{pk_{\mathsf{TTP}}}(\mathcal{O}_b; label)$
$\mathsf{sign}_{V_a} = \mathtt{sign}_{sk_a}(V_a)$
$\mathsf{sign}_{V_a}, \mathbf{V}_a$
$\mathsf{sign}_{V_b}, \mathbf{V}_b$
$\mathcal{O}_a$
$\mathcal{O}_b$
$label = \ell||vk_a||\phi(\varepsilon)||id$
$\mathsf{V}_b = VE_{pk_{\mathsf{TTP}}}(\mathcal{O}_a; label)$
$\mathsf{sign}_{V_b} = \mathtt{sign}_{sk_b}(V_b)$

Fig. 1. Our framework to make a S2PC protocol fair. `GenGC` generates garbled circuit.

Preparation Phase:

1. Alice and Bob generate private-public key pairs (sk_a, vk_a) and (sk_b, vk_b), respectively, for an unforgeable signature scheme. They exchange the signature verification keys vk_a and vk_b.
   They jointly generate four equality-test constants $e_{a,0}, e_{a,1}, e_{b,0}$ and $e_{b,1}$ as described [21]. Equality test constants represent 0 and 1 for the left (a) and the right (b) wires of the input gates.
2. Alice and Bob separately generate the random numbers and commitments for the input and the output gates as shown in Figure 1.
   The computations of Alice and Bob for each *input gate* i are the following: input-gate numbers $(u_i \,\text{resp.}\, u'_i)$, the equality-test numbers $(\{t \in \{0,1\}, z \in \{a,b\} : m_{z_i,t} = e_{z,t}^{u_i}\} \,\text{resp.}\, \{t \in \{0,1\}, z \in \{a,b\} : m'_{z_i,t} = e_{z,t}^{u'_i}\})$, and the commitments $(\{t \in \{0,1\} : \mathsf{D}_{b_i,t} = m_{b_i,t}h^{r_{i,t}}\} \,\text{resp.}\, \{t \in \{0,1\} : \mathsf{D}'_{a_i,t} = m'_{a_i,t}h^{r'_{i,t}}\})$. They are used in the input equality test to show the same inputs are used for both garbled circuits.
   They generate output labels $((\delta_j, \epsilon_j) \,\text{resp.}\, (\delta'_j, \epsilon'_j))$ for each row of garbled-output gate j and their commitments $(\mathsf{S}_j \,\text{resp.}\, \mathsf{S}'_j)$ for the *output gates*. The sets of the commitments are $\mathcal{S}_a = \{\mathsf{S}_j\} \,\text{resp.}\, \mathcal{S}_b = \{\mathsf{S}'_j\}$. The output labels are as unique identifiers for the rows of the constructor's garbled-output gates. Only the constructor knows which row they represent, which means only the constructor knows which output bit they correspond to. This makes sure that the evaluator cannot learn the output directly.

S2PC Phase:

1. [**Garbled Circuits:**] Alice and Bob construct their garbled circuits by following the rules of the underlying Γ protocol with little differences on the garbled tables of the input and the output gates.

Table 2. The garbled Input and Output Gate for an OR gate constructed by Alice. Encryption scheme is same as the underlying protocol Γ.

Row	Garbled Input Gate	Garbled Output Gate
00	$E_{k_{a_i,0}}E_{k_{b_i,0}}(r_{i,0}, k_0)$	$E_{k_{a,0}}E_{k_{b,0}}(\delta_j, \varepsilon_j)$
01	$E_{k_{a_i,0}}E_{k_{b_i,1}}(r_{i,1}, k_1)$	$E_{k_{a,0}}E_{k_{b,1}}(\delta_{j+1}, \varepsilon_{j+1})$
10	$E_{k_{a_i,1}}E_{k_{b_i,0}}(r_{i,0}, k_1)$	$E_{k_{a,1}}E_{k_{b,0}}(\delta_{j+2}, \varepsilon_{j+2})$
11	$E_{k_{a_i,1}}E_{k_{b_i,1}}(r_{i,1}, k_1)$	$E_{k_{a,1}}E_{k_{b,1}}(\delta_{j+3}, \varepsilon_{j+3})$

Input Gates: The difference is that each garbled-table row of an input gate i includes one more encryption besides the encryption of the output key. It is the encryption of either $r'_{i,0}$ or $r'_{i,1}$ representing the input of 0 and 1 for the wire of Alice in GC_b and either $r_{i,0}$, or $r_{i,1}$ representing the input of 0 and 1 for the wire of the Bob in GC_a. See Table 2 for the details.

