Efficiently Making Secure Two-Party Computation Fair

Handan Kılınç and Alptekin Küpçü

Koç University
handan.kilinc@epfl.ch , akupcu@ku.edu.tr

Abstract

Secure two-party computation cannot be fair in general 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 separately, 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, the security of the underlying protocol is still preserved.

Keywords: two party computation, garbled circuit, Yao's protocol, fair computation, optimistic model

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 [62] introduced the concept of secure two-party computation 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 [51, 61, 38, 44] 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 [55, 39, 56] or a trusted third party (TTP) [13, 42]. The ***gradual release*** protocols [23, 11, 7] 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 of this approach is that it is expensive since it requires many rounds of communication.

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 [4], 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 [13]. 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 [13, 39, 55, 56], fairness and security (non-fair sense) were proven *separately*, without *simulating the resolutions* with the TTP. However, it is important to prove them 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 4) that covers *both fairness and security.* In other words, **the fairness solution with the optimistic TTP should also be simulatable**. Besides, **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., [38, 13, 29]). Note that the state-of-the-art optimistic fairness solution [13] 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 together** accordingly, using a single simulator who also simulates the TTP.
- We show that the classic way of **proving security and fairness separately is not necessarily secure** (see Section 8).
- 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 identities of Alice and Bob, or their inputs or outputs. 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 almost the same steps in the protocol.
- The principles for fairness in **our framework can be adopted by any zero knowledge and Yao's construction based secure two party computation protocol**, 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.

2 Preliminaries

Yao's Two-Party Computation Protocol: We informally review Yao's construction [62], 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 $f : \{0,1\}^* \times \{0,1\}^* \rightarrow \{0,1\}^* \times \{0,1\}^*$. 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 [45] are achieved by the computation.

Because of the impossibility result on the fairness property without honest majority [18], 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., [27]).

3 Fairness Problem in 2PC

3.1 Failed Approaches

In the beginning it looks like adding *fairness* in a 2PC protocol based on gabled circuits and zero knowledge using TTP should not need a lot of work. However, we must be careful because it can break the *security* of the protocol. 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. There are multiple problems regarding this idea:

1. C can add wrong encryptions for the outputs of E, and hence E can learn wrong output result.

2. The wrong encryptions can leak information about E's output value to C according to the abortion of E.
3. When there is dispute between E and C, and E goes to the TTP for resolution, she cannot efficiently prove to the TTP that she evaluated C's garbled circuit correctly.

In the scenario above, C has some advantages. To remove C's advantage, E can be a constructor too. Thus, the idea is to make C and E construct their own circuits and then evaluate the other's circuit [49, 50]. The following problems occur in the two-constructor case:

1. One of the parties can construct a circuit for a different functionality, so the other party may learn a wrong output.
2. Even when both E and C construct the same circuit, they can cause the other one to learn a wrong output by using different inputs for the two circuits.
3. C and E can check their inputs are the same for both circuits by checking either input or output equality. If a malicious party uses different inputs between the two garbled circuits and then is able to learn whether or not respective two outputs are equal, (s)he can learn something illegitimate about the input of the honest party.

Besides the problems above, both C and E have to prove that they acted correctly during the protocol to learn output results from the TTP when there is dispute. These proofs should be efficient to reduce the workload on the TTP. In addition, the TTP should not learn anything about the inputs and outputs of the computation, as well as the identities of the participants. Additionally, we want to use the TTP in optimistic model so that he does not join the protocol if no dispute about fairness occurs. **Our solution uses TTP in optimistic model and achieves efficiency of and privacy against the TTP**.

3.2 Our Solution

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

- Alice and Bob employ **dual garbling technique** [49], 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 **commit to output labels** for their constructed circuits, and prove to each other that these match the garbled circuit. Alice also **signs her commitments**. These will be used for resolutions with the TTP.
- Alice and Bob exchange the garbled circuits and evaluate each others' circuits. In the end of evaluation, **Alice learns the output of Bob and Bob learns the output 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, Alice signs this success message for the TTP to check, and **verifiably escrows Bob's output labels**. This is essentially a guarantee for Bob that if Alice does not send his output labels later on, he can contact the TTP to get them.
- Now Bob has a guarantee, he should send Alice's output labels to her, who can translate them back to her actual outputs. Alice then responds with Bob's output labels, and he can

translate them himself. If there is a dispute about the fairness, they go to the TTP for resolution.

Overview of the resolution protocols is the following:

- **Bob Resolve:** 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 does this by providing Alice's output labels, and proving that they match the initial commitments Alice sent. The TTP also asks for Alice's signatures on the commitments and the equality test result, to ensure that Bob is not providing fake material. If so, the TTP provides the decryption for Bob, who can use it to translate back to his original output bits.
- **Alice Resolve:** We require Bob to resolve before some timeout. If Alice contacts the TTP afterward, and Bob has provided Alice's output labels, she gets them and translates back to her output bits. Otherwise, since Bob cannot resolve after the timeout, the protocol is aborted.

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.

4 Definitions

Here, we give the definition of ideal-real world for **both fairness and security**.

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

1. *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 U sends $\perp$ to both of the parties and halts.*
2. *Otherwise 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 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 [45, 31] 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 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 U is not real.

Definition 3 (Fair 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.

Note that in our ideal world, the moment the adversary sends his input, 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 and the adversary later 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 U in order to simulate the real protocol *before* it is guaranteed that in a real protocol both of the parties would 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 abort whereas the ideal Bob outputs the result of the computation. Clearly, the ideal and real worlds are *distinguishable* in this case. The proofs in [56, 39, 13] unfortunately fall into this pitfall.

The closest such definition was given by Cachin and Camenisch [13]. This definition also considers misbehaving TTP but their ideal world contacts with the TTP in the real world. It does not fit optimistic usage of TTP because TTP does not involve protocol if no dispute occurs in optimistic model. We prefer to use the Definition 3, which is more general (not specific to only the protocols with TTP) and intuitive, to prove our proposed protocol in Section 5 because we use the TTP in the optimistic model and we assume that the TTP is semi-honest while proving the protocol.

Definition 4. *(Verifiable Escrow) An escrow is a ciphertext under the public key of the TTP. A verifiable escrow [4, 14] 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 [60].*

Communication Model: We do *not* need private and authenticated channel 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 before the timeout.[1] We do not assume anything else about the communication model.

5 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 input x and gets output $f_a(x, y)$, and Bob has input y and gets 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

[1]We do not require tight synchronization. For example, if the timeout is six hours, and the parties' clocks differ by ten minutes, this is very acceptable.

