Appears in *Cryptographers' Track-RSA Conference (CT-RSA 2008)*, *Lecture Notes in Computer Science* 4964 (2008) 88–105. Springer-Verlag.

An Efficient Protocol for Fair Secure Two-Party Computation

Mehmet S. Kiraz and Berry Schoenmakers

Dept. of Mathematics and Computer Science, TU Eindhoven
P.O. Box 513, 5600 MB Eindhoven, The Netherlands
m.kiraz@tue.nl, berry@win.tue.nl

Abstract. [⋆] In the 1980s, Yao presented a very efficient constant-round secure two-party computation protocol withstanding semi-honest adversaries, which is based on so-called garbled circuits. Later, several protocols based on garbled circuits covering malicious adversaries have been proposed. Only a few papers, however, discuss the fundamental property of fairness for two-party computation. So far the protocol by Pinkas (Eurocrypt 2003) is the only one which deals with fairness for Yao's garbled circuit approach.

In this paper, we improve upon Pinkas' protocol by presenting a more efficient variant, which includes several modifications including one that fixes a subtle security problem with the computation of the so-called majority circuit. We prove the security of our protocol according to the real/ideal simulation paradigm, as Lindell and Pinkas recently did for the malicious case (Eurocrypt 2007).

1 Introduction

In secure two-party computation there are two parties who are interested in evaluating a public function $f(x, y) = (f_1(x, y), f_2(x, y))$ where x and y are their respective private inputs, and the first party wants to know the value $f_1(x, y)$ and the other party wants to know $f_2(x, y)$ without disclosing more information about their inputs than what is implied by the outputs. There might be only common output ($f_1(x, y) = f_2(x, y)$), or one party receiving no output (e.g., $f_2(x, y) = \perp$).

In his seminal paper, Yao [14] presented a protocol for secure two-party computation in the semi-honest model where the adversary follows the protocol specifications but stores all intermediate values which may be analyzed later to learn additional information. He used a tool called a *garbled circuit*, an encrypted form of a Boolean circuit that implements the function $f(x, y)$. Roughly speaking, in Yao's protocol, the garbled circuit is constructed by one party (Bob), and

[⋆] Mehmet Kiraz's PhD thesis (TU Eindhoven, August 2008) presents an improved version for our main protocol, essentially hiding the index r of the majority circuit from Bob. This eliminates a problem which was pointed out to us by Benny Pinkas and which—in retrospect—is present in all protocols derived from [11].

it is evaluated by the other party (Alice). Recently, several papers appeared, extending Yao's protocol to the malicious case by using *cut-and-choose* techniques [7,3,13,5,6]. However, these protocols do not ensure fairness. Informally speaking, a protocol is fair if either both parties learn their (private) outputs, or none of them learns anything. So, a fair protocol ensures that a malicious party cannot gain an advantage by aborting the protocol before the other (honest) party gets its output. Pinkas [11] presented the first fair and secure two-party protocol based on Yao's garbled circuits, which is the starting point of this work.

One of the main ideas of [11] is that the evaluation of the garbled circuit, as performed by Alice, does not result in garbled values (one per output wire) but—roughly speaking—in *commitments* to these garbled values. Basically, Alice will hold commitments for Bob's output wires, and, v.v., Bob will hold commitments for Alice's output wires. The important point here is that both parties are convinced of the correctness of the values contained in the commitments held by the other party. For this we need some special protocol techniques. In [11], blind signatures are used as a building block to achieve correctness of these commitments. However, blind signatures make Pinkas' protocol rather complex and inefficient. In our protocol, we avoid the use of blind signatures, resulting in a savings of a factor k, where k is a security parameter.

Once the correctness of the commitments for the output values is guaranteed, both parties will gradually open their commitments (bit by bit). For this gradual release phase we will use the protocol of [4] implementing their "commit-prove-fair-open" functionality, which is actually proved correct according to the real/ideal simulation paradigm. We will use their results in a black-box way.

1.1 Our Contributions

There have been several recent advances for Yao's garbled circuit approach [3,5,13,6]. In this paper, we revisit fairness while borrowing several improvements and ideas from these recent papers. We thus improve upon the protocol by Pinkas [11] which is the only paper considering fairness for Yao's garbled circuit approach.

Pinkas presents an intricate method which involves a modification of the truth tables for the garbled circuits. A crucial part of his protocol is to let the parties convince each other of the correctness of the committed values for the output wires, as explained above. The difficulty is to show the correctness of Alice's commitments for Bob's output wires. Concretely, blind signatures are used in [11] as a building block for the verification of these commitments (where Bob is the signer, Alice is the receiver of the blind signatures, and a further cut-and-choose subprotocol is used to ensure that Bob only signs correctly formed (blinded) commitments). Instead, we use the well-known OR-proofs [2], to let Alice show that she committed correctly to the garbled output values that she obtained for each of Bob's output wires. Alice needs to render these OR-proofs only for the so-called majority circuit. As a consequence, whereas Pinkas' protocol uses $2\ell m\kappa$ blind signatures, our protocol only needs ℓ OR-proofs (but also ℓm homomorphic commitments), where ℓ is the number of output wires for Bob, m is the number of circuits used for cut-and-choose, and κ is another security parameter (used for cut-and-choose as well). Overall, this leads to an improvement by a factor of κ in computational complexity, and also a significant improvement in communication complexity.

The above application of OR-proofs also critically relies on a slight modification of the circuit representing f, where a new input wire (for Bob) is introduced for every output wire of Bob. Bob will use random values for these new input wires, to blind the values of his output wires. Nevertheless, Alice will still be able to determine a majority circuit. This modification was suggested to us by Pinkas to resolve a subtle problem for the protocol of [11], which we communicated to him [10]; the problem is that a corrupted Alice may learn Bob's private inputs.

The security of our protocol is analyzed according to the real/ideal simulation paradigm, following the proof by Lindell and Pinkas [6] for the malicious case. However, we note that the failure probability for their simulator is quite large in the case that Bob is corrupted; namely $2^{1-m/17}$, where m is the number of circuits used, for a certain type of failure—in addition to other types of failures. This requires a large value for m, even for a low level of security. By two further extensions to the circuit representing f (the first of which actually suffices for [6]), we are able to reduce the failure probability to $2^{-m/4}$ (which we believe is optimal for this type of cut-and-choose protocols relying on majorities).

Roadmap

In Section 2, we describe the problem which covers fairness, the importance of the majority circuit computation and the problem mentioned above for the protocol by Pinkas. In Section 3, we present our improved fair and secure two-party protocol, in Section 4 we analyze its performance and compare with the protocol by Pinkas. Finally, in Section 5 we analyze the security of our protocol, where we also describe how to modify the circuit for f in order to get a more efficient simulation.

2 Main Protocol Ideas

In this section we briefly review some issues for Pinkas' protocol, and we present the main ideas behind our new protocol. We refer to [11] for a list of references on fairness.

Gradual Release. Let us first mention that for the gradual release phase, which concludes our protocol, we will use the protocol for the "commit-prove-fair-open" functionality of [4]. Note that Pinkas [11, Appendix A] uses a special protocol for this, based on *timed-commitments*. We will use the "commit-prove-fair-open" functionality in a black-box manner. The only thing we need to take care of before reaching the gradual release phase is that Alice and Bob hold commitments to the correct values for each other's output wires.

