Fast Optimistically Fair Cut-and-Choose 2PC

Alptekin Küpçü
Koç University, TURKEY
akupcu@ku.edu.tr

Payman Mohassel
Yahoo Labs, USA
pmohassel@yahoo-inc.com

February 25, 2016

Abstract

Secure two party computation (2PC) is a well-studied problem with many real world applications. Due to Cleve's result on general impossibility of fairness, however, the state-of-the-art solutions only provide security with abort. We investigate fairness for 2PC in presence of a trusted Arbiter, in an optimistic setting where the Arbiter is not involved if the parties act fairly. Existing fair solutions in this setting are by far less efficient than the fastest unfair 2PC.

We close this efficiency gap by designing protocols for fair 2PC with covert and malicious security that have competitive performance with the state-of-the-art unfair constructions. In particular, our protocols only requires the exchange of a few extra messages with sizes that only depend on the output length; the Arbiter's load is independent of the computation size; and a malicious Arbiter can only break fairness, but not covert/malicious security even if he colludes with a party. Finally, our solutions are designed to work with the state-of-the-art optimizations applicable to garbled circuits and cut-and-choose 2PC such as free-XOR, half-gates, and the cheating-recovery paradigm.

Keywords: secure two-party computation, covert adversaries, cut-and-choose, garbled circuits, fair secure computation, optimistic fair exchange.

1 Introduction

In electronic commerce, privacy and fairness are two sought-after properties as depicted in work related to contract signing and fair exchange [5, 14, 53, 9, 6, 8, 21]. Fair exchange is used in electronic payments to buy or barter items [12, 46] and contract signing is often used to ensure fairness: either all parties sign and agree on the contract, or the contract is invalid.

Fair secure two-party computation (2PC), a fundamental problem in cryptography, can be used to address both the privacy and fairness concerns, simultaneously. Alice and Bob would like to jointly compute a function of their private inputs, such that nothing other than the output leaks, and either both parties learn the output or either do (*e.g.* two banks trying to calculate a joint credit score for a customer, without giving away critical private information). Unfortunately, however, there is a significant efficiency gap between secure 2PC that achieve fairness and their unfair counterparts that have been the subject of many recent implementations and optimizations.

2PC Without Fairness: Yao [68] introduced the first 2PC with security against honest-but-curious adversaries [49] and a large body of recent work has focused on making 2PC practical in presence of stronger (covert and malicious) adversaries [54, 50, 7, 58, 48, 24].

The *cut-and-choose* paradigm is a popular method for enhancing the security of Yao's garbled circuit protocol to the case of *malicious* (or covert) adversaries where the players can deviate arbitrarily. In a nutshell, in this paradigm, one player (Alice) garbles many circuits while the other player (Bob) checks a randomly chosen subset (to ensure that the garbling was done correctly) and evaluates the rest. Until recently, many existing solutions (*e.g.* [55, 50, 51, 65, 56, 66]) required garbling at least $3s$ circuits to detect cheating with probability $1-2^{-s}$. The high number of garbled circuits is due to the fact that all these constructions ask that the evaluator computes a "majority output" and, for it to be valid, require that more than half of the evaluated circuits are correct. For the majority output to be valid, parties also need to enforce equality of the garbler's input to the majority of the circuits evaluated. This is often handled via a procedure called **input-consistency** check.

The recent work of Lindell [48] shows how to reduce the number of garbled circuits by a factor of 3.[1] In this approach, the second player evaluates the unchecked circuits, but is content with computing only one correct output (instead of a majority output) due to a **cheating-detection** component. This allows one to reduce the number of circuits to s and still achieve $1-2^{-s}$ security.

A more modest security guarantee for 2PC is **covert security**, proposed by Aumann and Lindell [7], which provides a practical alternative to the malicious setting. In this setting, the adversary can cheat with some small but non-negligible probability. The rationale is that a reputable real-world entity will not risk getting caught with non-negligible probability due to loss of reputation or the legal/economical costs. The protocols in the covert setting are more efficient than their malicious counterparts. For instance, in the cut-and-choose paradigm, one can settle for only garbling $s = 5$ circuits if $1 - 1/s = 4/5$ probability of getting caught is prohibitive enough. Back to our two banks computing a customer's credit score scenario, the financial losses when a bank gets caught cheating can be seen as prohibitive as a negligible probability of cheating.

2PC with Fairness: All the above-mentioned work focus on security with abort, where the malicious party is allowed to abort the protocol *after* he learns the output of the computation, but *before* the honest party obtains the output, because it is known that achieving general fairness is impossible [20]. This limits the real world applicability of the most efficient solutions. An interested corporation is less likely to adopt 2PC solutions if it has to risk being at a competitive disadvantage by revealing the outcome of the computation to a competitor without learning it itself.

There are two main approaches for achieving fairness in general-purpose 2PC.[2] (i) **Gradual release**-based approaches let Alice and Bob reveal each other's output piece by piece, using super-constant rounds [60, 36, 64, 63]. (ii) **Arbiter**-based approaches achieve constant round complexity by assuming that a trusted third party is available when needed [18, 47, 35, 34]. **Optimistic** approaches employ the Arbiter only if there is a dispute among the parties [5].

The most relevant work to ours is that of Cachin and Camenisch [18], and the follow up work of [35], in the same optimistic Arbiter-based setting. Both constructions utilize zero-knowledge proofs that require public-key operations, and hence have a high computational cost compared to the state-of-the-art cut-and-choose 2PC. Furthermore, in [18], the Arbiter may need to redo almost the whole computation in case of a malicious behavior, which creates a bottleneck in the system.

Lindell's optimistic framework [47], on the other hand, necessitates an electronic payment system. It is possible that one party obtains the output of the computation, whereas the other obtains a payment. [1, 2, 15, 33, 40, 46] also employ such penalty-based fairness models. These

[1] An alternative approach for reducing the number of circuits by a factor of 1.5 was introduced by [30].

[2] A different line of work focuses on achieving fairness not in general but for specific applications [27, 3, 16, 22, 17].

constructions are incomparable to ours as they work in a different setting and make different assumptions. See Table 1 for a list of the main differences between these work and ours.

[18]	[47]	[35]
Resolutions with Arbiter take time proportional to the circuit size	Requires a payment system and employs penalty-based fairness.	Efficiently adds fairness, but to zero-knowledge based 2PC protocols only.

Table 1: Comparison to the most related previous work.

Our Contribution: In this paper we investigate fairness for 2PC in presence of a trusted Arbiter in an optimistic setting, where the Arbiter is not involved if the parties act fairly. We design efficient protocols for fair 2PC with security against covert and malicious adversaries. Our constructions follow the cut-and-choose paradigm, and for the first time, close the efficiency gap between fair 2PC and the state-of-the-art unfair solutions, in this setting. In particular:

- ✓ The overhead of our protocols against state-of-the-art unfair solutions is small; only a constant number of extra rounds and a few messages with sizes that *only* depend on the output length.
- ✓ The Arbiter's load is minimal, and *independent* of the size of computation.
- ✓ A malicious Arbiter can *only* break fairness, but not covert/malicious security even if he colludes with a party. We prove this via a simulator for the usual security with abort definition, when the adversary is also controlling the Arbiter.
- ✓ Our protocols are *compatible* with optimizations applicable to cut-and-choose 2PC such as free-XOR [39], FleXor [38], and half-gates [69]. It also utilizes the cheating-recovery paradigm, and hence uses a reduced number of garbled circuits. These render our protocols *the most efficient* fair secure computation protocols to date.
- ✓ Our work is the first to consider fairness in the covert adversary model.

2 Overview of Our Constructions

We review the high level ideas behind our covert and malicious 2PC constructions next, emphasizing the non-trivial parts. Our starting point in each case is the state-of-the-art protocol with security with abort (in the cut-and-choose paradigm). We then show how to enhance and modify each at very low cost in order to obtain fairness in the presence of an Arbiter.

Some of our techniques are similar to that of Kılınç and Küpçü [35] who also provide an efficient solution for fair 2PC in the same setting. Similar to ours, their solution employs commitments to output labels, and verifiable escrows. But they instantiate these using zero-knowledge proofs of knowledge. In fact, verifiable escrow inherently employs zero-knowledge proofs. When one switches to the cut-and-choose setting, it is unclear how to deal with the multitude of such commitments and verifiable escrows, and still preserve correctness and efficiency. Our solutions are the first to combine optimistic Arbiter-based fairness and the cut-and-choose paradigm efficiently.

2.1 Fair Covert 2PC

There are various ways of combining fairness and covert security in a simulation-based definition. In this paper we consider the natural notion where both fairness and correctness/privacy are guaranteed with a reasonable (not all-but-negligible) probability $1 - \epsilon$ but both fairness and correctness/privacy are lost with probability ϵ against active cheating. A related notion to fairness in the covert setting is $1/p$ security [26, 31]. In that line of work [28, 57, 11, 10], the ideal world provides complete fairness (as in our case for malicious adversaries), but the simulation only needs to achieve $1/p$ indistinguishability between the ideal and real worlds. Our approach is slightly different: we directly take the covert adversary model [7, 4], and modify it to preserve fairness unless the adversary cheats and remains undetected. Note that the $1/p$ security does not explicitly model detection of the adversary's misbehavior. It is an interesting question to understand the relation between the two notions. Next, we review the main technical difficulties in our covert construction.

Security with Abort. Recall the covert 2PC protocol of Aumann and Lindell [7]. Alice generates s garbled circuits $\mathsf{GC}_1, \ldots, \mathsf{GC}_s$. Then, the parties perform ℓ (number of input bits) oblivious transfers (OTs) for Bob to learn his garbled inputs (this is intentionally done for all s circuits and before the opening). Alice sends the s garbled circuits to Bob. Parties then perform a coin-toss to choose a random index $e \in \{1, \ldots, s\}$. Alice opens the secrets for all garbled circuits and OTs except for the e^{th} one. Bob checks correctness of the opened circuits and the corresponding OTs, and aborts if cheating is detected. Else, Alice sends her garbled inputs for the e^{th} circuit. Bob evaluates the circuit and learns his own output. He also obtains the garbled values for Alice's output, which he sends to her for translation.

It is easy to see that the above construction is not fair. We now highlight the main changes we make to this protocol to achieve fairness.

Delay Evaluator's Output Translation. Note that Bob can abort the protocol immediately after learning his output and without forwarding Alice's output to her. Therefore, we modify the protocol so that Alice does *not* send to Bob the translation table for his output (mapping output labels to actual bits) *until* he sends Alice's garbled output to her. But note that this trivial change fails since now Alice can abort before sending the translation table to Bob.