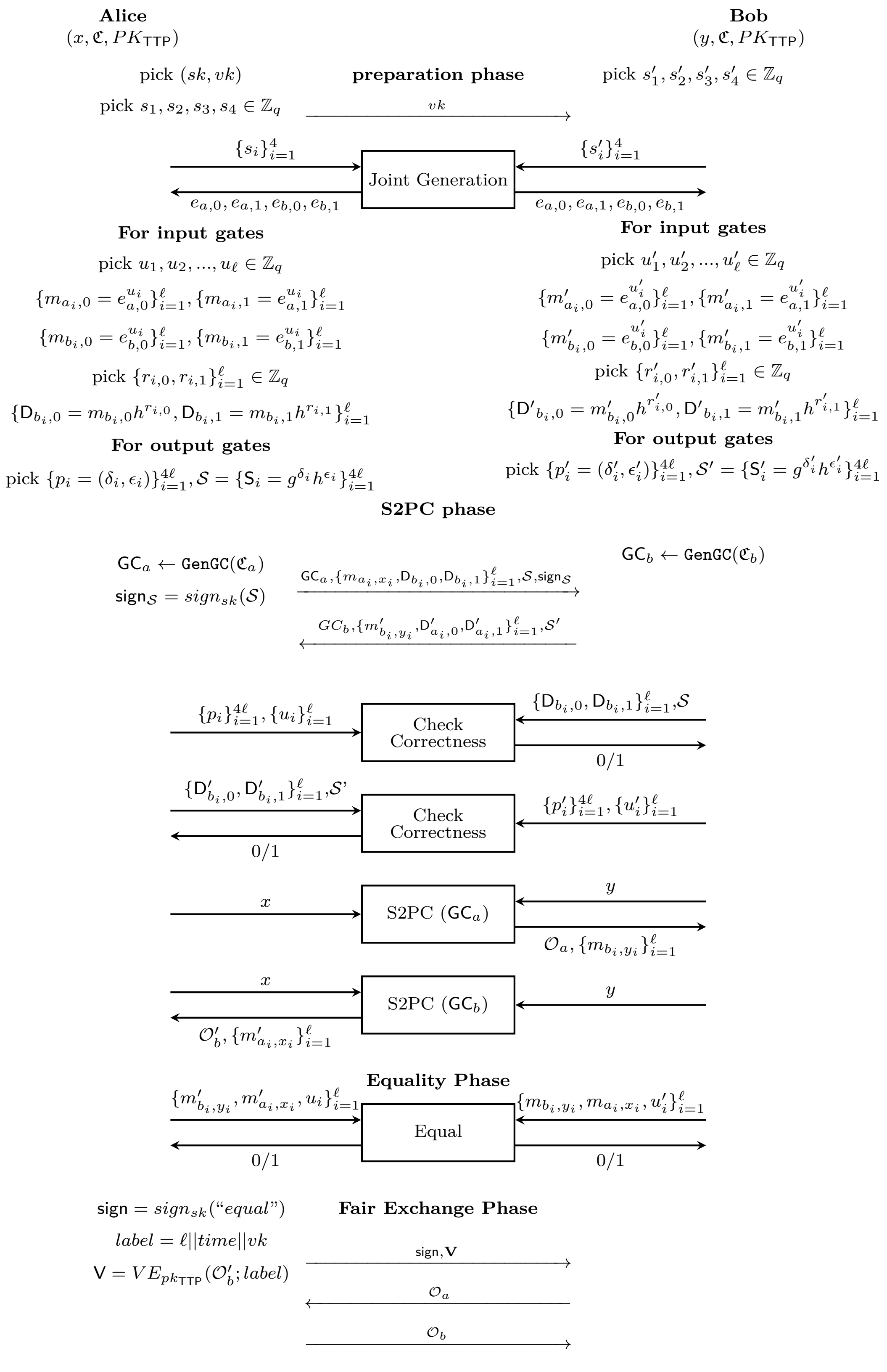
Alice
$(x, \mathfrak{C}, PK_{\mathsf{TTP}})$
Bob
$(y, \mathfrak{C}, PK_{\mathsf{TTP}})$
pick (sk, vk)
preparation phase
pick $s'_1, s'_2, s'_3, s'_4 \in \mathbb{Z}_q$
pick $s_1, s_2, s_3, s_4 \in \mathbb{Z}_q$
vk
$\{s_i\}_{i=1}^4$
Joint Generation
$\{s'_i\}_{i=1}^4$
$e_{a,0}, e_{a,1}, e_{b,0}, e_{b,1}$
$e_{a,0}, e_{a,1}, e_{b,0}, e_{b,1}$
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 $\{p_i = (\delta_i, \epsilon_i)\}_{i=1}^{4\ell}, \mathcal{S} = \{\mathsf{S}_i = g^{\delta_i}h^{\epsilon_i}\}_{i=1}^{4\ell}$
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 $\{r'_{i,0}, r'_{i,1}\}_{i=1}^{\ell} \in \mathbb{Z}_q$
$\{\mathsf{D}'_{b_i,0} = m'_{b_i,0}h^{r'_{i,0}}, \mathsf{D}'_{b_i,1} = m'_{b_i,1}h^{r'_{i,1}}\}_{i=1}^{\ell}$
For output gates
pick $\{p'_i = (\delta'_i, \epsilon'_i)\}_{i=1}^{4\ell}, \mathcal{S}' = \{\mathsf{S}'_i = g^{\delta'_i}h^{\epsilon'_i}\}_{i=1}^{4\ell}$
S2PC phase
$\mathsf{GC}_a \leftarrow \mathtt{GenGC}(\mathfrak{C}_a)$
$\mathsf{sign}_{\mathcal{S}} = sign_{sk}(\mathcal{S})$
$\mathsf{GC}_b \leftarrow \mathtt{GenGC}(\mathfrak{C}_b)$
$\mathsf{GC}_a, \{m_{a_i,x_i}, \mathsf{D}_{b_i,0}, \mathsf{D}_{b_i,1}\}_{i=1}^{\ell}, \mathcal{S}, \mathsf{sign}_{\mathcal{S}}$
$GC_b, \{m'_{b_i,y_i}, \mathsf{D}'_{a_i,0}, \mathsf{D}'_{a_i,1}\}_{i=1}^{\ell}, \mathcal{S}'$
$\{p_i\}_{i=1}^{4\ell}, \{u_i\}_{i=1}^{\ell}$
Check Correctness
$\{\mathsf{D}_{b_i,0}, \mathsf{D}_{b_i,1}\}_{i=1}^{\ell}, \mathcal{S}$
0/1
$\{\mathsf{D}'_{b_i,0}, \mathsf{D}'_{b_i,1}\}_{i=1}^{\ell}, \mathcal{S}'$
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
$\mathsf{sign} = sign_{sk}(\text{“}equal\text{”})$
$label = \ell||time||vk$
$\mathsf{V} = VE_{pk_{\mathsf{TTP}}}(\mathcal{O}'_b; label)$
Fair Exchange Phase
$\mathsf{sign}, \mathbf{V}$
$\mathcal{O}_a$
$\mathcal{O}_b$

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

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.
Randomized 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.
Random Row Pair	(δ, ε)	They are randomly chosen pairs, each representing a row of the garbled output gates.

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 [25, 22] and Pedersen commitments [54].

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 a group $\mathcal{G}_1$ which is the group that is used in Γ and picks generators $g, h \in \mathcal{G}_1$, secret and public key pair $sk_{TTP}, pk_{TTP} \in \mathcal{G}_1$. 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). Then, he announces his public key $PK_{\mathsf{TTP}} = [pk_{TTP}, (\mathcal{G}_1, g, h), (\mathcal{G}_2, q, g_0, g_1, g_2)]$.

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

Preparation Phase:

1. Alice generates private-public key pair (sk, vk) for an unforgeable signature scheme. She shares the signature verification key vk with Bob.
   They jointly generate four equality test constants $e_{a,0}, e_{a,1}, e_{b,0}$ and $e_{b,1}$. (see Appendix A.1). Equality test constants represent 0 and 1 for the left (a) and right (b) wires of input gates.
2. Alice (and Bob) separately generates the random numbers and commitments for input and output gates as shown in Figure 1.
   The computations for the *input gates* are input gate numbers, randomized equality test numbers (see Table 1), and commitments. They are used in the input equality test to show having same inputs are used for both garbled circuits.
   They generate random row pairs (see Table 1) and their commitments for the *output gates*. These can be thought 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. Alice and Bob construct their garbled circuits by following the rules of the underlying Γ protocol with little differences on the garbled tables of input gates and output gates. The garbled circuit constructed by Alice is GC_a and the garbled circuit constructed by Bob is GC_b. The difference in the *Input Gate* 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 the encryption of either

Table 2: 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	$E_{k_{a_i,1}}E_{k_{b_i,1}}(r_{i,1}, k_1)$	$E_{k_{a,1}}E_{k_{b,1}}(\delta_{j+3}, \varepsilon_{j+3})$

$r_{i,0}$ 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 partial decommitments because they only need to learn randomized equality test numbers (m values) that represent their input bits and we do *not* want to reveal input gate numbers (u values) since it cause the evaluator to learn the constructor's input.
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 randomized equality test numbers which represent 0 and 1 for this gate. Alternatively, they can agree to construct a circuit using only NAND gates [13].
The garbled tables of *Output Gates* are constructed differently as well. Each row of the garbled output gate includes the encryption of corresponding random row pairs instead of encryption of real output bits. See Table 2.