The intuition behind these gradual release protocols (which do not rely on a trusted party) is that Alice and Bob open the commitments bit-by-bit in such a way that the other party can directly check the correctness of each bit so released. Also, a party prematurely aborting the protocol cannot perform the search for the remaining bits in parallel, and [4] even shows how this can be done in a simulatable way.

Majority Circuits. We consider protocols which let Alice decide (at random) for each of m garbled circuits constructed by Bob, whether such a circuit must be opened by Bob (for checking by Alice), or whether the circuit will be evaluated by Alice. Exactly $m/2$ circuits will be checked by Alice, and she will evaluate the remaining half. Among the evaluated circuits there may still be a few incorrect circuits (not computing function f), but with high probability the majority of the evaluated circuits is correct.

We actually need to deal with two types of majority circuits. One is related to Bob's output wires (which we will indicate by index r throughout the paper; to ensure that Bob cannot get any information on Alice's inputs), and one is related to Alice's output wires (to ensure that Alice is guaranteed to get correct output values). These majority circuits can be characterized as follows. First, the *majority value* for an output wire is the bit value that occurs most frequently for that wire when considered over all $m/2$ evaluated circuits (ties can be broken arbitrarily). Further, an output wire is called a *majority wire* if its value is equal to the majority value. And, finally, a circuit is called a *majority circuit for Alice resp. Bob* if all of Alice's resp. Bob's output wires are majority wires.

Next, we explain why Alice should to send the output values only for a majority circuit, rather than sending Bob the output values for all evaluated circuits. For example, Bob could have constructed $m - 1$ garbled circuits that compute the function $f(x, y)$ and a single circuit which simply outputs Alice's input values. Since Alice evaluates $m/2$ out of m circuits, the probability would be $1/2$ that this single circuit is selected by Alice. Therefore, Bob would receive the outputs of the incorrect circuit from Alice with probability $1/2$ at the end of the evaluation.

Note that the computation of majority circuit for Bob can be avoided altogether for a protocol tolerating malicious adversaries but not achieving fairness. Namely, Lindell and Pinkas in [6, Section 2.2] show that it suffices to consider the case that only Alice receives private output. It is not clear, however, whether this protocol can be extended to cover fairness as well.

Problem with Pinkas' Computation of Majority Circuit for Bob. Omitting details, the protocol by Pinkas reaches a state in which for each evaluated garbled circuit GC_j and for each output wire i of Bob, Bob knows a random bit k_{ij} and Alice knows the value $b_{ij} \oplus k_{ij}$, where b_{ij} is the actual output bit for wire i in circuit GC_j. Alice and Bob then use these values to determine a majority circuit r for Bob. Pinkas proposes that Alice can be trusted to construct a garbled circuit for this task, as Alice needs this majority circuit to prevent Bob from cheating. But this way, nothing prevents Alice from constructing an arbitrary circuit which reveals some of Bob's input values and hence some of the $k_{i,j}$ values. Then Alice learns Bob's actual output bits, which should not be possible. Of course, this problem can be solved by running any two-party protocol which is secure against malicious parties (e.g., [6]). However, in this paper, we will not require any additional protocol for computing a majority circuit for Bob. We present a simple modification to the circuit and in this way we show that a majority circuit can be computed without considerable additional cost.

Modified Circuit Randomizing Bob's Output Wires. Alice needs to be able to determine a majority circuit for Bob, but at the same time she should not learn Bob's actual output values. Let C_f denote a circuit computing function $f(x, y)$. We extend circuit C_f to a *randomized circuit* RC_f, as follows [10]. See Figure 1. Hence, for each output wire W_i of Bob, a new input wire W_i' is added as well as a new output wire W_i'', such that $W_i'' = W_i \oplus W_i'$.

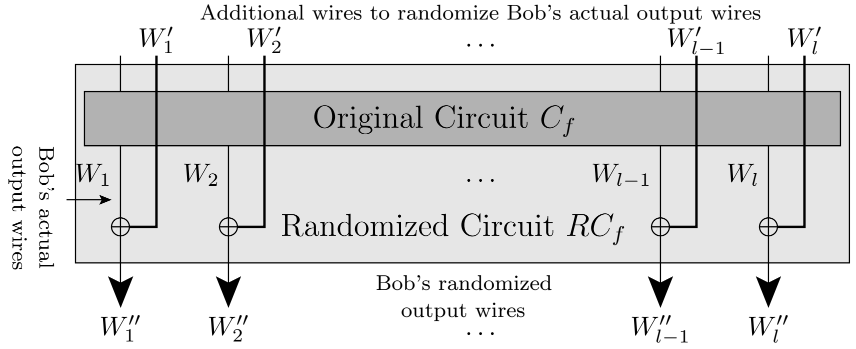

Additional wires to randomize Bob's actual output wires
W_1'
W_2'
...
W_{l-1}'
W_l'
Original Circuit C_f
Bob's actual output wires
W_1
W_2
...
W_{l-1}
W_l
Randomized Circuit RC_f
Bob's randomized output wires
W_1''
W_2''
...
W_{l-1}''
W_l''

Fig. 1. The randomized circuit RC_f made from the original circuit C_f.

Correctness of Garbled Input Values. Bob is not only providing Alice with m garbled circuits, but also with garbled values for each input wire for each circuit to be evaluated. It must be ensured that these garbled values are *correct* (i.e., correspond to the actual input value and fit with the garbled truth tables of the garbled circuits).

The correctness of Alice's garbled inputs will basically follow by definition of 1-out-of-2 oblivious transfer (OT) as in [11]. However, as pointed out in [5], one cannot use plain OT for the malicious case; rather a stronger form of OT such as committed OT, or the potentially weaker form of *committing OT* should be used. Here, committing OT is like plain OT with the added requirement upon completion of the protocol, the sender is committed to both of the input values it used during the protocol. The additional commitments output by the committing OT protocol will be opened by Bob (sender in OT) for the circuits which are checked by Alice.

The correctness of Bob's garbled inputs is not as straightforward to handle. Pinkas [11] originally used OR-proofs [2], whereas later papers [3,13,6]) aimed at using less expensive techniques relying on symmetric operations only (used in combination with cut-and-choose). In this paper, we use the *equality-checker scheme* of Franklin and Mohassel [3] for proving correctness of Bob's inputs.

The equality-checker scheme roughly runs as follows. For each pair of garbled circuits GC_j and $GC_{j'}$ with $1 \leq j < j' \leq m$, and for each input wire i for Bob, and for each $b \in \{0,1\}$, Bob commits to the garbled values $w_{i,j,b}$ and $w_{i,j',b}$. Alice will check the consistency of these commitments for each pair of garbled circuits that are both opened by Bob. This implies with overwhelming probability that Bob's inputs to any two circuits GC_j and $GC_{j'}$ are equal(see [3] for more details).

Correctness of Committed Outputs. In order that the two parties can safely enter the gradual release phase, one of the main problems that needs to be solved is that both parties are convinced of the correctness of the values contained in the commitments held by the other party. We treat this problem differently for Alice and Bob.