Hence, we need to ensure that if Alice aborts at this stage, Bob has enough information to invoke an output resolution protocol with the Arbiter and show evidence that he has been following the steps of the protocol and hence deserves to know the output. After checking Bob's claim, the Arbiter should be able to provide him with sufficient information to decode his output.

Prove Bob's Honesty to the Arbiter. Notice that efficiently proving this is a non-trivial task. For example, in [18], the Arbiter and the resolving party re-perform almost the whole computation for this purpose. In our case, Bob's proof of following through with the protocol will be the garbled output he computes for Alice's output. Note that due to the *output-authenticity* property of the garbling scheme, Bob cannot forge this value except if he honestly computes the output. In order to enable the Arbiter to check the validity of Bob's claimed output label, Alice will send hashes of her output labels (in permuted order) to Bob along with the garbled circuits, and a signature for the e^{th} one. Bob verifies validity of these hashes for the opened circuits. Now when he goes to the Arbiter, he shows both the output labels he obtained for Alice's output, and the signed hashes for the e^{th} circuit. The Arbiter can verify that the two are consistent, by ensuring that there is one output label provided per pair.

Equip the Arbiter with the Translation Table for Bob's Output. Furthermore, the Arbiter should have sufficient information to pass along to Bob for decoding his output. Hence, Alice encrypts the translation table for Bob's output under the Arbiter's public key and sends it to Bob along with the garbled circuits, and a signature for the e^{th} one. Bob checks validity of these encryptions for the opened circuits. Once Bob's claim of behaving honestly is verified, the Arbiter can decrypt the translation table, and send it to Bob for him to decode his output. The signature is needed to make sure that Bob is sending a legitimate decoding table for decryption. Since Bob verified the opened ones, he is ensured, with good probability, that the e^{th} decoding table is proper.

Simulation-based Proof with Fairness. One important difference between our proof and those of standard 2PC is that in our case the ideal trusted party must only be contacted by the simulator once it is certain that both parties can obtain the output, as first observed by Kılınç and Küpçü [35] for indistinguishability of the ideal and real world outputs. Therefore, to overcome this difficulty, Alice also commits to Bob's output translation tables as c_i^B using a trapdoor commitment, and opens them for the opened circuits. Bob ensures that the committed and encrypted translation tables are the same (in fact, we encrypt the commitment openings). For the e^{th} circuit, she opens c_e^B at the last step of the protocol. The reason we need these commitments is that, unlike standard covert 2PC, the Alice simulator in the proof for the case of corrupted Bob does not have $f_B(x_A, x'_B)$ when sending the garbled circuits (since in the fair protocol neither party may learn the output at this stage), and hence cannot embed the output in the e^{th} one at that stage. With trapdoor commitments, at a later stage, she is able to open the translation to something different in order to ensure the "fake" evaluation circuit evaluates to the correct output $f_B(x_A, x'_B)$. The hiding property of the commitment scheme ensures indistinguishability of the simulator's actions.

Handle Premature Resolutions. The parties have the right to contact the Arbiter. But they may choose to do so at a stage other than the prescribed one. For example, Bob may invoke the output resolution before he sends Alice's output labels to her. This behavior is mitigated by requiring that Bob provides the Arbiter with Alice's output labels that match the signed decoding table. Due to output authenticity of the garbling scheme and unforgeability of the signature scheme, Bob cannot cheat against the Arbiter and must provide correct labels. Later on, Alice can recover her output through her own output resolution protocol. A timeout mechanism ensures that Bob must contact the Arbiter during a predefined time[3], and immediately after that Alice can contact the Arbiter, without waiting indefinitely.

A Note on Synchronicity. Observe that we employ a timeout for resolutions with the Arbiter. Katz et al. [32] define a very nice framework for integrating synchronicity in the Universal Composability [19] framework. They provide a clock functionality which allows all honest parties to proceed further once a particular clock signal is reached, allowing for synchronous protocols. In that setting, they show input completeness and guaranteed termination can be achieved together (though not necessarily fairness). In our protocols, the only place we employ loosely synchronized clocks is for resolutions with the Arbiter. The remaining (optimistic) part of the protocol employs no synchronicity assumptions (just local network timeouts). There are two main reasons we choose to proceed this way: (1) Due to a result by Küpçü and Lysyanskaya [44] (see also [43]), if one would like to employ multiple autonomous (independent) entities to replace a single trusted Arbiter, we are forced to employ timing models. (2) Optimistic fair exchange literature shows that the timeout-based resolutions can be exchanged with slightly more expensive protocols (with one more round) that provide fairness without requiring timeouts (see *e.g.* [5, 46]). We believe a similar methodology may be employed here to replace the timeouts, and leave such an extension to our protocols as future work.

Proof overview. We obtain security against malicious Bob as follows: The simulator acts as Alice, except that she commits to and encrypts random values instead of actual output decoding

[3]Such timeout mechanisms are easy to implement and standard in the optimistic fair exchange literature (see *e.g.* [5, 46]).

table in c_e^B, d_e^B. Towards the end, if the simulator obtains proper output labels for Alice's output from the adversarial Bob, then she contacts the ideal trusted party to learn Bob's output and simulate opening of c_e^B to the actual values. Hiding commitments ensure indistinguishability of Alice's behavior. If, instead of sending them directly to Alice, Bob contacts the Arbiter and performs a proper resolution, the simulator simulates the Arbiter, and upon receiving proper output labels for Alice, contacts the ideal trusted party for obtaining Bob's output. She then sends the corresponding decoding table as if it was the decryption of d_e^B. Semantic security ensures indistinguishability of Alice's and Arbiter's behavior.

For security against covert Alice, different from the unfair scenario, the simulator contacts the ideal trusted party if Alice acts properly, and obtains Alice's output. He sends the corresponding labels back to Alice. He simulates by himself Bob's Arbiter resolution should Alice not respond back with Bob's output labels' openings. If Alice later contacts the Arbiter for resolution, he returns back Alice's output labels again.

2.2 Fair Malicious 2PC

Security with Abort. Our starting point is the cut-and-choose 2PC of Lindell [48], which contains a *cheating-detection* component to remove the requirement that majority of the circuits are correct, and hence reduce the number of circuits by a factor of 3.

In this protocol, Alice garbles s circuits $\mathsf{GC}_1, \ldots, \mathsf{GC}_s$ with the exception that she uses *the same output labels for all circuits*. Parties also perform ℓ OTs for Bob to learn his input labels. Bob then chooses a random subset of these circuits to be evaluated, and the rest are to be opened and checked for correctness later. Bob evaluates the evaluation circuits. Since output labels are reused for all circuits, Bob expects to retrieve the same labels from all evaluations. If this is indeed the case, he only needs to ensure that one of the evaluations was correct in order to make sure he has the correct output. If Bob obtains different labels for at least a single output wire, he uses the two distinct labels W_0 and W_1 corresponding to values 0 and 1, as his proof of Alice's cheating.

At this stage, parties engage in a cheating-detection phase, which itself is a malicious cut-and-choose 2PC for evaluating a cheating-detection (CD) circuit. This cut-and-choose is performed using $3s$ circuits (and a majority output), but since the CD circuit is significantly smaller, this will be a small overhead, independent of the actual circuit's size. The CD circuit takes W_0 and W_1 as Bob's input (his evidence of Alice's cheating), and takes Alice's input x_A to the original computation as her input. If Bob's two labels are valid proofs (Alice embeds the output labels in the CD circuits, and the CD circuit checks whether Bob's two labels are among them), he learns Alice's input x_A and can compute $f(x_A, x_B)$ on his own. Otherwise he learns a random value. It is important that Alice does not know whether Bob learned the output by evaluating the computation circuits or the cheating-detection circuits. Alice then opens the check circuits and Bob aborts if any of the checks fail. Else, he sends Alice's output labels to her.

Handle Alice's Input Consistency. Deviating from [48], we handle the consistency of Alice's inputs using the technique of [66], as it seems more suitable for the tweaks we need to make to input-consistency. In this approach a universal hash function (UH) is evaluated on her input inside the circuits, and Bob verifies that the output of this function is the same in all circuits. Alice's input is padded with a short random string r_x in order to increase its entropy and reduce the amount of information that can be learned about the input from the output of the UH. Let ℓ be the input length and s' be a security parameter. [66] shows that a random matrix of dimensions $s' \times (\ell + 2s' + \log s')$ over $GF(2)$ can be used as a UH, where the evaluation consists of multiplying this matrix with the input vector (and getting a vector of length s').

Delay Bob's Output via a One-time Pad. Similar to the covert 2PC, it is easy to see that the above construction is not fair. In particular, Bob can abort the protocol immediately after learning his output and without forwarding Alice's output to her. But unlike our fair covert 2PC, delaying the transmission of the output translation table is *not* sufficient for preventing Bob from learning his output early. Since the same output labels are used for all circuits and a fraction of them are opened, Bob can reconstruct the translation table on his own after the opening phase, and learn his output.

To overcome this issue, we encrypt Bob's output using a one-time pad pad_B that is Alice's additional input to the computation circuit. In particular, the circuit returns $f_B(x_A, x_B) \oplus pad_B$ as Bob's output, and the pad_B itself is only revealed in the final step of the protocol. Alice's output in the circuit is also encrypted using a separate pad pad_A of her choice, to prevent Bob from learning her output even after the opening.

Commit to the Consistent Pad. Note that simply revealing the pad_B to Bob does not provide Bob with sufficient guarantee that it is the same pad_B Alice used in the computation. Hence, for each circuit Alice sends a trapdoor commitment c_i^B to the translation table PadDec_i for the input wires associated with pad_B. She also encrypts the opening of this commitment as d_i^B using the Arbiter's public key, and signs it for resolution purposes. For the opened circuits, c_i^B and d_i^B are opened and checked. In the final stage, in order to reveal pad_B, Alice opens c_i^B for the evaluated circuits. But, we are not done yet. Bob learns one or more pad values used in the evaluation circuits, and needs to determine which is the correct one for decrypting his output. To facilitate this, we apply a separate UH to pad_B (*i.e.* $M_p \cdot (pad_B \| r_p)$ for a random matrix M_p) in the computation circuits, which Bob uses in the final stage to determine the "correct" pad among those retrieved. Without this, we could not have guaranteed correctness.