Remark: Alice and Bob just encrypt the partial decommitments of $\mathsf{D}_{b_i,0}, \mathsf{D}_{b_i,1}$ and $\mathsf{D}'_{a_i,t}\mathsf{D}'_{a_i,1}$, respectively because they only need to learn equality-test numbers (m values) that represent their input bits. They do *not* want to reveal input-gate numbers (u values) since it causes the evaluator to learn the constructor's input.

Remark: Note that there can be just *one* input wire of a gate (e.g., NOT gate for negation). In this case, there will be two equality-test numbers which represent 0 and 1 for this gate. Alternatively, they can agree to construct a circuit using only NAND gates [6].

Output Gates: Each row of the garbled output gate includes the encryption of corresponding output labels instead of encryption of real output bits (see Table 2). This is to hide the actual output from the evaluator.

2. [**Exchange:**] They exchange the constructed garbled tables along with the commitments, the signature of all commitments of the output labels ($\mathsf{sign}_{\mathcal{S}_a}$ resp. $\mathsf{sign}_{\mathcal{S}_b}$) and equality-test numbers that represents their input bits as in Figure 1.
3. [**Check Correctness:**] They prove to each other that they performed the input and the output gates' construction honestly, via efficient zero-knowledge proofs (see the full version of the paper [21]):
   - *Proof of Input Gates* to prove that the garbled input gates contain the correct decommitment values. This is basically done in three steps: Firstly, prover proves that (s)he knows the decommitmets of all commitments denoted by D [7]. Secondly, prover proves that each commitment pair $\mathsf{D}_{z,0}$ and $\mathsf{D}_{z,1}$ commits the same value under the different bases $e_{z,0}$ and $e_{z,1}$ respectively. If the prover is Alice then $z = b_i$, if the prover is Bob then $z = a_i$. Lastly, the prover proves that each input-garbled table includes the double-encryption of partial decommitment of $\mathsf{D}_{z,0}$ and $\mathsf{D}_{z,1}$.
   - *Proof of Output Gates* to prove that the garbled output gates encrypt the committed output labels.

   If there is a problem in the proofs, they abort. Otherwise, they continue.
4. [**S2PC:**] Alice and Bob execute Γ, and evaluate the garbled circuit they were given. While executing Γ, Alice and Bob prove that they correctly construct their garbled circuits that evaluate f by zero-knowledge proofs described in the protocol Γ. If all zero-knowledge proofs are verified, at the end of the evaluation, Alice learns the set $\mathcal{O}_b$ representing f_b, Bob learns the set $\mathcal{O}_a$ representing f_a, each including ℓ output labels. Besides, each party learns the set that includes equality-test numbers that represents her/his input (from the decryption of input-garbled gates). Otherwise, they abort.

Equality Phase: *This phase is necessary to test whether or not Alice and Bob used the same input bits for both circuit evaluations.* We use unfair version of equality test by Boudot et al. [3]; the unfair version is sufficient for our purpose.

Alice and Bob want to check, if $x_i^* = x_i$ and $y_i^* = y_i$ for the encryptions $E_{k'_{a_i,x_i}}(E_{k'_{b_i,y_i}}(k'))$ and $E_{k_{a_i,x_i^*}}(E_{k_{b_i,y_i^*}}(k))$ in each garbled input gate i, such that the first one was decrypted by Alice and the second one was decrypted by Bob. For this purpose, Alice and Bob will use the equality-test numbers $\{m_{z_i,t}, m'_{z_i,t}\}_{z\in\{a,b\},t\in\{0,1\}}$.

Assume Alice decrypted a row for an input gate i and learned equality-test numbers m'_{a_i,x_i} and she knows m'_{b_i,y_i} since Bob sent his equality-test numbers that represents his input in the exchange step of the S2PC phase. Also assume

Bob decrypted the corresponding garbled gate and similarly learned m_{b_i,y_i^*} and he knows m_{a_i,x_i^*} since Alice sent it in the exchange step of the S2PC phase. If they both used consistent input bits for both GC_a and GC_b, then we expect to see that the following equation is satisfied:

$$(m'_{a_i,x_i} m'_{b_i,y_i})^{u_i} = (m_{a_i x_i^*} m_{b_i,y_i^*})^{u'_i} \tag{1}$$

The left hand side of the equation (1) is composed of values Alice knows since she learned m' values and generated u_i herself. Similarly, the right hand side values are known by Bob since he learned m values and generated u'_i himself. This equality should hold if $x_i^* = x_i$ and $y_i^* = y_i$ since $m'_{a_i,x_i^*} = e^{u'_i}_{a,x_i^*}$, $m'_{b_i,y_i^*} = e^{u'_i}_{b,y_i^*}$ and $m_{a_i,x_i} = e^{u_i}_{a,x_i}$, $m_{b_i,y_i} = e^{u_i}_{b,y_i}$.