Bob's commitments to Alice's outputs will be guaranteed to be correct by cut-and-choose, exactly as in [11]. For Alice's commitments to Bob's outputs, however, we will use a different approach than in [11], which used blind signatures for this purpose. In our protocol, Alice will first obtain the garbled values for Bob's outputs for all evaluated circuits, and she commits to all of these values. At that point, Bob can safely reveal both garbled values for each output wire (of the randomized circuit, as described above). Additional commitments from Bob will ensure that these garbled values are correct. Finally, Alice proves that she committed to one of these garbled values, from which Bob deduces that Alice indeed committed to correct values for Bob's output wires.

Concretely, we let Alice produce an OR-proof as follows. Suppose Alice committed to a garbled value $w_{i,j}$ for output wire i of Bob in circuit GC_j, and that she received garbled values $w_{i,j,0}$ and $w_{i,j,1}$ from Bob. Using homomorphic commitments, such as Pedersen commitments [9], Alice can prove that either $w_{i,j} = w_{i,j,0}$ or $w_{i,j} = w_{i,j,1}$ without revealing which is the case, applying [2] to the Chaum-Pedersen protocol for proving equality of discrete logarithms [1]. We will use the Fiat-Shamir heuristic to make these proofs non-interactive (and provably secure in the random oracle model).

3 A Fair Secure Two-Party Protocol

The object of the protocol is to evaluate a function of the form $f(x,y) = (f_1(x,y), f_2(x,y))$ securely, where Alice holds input x and gets output $f_1(x,y)$ and Bob holds input y and gets output $f_2(x,y)$. For simplicity, we assume that these inputs and outputs are all of equal length, i.e., $f : \{0,1\}^\ell \times \{0,1\}^\ell \rightarrow \{0,1\}^\ell \times \{0,1\}^\ell$, for some integer value ℓ.

Let RC_f denote the randomized boolean circuit for function f, see Figure 1. We use I_A resp. O_A to denote the set of Alice's input resp. output wires, and I_B resp. O_B to denote the set of Bob's input resp. output wires. Furthermore, we use I'_B to denote the additional input wires for Bob, used in the construction of RC_f from C_f. Note that $|I_A| = |I_B| = |I'_B| = |O_A| = |O_B| = \ell$. Accordingly, we write $x = \langle x_i \in \{0,1\} : i \in I_A \rangle$ for Alice's input, $y = \langle y_i \in \{0,1\} : i \in I_B \rangle$ for Bob's input, and $z = \langle z_i \in \{0,1\} : i \in I'_B \rangle$ for Bob's random input to RC_f. Further, $|RC_f|$ denotes the number of gates in the circuit RC_f, and we $\mathcal{W}$ denote the set of all wires in the circuit RC_f. Hence, $I_A \cup I_B \cup I'_B \cup O_A \cup O_B \subseteq \mathcal{W}$.

In Phase 3 of the protocol, Bob will generate m garbled versions of the circuit RC_f, where m is a security parameter. We will denote these garbled circuits by GC_j, for $j = 1, \ldots, m$. A garbled circuit GC_j for RC_f is completely determined by the pair of garbled values $(w_{i,j,0}, w_{i,j,1})$ assigned by Bob to each wire $i \in \mathcal{W}$. Here, $w_{i,j,b} \in \{0,1\}^k$ corresponds to bit value $b \in 0, 1$, where k is another security parameter, denoting the length of the garbled values.

We abstract away most of the details of how garbled circuits are generated and evaluated. For our purposes it suffices to view a garbled circuit (as to be evaluated by Alice) as a concatenation of permuted-encrypted-garbled-4-tuples, one for each (binary) gate in RC_f, and of permuted ordered pairs, one for each output wire of Alice: $GC_j = \langle\langle \text{PEG-4-tuple}_{n,j} : 1 \leq n \leq |RC_f| \rangle, \langle POP_{i,j} : i \in O_A \rangle\rangle$. The permuted ordered pairs $POP_{i,j}$ are generated at random by Bob, using the garbled values $w_{i,j,0}$ and $w_{i,j,1}$ assigned to wire $i \in O_A$ in circuit j: $POP_{i,j} = w_{i,j,\sigma_{i,j}}, w_{i,j,1-\sigma_{i,j}})$, where $\sigma_{i,j} \in_R \{0,1\}$.

In the protocol we use two types of commitments, namely homomorphic ('asymmetric') commitments, e.g., Pedersen commitments [9], and other ('symmetric') commitments, e.g., constructed from pseudorandom generators [8]. We let $\text{commit}_P(m; r)$ denote a symmetric commitment to a message m using randomness r generated by party P, and we use $\text{commit}^h_P(m; r)$ to denote homomorphic commitments.

The protocol consists of 10 phases (see also Appendix A for a protocol diagram).

Phase 1 [Generation of garbled input values.] Bob generates garbled values $w_{i,j,b} \in_R \{0,1\}^k$, for $i \in I_A$, $j = 1, \ldots, m$, and $b \in \{0,1\}$.

Phase 2 [Committing OT.] Committing OT is run in order for Alice to learn her garbled input values. Bob is the sender with private input $w_{i,j,0}, w_{i,j,1}$ for $i \in I_A$ and $j = 1, \ldots, m$ which were generated in Phase 1, and Alice is the receiver with private input $x_i \in \{0,1\}$. At the end of committing OT Alice receives w_{i,j,x_i} and Bob gets no information about which of his inputs is chosen. Also, the common output is $A^{OT}_{i,j,b} = \langle\ \text{commit}_B(w_{i,j,b}; \alpha_{i,j,b})\ \rangle$ for $i \in I_A$, $j = 1, \ldots, m$, $b \in \{0,1\}$. These commitments will be checked by Alice later on, in order to prevent cheating by Bob; in particular to avoid the protocol issue addressed in [5].

Phase 3 [Construction.] In this phase Bob does the following:
−He prepares the garbled circuits GC_j for $j = 1, \ldots, m$ such that the garbled values $w_{i,j,0}, w_{i,j,1}$ for $i \in I_A$ and $j = 1, \ldots, m$ which are generated in Phase 1 are used for the corresponding wires.
−He also generates the commitments $B_{i,j,j',b}$=$\text{commit}_B(w_{i,j,b}, w_{i,j',b}; \beta_{i,j,j',b})$ for $i \in I_B \cup I'_B$ and j, j' such that $1 \leq j < j' \leq m, b \in \{0,1\}$ for the equality-checker scheme. $B_{i,j,j',b}$'s are committed to Bob's garbled input values and they are generated to ensure that Bob's input is consistent for all the circuits (see

Section 2).
$-$He also computes the commitments $C_{i,j}$=commit$_B(\sigma_{i,j};\gamma_{i,j})$ for $i \in O_A$ and $j = 1, \ldots, m$ where $\sigma_{i,j} \in_R \{0, 1\}$. These committed values are used to permute Alice's output values and the correctness will be proved by the cut-and-choose technique, by opening half of them in the opening phase.
$-$Finally in this phase, the commitments $D_{i,j}$=commit$_B(w_{i,j,0}, w_{i,j,1}; \delta_{i,j})$ for $i \in O_B$ and $j = 1, \ldots, m$ are computed. The $D_{i,j}$'s are committed to Bob's garbled output values and they are generated so that Alice can determine a correct majority circuit.