2. They exchange the constructed garbled tables along with the commitments as in the Figure 1.
3. [**Check Correctness:**] They prove to each other that they performed the input and output gates' construction honestly (details in [38] or appendix of [41]).
   (see Appendix A.2):
   - *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 [14]. Secondly, prover proves that each commitment pair $\mathrm{D}_{z,0}$ and $\mathrm{D}_{z,1}$ 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, prover proves that each input garbled table includes the two encryption of partial decommitment of $\mathrm{D}_{z,0}$ and $\mathrm{D}_{z,1}$.
   - *Proof of Output Gates* to prove that the garbled output gates encrypt the committed random row pairs.

   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. At the end of the evaluation, Alice learns the set $\mathcal{O}'_b$, Bob learns the set $\mathcal{O}_a$, each including ℓ random row pairs. Besides, each party learns the set that includes randomized equality test numbers that represents her/his input. If Γ protocol ends successfully they continue. 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. [9]; the unfair version is sufficient for our purpose.

Alice and Bob want to check, if $t^*_b = t_b$ and $t^*_a = t_a$ for the encryptions $E_{k_{a_i,t_a}}(E_{k_{b_i,t_b}}(k))$ and

$E_{k'_{a_i,t_a^*}}(E_{k'_{b_i,t_b^*}}(k'))$ in each garbled input gate i, such that the first one was decrypted by Bob and the second one was decrypted by Alice. For this purpose, Alice and Bob will use the randomized 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 randomized equality test numbers m'_{a_i,t_a} and she knows m'_{b_i,t_b} since Bob sent his randomized equality test numbers that represents his input in the beginning of S2PC phase. Also suppose Bob decrypted the same garbled gate and similarly learned m_{b_i,t_b^*} and he knows m_{a_i,t_a^*} since Alice sent it in the beginning of S2PC phase. If both used the same input bits for the both S2PC protocol for GC_a and for GC_b, then we expect to see that the following equation is satisfied:

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

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

After computing their side in Equality (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 [9] with their calculated hash values as their inputs.

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

Remark: Remember that the constructor did not prove that (s)he added randomized equality test numbers to the correct row of the encryption table. Suppose that the constructor encrypted the randomized 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 [44, 39, 46, 59, 49, 50] 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 could learn their outputs).

1. Alice firstly prepares a signature $\mathsf{sign_e}$ showing that equality test is successful. Secondly, she creates a verifiable escrow with the non-malleable label as in Figure 1. Then, she sends the signature $\mathsf{sign_e}$ and the verifiable escrow V.
   With the verifiable escrow, she proves that there are ℓ different decommitments in the escrow that correspond to ℓ of the commitments S_1,...,$\mathrm{S}_{4\ell} \in \mathcal{S}$ [37, 20, 10]. 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 no information about which row was decrypted. If V fails to verify, or if the public signature verification key was different from what was used in the previous steps, or the *time*

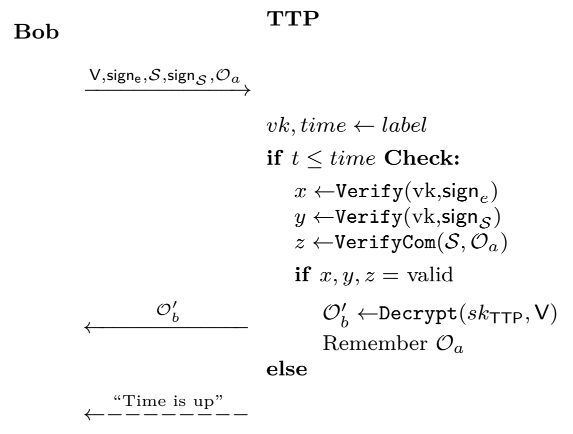
Bob
TTP
V,sign$_e$,$\mathcal{S}$,sign$_\mathcal{S}$,$\mathcal{O}_a$
$vk, time \leftarrow label$
if $t \leq time$ **Check:**
$x \leftarrow$Verify(vk,sign$_e$)
$y \leftarrow$Verify(vk,sign$_\mathcal{S}$)
$z \leftarrow$VerifyCom($\mathcal{S}, \mathcal{O}_a$)
if $x, y, z =$ valid
$\mathcal{O}'_b$
$\mathcal{O}'_b \leftarrow$Decrypt($sk_{\mathsf{TTP}}$, V)
Remember $\mathcal{O}_a$
else
"Time is up"

Figure 2: Bob Resolve. t is the time that Bob does resolve. TTP sends either the message on the arrow or the message on the dashed arrow.

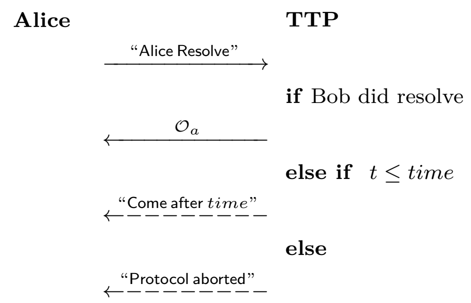

Figure 3: Alice Resolve. t is the time that Alice does resolve. TTP responds with either one of the arrows.

value is not what Bob was expecting (see [40]), then Bob aborts. Otherwise, Bob continues with the next step.

2. Bob sends $\mathcal{O}_a$ that he learned from the evaluation of Alice's circuit GC_a.
3. Alice checks if the random row pairs in $\mathcal{O}_a$ that he received from Bob are correct. The random row pairs are correct if ℓ of them are the pairs that are generated by Alice. If they are correct then she sends $\mathcal{O}_b$ that she learned from the evaluation of Bob's circuit GC_b. If at least one of the random row pairs is not correct or she does not receive any message from Bob (until some short message timeout based on round trip delays), then she does "Alice Resolve" with the TTP.
4. Similarly, Bob checks if the random row pairs in $\mathcal{O}_b$ that he received from Alice are correct. If they are not correct, or if Alice does not respond (until some short message timeout based on round trip delays), then he does "Bob Resolve" with the TTP. Otherwise the protocol ends.

5.1 Resolutions with the TTP

Bob Resolve: Bob has to contact the TTP before the timeout to get an answer. Considering he contacts before the timeout (verified using the *time* in the label of V), he sends the values in Figure 2. $\mathcal{S}$ and $\mathsf{sign}_\mathcal{S}$ are to prove that they are the correct representation of the decrypted row for the output values of Alice. The TTP checks if all signatures and decommitments are correct. If there is no problem, then the TTP decrypts the verifiable escrow V with sk_{TTP} and sends the values inside to Bob. Since Bob knows which output wire he put these values in 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 Resolve: When Alice contacts the TTP, she asks for her random row pairs. If the TTP has them, then he sends all random row pairs $\mathcal{O}_a$ as in Figure 3. If Bob Resolve has not been performed yet, then Alice should come after the timeout. When Alice connects after the timeout, if the TTP has the output of Alice, then he sends them to Alice. When Alice obtains $\mathcal{O}_a$ from the TTP, she can learn her real output results since these random row pairs uniquely define rows of the output gates of the garbled circuit she constructed. Otherwise, if Bob never resolved until the timeout, she will not get any output, but she can rest assured that Bob also cannot learn any output values: The protocol is aborted.