Simulate with Fairness. For simulation purposes, similar to the fair covert 2PC, the fact that the simulator can open c_i^B to an arbitrary pad in the final stage allows the simulation to go through by postponing the query to the ideal trusted party for obtaining the output, until we are sure both parties can learn the output. Remember that such a simulation is a necessity for simulating fair secure computation protocols properly [35].

Commit to Alice's Output Early. Similar to the fair covert 2PC, we also need to ensure that if Alice aborts before revealing pad_B, Bob has enough information to invoke an output resolution protocol with the Arbiter and show evidence that he has been following the steps of the protocol. In the covert protocol, we used the output-authenticity property of the garbling scheme for this purpose, but in the current protocol, output-authenticity is lost after the opening stage, since all circuits use the *same output labels*. To circumvent this issue, we have Bob commit to the output labels for Alice's output *before* the opening stage, and have Alice sign the commitment. In case of a resolution, Bob opens the signed commitment for the Arbiter, who checks its correctness and consistency with a signed translation table provided by Alice, and only then decrypts d_i^B escrows for Bob to learn the pads and obtain his output.

Fix Cheating-Detection. Note that in regular cheating-detection, Bob only learns x_A and hence the plaintext version of Alice's output $f_A(x_A, x_B)$. But, Bob needs to commit to the output labels for Alice's output, and because the output translation table of Alice corresponds to a padded output, knowing $f_A(x_A, x_B)$ is not sufficient for simulation. Therefore, we need to modify the CD circuit as well. We fix this by having the CD circuit also output pad_A. Bob can now compute $f_A(x_A, x_B) \oplus pad_A$, and use Alice's output translation table GDec^A to determine which evaluated circuit returned the correct output (we know there is at least one such circuit with all but negligible probability). He commits to those labels as Alice's output labels.

Proof overview. Our proofs are very similar in essence to the above malicious Bob case. Simulator Alice would commit to and encrypt random values, and later when she obtains the actual output from the ideal trusted party, she would simulate opening them to the correct values. For malicious Alice case, simulator Bob also commits to random labels for Alice's outputs, and later simulates opening them to proper labels.

3 Preliminaries

Garbling Schemes. Bellare et al. [13] introduce the notion of a garbling scheme Garble as a cryptographic primitive. We refer the reader to their work for a complete treatment and only give a brief summary here. Besides the standard privacy guarantees, we heavily take advantage of the *output-authenticity* of a garbling scheme, which intuitively guarantees that the evaluator cannot forge valid output labels except by honestly evaluating the garbled circuit.

Committing Oblivious Transfer. In a standard oblivious transfer (OT) protocol [61], the receiver has a selection bit σ, and the sender has two messages a_0, a_1. At the end of the protocol, the receiver learns a_σ while the sender does not learn anything. In a committing oblivious transfer [37, 65], at the end of the interaction, the receiver also receives a commitment to the sender's input messages, and the sender obtains the opening to those commitments. As a result, the receiver can ask the sender to open his messages at a later stage. Efficient constructions for committing OT were proposed in [65] and [56].

3.1 Security Definitions

In secure two-party computation, it is required that the parties do not learn anything beyond what is revealed by the output of the computation. Alice and Bob are trying to compute a function $f : \{0,1\}^* \times \{0,1\}^* \to \{0,1\}^* \times \{0,1\}^*$ on their private inputs x and y, respectively. The function computes $f(x,y) = (f_A, f_B)$, where f_A is the output Alice should obtain, and f_B is Bob's output.
(Unfair) Security against Malicious Adversaries: Security is captured by a simulator that can simulate the messages, as well as the outputs, without knowing the input of the party it simulates. Since it is known that a general two-party protocol cannot be fair [20], most existing work focus on the unfair definition where the malicious party is allowed to abort *after* she learns her output but *before* the honest party learns his output. There is a real world, where an honest user $\mathcal{H}$ interacts with the adversary $\mathcal{C}$. Then, we need a simulator in the corresponding ideal world who can interact with the same adversary $\mathcal{C}$ on behalf of the user, and submits his input to the ideal trusted party $\mathcal{U}$ to obtain the output. If those two worlds are indistinguishable, since the only value $\mathcal{U}$ reveals is the output of the computation, security is satisfied. This concept is formalized below (see *e.g.* [29]).

Ideal World: Players are the adversary $\mathcal{C}$, the honest party $\mathcal{H}$, and the ideal trusted party $\mathcal{U}$.

1. $\mathcal{U}$ receives input x or ABORT from $\mathcal{C}$, and y or ABORT from $\mathcal{H}$. If any ABORT message is received, $\mathcal{U}$ sends ABORT to both of the parties and halts.
2. Otherwise $\mathcal{U}$ computes $f(x,y) = (f_C, f_H)$ and sends f_C to $\mathcal{C}$.
3. $\mathcal{C}$ responds with either CONTINUE or ABORT.
4. If $\mathcal{C}$ says CONTINUE, $\mathcal{U}$ sends f_H to $\mathcal{H}$. Otherwise, $\mathcal{U}$ sends ABORT to $\mathcal{H}$.

Real World: This is the case where the honest user $\mathcal{H}$ interacts with the adversary $\mathcal{C}$ according to the protocol specification.

Definition 3.1 (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 securely if for every PPT real world adversary $\mathcal{C}$ with auxiliary input z, there exists a PPT ideal world simulator $\mathcal{S}$ given the same auxiliary input z so that for every auxiliary info $z \in \{0,1\}^*$ and for every pair of inputs $x, y \in \{0,1\}^*$, the ideal and real world outputs of both the adversary and the honest party are computationally indistinguishable.*

(Unfair) Security against Covert Adversaries: An almost identical definition can be used for the covert adversary setting [7], mainly by changing the ideal world definition. Let $\epsilon \in [0,1]$ be a probability value. The ideal world is now as follows:

Ideal World: Players are the adversary $\mathcal{C}$, the honest party $\mathcal{H}$, and the ideal trusted party $\mathcal{U}$.

1. $\mathcal{U}$ receives input x or ABORT or CORRUPTED or CHEAT from $\mathcal{C}$, and y or ABORT from $\mathcal{H}$. If any ABORT message is received, $\mathcal{U}$ sends ABORT to both of the parties and halts.
2. If $\mathcal{C}$ sent CORRUPTED, then $\mathcal{U}$ sends CORRUPTED to $\mathcal{H}$ and halts. This is to enable the adversary to voluntarily disclose itself.
3. If $\mathcal{C}$ says CHEAT, with ϵ probability, $\mathcal{U}$ sends CORRUPTED to $\mathcal{H}$ and halts (thus the adversary is caught), and with $1-\epsilon$ probability, $\mathcal{U}$ sends UNDETECTED and y to $\mathcal{C}$.
   (a) If $\mathcal{C}$ receives UNDETECTED, responds with some output o. Then $\mathcal{U}$ sends o to $\mathcal{H}$ and halts. This means, **when undetected, the adversary can break both security and correctness**.
4. Otherwise $\mathcal{U}$ computes $f(x,y) = f_C, f_H$ and sends f_C to $\mathcal{C}$.
5. $\mathcal{C}$ responds with either CONTINUE or ABORT.
6. If $\mathcal{C}$ says CONTINUE, $\mathcal{U}$ sends f_H to $\mathcal{H}$, and if $\mathcal{C}$ says ABORT, $\mathcal{U}$ sends ABORT to $\mathcal{H}$.

Fair Security against Malicious Adversaries: Note that in the unfair ideal worlds, the adversary may choose to abort the protocol after receiving her output, and before the honest party learns his output. To prevent this, we consider a *real world* trusted third party (called the **Arbiter**) to be optimistically available. *Optimistic* means that the Arbiter is contacted only if some dispute is to be resolved. Thus, the Arbiter is not involved if everything goes well. The parties in the real protocol have access to this Arbiter. Furthermore, the ideal world trusted party $\mathcal{U}$ would return the outputs to the adversary and the honest party simultaneously. When *fairness* is a requirement, we need the simulator to simulate the interactions with the Arbiter as well [35], since it is an honest party. In the fair security definitions below, the last 3 steps of the regular security ideal worlds above are condensed into one last step (which is the only difference between the ideal worlds when fairness is involved).

Ideal World: Players are the adversary $\mathcal{C}$, the honest party $\mathcal{H}$, and the ideal trusted party $\mathcal{U}$.

1. $\mathcal{U}$ receives input x or ABORT from $\mathcal{C}$, and y or ABORT from $\mathcal{H}$. If any ABORT message is received, $\mathcal{U}$ sends ABORT to both of the parties and halts.
2. Otherwise $\mathcal{U}$ computes $f(x,y) = f_C, f_H$. $\mathcal{U}$ sends f_C to $\mathcal{C}$ **and** f_H to $\mathcal{H}$.

Fair Security against Covert Adversaries: In the covert setting, remember that security and correctness hold unless the cheating is undetected. The same is true for fairness.

Ideal World: Players are the adversary $\mathcal{C}$, the honest party $\mathcal{H}$, and the ideal trusted party $\mathcal{U}$.

1. $\mathcal{U}$ receives input x or ABORT or CORRUPTED or CHEAT from $\mathcal{C}$, and y or ABORT from $\mathcal{H}$. If any ABORT message is received, $\mathcal{U}$ sends ABORT to both of the parties and halts.

2. If $\mathcal{C}$ sent CORRUPTED, then $\mathcal{U}$ sends CORRUPTED to $\mathcal{H}$ and halts. This is to enable the adversary to voluntarily disclose itself.
3. If $\mathcal{C}$ says CHEAT, with ϵ probability, $\mathcal{U}$ sends CORRUPTED to $\mathcal{H}$ and halts (thus the adversary is caught), and with $1 - \epsilon$ probability, $\mathcal{U}$ sends UNDETECTED and y to $\mathcal{C}$.
   (a) If $\mathcal{C}$ receives UNDETECTED, responds with some output o. Then $\mathcal{U}$ sends o to $\mathcal{H}$ and halts. This means, **when undetected, the adversary can break fairness, security, and correctness**.
4. Otherwise $\mathcal{U}$ computes $f(x, y) = f_C, f_H$. $\mathcal{U}$ sends f_C to $\mathcal{C}$ **and** f_H to $\mathcal{H}$.

4 Fair Covert Cut-and-Choose 2PC

In Figure 1 and 2 we provide a full description of our fair covert 2PC, and Figure 3 and 4 show the resolutions with the Arbiter for Bob and Alice, respectively. As discussed in the overview section (Section 2) our starting point is the covert 2PC protocol of [7]. The vertical lines in the figures represent parts that would not have existed in the unfair counterpart. Appendix 4.1 presents proof of security of the protocol. Finally, the protocol remains secure against covert adversaries even if the Arbiter actively cheats and colludes with one of the parties. See Appendix 4.2 for proof of security of the malicious Arbiter case.