He sends the circuits and the commitments generated above to Alice. Each pair of commitments $(B_{i,j,j',0}, B_{i,j,j',1})$ is sent in random order, in order that Alice does not learn Bob's inputs when Bob opens one commitment for each of these pairs later on in the evaluation phase.

Phase 4 [Challenge.] Alice and Bob run the challenge phase (a coin-tossing protocol) in order to choose a random bit string $\ell = \ell_1 \ldots \ell_m \in_R \{0, 1\}^m$ that defines which garbled circuits and which commitments will be opened.

Phase 5 [Opening & Checking.] $-$Alice asks Bob to open the circuits GC_j for j such that $\ell_j = 1$ which are called *check-circuits*. Similarly, the circuits GC_j for j such that $\ell_j = 0$ will be called *evaluation-circuits*. She also asks Bob to open the corresponding commitments for j such that $\ell_j = 1$. Bob sends the following for opening:
$-$Bob sends the opening set $\widetilde{GC}_j$= $\langle$ $w_{i,j,b}$: $i \in \mathcal{W}$ $\rangle$, for j such that $\ell_j = 1$, $b \in \{0, 1\}$ to open the check-circuits.
$-$He also sends $\widetilde{A}^{OT}_{j,b}$= $\langle$ $\alpha_{i,j,b}$: $i \in I_A$ $\rangle$ for j such that $\ell_j = 1$, $b \in \{0, 1\}$ in order to open the corresponding commitments $A^{OT}_{i,j,b}$.
$-$He also sends the opening set $\widetilde{B}_{j,j',b}$= $\langle$ $\beta_{i,j,j',b}$: $i \in I_B \cup I'_B$ $\rangle$ for j, j' such that ℓ_j=$\ell_{j'}$ =1, $1 \leq j < j' \leq m$, $b \in \{0, 1\}$ to open the corresponding commitments $B_{i,j,j',b}$.
$-$The opening set $\widetilde{C}_j$= $\langle$ $\sigma_{i,j}$,$\gamma_{i,j}$: $i \in O_A$ $\rangle$ for j such that $\ell_j = 1$ is also sent in this phase to open the corresponding commitments $C_{i,j}$.
$-$The opening set $\widetilde{D}_j$= $\langle$ $\delta_{i,j}$: $i \in O_B$ $\rangle$ for j such that $\ell_j = 1$ is also sent to open the corresponding commitments $D_{i,j}$.
$-$The opening set $\widetilde{B}^{input}_{j,j'}$= $\langle$ w_{i,j,y_i},w_{i,j',y_i}, β_{i,j,j',y_i}, $w_{i',j,z_{i'}}$,$w_{i',j',z_{i'}}$, $\beta_{i',j,j',z_{i'}}$: $i \in I_B$, $i' \in I'_B$ $\rangle$ for j, j' such that ℓ_j=$\ell_{j'}$ =0, 1$\leq$ j <j' $\leq m$. This set contains the garbled values of Bob's input wires for the evaluation-circuits, and sent to Alice which is a part of the equality-checker scheme.

Alice verifies the circuits and the commitments. Note that the consistency check of Bob's input is done now by the equality-checker scheme with the commitment set $\widetilde{GC}_j$ (contains all garbled values) for j such that $\ell_j = 1$ and $b \in 0, 1$ and the set $\widetilde{B}_{j,j',b}$ for j, j' such that $\ell_j = \ell_{j'} = 1, 1 \leq j < j' \leq m$ and $b \in \{0, 1\}$. Note that the opening sets $\widetilde{A}^{OT}_{j,b}$, $\widetilde{B}_{j,j',b}$ and $\widetilde{D}_j$ contain only randomness since the corresponding garbled values comes already from the set $\widetilde{GC}_j$. If any of the verifications fail Alice aborts the protocol.

Phase 6 [Evaluation.] Alice does the following in the evaluation phase:
$-$She first evaluates the circuits GC_j for $\ell_j = 0$ and computes garbled output values.
$-$She then commits to Bob's output values as $E_{i,j} = \text{commit}^h_A(w_{i,j}; \zeta_{i,j})$ for $i \in O_B$ and j such that $\ell_j = 0$ and sends them to Bob. Note that the commitments $E_{i,j}$ are generated to assure Bob that the committed values in $E_{i,j}$ are circuit values. If, for example, Alice commits to values different from garbled output values then she will be detected in OR-proofs in Phase 9. The crucial property we need here is that these commitments are homomorphic in order to be able to use in OR-proofs.

Phase 7 [Opening of Bob's ordered output.] After Bob receives the commitments $E_{i,j}$ for $i \in O_B$ and j such that $\ell_j = 0$ he opens the commitments $D_{i,j}$ by sending the opening set $\widetilde{D_j} = \langle\ w_{i,j,0}, w_{i,j,1}, \delta_{i,j}\text{: } i \in O_B\ \rangle$ for j such that $\ell_j = 0$. Note that the commitments $D_{i,j}$ can be opened since Bob's outputs are randomized (see Figure 1); hence Alice can only see which outputs match (and determine a majority circuit), but she does not learn Bob's output $f_2(x, y)$.

Phase 8 [Decision of majority circuit.] Alice determines a majority circuit GC_r for some r such that $\ell_r = 0$. Note that she can determine a correct majority circuit GC_r without further interaction with Bob since the additional input values that were used to randomize Bob's output wires are all identical for the same wire in each circuit GC_j for j such that $\ell_j = 0$.

Phase 9 [Verification of Alice's commitments.] They run OR-proofs where Alice is the prover, Bob is the verifier. Alice proves that the committed value inside $E_{i,r}$ is either equal to $w_{i,r,0}$ or $w_{i,r,1}$. Alice cannot cheat here, otherwise she has to guess a garbled value, which has chances at most negligibly in k.

Phase 10 [Gradual release.] They then run the protocol for the gradual release to open their respective commitments, namely, $C_{i,j}$'s and $E_{i,j}$'s. At the end of the gradual release:
$-$Alice learns all $\sigma_{i,j}$ for $i \in O_A$ and $j \in U$ and applies it to $POP_{i,j}$ to learn her actual outputs for j circuits. She takes the majority of j circuits which will result in $f_1(x, y)$.
$-$Bob also matches the garbled output values that are received from Alice and the additional wire values in terms of bits (he knows the garbled values and the corresponding bits). Bob finally computes his output $f_2(x, y)$ by XORing his randomized output wire for the circuit GC_r with the corresponding additional wires.

4 Performance Analysis

We analyze the overall communication and computational complexity of our protocol, and compare with Pinkas' protocol by ignoring the constructions that are used in both protocols. We assume that Pinkas' protocol also uses the equality-checker for consistency of Bob's input. We also assume that Pinkas' protocol uses committing OT to fix the protocol issue [5]. Note that by the modification presented in Figure 1 we need ℓ additional XOR gate for each output wire of Bob which has only negligible affect to the overall complexity.

As we said before, the problem of Pinkas' protocol with majority circuit computation can be fixed by running any two-party protocol considering malicious adversaries. For example, if the protocol in [6] is used then the communication complexity of majority circuit computation is $O(\ell m^2 \log m)$. We note that there is no need to use such a protocol in our case.