After computing their side locally in equation (1) for each input gate, they concatenate the results in order to hash them, where the output range of the hash function is $\mathbb{Z}_q$. Then Alice and Bob execute *Proof of Equality* protocol in [3] with the hashes.

If the equality test succeeds, they continue with the next phase.

Remark: Remember that the constructor did not prove that (s)he added equality-test numbers to the correct row of the encryption table. Suppose that the constructor encrypted the equality-test number that represents 0 where the evaluator's encryption key represents 1. In this case, it is sure that the equality test will fail, but the important point is that the constructor *cannot* understand which row is decrypted by the evaluator, and thus does not learn any information because he cannot cheat just in one row. If he cheats in one row, he has to change one of the other rows as well, as otherwise he fails the "Proof of Input Gates". Thus, even if the equality test fails, the evaluator might have decrypted any one of the four possibilities for the gate, and thus might have used any input bit. This also means that the equality test can be simulated, and hence reveals nothing about the input.

Note that there are some techniques to check input equality in the literature as in [26, 23, 28, 35, 29, 30] but they are based on cut-and-choose. Since the underlying protocol Γ does not use cut-and-choose to guarantee the security, the equality test we used is more suitable here.

Fair Exchange Phase: In this phase, Alice and Bob exchange the outputs. Remember that the outputs are indeed randomized, and only the constructor knows their meaning. Thus, if they do not perform this fair exchange, no party learns any information about the real output (unless they resolve with the TTP, in which case they both learn their outputs).

1. Alice first picks a value ω from the domain of the one way-function ϕ and computes $\phi(\omega)$. Next, she creates a verifiable escrow V_a including $\mathcal{O}_b$ with non-malleable label $(\ell||vk_a||\phi(\varepsilon)||id)$ as in Figure 1. Finally, she signs V_a with sk_a and sends the signature sign_{V_a} and V_a.

   With the verifiable escrow, she proves that there are ℓ different decommitments in the escrow that correspond to ℓ of the commitments in $\mathcal{S}_b$ [18, 9, 4]. Since Alice can just decrypt one row for every gate and so she only has one pair of keys for each gate, this proof shows that Alice decrypted Bob's garbled output tables correctly, and the verifiable escrow has the evaluation result of GC_b. If V_a or sign_{V_a} fails to verify, or if the label is not correct, then Bob aborts. Otherwise, Bob continues with the next step.

Remark: ω is used in the Alice Abort protocol with the TTP to prevent Bob from claiming to be Alice and aborting after Bob Resolve. Since only Alice knows ε that is a pre-image of $\phi(\varepsilon)$, Bob cannot convince the TTP.

2. Bob creates a verifiable escrow V_b including $\mathcal{O}_a$ with non-malleable label the same as Alice created. He signs V_b with sk_b and sends the signature sign_{V_b} and V_b.

   With the verifiable escrow, he proves that there are ℓ different decommitments in the escrow that correspond to ℓ of the commitments in $\mathcal{S}_a$ [18, 9, 4]. If V_a or sign_{V_a} fails to verify, or if the label is not correct, then Alice runs "Alice Abort" protocol with the TTP. Otherwise, Alice continues with the next step.
3. Alice sends $\mathcal{O}_b$ to Bob.
4. Bob checks if the output labels in $\mathcal{O}_b$ are correct. The output labels are correct if ℓ of them are the pairs that are generated by Bob. If they are correct, then he sends $\mathcal{O}_a$. If at least one of the output labels is not correct, then he does "Bob Resolve" with the TTP.
5. Alice checks if the output labels in $\mathcal{O}_a$ are correct. If they are not correct, then she does "Alice Resolve" with the TTP. Otherwise the protocol ends.

Alice and Bob Resolve (See Figure 2): We explain Bob Resolve below. Alice Resolve is the same where the verifiable escrow, the signatures and $\mathcal{O}$ are Bob's values.