Performance. Our protocol confirms that the cost of achieving fairness is very small, and the amount of work the Arbiter needs to perform (in case of a resolution) is *independent* of the circuit size. Mainly, we require Alice to commit to and encrypt Bob's output translation table, and sign it. Thus, compared to an unfair state-of-the-art covert protocol, Alice needs to perform s commitments and encryptions of size proportional to Bob's output length, and s signatures on those. Bob, of course, needs to verify all these. Finally, at the last step, Alice needs to send an extra message as the opening of the e^{th} commitment to Bob's decoding table, hence adding a single round of interaction to the protocol. Table 2 summarizes this overhead.

Extra Rounds	Extra Messages' Size	Operation Type
1	$O(sm)$	Public Key

Table 2: Overhead for fairness (Covert). Round is a single message. s is the statistical security parameter, m is Bob's output length.

Optimistic Fair Exchange Considerations. The optimistic fair exchange literature includes many implementation details we skip here for the sake of clarity (see *e.g.* [5, 45, 44]). For example, to ensure that the Arbiter treats different computations separately, the signatures must include a **unique session identifier** *sid*. This session identifier can be generated jointly randomly by Alice and Bob, put into the signatures by Alice, and verified by Bob. Furthermore, the **resolution timeouts** (*i.e.* deadlines, which are current time plus allowed timeout) can be put in the signature as well. Bob must abort if the resolution timeout is very different from what he was expecting, and the Arbiter checks this deadline during resolutions.[4] Note that a *message timeout* (to decide that the other party did not respond back) is different from the resolution timeout, and is much shorter.[5] Finally, obviously there is a *computation timeout* in this protocol, which is related to the reasonable

[4]No tight synchronization is necessary. For example, if Bob has half an hour until the deadline, and the parties' clock differs by 5 minutes, it may still be considered to be within allowable range.

[5]For example, consider it as a web request timeout, around 2 minutes.

amount of time for Bob to finish computing the garbled circuit. We do not complicate our protocol with such a computation timeout description, since **until the resolution timeout is defined during the output exchange phase of our protocol, any party can locally abort the protocol without breaking fairness**. If Alice already registered her **signature verification key** with the Arbiter, our protocol can be employed as is. But, if we want anonymity, then the Arbiter must have a way of obtaining this verification key. The standard mechanism is to put it into the label of a labeled encryption scheme. In our case, Alice can generate a new key pair for each computation (or even circuit) and put the verification key into the label of d_i^B encryptions, and Bob can verify the signatures using this verification key. The Arbiter can then also use this key for verification. Details such as these are well-discussed in the previous work [5, 46, 42, 41], and hence we do not repeat for the sake of space.

4.1 Proof of Fair Covert 2PC Protocol

Theorem 4.1. *If the committed-OT protocol used is secure against malicious adversaries[37], the garbling scheme is secure and provides output authenticity [13], the (labeled) public key encryption scheme employed is semantically secure [67], the hash function used is second-preimage resistant [62], the trapdoor commitment scheme is hiding and binding [23, 52], and the signature scheme is existentially unforgeable [25], then our construction is fair and secure against malicious Bob and covert Alice.*

4.1.1 Proof

Security against *malicious* Bob: For any adversary $\mathcal{B}$ corrupting Bob in the real protocol, we describe a simulator $\mathcal{S}_B$ in the ideal world such that the joint outputs of both parties in the two worlds are indistinguishable.

1. $\mathcal{S}_B$ generates a key pair on behalf of the Arbiter, and another on behalf of Alice, and provides the public keys to $\mathcal{B}$. She further generates the commitment parameters and keeps a trapdoor.[6]
2. $\mathcal{S}_B$ runs $\mathcal{B}$ and plays the ideal trusted party $\mathcal{U}_{COT}$ in the committed oblivious transfer to receive $\mathcal{B}$'s input x'_B. If $\mathcal{B}$ aborts at this stage, $\mathcal{S}_B$ sends ABORT to the ideal trusted party $\mathcal{U}_f$ for f and simulates honest Alice aborting. Else, she continues as honest Alice would in the protocol.
3. $\mathcal{S}_B$ learns the challenge index e chosen by $\mathcal{B}$ by playing the role of the ideal trusted party in the OT-based challenge generation. She picks a random x'_A as Alice's input and uses it during the OT-based challenge generation. She generates the circuits, commitments, decoding tables, encryptions all as an honest Alice would.
4. $\mathcal{S}_B$ continues as honest Alice would, until she receives output labels for Alice's output from $\mathcal{B}$.
    (a) If they are correct, then she sends x'_B to $\mathcal{U}_f$, and receives $f_B(x_A, x'_B)$ as output, and hence learns the correct out'_B labels. Note that $\mathcal{S}_B$ prepared everything using a random input x'_A for Alice, whereas the output labels out'_B are based on the actual input x_A of Alice. Therefore, $\mathcal{S}_B$ now needs to fake the opening of her commitment. She generates

[6] e.g., Pedersen commitments [59] have a trapdoor $dlog_g h$, which the simulator can use in the CRS model by creating the commitment parameters.

Alice's input: $x_A \in \{0,1\}^\ell$. **Bob's input:** $x_B \in \{0,1\}^\ell$.
Common input: Alice and Bob agree on the description of a circuit C, where $C(x_A, x_B) = f(x_A, x_B) = (f_A(x_A, x_B), f_B(x_A, x_B))$, and a second-preimage resistant hash function $H : \{0,1\}^* \to \{0,1\}^\ell$.
s is a statistical security parameter that is inversely proportional to the bound on the cheating probability. L is a computational security parameter, so, for example, each key label is L-bits long. Let $\mathsf{TCommit}(\cdot)$ be a trapdoor commitment scheme.
Setup: Let (PK_T, SK_T) be the Arbiter's key pair for a public key encryption, and (SK_A, VK_A) be the signing-verification key-pair for a digital signature scheme for Alice. At the beginning of the protocol, both parties obtain the Arbiter's public key from the Arbiter. Alice sends her verification key to Bob and the Arbiter.
Output: Alice learns an m-bit string $f_A(x_A, x_B)$ and Bob learns an m-bit string $f_B(x_A, x_B)$.

Alice Prepares the Garbled Circuits.
1. For $1 \le i \le s$, Alice computes $\mathsf{GC}_i \leftarrow \mathsf{Garble}(C)$.
2. Let $\mathsf{in}_b^{A,i,j}$ denote the key for bit b for Alice's j^{th} input wire in the i^{th} garbled circuit for $b \in \{0,1\}$, $1 \le i \le s$, and $1 \le j \le \ell$. $\mathsf{in}_b^{B,i,j}$ is defined similarly for Bob's input labels.
3. Let $\mathsf{out}_b^{A,i,j}$ denote the key for bit b for Alice's j^{th} output wire in the i^{th} garbled circuit. $\mathsf{out}_b^{B,i,j}$ is used for Bob's output labels.
4. Alice lets $\mathsf{GDec}_i^B = \left\{\mathsf{out}_0^{B,i,j}, \mathsf{out}_1^{B,i,j}\right\}_{j=1}^m$ be the decoding table for Bob's output.
5. | Alice computes $\mathsf{GDec}_i^A = \left\{H(\mathsf{out}_{b_{i,j}}^{A,i,j}), H(\mathsf{out}_{\neg b_{i,j}}^{A,i,j})\right\}_{j=1}^m$ for random bits $b_{i,j}$ as the output validity-checking table for her output. [This table is randomly permuted based on the bits $b_{i,j}$ such that in case of output resolution, the Arbiter can check validity of output labels without learning their actual value.]
6. | She computes $c_i^B = \mathsf{TCommit}(\mathsf{GDec}_i^B)$ as the commitment to Bob's output decoding table. Let $\mathsf{GDecOpen}_i^B$ be the opening of this commitment. She encrypts this opening as $d_i^B = E_{PK_T}(\mathsf{GDecOpen}_i^B)$ using the Arbiter's public key. [GDec_i^A and d_i^B will be used by the Arbiter to verify Bob's honesty, and give back Bob's output decoding table, respectively. c_i^B will be employed by Bob to ensure that Alice behaves honestly at the last step. Note that GDec_i^A is computed using a second-preimage resistant hash function, whereas c_i^B is computed via trapdoor commitments.]
7. | Alice signs those as $\sigma_i = Sign(SK_A, (sid, \mathsf{GDec}_i^A, d_i^B))$. [This signature will tie the two decryption tables to the same circuit, and be checked by the Arbiter. sid is the unique session identifier.]

Oblivious Transfer for Bob's Input.
1. Alice and Bob engage in ℓ committed OTs, where in the j^{th} OT, Bob's input is $x_{B,j}$ and Alice input is a pair where the first component is $[\mathsf{in}_0^{B,1,j}, \ldots, \mathsf{in}_0^{B,s,j}]$ and the second component is $[\mathsf{in}_1^{B,1,j}, \ldots, \mathsf{in}_1^{B,s,j}]$. As a result, Bob learns $\mathsf{in}_{x_{B,j}}^{B,i,j}$ for $1 \le i \le s$, $1 \le j \le \ell$.

Alice Sends the Circuits.
1. Alice sends $\left\{\mathsf{GC}_i, \mathsf{GDec}_i^A, c_i^B, d_i^B\right\}_{i=1}^s$ to Bob.

OT-based Challenge Generation.
1. Bob picks a random challenge index e, lets $b_e = 0$, and $b_i = 1$ for all $i \neq e$.
2. Alice and Bob run s OTs where Bob's input (as the receiver) in the i^{th} OT is b_i while Alice (as the sender) inputs a pair where the first component is her garbled inputs $\{\mathsf{in}_{X_{A,j}}^{A,i,j}\}_{j=1}^\ell$ along with σ_i, and the second component is the openings for $\mathsf{GC}_i, \mathsf{GDec}_i^A, c_i^B, d_i^B$ and the input and randomness she used in the i^{th} committed OT above. In other words, for $i \neq e$ Bob learns openings for everything about the circuits, and for $i = e$ he learns Alice's input labels and her signature.

Bob Verifies Check Circuits.
1. For $i \neq e$, Bob uses the openings he obtained in the challenge generation phase to check the correctness of GC_i. He verifies the consistency of c_i^B with d_i^B both of which he has openings for. He checks that GDec_i^B is consistent with GC_i.
2. | For $i = e$, he verifies the signature σ_e.