We next consider the parts related to fairness. Note that the way we generate Bob's commitments to Alice's outputs is the same as in Pinkas' protocol (namely, there are $O(\ell m)$ commitments to permutations $\sigma_{i,j}$'s). However, for Alice's commitments to Bob's outputs is much different: Pinkas' protocol has $O(\ell m \kappa)$ commitments which are generated by Alice in order to be blindly signed by Bob, where κ is the security parameter (which are actually timed commitments for the gradual opening). In our protocol, there are $O(\ell m)$ homomorphic commitments, and ℓ OR-proofs (namely, one proof for each wire). These homomorphic commitments will also be used in the protocol of [4] in order to verify the correctness before the gradual opening. Note that one can also use the protocol presented by Schoenmakers and Tuyls [12] which is a weaker version of [4] but relatively more efficient.

The major difference between our approach and the construction by Pinkas [11] is in the removal of the blind signatures and of the majority circuit computation. This leads to an improvement by a factor of κ for the computational complexity. The reason is that for every output wire of Bob 2κ blind signatures are needed in Pinkas' protocol while in ours only one proof of knowledge is needed together with a simple modification to the circuit.

5 Security Analysis

In our security analysis we want to take advantage of the frameworks established by [6] and [4] for the real/ideal simulation paradigm, resp., for the malicious case in secure two-party computation (and Yao's protocol in particular) and for the case of fair protocols. To do so we will actually focus on analyzing a variant of our protocol, in which Phase 10 is replaced by Phase 10$'$:
Phase 10$'$ **[Trivial opening.]** Alice opens the commitments $E_{i,r}$ for $i \in O_B$ and Bob opens the commitments $C_{i,j}$ for $i \in O_A$ and j such that $\ell_j = 0$.

This adapted protocol is not fair, but it withstand malicious adversaries. We will argue so by showing how to simulate it, following [6]. From this we conclude that the commitments upon entering Phase 10 in the protocol are correct, as a consequence of which the framework of [4] applies and the simulatability of our protocol follows.

Before we give a simulation we present the following two additional modifications over the circuit RC_f, in order to have an efficient simulation.
Modification 1. We first modify Bob's input wires of the circuit RC_f in the following way: for each input wire of Bob (say W_B), we add an AND gate and an OR gate as shown in Figure 2 in such a way that the AND gate has one new input wire for Alice (say W_A) and the original input wire from Bob (W_B). This composition of gates always reproduces the value of the wire W_B independently of the value of W_A ($(W_A \wedge W_B) \vee W_B = W_B$).

This modification is applied so that the simulator is able to learn the input of the corrupted Bob. Roughly speaking, if the simulator knows the garbled circuit, two garbled values for each input wire of Alice (together with their corresponding bit value) and a garbled value for each input wire of Bob (where he does not know the corresponding bit value) then it is possible to compute the bit value of the garbled value for each input wire of Bob.

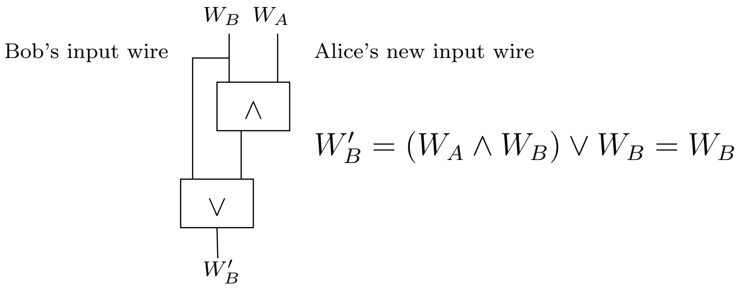
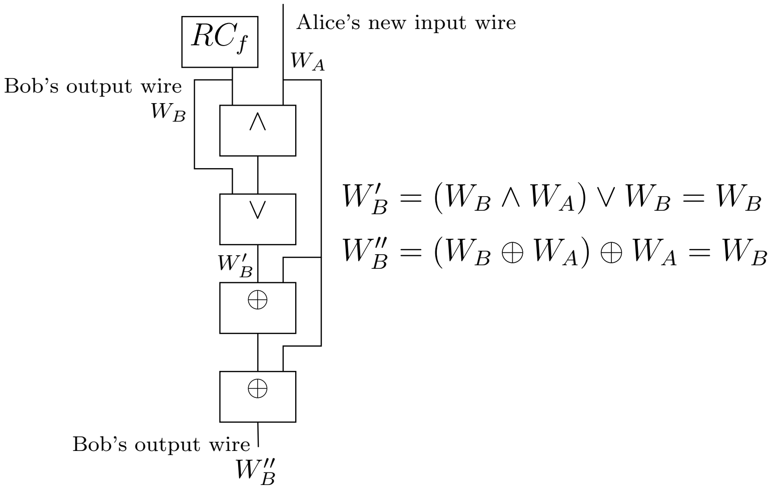


Fig. 2. Additional gates for each of Bob's input wires

We now proceed with the details. Let $w_{A,i,0}$ and $w_{A,i,1}$ denote Alice's garbled input values (for 0 and 1) for $i \in I_A$, and $w_{B,i,b}$ Bob's garbled input value for $i \in I_B \cup I'_B$ and for some $b \in \{0,1\}$. If the garbled values $w_{A,i,0}, w_{A,i,1}, w_{B,i,b}$ and the garbled circuit are given then by evaluating the garbled AND gate with the garbled inputs $(w_{A,i,0}, w_{B,i,b})$ and $(w_{A,i,1}, w_{B,i,b})$ it is possible to decide the bit value of $w_{B,i,b}$ (i.e., learn the bit value b). Namely, if after these two evaluations the same garbled value is obtained, it means that Bob's garbled input corresponds to the bit 0; otherwise, Bob's input bit is 1. We note that this deduction process does not work for an arbitrary Boolean gate, and this is the reason why we modified the circuit in such a way that the input gates are AND gates. For example, evaluation of an XOR gate (has Alice's input wire and Bob's input wire) using $(w_{A,i,0}, w_{B,i,b})$ and $(w_{A,i,1}, w_{B,i,b})$ would always result in two different garbled values from which one cannot conclude the input bit of the corrupted Bob.
Modification 2. We next modify Bob's output wires of the circuit RC_f in the

following way: for each output wire of Bob, we add the construction presented in Modification 1 and add two XOR gates as shown in Figure 3 in such a way that new input wire for Alice is added. This composition of gates always reproduces the original output bit of Bob in the garbled circuit independently of the value of Alice's additional input (i.e., the bit value of the wires W_B, W'_B and W''_B in Figure 3 is the same regardless of Alice's input.). The simulator has to learn the garbled output values of the corrupted Bob together with their corresponding bit (we describe this during the simulation), and by this modification the simulator would be able to learn them.



$$W'_B = (W_B \wedge W_A) \vee W_B = W_B$$
$$W''_B = (W_B \oplus W_A) \oplus W_A = W_B$$

Fig. 3. Additional gates for each of Bob's output wires

If the garbled values $w_{A,i,0}, w_{A,i,1}, w_{B,i,b}$ and the garbled circuit are given then by evaluating the modification in Figure 3 it is possible to compute the garbled values for each output wire of Bob together with their corresponding bit. More precisely, as described above, by evaluating the garbled AND gate with the garbled inputs $(w_{A,i,0}, w_{B,i,b})$ and $(w_{A,i,1}, w_{B,i,b})$ it is possible to compute the output bit value of Bob's garbled output value $w_{B,i,b}$ for the wire W_B in Figure 3 (i.e., learn the bit value b). We here note that the bit value of W'_B is the same as the bit value of W_B, so the bit value of W'_B is also known.