Bob contacts with the TTP and sends the values $\mathsf{V}_a, \mathsf{sign}_{V_a}, \mathcal{S}_a, \mathsf{sign}_{\mathcal{S}_a}, \mathcal{O}_a$. He sends $\mathsf{sign}_{\mathcal{S}_a}$ to prove that $\mathcal{S}_a$ is generated by the same party who generates V_a. The TTP checks if all signatures are correct and the decommitments in $\mathcal{O}_a$ correspond to ℓ of the commitments in $\mathcal{S}_a$. If there is no problem, then the TTP decrypts V_a with sk_{TTP} and sends the values inside V_a to Bob. Since Bob knows the meaning of the output labels of the garbled circuit he constructed, he effectively learns his output. The TTP remembers Alice's output $\mathcal{O}_a$, given and proven by Bob, in his database.

Alice Abort (See Figure 3): When Alice contacts the TTP for abort, she sends V_a and sign_a, together with ε. The TTP checks that the signature is valid and $\phi(\varepsilon)$ matches the label of V_a. If Bob did resolve before, the TTP sends $\mathcal{O}_a$ as in Figure 3 so that Alice can also learn her output. Otherwise, the protocol is aborted and the TTP will not honor resolution requests for this exchange.

Remark that Alice and Bob **do not** re-do the zero-knowledge proofs in the S2PC phase to the TTP because sign_{V_a} and sign_{V_b} show that both Alice and Bob execute everything correctly until the end of Equality Phase.

Theorem 1. *Let $f : \{0,1\}^* \times \{0,1\}^* \rightarrow \{0,1\}^* \times \{0,1\}^*$ be any probabilistic polynomial time (PPT) two-party functionality. The protocol above for computing f is secure and fair according to Definition 3, assuming that the TTP is semi-honest, the subprotocols that are stated in the protocol are all secure (sound and zero-knowledge), all commitments are hiding and binding [13, 11], the signa-*

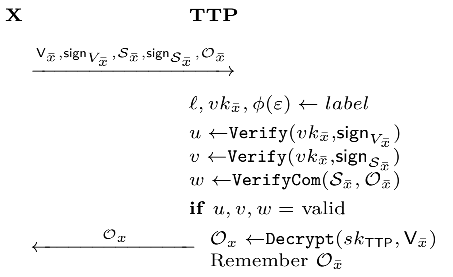
X
TTP
$\mathsf{V}_{\bar{x}}, \mathsf{sign}_{V_{\bar{x}}}, \mathcal{S}_{\bar{x}}, \mathsf{sign}_{\mathcal{S}_{\bar{x}}}, \mathcal{O}_{\bar{x}}$
$\ell, vk_{\bar{x}}, \phi(\varepsilon) \leftarrow label$
$u \leftarrow \texttt{Verify}(vk_{\bar{x}}, \mathsf{sign}_{V_{\bar{x}}})$
$v \leftarrow \texttt{Verify}(vk_{\bar{x}}, \mathsf{sign}_{\mathcal{S}_{\bar{x}}})$
$w \leftarrow \texttt{VerifyCom}(\mathcal{S}_{\bar{x}}, \mathcal{O}_{\bar{x}})$
if $u, v, w =$ valid
$\mathcal{O}_x$
$\mathcal{O}_x \leftarrow \texttt{Decrypt}(sk_{\mathsf{TTP}}, \mathsf{V}_{\bar{x}})$
Remember $\mathcal{O}_{\bar{x}}$

Fig. 2. X Resolve where X $\in$ {Alice,Bob}. If X is Alice, $\bar{x}$ is b, otherwise $\bar{x}$ is a.

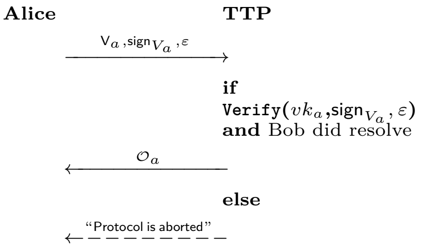
Alice
TTP
$\mathsf{V}_a, \mathsf{sign}_{V_a}, \varepsilon$
if
$\texttt{Verify}(vk_a, \mathsf{sign}_{V_a}, \varepsilon)$
and Bob did resolve
$\mathcal{O}_a$
else
"Protocol is aborted"

Fig. 3. Alice Abort

ture scheme used is unforgeable [17], and the Γ is a 2PC protocol secure against malicious adversaries based on Yao's garbled circuits and zero knowledge proofs.

Proof Sketch. A full proof exists in the full version of the paper [21]. The important point in our proof is that after learning the input of the adversary in the real world, the simulator does *not* learn the output of the adversary from the ideal world universal party $\mathfrak{U}$ until it is guaranteed that both parties can obtain their outputs.