Bob Evaluates.
1. Note that Bob has Alice's input labels for the e^{th} circuit via the OT-based challenge generation and his own input labels via the committed OT. He evaluates GC_e.

Figure 1: Optimistic Fair **Covert** 2PC

Output Exchange.

1. Bob tells Alice that he is done with the evaluation.
2. Alice responds by sending $\sigma_t = Sign(SK_A, (sid, deadline))$. Bob checks the timeout and the session identifier are consistent with the agreed upon values, and aborts otherwise.
3. Denote the labels for Alice's output by $\left\{\mathsf{out}^{A,e,j}_{out_{A,j}}\right\}_{j=1}^{m}$. Bob sends these to Alice, along with σ_e so that Alice will learn the evaluated circuit identifier e, and then she translates them to her actual output on her own. If Alice does not receive the correct labels in time, she contacts the Arbiter for resolution.
4. Alice opens c^B_e to the decoding table GDec^B_e, which Bob uses to decode his actual output. If Bob does not receive the correct decoding table in time, he contacts the Arbiter for resolution.

Figure 2: Optimistic Fair **Covert** 2PC (cnt'd)

1. Bob sends $\mathsf{GDec}^A_e, d^B_e, \sigma_e, \sigma_t$ to the Arbiter. He also sends labels for Alice's output *i.e.* $\left\{\mathsf{out}^{A,e,j}_{out_{A,j}}\right\}_{j=1}^{m}$.
2. The Arbiter verifies the signatures, checks that the time is earlier than the deadline in σ_t and the session identifiers are matching. He also makes sure $\mathsf{out}^{A,e,j}_{out_{A,j}}$ values are consistent with GDec^A_e. Essentially, one output label per pair must be provided. He aborts if any of the checks fail.
3. In case of no fails, the Arbiter decrypts d^B_e and sends $\mathsf{GDecOpen}^B_e$ to Bob. He stores $\left\{\mathsf{out}^{A,e,j}_{out_{A,j}}\right\}_{j=1}^{m}$ for Alice.
4. Bob checks that $\mathsf{GDecOpen}^B_e$ is the correct opening for c^B_e,[a] and uses GDec^B_e in the opening to translate his output labels to actual outputs.

[a]This check is necessary against potentially malicious Arbiter to preserve correctness.

Figure 3: Resolution for Bob (for Optimistic Fair **Covert** 2PC)

1. If Alice contacts the Arbiter *before* the timeout and Bob has *not* contacted the Arbiter yet, the Arbiter tells Alice to come after the timeout.
2. If Alice contacts the Arbiter *after* the timeout and Bob has *not* contacted the Arbiter yet, the protocol is aborted and no party obtains the actual output.
3. Else (Bob already contacted the Arbiter and resolved), the Arbiter sends $\left\{\mathsf{out}^{A,e,j}_{out_{A,j}}\right\}_{j=1}^{m}$ obtained via Bob's resolution to Alice (after making sure she is the same Alice, *e.g.* by asking for the input to a one way function whose output was in the associated signature given by Bob, see *e.g.* [5]).
4. Alice translates $\left\{\mathsf{out}^{A,e,j}_{out_{A,j}}\right\}_{j=1}^{m}$ to actual outputs on her own.

Figure 4: Resolution for Alice (for Optimistic Fair **Covert** 2PC)

the GDec_e^B according to out'_B, and sends it to Bob together with a simulated opening for c_e^B using the trapdoor.

(b) If the labels were problematic (or was not received within a reasonable message timeout), $\mathcal{S}_B$ waits until the resolution timeout. During this time, if Bob wants to perform a resolution with the Arbiter, she acts as the Arbiter.

i. If Bob provides correct labels to the Arbiter, then she acts identical to the step above (*i.e.* contacts $\mathcal{U}_f$, providing x'_B and obtaining back the output, preparing GDec_e^B via the corresponding out'_B, and giving the corresponding $\mathsf{GDecOpen}_e^B$ to Bob using the trapdoor).

ii. If Bob did not contact the Arbiter until the timeout, $\mathcal{S}_B$ sends ABORT to $\mathcal{U}_f$ and aborts. In this case, no party learns the actual output.

Indistinguishability: $\mathcal{S}_B$ is indistinguishable from real Alice, as she acts exactly the same with small exceptions.

- In step 2, the hybrid model allows her to act as the trusted party for the committed oblivious transfer.
- In step 3, similarly she acts as the trusted party for the OT-based challenge generation, using a random input for Alice. But due to security of the garbling scheme this remains indistinguishable to Bob. We want to remind the reader that the circuit is outputting random labels, which will only make sense once the translation table is obtained. Therefore, until the opening of the translation table, Bob cannot distinguish anything.
- $\mathcal{S}_B$ also behaves exactly like the Arbiter when Bob resolves.
  - Note that due to the output-authenticity of the garbling scheme and the fact that H is a second-preimage resistant hash function, Bob cannot forge another valid output label except with negligible probability in L. Because Bob only learns one output label per pair of hashes he is given per bit, due to the output authenticity of the garbling scheme. He cannot use the hash values to break output authenticity by finding a different output label, neither by finding a second preimage to the output label he computed, nor by finding a preimage to the hash value corresponding to the other bit.
  - When the Arbiter decrypts, it may be to a different $\mathsf{GDecOpen}$, but semantic security of encryption makes sure that this is indistinguishable.
- Lastly, the trapdoor commitment scheme allows $\mathcal{S}_B$ to fake the opening to the correct GDec_e^B after contacting $\mathcal{U}_f$.

Hence, outputs in the two worlds would be identical (*i.e.* either abort or the correct output), except with negligible probability. Finally, note that the simulator contacts the ideal trusted party for the output *only* when both parties can obtain their outputs in the real world.

Security against *covert* Alice. For any adversary $\mathcal{A}$ corrupting Alice in the real protocol, we describe a simulator $\mathcal{S}_A$ in the ideal world such that the joint outputs of both parties in the two worlds are indistinguishable.

1. $\mathcal{S}_A$ generates a key pair on behalf of the Arbiter, and provides the public key to $\mathcal{A}$. $\mathcal{S}_A$ obtains the signature public key of $\mathcal{A}$.
2. $\mathcal{S}_A$ picks a random input x'_B on behalf of Bob, and plays the role of the ideal trusted party $\mathcal{U}_{COT}$ in the committed OT in order to obtain $\mathcal{A}$'s input to the OT (all labels for Bob's input wires). If $\mathcal{A}$ aborts, $\mathcal{S}_A$ sends ABORT to the ideal trusted party $\mathcal{U}_f$ for f and simulates honest Bob aborting.
3. $\mathcal{S}_A$ receives from $\mathcal{A}$ her inputs to the s OTs for the OT-based challenge generation. In other

words, he learns the openings for $\mathsf{GC}_i, \mathsf{GDec}_i^A, c_i^B, d_i^B$ and the i^{th} committed OT, as well as $\mathcal{A}$'s input labels $\{\mathsf{in}_{X_{A,j}}^{A,i,j}\}_{j=1}^{\ell}$ along with σ_i, for all $i \in \{1, \ldots, s\}$.

4. We call an index i valid if all the openings for $\mathsf{GC}_i, \mathsf{GDec}_i^A, c_i^B, d_i^B$ and the i^{th} committed OT for Bob's input are correct and consistent. We now have three different cases:
   (a) All indices are valid: $\mathcal{S}_A$ lets the challenge index be a random $e \in \{1, \ldots, s\}$. In this case, $\mathcal{S}_A$ uses $\mathcal{A}$'s input labels for the e^{th} circuit (he obtained them from OT-based challenge generation) to extract her actual input x'_A. If the signature σ_e also verifies, he tells $\mathcal{A}$ that he is done.
      i. If a correct signature σ_t with a good timeout is received, he sends x'_A to $\mathcal{U}_f$ and receives $f_A(x'_A, x_B)$ back. He uses it and the openings for e^{th} circuit to derive Alice's output labels out'_A for the e^{th} circuit and sends them to her. If Alice does not respond back with the correct labels for Bob's output, he decrypts d_e^B and obtains Bob's output (simulating honest Bob resolving with the Arbiter). If Alice contacts the Arbiter afterward, he sends Alice's output labels again.
      ii. In any other case, or if the decryption of d_e^B was problematic, he sends ABORT to $\mathcal{U}_f$ and halts. Note that d_e^B being faulty only happens with at most $1/s$ probability, in which case the definition allows the adversary to break fairness as well.
   (b) Two or more indices are invalid. In this case, $\mathcal{A}$ is always caught in the real protocol. $\mathcal{S}_A$ lets the challenge index be a random $e \in \{1, \ldots, s\}$ and sends CORRUPTED to $\mathcal{U}_f$, outputting whatever $\mathcal{A}$ does.
   (c) All but one of the indices are valid. $\mathcal{S}_A$ sends CHEAT to $\mathcal{U}_f$. With probability $1 - 1/s$ the ideal trusted party returns CORRUPTED. If so, $\mathcal{S}_A$ chooses the challenge index to be one of the valid ones. He then simulates honest Bob aborting and outputs whatever $\mathcal{A}$ does. With the remaining probability of $1/s$, $\mathcal{U}_f$ returns UNDETECTED together with honest Bob's input x_B. Then $\mathcal{S}_A$ lets the challenge index be the invalid index. $\mathcal{S}_A$ then simulates honest Bob identically to the real protocol (using his input x_B) and outputs whatever $\mathcal{A}$ does.

Indistinguishability: $\mathcal{S}_A$ is indistinguishable from real Bob, as he behaves exactly the same.

- One exception is that he picks a random input for Bob. But, the output he receives from $\mathcal{U}_f$ is created using the correct input of Bob, and hence will be identical in both worlds.
- $\mathcal{S}_A$ also behaves exactly like the Arbiter during resolutions, and only aborts whenever a real Bob would.
- The joint output distributions would only be distinguishable if $\mathcal{A}$ manages to forge a signature, which only happens with a negligible probability.

4.2 Security against Colluding Arbiter - Covert Protocol

Theorem 4.2. *If the committed-OT protocol used is secure against malicious adversaries [18], the garbling scheme is secure and provides output authenticity [13], the (labeled) public key encryption scheme employed is semantically secure [67], the hash function used is second-preimage resistant [62], the trapdoor commitment scheme is hiding and binding [23, 52], and the signature scheme is existentially unforgeable [25], then our construction is secure (with abort) against malicious Bob and covert Alice, even when the Arbiter acts maliciously and colludes with one of them.*

4.2.1 Proof

Note that by security here, we mean the usual unfair definition that allows the adversary to abort after receiving his output but before the honest party receives her output. The corresponding security definition was already given. We prove for Alice and Bob separately, assuming the Arbiter colludes with the malicious one. This means, the simulator is *not* allowed to act on behalf of the Arbiter in its simulation.