6 Security of the Protocol

Theorem 1. *Let $f : \{0,1\}^* \times \{0,1\}^* \to \{0,1\}^* \times \{0,1\}^*$ 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 [25, 22], the signature scheme used is unforgeable [32], and the Γ is a 2PC protocol secure against malicious adversaries based on Yao's garbled circuits and zero knowledge proofs.*

7 Security of the Protocol

7.1 Security against Malicious Alice

Lemma 1. *Under the assumptions in Theorem 1, our protocol is secure and fair against malicious Alice.*

Proof. We construct a simulator that interacts with Alice in the real world as Bob, and with the universal trusted third party U in the ideal world as Alice:

1. **Preparation Phase:** S_B generates same values as in the preparation phase. S_B also generates a key pair on behalf of the TTP, and provides the public key to Alice.
2. **S2PC Phase:** S_B constructs the garbled circuit same as the simulator of Γ. For the input and output gates, it follows the same process as in the framework. Then $\mathcal{A}$ and S_B exchange the garbled circuits and commitments.
   $\mathcal{A}$ performs check correctness phase. Here, S_B learns all input gate numbers $\{u_i\}_{i=1}^{\ell}$ from the proof of input gates and all random row pairs $\{(\delta_i, \epsilon_i)\}_{i=1}^{4\ell}$ from the proof of output gates using soundness extractors. If $\mathcal{A}$'s proofs are not valid, then S_B sends message ABORT to U and stops. Otherwise, they continue with the evaluation of circuits.
   S_B learns the input of $\mathcal{A}$ $x = (x_1, x_2, ..., x_\ell)$ by using the simulator of Γ.
3. **Equality Phase:** S_B acts as the random oracle, so when $\mathcal{A}$ asks for the hash of her equality test input, S_B answers her randomly and sees randomized equality test numbers that corresponds $\mathcal{A}$'s input, which consist of m' values. Since these were generated by S_B, he can check if they are the expected ones because S_B knows his own inputs and Alice's inputs for S_B's garbled circuit, so he knows which row $\mathcal{A}$ has to decrypt.
   In addition S_B checks if $\mathcal{A}$ used the same input as himself for the garbled circuit evaluation. He knows which row $\mathcal{A}$ should have decrypted, as explained above. He learned the input gate numbers $\{u_i\}_{i=1}^{\ell}$ in the previous step. So S_B can learn which equality test constants of the row that he decrypted are used, and hence he can learn which input $\mathcal{A}$ is used in $\mathcal{A}$'s garbled circuit. Finally, S_B can check if the inputs that $\mathcal{A}$ used in both circuits are equal.
   They begin the equality test protocol. If the proofs of $\mathcal{A}$ are not valid or the input values are not the expected ones, then S_B sends ABORT message to U. S_B simulates all zero-knowledge proof of knowledge protocols in the equality test, similarly as in [9]. If the test was supposed to return true, he simulates that way, otherwise he simulates with another random value whose hash is random. The probability that the adversary queries the random oracle at this random value is negligible. If the result is false, then he sends ABORT message to U and the simulation ends.
   Remark: Our security theorem intentionally does not mention the random oracle model. As is specified above and in [9], the equality test and the use of the hash function here necessitates the use of the random oracle model. Yet, one may convert this part to be secure

in the standard model, with a loss of efficiency. First, all zero-knowledge proofs of knowledge must be done interactively. Second, instead of hashing the values and performing only a single equality test, the values must be compared independently, using $2l$ (sequential) equality tests. Performing these equality tests independently does not harm security, since one may not cheat in the other party's input, but can only cheat with own input, and therefore learns nothing from the (in)equality.

4. **Fair Exchange Phase:** If the equality test is successful, then $\mathcal{A}$ should send the signature $\mathsf{sign_e}$ the verifiable escrow and if verification fails, then S_B sends ABORT message to U. Otherwise, the simulation continues with the next step.
   At this point S_B is sure that both parties can obtain the output, because honest real Bob could have obtained his output using the Bob Resolve, and thus there is no more need for aborting. Hence, S_B hands Alice's input x to U and U sends $f_a(x, y)$ to the simulator. Remember that he extracted all random row pairs of $\mathcal{A}$ from the *Proof of Output Gates*. In addition, he learned which bit the row represents from the *Proof of Garbled Construction*. Then, he picks random row pairs corresponding to the $\mathcal{A}$'s output $f_a(x, y)$ and sends all to $\mathcal{A}$.
5. **Resolutions**: S_B waits for the response from $\mathcal{A}$. If $\mathcal{A}$ does not respond or sends invalid random row pairs corresponding to his output, then he simulates *Bob Resolve* by decrypting the verifiable escrow. Later on, if Alice contacts the TTP for resolution, S_B simulates the TTP using the random row pairs in the previous step.
6. S_B outputs whatever $\mathcal{A}$ outputs.

□

Claim 1. *The view of $\mathcal{A}$ in his interaction with the simulator S_B is indistinguishable from the view in his interaction with real Bob and real TTP.*

Proof. We need to check the behaviors of the S_B that are different from Bob's behavior in the protocol, since these can affect the distribution.

- The construction garbled circuit of S_B is indistinguishable than the real one thanks to the simulator in protocol Γ.
- S_B did not commit to the real randomized equality test numbers as real Bob would do. Since the commitment scheme is hiding, the fake commitments are indistinguishable from the real ones. In addition, S_B simulates the zero-knowledge proofs in the protocol. Their indistinguishably comes from the zero-knowledge simulatability.
- The simulator only aborts when the real Bob would abort. Thus, the abort actions are indistinguishable as well.
- The simulator can perfectly simulate the TTP as well, since he generated its keys.

We emphasize one more time that a simulation proof for a **secure and fair two-party computation protocol needs to make sure that the simulator learns the output only when it is guaranteed that the other party can also obtain the output, and not before that point**. In real world, it is sure for honest Bob that he will learn the functionality output after Alice sends the correct verifiable escrow, because even if Alice does not send the output to him in the fair exchange phase, Bob can learn them from the TTP via *Bob Resolve*. For this reason S_B sends input x to U after receiving the valid verifiable escrow. After that Bob and S_B learn their outputs in the ideal world. As mentioned, Bob learns his output certainly in the end of the protocol in the real world after this point. Since Bob is honest, $\mathcal{A}$ learns her output too, in the real world.

Overall, we proved that the joint distribution of $\mathcal{A}$ and the honest Bob's output in the real execution is indistinguishable from the joint distribution of S_B and the honest Bob's output in the ideal model. □

Therefore, combined with Claim 1, the proof of Lemma 1 is complete.