We next show that it is possible to compute both garbled output values (0 and 1) for the output wires of XOR gates from the second construction (for the wires W'_B and W''_B in Figure 3), together with their corresponding bit. Let $\hat{w}$ be the evaluated garbled output value of the OR gate for wire W'_B. By evaluating the garbled XOR gate with the garbled inputs $(w_{A,i,0}, \hat{w})$ and $(w_{A,i,1}, \hat{w})$ where the bit value of $\hat{w}$ is known one can learn both garbled values for the output wires of XOR gates (W'_B and W''_B), and their corresponding bit. Namely, these two evaluations always result in two different garbled values from which it is easy to learn the corresponding bit.

We stress that our protocol is applied to this final modified circuit together with the above modifications, and in this way we show that we have an efficient simulator.

We are now ready to simulate the protocol (the one with the trivial opening) assuming that either Bob or Alice is corrupted.

Case 1- Assume Bob is corrupted

Let R_B be an adversary corrupting Bob; we construct a simulator S_B as follows. Since we assume that the committing OT protocol is secure, we analyze the security of the protocol in the hybrid model with a trusted party computing the committing OT functionality.

The simulator.

1. The simulator S_B chooses a fixed input $x' = 0$ for Alice and uses it only in the beginning of the protocol (namely, to run the OT phase) but it is not used later on.

2. S_B invokes R_B and obtains the garbled input values $w_{i,j,0}$ and $w_{i,j,1}$ for $i \in I_A$ and $j \in \{1, \ldots, m\}$ which are R_B's inputs from the committing OT protocol (in the hybrid model).

3. S_B receives all of the garbled circuits and the commitments from R_B.

4. S_B then runs the challenge phase to generate the random challenge values.

5. Now the input of R_B will be extracted as follows. The simulator S_B receives all of the required decommitments from R_B based on the challenge values, including the garbled values that correspond to Bob's input. Let $w_{i,j}$ be Bob's garbled input value for $i \in I_B \cup I'_B$ and j such that $\ell_j = 0$. S_B verifies that all the commitments are correct as Alice would do in Phase 5. If any of the checks fail, S_B sends an abort message to R_B, sends $\perp$ to the trusted party and halts, outputting whatever R_B outputs. If none of the checks fail, S_B obtains $m/2$ input for Bob for $m/2$ circuits because of Modification 1. More precisely, the simulator knows $w_{i,j,0}$, $w_{i,j,1}$ for $i \in I_A$ and $w_{i,j}$ for $i \in I_B \cup I'_B$ for j such that $\ell_j = 0$, and by Modification 1 the simulator can learn the input bit of Bob for each $w_{i,j}$ for $i \in I_B \cup I'_B$ for j such that $\ell_j = 0$. (In the real case, this does not happen since Alice learns only one garbled input value from OT for each her input wire.) If no input value appears more than $m/4$ times, then S_B outputs fail [**]. We show below that fail also does not occur with high probability. Otherwise, S_B sets y to be the value that appears more than $m/4$ times and sends it to the trusted party. Trusted party replies with $f_2(x, y)$ to S_B.

6. Now the simulator knows $f_2(x, y)$ but it has to convert this value into the corresponding garbled values. The simulator S_B first computes the evaluation-circuits as in the real protocol and obtains one garbled output value per wire. The complementary values will appear as well which are in general not the correct ones since the simulator computes the garbled circuit in the case that $x' = 0$. However, Modification 2 has been applied in order to learn both garbled output values of Bob, and the corresponding bits. As we described above, the simulator learns the output bit of $w_{i,j}$ for $i \in I_B \cup I'_B$ and j such that $\ell_j = 0$ from the AND

[**] The majority of inputs are computed in order to have a correct output by the cut-and-choose technique.

gate in Figure 3 (for the wire W_B). This bit value is the same as the bit value for the wire W'_B in Figure 3. Then, by decrypting the XOR gates the simulator learns both garbled values, and their corresponding bits. In the real case, this does not happen since Alice learns only one garbled input value from OT for each her input wire. Hence, since the simulator knows the private output of the corrupted party and corresponding garbled output values it then computes the commitments $\text{commit}_A(w_{i,j};\zeta_{i,j})$ for $i \in O_B$ and j such that $\ell_j = 0$ as Alice does in the real protocol in Phase 6 and sends to R_B.
7. The commitments $\text{commit}_B(w_{i,j,0}, w_{i,j,1}; \delta_{i,j})$ for $i \in O_B$ and j such that $\ell_j = 0$ are opened by R_B for the evaluation-circuits as in the real protocol.
8. The simulator then determines the majority circuit since it knows the garbled output values and the corresponding bits as in the real protocol.
9. Since the simulator knows the values in the commitments $E_{i,r}$ it can produce the proof.
10′. Alice opens the commitments $E_{i,r}$ for $i \in O_B$ and Bob opens the commitments $C_{i,j}$ for $i \in O_A$ and j such that $\ell_j = 0$.

Analysis. We claim that the view of R_B in the simulation with S_B is statistically close to its view in a hybrid execution of the protocol with a trusted party computing the committing OT protocol. (Note that our protocol is not statistically secure since the simulation is in the hybrid model for committing OT functionality, and it depends on the implementation of OT subprotocol, the commitment schemes and the OR-proofs used).

First of all, we show that if Alice aborts the protocol depending a cheating behavior by Bob, then Bob does not get any information about Alices input. This is only possible either at Phase 5 (at the opening & checking phase) or at Phase 7 (at the opening of $D_{i,j}$'s) while checking the correctness of the circuits and the commitments. In this case, the decision to abort is based on Bob's construction of the circuits as well as commitments (including commitments from the OT phase), and on the random inputs of the parties, and is independent of Alice's input. Thus, Bob does not get any information if Alice aborts the protocol. Thus, we know that the difference between Alice receives "abort" in an ideal execution with S_B and that Alice outputs "abort" in a real execution with R_B is negligible. From here on, we therefore consider the case that Alice does not abort the protocol.

We now prove that the circuits and the commitments are correct with overwhelming probability. First of all, we note that the additional modifications does not compromise the security of the garbled circuit since, by definition of garbled circuit, having one garbled value for each input wire for a gate results in always one garbled output value, which ensures privacy. If Alice does not abort then with probability $2^{-m/4}$ at most $m/4$ of the circuits are bad (including the commitments). Also, we know that the equality-checker scheme [3] assures with high probability a majority of the evaluation-circuits obtain the same input and OT assures with high probability that the values received from OT are correct garbled values, and therefore fail does not occur with negligible probability. The simulator S_B can then decide on a majority circuit, prove that the commitments are committed to the garbled values of output wires of R_B and open the commitments for only this circuit.