Security against *malicious* Bob and colluding Arbiter: For any adversary $\mathcal{B}$ corrupting Bob in the real protocol, we describe a simulator $\mathcal{S}_B$ in the ideal world such that the joint outputs of both parties in the two worlds are indistinguishable.

1. $\mathcal{S}_B$ generates a key pair on behalf of Alice, and provides the signature verification key to $\mathcal{B}$. She learns the Arbiter's public key. She does not need a commitment trapdoor for this proof.
2. $\mathcal{S}_B$ runs $\mathcal{B}$ and plays the ideal trusted party $\mathcal{U}_{COT}$ in the committed oblivious transfer to receive $\mathcal{B}$'s input x'_B. If $\mathcal{B}$ aborts at this stage, $\mathcal{S}_B$ sends ABORT to the ideal trusted party $\mathcal{U}_f$ for f and simulates honest Alice aborting. Else, she continues as honest Alice would in the protocol.
3. $\mathcal{S}_B$ sends x'_B to the ideal trusted party $\mathcal{U}_f$ for f, and receives $f_B(x_A, x'_B)$ as output, and hence learns the correct out'_B labels.
4. $\mathcal{S}_B$ learns the challenge index e chosen by $\mathcal{B}$ by playing the role of the ideal trusted party in the OT-based challenge generation. For all $i \neq e$, she generates the circuits, commitments, decoding tables, encryptions all as an honest Alice would. For GC_e, he garbles a circuit that always outputs out'_B labels as Bob's output, but does the normal computation for Alice's output, and prepares c_e^B, d_e^B according to out'_B.
5. $\mathcal{S}_B$ continues the same as honest Alice until she receives output labels for Alice's output from $\mathcal{B}$. If they are correct, then she opens c_e^B to GDec_e^B (previously prepared according to out'_B) to Bob. She also sends CONTINUE to U_f and halts.
6. If the labels were problematic (or was not received in a reasonable amount of time –*e.g.* a few round-trip message delays–), $\mathcal{S}_B$ waits until the resolution timeout and then contacts the Arbiter. If the resolution is successful, she sends CONTINUE to U_f, else she sends ABORT to U_f and halts.

Indistinguishability: $\mathcal{S}_B$ is indistinguishable from real Alice, as she acts exactly the same with one difference: She uses random inputs for Alice in the garbled circuits. But due to security properties of the garbling scheme this remains indistinguishable to Bob. The simulator only aborts in the ideal world when the real adversary aborts. Hence, outputs in the two worlds would be identically distributed (*i.e.* either abort or the correct output), except with negligible probability.

Security against *covert* Alice and colluding Arbiter. For any adversary $\mathcal{A}$ corrupting Alice in the real protocol, we describe a simulator $\mathcal{S}_A$ in the ideal world such that the joint outputs of both parties in the two worlds are indistinguishable.

1. $\mathcal{S}_A$ obtains the signature public key and the Arbiter's public key from $\mathcal{A}$.
2. $\mathcal{S}_A$ picks a random input x'_B on behalf of Bob, and plays the role of the ideal trusted party $\mathcal{U}_{COT}$ in the committed OT in order to obtain $\mathcal{A}$'s input to the OT (all labels for Bob's input wires). If $\mathcal{A}$ aborts, $\mathcal{S}_A$ sends ABORT to the ideal trusted party $\mathcal{U}_f$ for f and simulates honest Bob aborting.
3. $\mathcal{S}_A$ receives from $\mathcal{A}$ her inputs to the s OTs for the OT-based challenge generation. In other words, he learns the openings for $\mathsf{GC}_i, \mathsf{GDec}_i^A, c_i^B, d_i^B$ and the i^{th} committed OT, as well as $\mathcal{A}$'s input labels $\{\mathsf{in}_{X_{A,j}}^{A,i,j}\}_{j=1}^{\ell}$ along with σ_i, for all $i \in \{1, \dots, s\}$.

4. We call an index i valid if all the openings for $\mathsf{GC}_i, \mathsf{GDec}_i^A, c_i^B, d_i^B$ and the i^{th} committed OT for Bob's input are correct and consistent. We now have three different cases:
   (a) All indices are valid: $\mathcal{S}_A$ lets the challenge index be a random $e \in \{1, \ldots, s\}$. In this case, $\mathcal{S}_A$ uses $\mathcal{A}$'s input labels for the e^{th} circuit (he obtained them from OT-based challenge generation) to extract her actual input x'_A. If the signature σ_e also verifies, he tells $\mathcal{A}$ that he is done.
      i. If a correct signature σ_t with a good timeout is received, he sends x'_A to $\mathcal{U}_f$ and receives $f_A(x'_A, x_B)$ back. He uses it and the openings for e^{th} circuit to derive Alice's output labels out'_A for the e^{th} circuit and sends them to her. If Alice does not respond back with the correct labels for Bob's output, he tries to resolve with the Arbiter. If he successfully obtains the correct opening of c_e^B via the decryption of d_e^B, he sends CONTINUE to U_f.
      ii. In any other case, or if the decryption of d_e^B was problematic, he sends ABORT to $\mathcal{U}_f$ and halts. Note that d_e^B being faulty only happens with at most $1/s$ probability, in which case the definition allows the adversary to break fairness as well.
   (b) Two or more indices are invalid. In this case, $\mathcal{A}$ is always caught in the real protocol. $\mathcal{S}_A$ lets the challenge index be a random $e \in \{1, \ldots, s\}$ and sends CORRUPTED to $\mathcal{U}_f$, outputting whatever $\mathcal{A}$ does.
   (c) All but one of the indices are valid. $\mathcal{S}_A$ sends CHEAT to $\mathcal{U}_f$. With probability $1 - 1/s$ the ideal trusted party returns CORRUPTED. If so, $\mathcal{S}_A$ chooses the challenge index to be one of the valid ones. He then simulates honest Bob aborting and outputs whatever $\mathcal{A}$ does. With the remaining probability of $1/s$, $\mathcal{U}_f$ returns UNDETECTED together with honest Bob's input x_B. Then $\mathcal{S}_A$ lets the challenge index be the invalid index. $\mathcal{S}_A$ then simulates honest Bob identically to the real protocol (using his input x_B) and outputs whatever $\mathcal{A}$ does.

Indistinguishability: $\mathcal{S}_A$ is indistinguishable from real Bob, as he behaves exactly the same, and aborts exactly at the same locations. The hybrid model allows $\mathcal{S}_A$ to act as the trusted parties in the oblivious transfers.

5 Fair Malicious Cut-and-Choose 2PC

In this section we describe an efficient malicious 2PC protocol that also achieves fairness. As discussed in the overview (Section 2), we can employ stat-of the-art techniques such as cheating-detection, Free-XOR, FleXor, and Half-gates [48, 39, 38, 69]. Figures 5, 6 and 7 describe the full protocol, and Figures 8 and 9 show the resolutions with the Arbiter for Bob and Alice, respectively. The vertical lines in the figures represent parts that would not have existed in the unfair counterpart. The protocol remains secure against malicious adversaries even if the Arbiter actively cheats and colludes with one of the parties. See Appendix 5.1 for the proof of security. Appendix 5.2 prove that even if the Arbiter maliciously colludes with the adversarial party, only fairness is lost but correctness/privacy is preserved.

Performance. Our protocol confirms that the cost of achieving fairness is very small, and the amount of work the Arbiter needs to perform (in case of a resolution) is *independent* of the circuit size. We require Alice to extend her input length by $2m + t$ bits for the pads (pad_A, pad_B) used to encrypt the outputs and the random input r_p used for input consistency of pad_B. This also requires extending the circuit for the one time pad computations, but free XOR technique [39] enables us to obtain this for free. The input length to the cheating-detection circuit is also extended by m bits for the pad for Alice's output.

As in the fair covert protocol, we require Alice to commit to and encrypt Bob's output translation table, and sign it. Thus, compared to an unfair state-of-the-art covert protocol, Alice needs to perform s commitments and encryptions of size proportional to Bob's output length m, and $s/2$ signatures on those, plus 2 other signatures. Bob, of course, needs to verify all these. In addition, we need Bob to commit to Alice's m output labels. Finally, at the last step, Alice needs to send an extra message as the opening of $s/2$ commitments to Bob's decoding tables (though commitments can be constant size using a collision-resistant hash function, their openings must be of size $O(m)$). Table 3 summarizes this overhead.

Extra Rounds	Extra Input Length	Extra Messages' Size	Operation Type
3	$O(m+t)$	$O(s(m+t))$	Public Key

Table 3: Overhead for fairness (Malicious). Round is a single message. s is the statistical security parameter, m is the output length, t is a security parameter for input consistency.

5.1 Proof of Fair Malicious 2PC Protocol

Theorem 5.1. *If the committed-OT protocol used is secure against malicious adversaries [18], the garbling scheme is secure and provides output authenticity [13], the (labeled) public key encryption scheme employed is semantically secure [67], the hash function used is second-preimage resistant [62], the trapdoor commitment scheme is hiding and binding [23, 52], and the signature scheme is existentially unforgeable [25], then our construction is fair and secure against malicious Bob and malicious Alice.*

5.1.1 Proof

Security against *malicious* Bob: For any adversary $\mathcal{B}$ corrupting Bob in the real protocol, we describe a simulator $\mathcal{S}_B$ in the ideal world such that the joint outputs of both parties in the two worlds are indistinguishable.