7.2 Security against Malicious Bob

Lemma 2. *Under the assumptions in Theorem 1, our protocol is secure and fair against malicious Bob.*

Proof. The proof is similar to that of Lemma 1. We construct a simulator S_A that interacts as Alice with the adversary $\mathcal{B}$ that controls Bob in the real world, and as Bob with the U in the ideal world. S_A simulates the protocol as following:

1. **Up to Fair Exchange Phase:** All simulation actions are the same as the simulator S_B above until the end of *equality test.* There is just a minor difference, where initially S_A creates a key-pair for the signature scheme and sends her verification key vk_A to $\mathcal{B}$, and additionally sends the signature of the commitments of all the random row pairs to $\mathcal{B}$, as in the protocol.
2. **Fair Exchange Phase:** S_A prepares the signature $\mathsf{sign_e}$ and verifiable escrow (if the equality test was indeed successful). Since S_A does not know $f_a(x, y)$, she adds random values in the escrow. She labels the escrow with the verification key vk of the signature scheme and the correct timeout value. S_A simulates the proof by using verifiable escrow simulator [14].
   S_A waits for an answer in the output exchange phase. If $\mathcal{B}$ sends random row pairs, firstly S_A checks if they are random row pairs that she encrypted in its garbled circuit. If they are correct, we are guaranteed that the other party can obtain the output and the protocol will not be aborted. S_A sends y, which he learned via the Γ simulator, to U, and gets $f_b(x, y)$ from U. This also means that U sends $f_a(x, y)$ to Alice in the ideal world. Since S_A learned all decommitments of random row pairs of Bob from *Proof of Output Gates*, S_A prepares the random row pairs of $\mathcal{B}$ according to $f_b(x, y)$ and sends them to $\mathcal{B}$.
3. **Resolutions:** If there was a problem with the random row pairs that $\mathcal{B}$ sent, then resolution needs to take place. If before the timeout $\mathcal{B}$ runs *Bob Resolve*, S_A behaves like the TTP. First, S_A checks whether or not the verifiable escrow provided by $\mathcal{B}$ is the same as the one S_A sent, signature $\mathsf{sign_e}$ given by the adversary verifies under the public key of S_A in the verifiable escrow, and the random row pairs as proven by $\mathcal{B}$ are correct.
   If they are correct, then again it is guaranteed that both parties would obtain the output, and thus S_A sends y to U and gets $f_b(x, y)$ (and U sends $f_a(x, y)$ to ideal Alice). Then, S_A picks the correct random row pairs according to the output value $f_b(x, y)$. He sends these pairs to $\mathcal{B}$ as if they are the decryption of the verifiable escrow.
   If $\mathcal{B}$ did not send output values to S_A, or did not properly perform *Bob Resolve* until the timeout, then S_A sends ABORT message to U.
4. S_A outputs whatever $\mathcal{B}$ outputs.

□

Claim 2. *The view of $\mathcal{B}$ in his interaction with the simulator S_A is indistinguishable from the view in his interaction with real Alice and real TTP.*

Proof. Honest Alice learns the output of the function if malicious Bob sends it in the fair exchange phase, or if Bob successfully performs *Bob Resolve* in the real world. For this reason, the simulator

S_A learns the output from U in the ideal world exactly at these points. The abort cases also follow the real world.

As the rest of the simulation is similar to the case in Claim 1, the joint distribution of $\mathcal{B}$ and honest Alice's output in the real execution is indistinguishable from the joint distribution of S_A and honest Alice's output in the ideal model. $\square$

This completes the proof of Theorem 1.

TTP Analysis: As we claim, semi-honest TTP is sufficient in our protocol because TTP only learns random row pairs that their meaning is only known by the owners (Alice or Bob) and a signature. In addition, (s)he does not receive anything related with the input of Alice or Bob. Furthermore, the signature key can be specific to each circuit, and thus we do *not* require a public key infrastructure, and it does not give away Alice's identity. Hiding identities and handling timeouts in such resolution protocols are easy, as done by previous work [4, 40].[2] Therefore, if TTP follows the protocol but also tries to learn extra information about the parties (input or output), (s)he cannot succeed.

We do not assume in our protocol TTP is malicious, but even if (s)he is malicious, (s)he can only break the fairness property of the protocol. A malicious TTP can collude with Alice or Bob and s(he) can help to the party to make him or her learn extra information from the protocol or break the correctness property of the protocol. If Alice or Bob colludes with the TTP, they cannot break the correctness property becaouse in honest Bob case, he cannot receive wrong output since Alice can only learn one random row pair for each output gate so she can put only the learned random row pairs which means TTP cannot give different ones and break the correctness property. Similarly, honest Alice cannot receive wrong output. We defer a full proof to the full version.

8 Proving Security and Fairness Together

In this section we show the importance of proving security and fairness together according to Definition 3. We give a weird modified version of the protocol in Section 5. The modified protocol is almost the same as the original. One change is in the construction of garbled circuit. Both Alice and Bob construct the garbled circuit that computes both of their outputs. Here, Alice sends all signatures of the commitments of random row pairs $\{\mathsf{S}_i, sign_{sk}(\mathsf{S}_i)\}_{i=1}^{8\ell}$.

The important changes are in the resolve protocols. In *New Bob Resolve*, Bob contacts the TTP before the timeout and gives all items as in the old Bob Resolve. In addition, he gives all signatures of $\{\mathsf{S}_i\}_{i\in\{1,\ldots,8\ell\}}$ (in the original scheme he provided only 4ℓ signatures) and ℓ random row pairs that correspond to Alice's output along with ℓ **random row pairs that correspond to his output**. In short, he gives information that can be used to identify both his and Alice's outputs as computed from Alice's circuit. Then, the TTP checks the correctness of the random row pairs by checking if they are decommitments of properly signed $\{\mathsf{S}_i\}$ values. In return, the TTP gives the decryption of the verifiable escrow as in the old Bob Resolve.

New Alice Resolve protocol's only difference from the old one is that the TTP gives Alice not only Alice's random row pairs but also Bob's random row pairs he obtained through the *New Bob Resolve*. Thus, Alice can learn both her and Bob's outputs. **These resolution protocols obviously violate privacy.**

[2]The discussion about the usage of unique identifiers for the TTP to identify different evaluations between some Alice and some Bob (with anonymity and unlinkability guarantees), as well as the timeout not harming the system, already exists in the optimistic fair exchange literature [4, 40], and therefore we are not complicating our presentation with such issues.

We now show that this modified protocol **can be proven secure and fair separately**, via the standard method used in most of the previous work: prove security with the unfair simulation paradigm (without simulating resolutions), and argue fairness separately. We show that such a protocol, and any of its privacy-violating variants, **cannot be proven secure and fair simultaneously** according to Definition 3.

Prove Security then Fairness: We first prove that the protocol is secure in the standard (unfair) ideal/real world definition without considering TTP (as done in the previous works [39, 56] who prove security without considering gradual release and then do a fairness analysis). The security proof of this protocol is almost the same as our protocol's proof. The simulator learns the input of the adversary in the real world as the simulator of Γ and gives this input to U in the ideal world. Afterwards, the simulator receives output from U and continues the simulation as explained. If the adversary aborts the real protocol, then the simulator sends ABORT message to U.

For fairness of this protocol, note that either both parties receive the output or none of them receives, because if the protocol terminates before the verifiable escrow, no party can learn any useful information, and if the protocol finishes without problem, both of them learn the outputs. If the verifiable escrow is received, but the last phase of exchanging random row pairs was problematic, then the resolution protocols will ensure fairness such that both parties will learn their corresponding outputs if they resolve. Hence **according to this type of separate proofs, this contrived protocol is fair and secure**.

The problem arises since the fairness definition is only concerned that either Alice receives her output and Bob receives his, or no one receives anything. Fairness is achieved by the contrived protocol. But, during resolutions, Alice learns some extra information. To prevent learning extra information during fairness extensions, one must also ensure that the fairness parts (resolutions with the TTP) are simulatable. If the simulator is not concerned about fairness, one may prove such a contrived protocol fair, as we did above.

Try to Prove Security and Fairness: Let us try to prove security and fairness of such a contrived protocol using our joint definition. Assume, for simplicity, Alice is malicious. As in the proof of our original scheme, the simulator S_B learns the input x via Γ simulator, and then continues to simulate the view of the protocol using a random input y'. When the fairness is guaranteed, S_B sends Alice's input x to U, who responds back with $f_a(x, y)$.