We next show that if S_B does not output any fail message, the simulated view of R_B is identically distributed to its view in an execution of the protocol. Actually, they are identical since S_B just runs the honest Alice's instructions when interacting with corrupted Bob. Since S_B uses independent random coins in the challenge phase and follows Alice's instructions each time, the above process results in a distribution that is identical to the view of R_B in a real execution with Alice. As we mentioned before the protocol is not statistically secure since the simulation is considered in the hybrid model for committing OT functionality, and it depends on the implementation of OT subprotocol, the commitment schemes and the OR-proofs.

Case 2- Assume Alice is corrupted

The security analysis when Alice is corrupted is very similar to the proof of [6]. During the protocol Alice sees the circuits and the commitments and they run a secure committing OT where she gets only the garbled values corresponding to her input bits. On a high level, in the simulation, the simulator first extracts the input of Alice from OT functionality in the hybrid model and then sends the input x to the trusted party and learn the output value. Given the output, the simulator constructs the garbled circuits. The simulator constructs the garbled circuits where some of them correctly computes $f(x, y)$ and some of them compute a constant function which always outputs Alice's real output. Namely, the output of this garbled circuit is always equal to the value which is received from the trusted party. The simulator then chooses the challenge value in such a way that all the check-circuits correctly compute the function $f(x, y)$ while all the other circuits (representation of constant function) are going to be evaluation-circuits which compute the constant function. We refer to [6] for details.

Remark. As we said before, in one part of the proof of [6], the failure probability of the simulator is bounded above by $2^{1-m/17}$. The reason is that the rewinding process is used in the case that Bob is corrupted. We note that Modification 1 is sufficient to have a better bound of [6]. Modification 2 is not necessary for [6] since the way our protocol permits two private outputs is different from theirs. In our case, once Alice evaluates the circuits we know that she can compute only one garbled output value. And, Bob accepts it as output if and only if the value is the same as the circuit garbled value. In our security analysis, Modification 2 lets the simulator learn the corresponding garbled value and in this way, we avoid running the rewinding procedure in [6], which results in more efficient simulation.

Also, note that sending garbled output values or actual bit values to Bob in [6] does not compromise the security of the protocol, however, in our protocol Alice has to send garbled output values but not the actual values (bits). Therefore, we highlight that in our protocol the correctness of outputs comes from checking whether the received values are garbled values of the circuit.

Acknowledgements. We would like to thank Peter van Liesdonk and José Villegas for their comments on the presentation.

References

1. D. Chaum and T. P. Pedersen. Wallet databases with observers. In *Advances in Cryptology–Crypto 92*, volume 740 of *Lecture Notes in Computer Science*, pages 89–105, Berlin, 1993. Springer-Verlag.
2. R. Cramer, I. Damgård, and B. Schoenmakers. Proofs of partial knowledge and simplified design of witness hiding protocols. In *Advances in Cryptology–Crypto 1994*, volume 839 of *Lecture Notes in Computer Science*, pages 174–187. Springer-Verlag, 1994.
3. M. Franklin and P. Mohassel. Efficiency tradeoffs for malicious two-party computation. In *Public Key Cryptography–PKC 2006*, volume 3958 of *Lecture Notes in Computer Science*, pages 458–473, 2006.
4. J. A. Garay, P. D. MacKenzie, M. Prabhakaran, and K. Yang. Resource fairness and composability of cryptographic protocols. In *Theory of Cryptography Conference – TCC 2006*, volume 3876 of *Lecture Notes in Computer Science*, pages 404–428, Berlin, 2006. Springer-Verlag. `http://eprint.iacr.org/2005/370`.
5. M. S. Kiraz and B. Schoenmakers. A protocol issue for the malicious case of Yao's garbled circuit construction. In *The 27th Symposium on Information Theory in the Benelux*, pages 283–290, 2006.
6. Y. Lindell and B. Pinkas. An efficient protocol for secure two-party computation in the presence of malicious adversaries. In *Advances in Cryptology–Eurocrypt 2007*, volume 4515 of *Lecture Notes in Computer Science*, pages 52–78. Springer-Verlag, 2007.
7. D. Malkhi, N. Nisan, B. Pinkas, and Y. Sella. Fairplay – a secure two-party computation system. In *USENIX Security*, pages 287–302, 2004.
8. M. Naor. Bit commitment using pseudorandomness. In *Journal of Cryptology*, volume 4, pages 151–158, 1991.
9. T. Pedersen. A threshold cryptosystem without trusted party. In *Advances in Cryptology–Eurocrypt 1991*, volume 547 of *Lecture Notes in Computer Science*, pages 522–526. Springer-Verlag, 1991.
10. B. Pinkas. Personal communication, 2005.
11. B. Pinkas. Fair secure two-party computation. In *Advances in Cryptology–Eurocrypt 2003*, volume 2656 of *Lecture Notes in Computer Science*, pages 87–105. Springer-Verlag, 2003.
12. B. Schoenmakers and P. Tuyls. Practical two-party computation based on the conditional gate. In *Advances in Cryptology–Asiacrypt 2004*, volume 3329 of *Lecture Notes in Computer Science*, pages 119–136. Springer-Verlag, 2004.
13. D. P. Woodruff. Revisiting the efficiency of malicious two-party computation. In *Advances in Cryptology–Eurocrypt 2006*, volume 4515, pages 79–96. Springer-Verlag, 2007. `http://eprint.iacr.org/2006/397`.
14. A. Yao. How to generate and exchange secrets. In *27th IEEE Symposium on Foundations of Computer Science*, pages 162–168, 1986.

A Protocol Diagram

Common Input: f

Compute: $f(x,y) = (f_1(x,y), f_2(x,y))$

Alice | **Bob**

Private Input: $x = \langle x_i \in \{0,1\}, i \in I_A \rangle$ | **Private Input:** $y = \langle y_i \in \{0,1\}, i \in I_B \rangle$

Phase 1: Generation of garbled input values.

Generate $w_{i,j,b} \in_R \{0,1\}^k, i \in I_A, j = 1, \ldots, m, b \in \{0,1\}$.

Phase 2: Committing OT Run in parallel, for $i \in I_A$.

Receiver | **Sender**

Private Input: x_i | **Private Input:** $\langle w_{i,j,0}, w_{i,j,1} : j = 1, \ldots, m \rangle$

Committing OT subprotocol

$\longleftrightarrow$

Private Output: $\langle w_{i,j,x_i} : j = 1, \ldots, m \rangle$ | **Private Output:** $\langle \alpha_{i,j,0}, \alpha_{i,j,1} : j = 1, \ldots, m \rangle$

Common Output:

$A_{i,j,b} = \langle \ \text{commit}_B(w_{i,j,b}; \alpha_{i,j,b}) \ \rangle$ for $j = 1, \ldots, m, b \in \{0,1\}$

Phase 3: Construction.

Compute GC_j for $j = 1, \ldots, m$ s.t. for all $i \in I_A$ $\langle w_{i,j,0}, w_{i,j,1} : j = 1, \ldots, m \rangle$ are used for the corresponding wires in GC_j.

Compute $B_{i,j,j',b}$=commit$_B(w_{i,j,b}, w_{i,j',b}; \beta_{i,j,j',b})$ for $i \in I_B \cup I'_B$ and $1 \leq j < j' \leq m, b \in \{0,1\}$.