1. $\mathcal{S}_B$ generates a key pair on behalf of the Arbiter, and another on behalf of Alice, and provides the public keys to $\mathcal{B}$. She further generates the commitment parameters and keeps a trapdoor.
2. $\mathcal{S}_B$ picks $x_A^C = pad_B \| r_p \| x'_A \| pad_A \| r_x$ randomly, and emulates honest Alice all the way upto the oblivious transfer for Bob's inputs. In these OTs, $\mathcal{S}_B$ plays the ideal trusted party $\mathcal{U}_{COT}$ in the committed oblivious transfer to receive $\mathcal{B}$'s input x'_B. She continues as an honest Alice would.
3. $\mathcal{S}_B$ plays the honest Alice in the coin-toss and learns the evaluation sets $\mathbb{E}$ and $\mathbb{E}'$ as a result. She generates and sends the circuits, commitments, encryptions all as an honest Alice would.
4. Until the output exchange phase, $\mathcal{S}_B$ simulates honest Alice. If $\mathcal{B}$ aborts, $\mathcal{S}_B$ sends ABORT to the ideal trusted party $\mathcal{U}_f$ for f and simulates honest Alice aborting.
5. $\mathcal{S}_B$ finally receives the openings for C_A.
   (a) If the openings are proper, she sends x'_B to $\mathcal{U}_f$, and receives $out'_B = f_B(x_A, x'_B)$ as output. Together with the pad_B that she picked, this now allows her to figure out which W labels would be the correct ones. She responds to Bob with simulating the openings of c_i^B for $i \in \mathbb{E}$ in such a way that the pads translate correctly.

Alice's input: $x_A \in \{0,1\}^\ell$. **Bob's input:** $x_B \in \{0,1\}^\ell$.
Common input: Alice and Bob agree on the description of a circuit C, where $C(x_A, x_B) = f(x_A, x_B) = (f_A(x_A, x_B), f_B(x_A, x_B))$, and a second-preimage resistant hash function $H : \{0,1\}^* \to \{0,1\}^\ell$.
s is a statistical security parameter that represents the bound on the cheating probability. L is a computational security parameter, so, for example, each key label is L-bits long. s' is a statistical security parameter associated with the input-consistency matrix. Let $t = 2s' + \log s'$, and $\ell' = 2m + \ell + 2t$.
Let $\mathsf{TCommit}(\cdot)$ be a trapdoor commitment scheme, and $\mathsf{Commit}(\cdot)$ be a regular commitment scheme.
Setup: Let (PK_T, SK_T) be the Arbiter's key pair for a public key encryption, and (SK_A, VK_A) be the signing-verification key-pair for a digital signature scheme for Alice. At the beginning of the protocol, both parties obtain the commitment parameters, and the Arbiter's public key from the Arbiter. Alice sends her verification key to Bob and the Arbiter.
Output: Alice learns an m-bit string $out_A = f_A(x_A, x_B)$ and Bob learns an m-bit string $out_B = f_B(x_A, x_B)$.

Alice Prepares Input/Output Labels.

1. Alice chooses s PRF seeds $sd_1^A, \ldots, sd_s^A$, and commits to them using $\mathsf{Commit}(sd_1^A), \ldots, \mathsf{Commit}(sd_s^A)$. All the randomness Alice will use for generating the i^{th} garbled computation circuit and its input labels will be derived from sd_i^A. Similarly, she chooses $3s$ PRF seeds $sd_1'^A, \ldots, sd_{3s}'^A$, and commits to them, where the randomness she uses for generating the i^{th} garbled cheating-detection (CD) circuit and its input labels will be derived from $sd_i'^A$.
2. Alice chooses $r_x, r_p \in_R \{0,1\}^t, pad_A, pad_B \in_R \{0,1\}^m$ and sets $x_A^C = pad_B \| r_p \| x_A \| pad_A \| r_x$. She will be using x_A^C as her input to the computation circuits instead of x_A. We denote the j^{th} bit of x_A^C by $x_{A,j}^C$.
3. Alice chooses $\mathsf{in}_b^{A,i,j} \in_R \{0,1\}^L$ for $b \in \{0,1\}$, $1 \le i \le s$ and $1 \le j \le \ell'$. $\mathsf{in}_b^{A,i,j}$ would be the b-key for Alice's j^{th} input wire in the i^{th} computation garbled circuit.
4. Alice sends $\mathsf{Commit}(H(\mathsf{in}_{x_{A,1}^C}^{A,i,1} \| \cdots \| \mathsf{in}_{x_{A,\ell'}^C}^{A,i,\ell'}))$ for $1 \le i \le s$, i.e. commitments to encoding of her inputs. This is intended to commit Alice to her inputs before the matrices associated with input-consistency are chosen.
5. Alice lets her input to the CD circuit be $x_A^{CD} = x_A \| pad_A \| r_x$. She chooses random labels for the associated input wires in the CD circuits and commits to the encoding of her inputs as she did for the computation circuits.
6. Alice chooses $W_b^{A,j} \in_R \{0,1\}^L$ for $j \in \{1, \ldots, m\}$ and $b \in \{0,1\}$. Similarly, she chooses $W_b^{B,j} \in \{0,1\}^L$. These correspond to labels for output wires corresponding to Alice's and Bob's output (padded with Alice's pads), respectively, and unlike the covert protocol, will be the *same* across all s circuits.
7. Alice lets $\mathsf{GDec}^B = \{H(W_0^{B,j}), H(W_1^{B,j})\}_{j=1}^m$ be the decoding table for Bob's output. She also lets $\mathsf{GDec}^A = \{H(W_0^{A,j}), H(W_1^{A,j})\}_{j=1}^m$. The translation table for other outputs of the circuit (i.e. outputs of the UH functions) will be created in the standard way and with different labels for each circuit.
8. Alice lets $\mathsf{PadDec}_i = \{\mathsf{in}_0^{A,i,j}, \mathsf{in}_1^{A,i,j}\}_{j=1}^{m+t}$ for the i^{th} circuit (note that the first $m + t$ input wires are associated with Alice's pad_B and r_p). This is essentially a decoding table for input wires for Alice's pad_B and r_p. Alice then commits to this table $c_i^B = \mathsf{TCommit}(\mathsf{PadDec}_i)$ using the trapdoor commitment scheme, and encrypts its opening as $d_i^B = E_{PK_T}(\mathsf{PadDecOpen}_i)$ using the Arbiter's public key.

Figure 5: Optimistic Fair **Malicious** 2PC

Alice Prepares the Garbled Circuits.

1. Alice and Bob jointly choose random binary matrices $M_x \in_R \{0,1\}^{s' \times \ell+m+t}, M_p \in_R \{0,1\}^{s' \times m+t}$. Let $C'(x_A^C, x_B) = (f_A(x_A, x_B) \oplus pad_A, (f_B(x_A, x_B) \oplus pad_B, M_x \cdot (x_A \| pad_A \| r_x), M_p \cdot (pad_B || r_p)))$. In other words, the circuit pads Alice and Bob's output with separate pads generated by Alice, and also outputs the result of applying the M_x and M_p to x_A and pad_B for input-consistency checks.
2. For $1 \le i \le s$, Alice computes $\mathsf{GC}_i \leftarrow \mathsf{Garble}(C')$ with the consideration that she uses the input and output labels she generated above for the garbling.
3. Denote by CD the cheating detection circuit. Alice's input to this circuit is $x_A^{CD} = x_A \| pad_A \| r_x$. Bob's input is an L-bit string pc, his (potential) proof of Alice's cheating. CD's computation is as outlined in Lindell [48] with the exception that in case of detected cheating x_A and pad_A are both revealed to Bob. In particular, CD has the labels $\{W_0^{B,j}, W_1^{B,j}\}_{j=1}^m$ and $\{W_0^{A,j}, W_1^{A,j}\}_{j=1}^m$ embedded in it and checks whether pc is the XOR of the 0-key and the 1-key for any of the wires. If so, it outputs to Bob $x_A \| pad_A$. Otherwise, it outputs a random string. CD also outputs $M_x \cdot (x_A \| pad_A \| r_x)$ to Bob. Alice has no output.
4. For $1 \le i \le 3s$, Alice computes $\mathsf{GCD}_i \leftarrow \mathsf{Garble}(CD)$ with the consideration that she uses the input labels she generated above for garbling. The translation tables for GCD_i are generated in the standard way.

Oblivious Transfer for Bob's Input to Computation Circuits. Alice and Bob engage in ℓ committed OTs, where in the j^{th} OT, Bob's input is $x_{B,j}$ and Alice's input is a pair where the first component is $[\mathsf{in}_0^{B,1,j}, \dots, \mathsf{in}_0^{B,s,j}]$ and the second component is $[\mathsf{in}_1^{B,1,j}, \dots, \mathsf{in}_1^{B,s,j}]$. As a result, Bob learns $\mathsf{in}_{x_{B,j}}^{B,i,j}$ for $1 \le i \le s$, $1 \le j \le \ell$.

Alice Sends the Garbled Circuits. Alice sends $\{\mathsf{GC}_i, c_i^B, d_i^B\}_{i=1}^s$ and GDec^A, GDec^B, and $\sigma_{\mathsf{out}^A} = Sign(SK_A, (sid, \mathsf{GDec}^A))$ to Bob (where sid is the unique session identifier). She also sends $\{\mathsf{GCD}_i\}_{i=1}^{3s}$ and the associated output translation tables.

Challenge Generation. Alice and Bob jointly run a simulatable coin-toss to generate a uniformly random s-bit string b and a uniformly random $3s$-bit string b'. Define the evaluation set $\mathbb{E}$ where $i \in \mathbb{E}$ if and only if $b_i = 0$, and the evaluation set $\mathbb{E}'$ similarly with respect to b'. Both parties learn $\mathbb{E}$ and $\mathbb{E}'$. Circuits are *not* opened immediately, though.

Bob Evaluates Computation Circuits in $\mathbb{E}$.

1. Alice sends her garbled input labels for x_A^C for all $\mathsf{GC}_i, i \in \mathbb{E}$, by opening the commitments she made to them earlier. Alice also sends $\sigma_{i,pad} = Sign(SK_A, (sid, d_i^B))$ for $i \in \mathbb{E}$. Bob uses these input labels and those of his own from the committed OTs to evaluate all GC_i where $i \in \mathbb{E}$.
2. If there is at least one circuit with a valid output, and all circuits with a valid output return the same output labels $W_{o_{A,1}}^{A,1}, \dots, W_{o_{A,m}}^{A,m}$ and $W_{o_{B,1}}^{B,1}, \dots, W_{o_{B,m}}^{B,m}$, Bob lets $C_A = \mathsf{TCommit}(W_{o_{A,1}}^{A,1}, \dots, W_{o_{A,m}}^{A,m})$. Note that through these, Bob can learn $o_A = out_A \oplus pad_A$ and $o_B = out_B \oplus pad_B$, but since he does not know the pads, these are useless. Bob lets pc be a random L-bit string.
3. If there are at least two circuits with valid but different outputs, Bob chooses the first output wire with different labels and denotes the two labels by W and W'. $pc = W \oplus W'$ will constitute Bob's input to the cheating detection circuits.
4. If all circuits are evaluated to *invalid* output labels (i.e. the obtained labels are not consistent with GDec^A and GDec^B) or if the output of the UHs in any two circuits are different Bob does not abort (until after the opening stage) but instead commits to a random string of appropriate length in C_A.