Malicious Alice: Simulator S_B creates a key pair on behalf of the TTP and shares the public key with Alice. S_B prepares the circuit as the simulator of Γ with a random input y' and simulates all the proofs, including the equality test. S_B extracts the input x of Alice as in the simulation of Γ. Here, he does not send the input of Alice directly to the ideal world trusted party $\mathfrak{U}$. He waits until it is guaranteed that Bob can also obtain his input. If, before V_a is received properly, the values Alice sends are not correct or equality test is not successful, he sends ABORT to $\mathfrak{U}$. If S_B receives the correct V_a, he sends Alice's input to $\mathfrak{U}$ and receives Alice's output. At this point, he sends V_b to Alice. Afterward, if he does not receive his correct output from Alice, he simulates *Bob Resolve* by decrypting V_a. If Alice performs "*Alice Abort*", then there are two options: if S_B already obtained the output from $\mathfrak{U}$, S_B sends output labels of Alice so that she resolves her output; otherwise, S_B sends ABORT to $\mathfrak{U}$.

Malicious Bob: Simulator S_A behaves almost the same as S_B. Since S_A does not know the actual output of Bob, she puts random values in V'_a and sends it to Bob, simulating the proof. Then S_A waits for V_b: If Bob does not send valid values but performs "*Bob Resolve*", then S_A gives Bob's input that she extracted to $\mathfrak{U}$ and learns Bob's output so that S_A is able to simulate "*Bob Resolve".* If Bob does not send valid values and does not perform "*Bob Resolve*", then S_A sends ABORT to $\mathfrak{U}$ (simulating "*Alice Abort*"). If Bob sends V_b, then S_A gives Bob's input to $\mathfrak{U}$ and $\mathfrak{U}$ sends back Bob's output, and finally S_A sends to Bob the correct output labels accordingly.

TTP Analysis: As we claim, a semi-honest TTP is sufficient in our protocol because the TTP only learns output labels where their meaning is only known by the circuit constructors (Alice or Bob), and a signature. In addition, (s)he does *not* receive anything else related to the input of Alice or Bob. Therefore, if the TTP follows the protocol but also tries to learn extra information about the parties (input or output), (s)he cannot succeed.

Even if the TTP is malicious, (s)he can only break the fairness property of the protocol. A malicious TTP can collude with Alice or Bob. As seen in Theorem 1, the protocol preserves the privacy if the TTP is malicious since the TTP does not have more power than Alice or Bob. He only knows his secret key which is only used in the Fair Exchange phase.

Malicious TTP also cannot break the correctness property. In the honest Bob case (same in honest Alice case), he cannot receive wrong output since Alice can only learn one output label per gate, so (s)he can use only them. It means TTP cannot give different ones (because (s)he only knows those that Alice provides) to Bob. Thus, the TTP cannot break the correctness property.

5 Proving Security and Fairness Together

In this section we show the importance of proving with *full simulation* according to Definition 3. First, we define what we mean by *partial simulation* more formally and then we give contrived versions of several protocols ([22, 34, 6]) including ours that are obviously insecure, but can be proven fair and secure with partial simulation while it cannot be proven fair and secure with full simulation.

Definition 5 (Partial Simulation). *Let f, h, g be the PPT functionalities where $f = g \circ h$, $h(x, y) = (h_b, h_a)$ and $g(h_b, h_a) = (f_a, f_b)$ and let π_f, π_h, π_g be the PPT protocols to compute f, h, g, respectively where the first input and output of a functionality correspond to one party (Alice) and the second input and output of a functionality correspond to other party (Bob). The* ***partial simulation*** *paradigm says that π_f computes f* ***fairly and securely*** *if there exists a PPT protocol π_h that is secure under simulation with abort [14] and there exists a PPT protocol π_g which achieves fairness [1, 24].*

Almost all previous works (See Table 3) prove their fairness and security with partial simulation: prove security with the unfair simulation paradigm (with abort) (corresponding to proving π_h to be a secure 2PC protocol with abort), and argue fairness (of the π_g part, either using TTP or gradual release) separately. This is risky. Consider the following three contrived protocols where Alice and Bob want to compute functionality $f = (f_a, f_b)$ fairly and securely:

- A modification on our protocol is that the TTP gives Alice's output *along with the input of Bob* whenever Alice contacts for resolution or abort, if Bob have done "Bob Resolve" before (honest Bob is required to provide his input to the TTP in "Bob Resolve"). Here, $h = (h_a, h_b)$ is a functionality where $h_a = \mathcal{O}_a$ and $h_b = \mathcal{O}_b$ (π_h is our protocol until the fair exchange phase, where parties only obtain random output labels), and g is a functionality where $g(\mathcal{O}_a, \mathcal{O}_b) = (f_x, f_y)$ (π_g is the fair exchange phase of our protocol with new "Alice Abort" and "Alice Resolve".). It is very easy to simulate π_h with abort, since parties essentially learn nothing. Also, it is easy to argue about fairness of this π_g without simulation, since at the end of resolutions, either both parties obtain their outputs or no one learns anything useful.