Then, S_B sends the corresponding random row pairs for $f_a(x, y)$ and waits for the answer from Alice. If Alice does not answer or does not give the correct random row pairs, S_B performs *New Bob Resolve* with the TTP. Differently, from the original protocol, S_B has to give random row pairs that correspond to *both* $f_a(x, y)$ *and* $f_b(x, y)$. However, he *cannot* do it because he does not know $f_b(x, y)$, or y, or anything that depends on y other than $f_a(x, y)$. The best that S_B can do is to give random row pairs that correspond to $f_b(x, y')$ and $f_a(x, y)$. After Alice comes for the *New Alice Resolve* and gets these random row pairs, she calculates the corresponding bits for each random row pair in her circuit and outputs $f_a(x, y), f_b(x, y')$. Unfortunately, this output is now **distinguishable** from the ideal world, since the ideal world of this protocol would correspond to outputting $f_a(x, y), f_b(x, y)$. We can conclude that this protocol is **not fair and secure according to our ideal/real world definition that captures security and fairness simultaneously**.

In general, using Definition 3, since the simulator S_B only knows $f_a(x, y)$ and nothing else that depends on the actual y, *no* fairness solution that leaks information on y (other than $f_a(x, y)$) can be simulated.[3]

[3] For symmetric functionalities where $f_a(x, y) = f_b(x, y)$ the simulation would be successful. But also notice that the contrived protocol is still secure and fair in those cases, since learning the other party's output does not provide any additional information the party could not obtain via her/his own output.

Consequently, it is risky to prove fairness separate from the ideal/real world simulation. We do not claim that previous protocols [13, 39, 56] have security problems, but we want to emphasize that the old proof technique does *not* cover all security aspects of a protocol and should not be used in newer approaches.

Importance of the Timing of the Simulator contacting the Universal Trusted Party: The proofs of the protocols [56, 39] are problematic since the simulator learns the output of the computation from 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.**

9 Related Works

Secure Two-Party Computation: Yao [62] presented a secure and efficient protocol for two-party computation in the *semi-honest model*, where the adversary follows the protocol's rules but he cannot learn any additional information from his view of the protocol. Since this protocol is not secure against malicious behavior, new methods were suggested based on proving parties' honesty via zero-knowledge proofs [28, 13, 38] or via cut-and-choose techniques [44, 51, 61, 43].

Another problem in Yao's protocol is *fairness.* The general solutions are based on gradual release timed commitments [55, 39, 57, 56] or a trusted third party (TTP) [13, 42].

Gradual Release: Pinkas [55] presents a method where the evaluation of the garbled circuit is executed on the *commitments* of the garbled values rather than the original garbled values. It uses cut-and-choose to prevent malicious behavior of the constructor and *blind signatures* [16] for the verification of the evaluator's commitments. However, this protocol is expensive because of the gradual release timed commitments. Moreover, the *majority circuit* evaluation can reveal the constructor's input values, as shown by Kiraz and Schoenmakers [39], who improve Pinkas's construction by removing the majority circuit computation and the blind signatures, and using OR-proofs instead [21]. Yet, the other inefficiency problems remain. Similarly, Ruan et al. [56] use Jarecki and Shmatikov construction [38], and instead of the proof of the correct garbled circuit, they employ the cut-and-choose technique. Gradual release in these protocols constitutes the weak part regarding the efficiency.

Optimistic Model: Cachin and Camenisch [13] present a 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**, creating a bottleneck in the system. Lindell [42] 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.*

Relaxed Fairness: Because of the impossibility result of fairness, the alternative simplified definitions arise as partial fairness [26, 5, 34, 30] and fairness in rational settings [52, 35, 36, 47, 1, 53].

Security Definitions: Pinkas [55] protocol does *not* have proofs in the ideal-real world simulation paradigm. The importance of ideal-real world simulation is explained by Lindell and Pinkas

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, NFS indicates simulation proof given but *not for fairness resolutions*, and finally FS indicates *full simulation* proof including fairness. A check mark ✓ is put for easily identifying better techniques.

	[55]	[56]	[39]	[42]	[13]	Ours
Malicious Behavior	CC	CC	CC	CC/ZK	ZK	ZK
Fairness	GR	GR	GR	OFE^P	OFE^I	OFE ✓
Proof Technique	NS	NFS	NFS	FS ✓	NFS	FS ✓

[45]. The standard simulation definition [31] does not include fairness because of the impossibility result of fairness without honest majority [18]. Garay et al. [26] relax the notion of fairness and define a "commit-prove-fair-open" functionality to simulate gradual release to prove security and fairness together. Then, Kiraz and Schoenmakers [39] and Ruan et al. [56] use this functionality in their security proofs, but the *output indistinguishabilty* of the ideal and real worlds is *not* satisfied in their proofs because fairness is proven separately, without simulation. Therefore, none of these protocols has a proof in ideal-real world simulation paradigm that shows the protocol is both secure and fair. Cachin and Camenisch [13] give an ideal-real world definition that includes fairness and security together for protocols that employ a TTP. This definition considers also malicious TTP but the definition does not use TTP in optimistic model since the universal TTP in the ideal world contacts with the real world TTP in their ideal world definition. Interestingly, they do *not* prove their proposed protocol according to their definition. Lindell [42] defines a new ideal world definition that captures security and fairness as defined above (i.e., exchanging money in return is considered fair as well [6, 40]) and proves fairness and security according to this definition. In Section 8, we show that proving fairness and security *separately* do *not* necessarily yield to **fair and secure** protocols. The security and fairness definition we use follows the same intuition as that of Canetti [15].

Fairness for Specific Functionalities:

Finally, there are very efficient constructions for specific functionalities [33, 9, 24, 2, 3, 19], but we are interested in computing general functionalities efficiently. We achieve this goal, fairly, in the malicious setting.

10 Conclusion

Due to lack of space, we defer the full related work and our detailed performance analysis to the Appendix and just provide Table 3. Briefly, our protocol is the only one achieving full fairness with an efficient optimistic TTP, and proven fully simulatable.

- ✓ All our overhead (TTP, Alice, Bob) are dependent only on the output size, and **independent** of the circuit size, in contrast to [13].
- ✓ We require a **constant** number of rounds for fairness, contrary to gradual release based solutions [55, 39, 56].
- ✓ 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 [42].
- ✓ Even when a dispute occurs, the parties remain **anonymous**, and their computations remain **unlinkable**, since we do not need a public key infrastructure, in comparison to the certificate authority in [42].

- ✓ Even if the TTP becomes malicious and colludes with one participant, he **cannot violate the security of the protocol**. On the other hand, in [42], while the Bank cannot violate 2PC security, it can maliciously deal with the balances, possibly causing a lot of headache.
- ✓ Finally, our protocol is proven secure in the **ideal/real simulation** paradigm (not in [55]) with **output indistinguishability** (not in [39, 56]), and by proving **fairness and security simultaneously** (not in [13]).

11 Performance

In order to have fairness property in Γ protocol, each party needs to compute extra 4ℓ exponentiations, 6ℓ commitments, 2 signatures, 8ℓ verifiable encryptions, 4ℓ proof of knowledges and a verifiable escrow containing ℓ values (only Alice). Among all these computation, the most time consuming one is verifiable encryption. According to [48], a verifiable encryption with two values takes almost 318 milliseconds (ms) and with one value takes 257 ms. For the verifiable escrow containing ℓ values, we have at most 0.25ℓ seconds [4]. Therefore, we can conclude that each party spends almost 2.3ℓ seconds for the extra computations which is not a big overhead compared to the time that is spend for Γ protocol.

If Bob Resolve is executed, TTP does one signature, ℓ commitment check and one verifiable escrow decryption. The most time consuming part for TTP is decryption of verifiable escrow which takes time around 5ℓ ms [48].