Compute $C_{i,j}$=commit$_B(\sigma_{i,j}; \gamma_{i,j})$ for $i \in O_A$ and $j = 1, \ldots, m$ where $\sigma_{i,j} \in_R \{0,1\}$.

Compute $D_{i,j}$=commit$_B(w_{i,j,0}, w_{i,j,1}; \delta_{i,j})$ for $i \in O_B$ and $j = 1, \ldots, m$.

$\langle GC_j : j = 1, \ldots, m \rangle, \langle C_{i,j} : i \in O_A, j = 1, \ldots, m \rangle, \langle D_{i,j} : i \in O_B, j = 1, \ldots, m \rangle,$
$\langle B_{i,j,j',b_{i,j,j'}}, B_{i,j,j',1-b_{i,j,j'}} : i \in I_B \cup I'_B, 1 \leq j < j' \leq m, b_{i,j,j'} \in_R \{0,1\} \rangle$

$\longleftarrow$

Phase 4: Challenge.

Secure coin-flipping subprotocol

$\longleftrightarrow$

Common Output: $\ell = \ell_1 || \ell_2 || \ldots || \ell_m$ s.t. $\ell_i \in_R \{0,1\}$

Phase 5- Opening & Checking.

$\widetilde{GC}_j = \langle w_{i,j,b} : i \in \mathcal{W}, b \in \{0,1\}\rangle, \ell_j = 1.$

$\widetilde{A}^{OT}_{j,b} = \langle \alpha_{i,j,b} : i \in I_A\rangle, \ell_j = 1, b \in \{0,1\}.$

$\widetilde{B}_{j,j',b} = \langle \beta_{i,j,j',b} : i \in I_B \cup I'_B\rangle,\ \ell_j = \ell_{j'} = 1, 1 \le j < j' \le m,$ $b \in \{0,1\}.$

$\widetilde{C}_j = \langle \sigma_{i,j}, \gamma_{i,j} : i \in O_A\rangle,\ \ell_j = 1.$

$\widetilde{D}_j = \langle \delta_{i,j} : i \in O_B\rangle$ for $\ell_j = 1.$

$\widetilde{B}^{input}_{j,j'} = \langle w_{i,j,y_i}, w_{i,j',y_i}, \beta_{i,j,j',y_i}, w_{i',j,z_{i'}}, w_{i',j',z_{i'}},$ $\beta_{i',j,j',z_{i'}} : i \in I_B, i' \in I'_B\rangle$ for $\ell_j = \ell_{j'} = 0, 1 \le j < j' \le m.$

$$\xleftarrow[]{\langle \widetilde{GC}_j : \ell_j = 1, b \in \{0,1\}\rangle, \langle \widetilde{B}_{j,j',b} : \ell_j = \ell_{j'} = 1, 1 \le j < j' \le m, b \in \{0,1\}\rangle, \langle \widetilde{C}_j : \ell_j = 1\rangle, \langle \widetilde{D}_j : \ell_j = 1\rangle, \langle \widetilde{B}^{input}_{j,j'} : \ell_j = \ell_{j'} = 0, 1 \le j < j' \le m\rangle}$$

Check GC_j for $\ell_j = 1$ using $\widetilde{GC}_j$.

$A^{OT}_{i,j,b} \overset{?}{=} \text{commit}_B(w_{i,j,b}; \alpha_{i,j,b})$ for $i \in I_A,\ \ell_j = 1, b \in \{0,1\}.$

$B_{i,j,j',b} \overset{?}{=} \text{commit}_B(w_{i,j,b}, w_{i,j',b}; \beta_{i,j,j',b})$ for $i \in I_B \cup I'_B,\ \ell_j = \ell_{j'} = 1, 1 \le j < j' \le m, b \in \{0,1\}.$

$C_{i,j} \overset{?}{=} \text{commit}_B(\sigma_{i,j}; \gamma_{i,j})$ for $i \in O_A,\ \ell_j = 1.$

$D_{i,j} \overset{?}{=} \text{commit}_B(w_{i,j,0}, w_{i,j,1}; \delta_{i,j})$ for $i \in O_B,\ \ell_j = 1.$

$B_{i,j,j',y_i} \overset{?}{=} \text{commit}_B(w_{i,j,y_i}, w_{i,j',y_i}; \beta_{i,j,j',y_i})$ for $i \in I_B, \ell_j = \ell_{j'} = 0, 1 \le j < j' \le m.$

$B_{i',j,j',z_{i'}} \overset{?}{=} \text{commit}_B(w_{i',j,z_{i'}}, w_{i',j',z_{i'}}; \beta_{i',j,j',z_{i'}})$ for $i' \in I'_B, \ell_j = \ell_{j'} = 0, 1 \le j < j' \le m.$

Phase 6- Evaluation.

Evaluate GC_j for $\ell_j = 0$, using $\widetilde{B}^{input}_{j,j'}$.

Compute $E_{i,j} = \text{commit}^h_P(w_{i,j}; \zeta_{i,j})$ for $i \in O_B$ and $\ell_j = 0.$

$$\xrightarrow{\langle E_{i,j} : i \in O_B, \ell_j = 0\rangle}$$

Phase 7: Opening of $D_{i,j}$.

$\widetilde{D}_j = \langle w_{i,j,0}, w_{i,j,1}, \delta_{i,j} : i \in O_B\rangle$ for $\ell_j = 0.$

$$\xleftarrow{\langle \widetilde{D}_j : \ell_j = 0\rangle}$$

$D_{i,j} \overset{?}{=} \text{commit}_B(w_{i,j,0}, w_{i,j,1}; \delta_{i,j})$ for $i \in O_B,\ \ell_j = 0$

Phase 8: Decision of majority circuit.

Determine a majority circuit GC_r for some r s.t. $\ell_r = 0$ where only majority of Bob's output wires are counted.

Phase 9: Verification of Alice's commitments. Run in parallel, for $i \in O_B$, r s.t. $\ell_r = 0$.

Prover		**Verifier**
Private Input: $w_{i,r}$	**Common Input:** $E_{i,r}, w_{i,r,0}, w_{i,r,1}, \epsilon_{i,r}$ OR-Proofs subprotocol $\longleftrightarrow$	**Private Input:** $\perp$
Private Output: $\perp$	**Common Output:** Proof of validity	**Private Output:** $\perp$

Phase 10: Gradual Release. Run in parallel, for $i \in O_A, i' \in O_B, \ell_j = 0$.

Private Input: $w_{i',r}, \zeta_{i',r}$	**Common Input:** $C_{i,j}, E_{i',r}$, Gradual Release subprotocol $\longleftrightarrow$	**Private Input:** $\sigma_{i,j}, \delta_{i,j}$
Private Output: $\sigma_{i,j}$		**Private Output:** $w_{i',r}$

Apply $\sigma_{i,j}$ to $POP_{i,j}$ for $i \in O_A$ and match $w_{i,r}$ with $(w_{i,j,0}, w_{i,j,1})$ for $i \in O_A$ and determine a majority circuit for Alice.

Private Output: $f_1(x,y)$

Match $w_{i',r}$ with $(w_{i',r,0}, w_{i',r,1})$ for $i' \in O_B$ to find the randomized output bits and compute XOR with the corresponding additional input wires.

Private Output: $f_2(x,y)$