- The protocol which is the same as Cachin and Camenisch's protocol [6] where the only difference is that the TTP gives the other parties' inputs to the party in the resolution protocols (with similar reasoning as above).
- The modified versions of Kiraz and Schoenmakers [23] or Ruan et. al [34] protocols where the only difference is Alice sends her *input* to Bob and vice versa at the end of the gradual release.

In [23, 34] partial simulation is provided only until the beginning of the gradual release phase, then fairness is argued via the fairness of the gradual release. Similarly, in [6] the partial simulation is provided for a functionality computation, then the fairness is discussed based on the parties' and TTP's behaviors. Using the same type of reasoning, their and our contrived versions can be proven fair and secure via partial simulation, and fairness can be argued since at the end of the gradual release or TTP resolutions, either both parties obtain their outputs or no one does. But, it is clear that the contrived protocols leak the inputs to the other party, becoming insecure. Observe that they can *never* be fully simulated, because *the simulator will not have access to the honest party's input* and so it cannot provide indistinguishability of ideal and real worlds.

Consequently, it is risky to argue fairness separate from the ideal/real world simulation. We do not claim that previous protocols [6, 23, 34] have security problems, but we want to emphasize that the partial simulation technique does *not* cover all security aspects of a protocol and should not be preferred anymore. Therefore, they should be proven with the full simulation technique.

Importance of the Timing of the Simulator contacting the Universal Trusted Party: The proofs of the protocols [34, 23, 6] are also problematic since the simulator learns the output of the computation from $\mathfrak{U}$ *before* it is guaranteed that the other party can also obtain the output. This behavior of the simulator *violates the indistinguishibility of the ideal and real worlds* because if the simulator does not receive his/her output in the real world while the parties already obtained the outputs in the ideal world, then the outputs in ideal and real worlds are distinguishable, and the simulation fails. Therefore, **the simulator must obtain the output from the universal trusted party in the ideal world, *only after* it is guaranteed that both parties can obtain the output in the real world.**

6 Conclusion

Table 3 presents a comparison with the most related works.

✓ All our overhead (TTP, Alice, Bob) are dependent only on the input and output size, and **independent** of the circuit size, in contrast to [6].

✓ We require a **constant** number of rounds for fairness, contrary to gradual release based solutions [33, 23, 34].

✓ We do **not** necessitate a payment framework. Our fairness definition is that either both parties obtain the output, or no one does, as opposed to [25].

✓ Even if the TTP becomes malicious and colludes with one participant, he **cannot violate the security of the protocol**. On the other hand, in [25], while the Bank cannot violate 2PC security, it can maliciously deal with the balances, possibly causing a lot of headache.

Table 3. Comparison of our protocol with previous works. CC denotes cut-and-choose, ZK denotes efficient zero-knowledge proofs of knowledge, GR denotes gradual release, OFE denotes efficient optimistic fair exchange, superscript I denotes *inefficient* TTP, superscript P denotes necessity of using a *payment* system, NS denotes *no* ideal-real simulation proof given, PS indicates *partial simulation* proof, and finally FS indicates *full simulation* proof including fairness. A check mark ✓ is put for easily identifying better techniques.

	[33]	[34]	[23]	[25]	[6]	Ours
Malicious Behavior	CC	CC	CC	CC/ZK	ZK	ZK
Fairness	GR	GR	GR	OFE^P	OFE^I	OFE ✓
Proof Technique	NS	PS	PS	FS ✓	PS	FS ✓

✓ Finally, our protocol is proven secure in the **ideal/real simulation** paradigm (not in [33]) with **output indistinguishability** (not in [23, 34, 6]), and by proving fairness and security simultaneously via a **full simulation proof** (none except [25]).

Acknowledgements

The authors acknowledge the support of TÜBİTAK, the Scientific and Technological Research Council of Turkey, under project number 111E019, and European Union COST Action IC1306.