Our protocol adds extra 10 messages with the joint generation, the equality test and the fair exchange while gradual release based solutions [55, 39, 56] require much more number of rounds at the final output exchange stage. When we examine our network overhead, we have extra 100.6ℓ KB (kilobyte) from verifiable encryptions and 18.8ℓ KB from verifiable escrow [48]. [5]

References

[1] I. Abraham, D. Dolev, R. Gonen, and J. Halpern. Distributed computing meets game theory: robust mechanisms for rational secret sharing and multiparty computation. In *Proceedings of the twenty-fifth annual ACM symposium on Principles of distributed computing*.

[2] G. Asharov. Towards characterizing complete fairness in secure two-party computation. In *Theory of Cryptography*, pages 291–316. Springer, 2014.

[3] G. Asharov, A. Beimel, N. Makriyannis, and E. Omri. Complete characterization of fairness in secure two-party computation of boolean functions. In *Theory of Cryptography*, pages 199–228. Springer, 2015.

[4] N. Asokan, V. Shoup, and M. Waidner. Optimistic fair exchange of digital signatures. *IEEE Journal on Selected Areas in Communications*, pages 591–610, 2000.

[5] D. Beaver and S. Goldwasser. Multiparty computation with faulty majority. In *CRYPTO*, 1990.

[6] M. Belenkiy, M. Chase, C. Erway, J. Jannotti, A. Küpçü, A. Lysyanskaya, and E. Rachlin. Making p2p accountable without losing privacy. In *ACM WPES*, 2007.

[7] D. Boneh and M. Naor. Timed commitments (extended abstract). In *CRYPTO*, 2000.

[8] F. Boudot. Efficient proofs that a committed number lies in an interval. In *EUROCRYPT*, 2000.

[9] F. Boudot, B. Schoenmakers, and J. Traoré. A fair and efficient solution to the socialist millionaires' problem. *Discrete Applied Mathematics*, (1-2):23–36, 2001.

[10] E. Bresson and J. Stern. Proofs of knowledge for non-monotone discrete-log formulae and applications. In *ISC*, 2002.

[4] The computation time of a verifiable escrow with ℓ values is less than 0.25ℓ seconds since it does not increase linearly but we assume here it is linear to guess average computation time.

[5] We do not consider other messages here since they are neglectable compared to verifiable encryption and escrow.

[11] E. F. Brickell, D. Chaum, I. Damgård, and J. v. d. Graaf. Gradual and verifiable release of a secret. In *CRYPTO*, 1987.

[12] Brownie cashlib cryptographic library. http://github.com/brownie/cashlib.

[13] C. Cachin and J. Camenisch. Optimistic fair secure computation. In *CRYPTO*, 2000.

[14] J. Camenisch and V. Shoup. Practical verifiable encryption and decryption of discrete logarithms. In *CRYPTO*, 2003.

[15] R. Canetti. Security and composition of multiparty cryptographic protocols. *Journal of Cryptology*, 13:143–202, 2000.

[16] D. Chaum. Blind signatures for untraceable payments. In *CRYPTO*, 1982.

[17] D. Chaum and T. P. Pedersen. Wallet databases with observers. In *CRYPTO*, 1993.

[18] R. Cleve. Limits on the security of coin flips when half the processors are faulty. In *STOC*, 1986.

[19] R. Cohen and Y. Lindell. Fairness versus guaranteed output delivery in secure multiparty computation. In *Advances in Cryptology–ASIACRYPT 2014*, pages 466–485. Springer, 2014.

[20] R. Cramer, I. Damgård, and B. Schoenmakers. Proofs of partial knowledge and simplified design of witness hiding protocols. In *CRYPTO*, 1994.

[21] I. Damgård. On sigma protocols. http://www.daimi.au.dk/ ivan/Sigma.pdf.

[22] I. Damgard and E. Fujisaki. A statistically-hiding integer commitment scheme based on groups with hidden order. In *ASIACRYPT*, 2002.

[23] I. B. Damgård. Practical and provably secure release of a secret and exchange of signatures. *Journal of Cryptology*, 8:201–222, 1995.

[24] C. Dong, L. Chen, J. Camenisch, and G. Russello. Fair private set intersection with a semi-trusted arbiter. In *DBSec*, 2013.

[25] E. Fujisaki and T. Okamoto. Statistical zero knowledge protocols to prove modular polynomial relations. In *CRYPTO*, 1997.

[26] J. A. Garay, P. MacKenzie, M. Prabhakaran, and K. Yang. Resource fairness and composability of cryptographic protocols. *Journal of cryptology*, 2011.

[27] O. Goldreich. *Foundations of Cryptography: Volume 2, Basic Applications*. Cambridge University Press, New York, NY, USA, 2004.

[28] O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game. In *STOC*, 1987.

[29] O. Goldreich, S. Micali, and A. Wigderson. Proofs that yield nothing but their validity or all languages in np have zero-knowledge proof systems. *Journal of ACM*, 38:728, 1991.

[30] S. Goldwasser and L. Levin. Fair computation of general functions in presence of immoral majority. In *CRYPT0*. 1991.

[31] S. Goldwasser and Y. Lindell. Secure computation without agreement. In *Distributed Computing*. Springer, 2002.

[32] S. Goldwasser, S. Micali, and R. Rivest. A digital signature scheme secure against adaptive chosen-message attacks. *SIAM Journal on Computing*, 17:281–308, Apr. 1988.

[33] S. D. Gordon, C. Hazay, J. Katz, and Y. Lindell. Complete fairness in secure two-party computation. *Journal of ACM*, 58, 2011.

[34] S. D. Gordon and J. Katz. Partial fairness in secure two-party computation. *Journal of cryptology*, 2012.

[35] A. Groce and J. Katz. Fair computation with rational players. In *EUROCRYPT*. 2012.

[36] J. Halpern and V. Teague. Rational secret sharing and multiparty computation. In *Proceedings of the thirty-sixth annual ACM symposium on Theory of computing*, 2004.

[37] R. Henry and I. Goldberg. Batch proofs of partial knowledge. In *ACNS*, 2013.

[38] S. Jarecki and V. Shmatikov. Efficient two-party secure computation on committed inputs. In *EUROCRYPT*, 2007.

[39] M. S. Kiraz and B. Schoenmakers. An efficient protocol for fair secure two-party computation. In *CT-RSA*, 2008.

[40] A. Küpçü and A. Lysyanskaya. Usable optimistic fair exchange. *Computer Networks*, 56:50–63, 2012.

[41] A. Küpçü. *Efficient Cryptography for the Next Generation Secure Cloud: Protocols, Proofs, and Implementation.* Lambert Academic Publishing, 2010.

[42] A. Y. Lindell. Legally-enforceable fairness in secure two-party computation. In *CT-RSA*, 2008.

[43] Y. Lindell. Fast cut-and-choose based protocols for malicious and covert adversaries. In *CRYPTO*, 2013.

[44] Y. Lindell and B. Pinkas. An efficient protocol for secure two-party computation in the presence of malicious adversaries. In *EUROCRYPT*, 2007.

[45] Y. Lindell and B. Pinkas. Secure multiparty computation for privacy-preserving data mining. *Journal of Privacy and Confidentiality*, 1, 2009.

[46] Y. Lindell and B. Pinkas. Secure two-party computation via cut-and-choose oblivious transfer. *Journal of cryptology*, pages 680–722, 2012.

[47] A. Lysyanskaya and N. Triandopoulos. Rationality and adversarial behavior in multi-party computation. In *CRYPTO*. 2006.

[48] S. Meiklejohn, C. C. Erway, A. Küpçü, T. Hinkle, and A. Lysyanskaya. Zkpdl: A language-based system for efficient zero-knowledge proofs and electronic cash. In *USENIX Security Symposium*, 2010.

[49] P. Mohassel and M. Franklin. Efficiency tradeoffs for malicious two-party computation. In *PKC*, 2006.

[50] P. Mohassel and B. Riva. Garbled circuits checking garbled circuits: More efficient and secure two-party computation. In *CRYPTO*. 2013.

[51] J. B. Nielsen and C. Orlandi. Lego for two-party secure computation. In *TCC*, 2009.

[52] S. J. Ong, D. C. Parkes, A. Rosen, and S. Vadhan. Fairness with an honest minority and a rational majority. In *Theory of Cryptography*. 2009.

[53] S. J. Ong, D. C. Parkes, A. Rosen, and S. Vadhan. Fairness with an honest minority and a rational majority. In *Theory of Cryptography*. 2009.

[54] T. P. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In *CRYPTO*, 1992.

[55] B. Pinkas. Fair secure two-party computation. In *EUROCRYPT*, 2003.

[56] O. Ruan, J. Chen, J. Zhou, Y. Cui, and M. Zhang. An efficient fair uc-secure protocol for two-party computation. *Security and Communication Networks*, 2013.

[57] O. Ruan, J. Zhou, M. Zheng, and G. Cui. Efficient fair secure two-party computation. In *IEEE APSCC*, 2012.

[58] C.-P. Schnorr. Efficient signature generation by smart cards. *Journal of Cryptology*, pages 161–174, 1991.

[59] C.-h. Shen et al. Two-output secure computation with malicious adversaries. In *EUROCRYPT*. 2011.

[60] V. Shoup and R. Gennaro. Securing threshold cryptosystems against chosen ciphertext attack. In *EUROCRYPT*, 1998.

[61] D. P. Woodruff. Revisiting the efficiency of malicious two-party computation. In *EUROCRYPT*, 2007.

[62] A. C. Yao. Protocols for secure computations. In *FOCS*, 1982.

A Sub Protocols

A.1 Joint Generation

For the joint generation, Bob first commits to four random elements from $\mathbb{Z}_n$ and proves the knowledge of the committed values and their ranges [12, 8]. Then, Alice chooses four random elements from the same range and sends them to Bob. Finally, Bob decommits. Then they both calculate the multiplication of pairs of these numbers, in $\mathbb{Z}_n$, and so jointly agree on random numbers $\rho_{a,0}, \rho_{a,1}, \rho_{b,0}, \rho_{b,1}$. In the end, they calculate equality test constants where $\{e_{z,t} = g^{\rho_{z,t}}\}_{z\in\{a,b\},t\in\{0,1\}}$.

A.2 Check Correctness

We present the subprotocols from Alice's viewpoint as the prover. Bob's proofs are symmetric.

Proof of Output Gates:

Inputs: Common inputs are the garbled tables of gate g that contains encryptions $E_{00}, E_{01}, E_{10}, E_{11}$, along with the commitments $\mathsf{S}_i^A, ..., \mathsf{S}_{i+3}^A$ of the random row pairs of g. Prover's private inputs are all decommitments. Prover performs following:

- $\boldsymbol{ZKVerifyEnc}(E_{00}, \mathsf{S}_i^A) \wedge \boldsymbol{ZKVerifyEnc}(E_{01}, \mathsf{S}_{i+1}^A) \wedge \boldsymbol{ZKVerifyEnc}(E_{10}, \mathsf{S}_{i+2}^A) \wedge \boldsymbol{ZKVerifyEnc}(E_{11}, \mathsf{S}_{i+3})$, where $\boldsymbol{ZKVerifyEnc}(E, \mathsf{S})$ proves that the encryption E contains the decommitment of S.

Proof of Correct Input Gates:

Inputs: Common inputs are the commitments $\mathsf{D}_{w_b,0}^A, \mathsf{D}_{w_b,1}^A$ and the garbled input gate table that has encryptions $E_{00}, E_{01}, E_{10}, E_{11}$. Prover knows the decommitments. Prover performs the following (details in [38] or appendix of [41]):

- According to prover's side, he performs one of the proofs below. If she is from the left side (in our protocol it is Alice's side) then she performs ***ZKVerifyEncRight***, otherwise he performs ***ZKVerifyEncLeft***. Each $\boldsymbol{ZKVerifyEnc}(E, \mathsf{D})$ shows that the encryption E contains the decommitment of the commitment D. This proof can be done as in [14].
  - $\boldsymbol{ZKVerifyEncRight} = \boldsymbol{ZKVerifyEnc}(E_{00}, \mathsf{D}_{w_b,0}^A) \wedge \boldsymbol{ZKVerifyEnc}(E_{01}, \mathsf{D}_{w_b,1}^A) \wedge \boldsymbol{ZKVerifyEnc}(E_{10}, \mathsf{D}_{w_b,0}^A) \wedge \boldsymbol{ZKVerifyEnc}(E_{11}, \mathsf{D}_{w_b,1}^A)$
  - $\boldsymbol{ZKVerifyEncLeft} = \boldsymbol{ZKVerifyEnc}(E_{00}, \mathsf{D}_{w_b,0}^A) \wedge \boldsymbol{ZKVerifyEnc}(E_{01}, \mathsf{D}_{w_b,0}^A) \wedge \boldsymbol{ZKVerifyEnc}(E_{10}, \mathsf{D}_{w_b,1}^A) \wedge \boldsymbol{ZKVerifyEnc}(E_{11}, \mathsf{D}_{w_b,1}^A)$
- $\boldsymbol{ZKComDL}(\mathsf{D}_{w_b,0}^A) \wedge \boldsymbol{ZKComDL}(\mathsf{D}_{w_b,1}^A)$ proving knowledge of the decommitments of these commitments.
- $\boldsymbol{ZKEqComDL}$ $(\mathsf{D}_{w_b,0}^A, \mathsf{D}_{w_b,1}^A)$ showing that the committed messages are equal for both commitments (under different bases, $e_{b,0}$ and $e_{b,1}$).

A.3 Equality Test

The adapted protocol of [9] is the following:

- Alice and Bob generate a cyclic abelian group G, which has a large prime order q, with generators g_0, g_1, g_2, g_3. Then, they compute the corresponding values of their input bits (for equality test) in $\mathbb{Z}_q$. In our case, these are the hashed values of all left (right) hand sides of equation 1. Say Alice's value is θ_a and Bob's is θ_b.
- Alice chooses a random $x_a \in \mathbb{Z}_q$ and computes $g_a = g_1^{x_a}$ and similarly Bob chooses a random $x_b \in \mathbb{Z}_q$ and computes $g_b = g_1^{x_b}$. Then, they send these values to each other and calculate $g_{ab} = g_a^{x_b} = g_b^{x_a}$. In addition, they prove knowledge of x_a and x_b using Schnorr's protocol [58].
- Alice selects $a \in_R \mathbb{Z}_q$, calculates $(P_a, Q_a) = (g_3^a, g_1^a g_2^{\theta_a})$, and sends (P_a, Q_a) to Bob. Bob does the symmetric version, computing and sending $(P_b, Q_b) = (g_3^b, g_1^b g_2^{\theta_b})$, where $b \in_R \mathbb{Z}_q$. Alice calculates $R_a = (Q_a/Q_b)^{x_a}$. Bob also calculates $R_b = (Q_a/Q_b)^{x_b}$. Then, Alice sends R_a with a proof that $log_{g_1} g_a = log_{Q_a/Q_b} R_a$ to Bob [17].
- Bob can now learn the result of the equality test by checking whether $P_a/P_b = R_a^{x_b}$. Then, he sends same proof as above to show $log_{g_1} g_b = log_{Q_a/Q_b} R_b$ so that Alice also learns the equality test result.

Note that if Bob does not send the last message, he will not obtain the equality signature s_{eq} in our protocol, and the whole protocol will be aborted without anyone learning the actual output.