Figure 6: Optimistic Fair **Malicious** 2PC (cnt'd)

Evaluating Cheating-Detection Circuits in $\mathbb{E}'$.

1. Alice and Bob engage in L committed OTs, where in the j^{th} OT, Bob's input is pc_j and Alice's input is a pair where the first component is the $3s$ input labels corresponding to 0 and the second component is the $3s$ labels corresponding to 1.
2. Alice sends her garbled input labels for x_A^{CD} for GCD_i where $i \in \mathbb{E}'$, by opening the commitments she made to them earlier.
3. Bob uses the input labels to evaluate all GCD_i with $i \in \mathbb{E}'$, and uses the translation tables to translate to plaintext outputs. If any two UH outputs are different or if they are different from those output in the computation circuits Bob postpones aborting until the opening stage, commits to a random string of appropriate length for C_A.
4. Else, he considers the majority output as the correct output. If Bob had a valid proof of cheating pc, he learns $x_A \| pad_A$. He computes $o_A = f_A(x_A, x_B) \oplus pad_A$ on his own. He then chooses a $\{W_{o_{A,j}}^{A,j}\}_{j=1}^{m}$ from the evaluation circuits that is consistent with $GDec^A$ and o_A, and lets $C_A = \mathsf{TCommit}(W_{o_{A,1}}^{A,1}, \ldots, W_{o_{A,m}}^{A,m})$ (with high probability there is at least one). [Note that in case Alice's opening of the check circuits are problematic, Bob will never decommit anyways.]

Bob Commits to Alice's Garbled Output.

1. Bob sends C_A as his commitment to Alice's output labels.
2. Alice sends back $\sigma_{o_A} = Sign(SK_A, (sid, deadline, C_A))$. Bob can use this in case of resolution to prove to the Arbiter that he computed Alice's output honestly.

Alice Opens Everything for Check Circuits.

1. For $i \notin \mathbb{E}$, Alice opens sd_i^A to open all secrets of GC_i. She also opens c_i^B, d_i^B, and the randomness used in committed OTs for Bob's input. Bob checks correctness of opened circuits and their consistency with $\mathsf{GDec}^A, \mathsf{GDec}^B$. He also verifies correctness c_i^B, d_i^B and the opened PadDec_i. He aborts if any of the checks fail.
2. For $i \notin \mathbb{E}'$, Alice opens $sd_i'^A$ to open all secrets of GCD_i. He also reveals the randomness used in committed OTs for Bob's input. Bob checks the correctness of the opened circuits and the OTs, and aborts in case of a fail.

Output Exchange.

1. Bob opens C_A to Alice's output labels. Alice translates these to her actual output out_A using pad_A and the translation table, on her own. In case of a problem, Alice resolves with the Arbiter.
2. Alice opens c_i^B for $i \in \mathbb{E}$. This allows Bob to learn the values Alice used for pad_B, r_p in all evaluated circuits. For each such value he computes $M_p \cdot (pad_B \| r_p)$ and checks if the result is equal to the unique UH output he obtained when evaluating the circuits. He chooses a pad meeting this requirement and uses it to decode his final output out_B. In case of a problem, Bob resolves with the Arbiter.

Figure 7: Optimistic Fair **Malicious** 2PC (cnt'd)

1. Bob sends $\mathsf{GDec}^A, \sigma_{\mathsf{GDec}^A}, C_A, \sigma_{o_A}$ to the Arbiter. He also sends $d_i^B, \sigma_{i,pad}$ for all $i \in \mathbb{E}$ to the Arbiter. He also opens C_A to $W_{o_{A,1}}^{A,j}, \ldots, W_{o_{A,m}}^{A,j}$.
2. The Arbiter verifies the signature, checks that the time is earlier than the deadline in the signature and the session identifiers match. He also makes sure the opened values $W_{o_{A,1}}^{A,j}, \ldots, W_{o_{A,m}}^{A,j}$ are consistent with C_A and GDec^A. Essentially, one output label per pair must be provided. He aborts if any of the checks fail.
3. In case of no fails, the Arbiter decrypts d_i^B for $i \in \mathbb{E}$ and sends $\mathsf{PadDecOpen}_i$ to Bob. He stores $W_{o_{A,1}}^{A,1}, \ldots, W_{o_{A,m}}^{A,m}$ for Alice.
4. Bob checks that $\mathsf{PadDecOpen}_i^B$ is the correct opening for c_i^B, for $i \in \mathbb{E}$, and then uses PadDec_i values to obtain his actual output out_B as in the last step of the main protocol.

Figure 8: Resolution for Bob (for Optimistic Fair **Malicious** 2PC)

1. If Alice contacts the Arbiter *before* the timeout and Bob has *not* contacted the Arbiter yet, the Arbiter tells Alice to come after the timeout.
2. If Alice contacts the Arbiter *after* the timeout and Bob has *not* contacted the Arbiter yet, the protocol is aborted and no party obtains the actual output.
3. Else (Bob already contacted the Arbiter and resolved), the Arbiter sends $W^{A,1}_{o_{A,1}}, \dots, W^{A,m}_{o_{A,m}}$ obtained via Bob's resolution to Alice.
4. Alice translates $W^{A,1}_{o_{A,1}}, \dots, W^{A,m}_{o_{A,m}}$ to her actual outputs on her own.

Figure 9: Resolution for Alice (for Optimistic Fair **Malicious** 2PC)

(b) If the openings are problematic (or were not received in a reasonable message timeout), $\mathcal{S}_B$ waits until the resolution timeout.
   i. During this time, if Bob wants to perform a resolution with the Arbiter, she acts as the Arbiter. If Bob provides correct labels to the Arbiter, then $\mathcal{S}_B$ acts the same as above (*i.e.* contacting $\mathcal{U}_f$ and then preparing and sending $\mathsf{PadDecOpen}_i$ values to Bob according to the correct pad).
   ii. If Bob did not resolve with the Arbiter until the timeout, $\mathcal{S}_B$ sends ABORT to $\mathcal{U}_f$ and aborts. In this case, no party learns the actual output.

Indistinguishability: $\mathcal{S}_B$ is indistinguishable from real Alice, as she acts exactly the same with small exceptions.

- In step 2, the hybrid model allows her to act as the trusted party for the committed oblivious transfer.
- She uses random inputs for Alice in the garbled circuits. But due to security properties of the garbling scheme this remains indistinguishable to Bob.
- $\mathcal{S}_B$ also behaves exactly like the Arbiter when Bob resolves.
  – Note that due to the output-authenticity of the garbling scheme and the fact that H is a second-preimage resistant hash function, Bob cannot forge a valid output label except with negligible probability in L.
  – When the Arbiter decrypts, it may be to a different $\mathsf{PadDecOpen}$, but semantic security of encryption makes sure that this is indistinguishable.
- Lastly, the trapdoor commitment scheme allows $\mathcal{S}_B$ to fake the opening to the correct PadDec after contacting $\mathcal{U}_f$.

Hence, outputs in the two worlds would be identical (*i.e.* either abort or the correct output). Finally, note that the simulator contacts the ideal trusted party *only* when both parties can obtain their outputs in the real world.

Security against *malicious* Alice: For any adversary $\mathcal{A}$ corrupting Alice in the real protocol, we describe a simulator $\mathcal{S}_A$ in the ideal world such that the joint outputs of both parties in the two worlds are indistinguishable.

1. $\mathcal{S}_A$ generates a key pair on behalf of the Arbiter, and provides the public key to $\mathcal{A}$. $\mathcal{S}_A$ obtains the signature public key of $\mathcal{A}$. He further generates the commitment parameters and keeps a trapdoor.
2. $\mathcal{S}_A$ runs $\mathcal{A}$ emulating an honest Bob with a random input x'_B all the way upto the oblivious transfer for Bob's inputs. In these OTs, $\mathcal{S}_A$ plays the role of the ideal trusted party $\mathcal{U}_{COT}$ in the committed OT in order to obtain $\mathcal{A}$'s input to the OT (all labels for Bob's input wires). If $\mathcal{A}$ aborts, $\mathcal{S}_A$ sends ABORT to the ideal trusted party $\mathcal{U}_f$ for f and simulates honest Bob aborting.
3. During committed OTs for cheating detection, $\mathcal{S}_A$ again acts as the ideal trusted party and

learns Alice's input, including x'_A and pad'_A.

4. $\mathcal{S}_A$ continues acting as the honest Bob until he sends the commitment C_A to Alice's output labels. At that point, he commits to random W labels, since he is not sure if the output exchange will complete yet, and hence cannot contact U_f. He then continues as honest Bob.
5. Upto the output exchange phase, $\mathcal{S}_A$ sends ABORT to the ideal trusted party $\mathcal{U}_f$ for f wherever honest Bob would have aborted. If everything went well until the output exchange phase, $\mathcal{S}_A$ is now sure that both parties can obtain the output (since, an honest Bob would send Alice's output back and in the worst case, would have resolved with the Arbiter to obtain his output). Hence sends sends x'_A to $\mathcal{U}_f$ and receives $out'_A = f_A(x'_A, x_B)$. He then computes $o'_A = out'_A \oplus pad'_A$ and picks the output labels $\{W^{A,j}_{o_{A,j}}\}_{j=1}^{m}$ accordingly (using the evaluated ones, since at least one is correct with high probability, and he can check via GDec^A) and sends them to Alice, simulating the opening of C_A with the trapdoor.
6. $\mathcal{S}_A$ then receives the openings for $c_i^B, i \in \mathbb{E}$. If the openings are problematic (or were not received in a reasonable message timeout), $\mathcal{S}_A$ simulates Bob's Arbiter resolution himself, and uses the Arbiter's secret key to decrypt $d_i^B, i \in \mathbb{E}$ to compute Bob's actual output as in the last step of the protocol.
7. If Alice contacts the Arbiter afterward, he sends the correctly computed output labels $\{W^{A,j}_{o_{A,j}}\}_{j=1}^{m}$ to Alice.

Indistinguishability: $\mathcal{S}_A$ is indistinguishable from real Bob, as he behaves exactly the same, except:

(i) if all the evaluated circuits (or the associated commitments/inputs, etc.) are bad and $\mathcal{A}$ is not caught. But this only happens with negligible probability of 2^{-s}.
(ii) if $\mathcal{A}$ manages to find two different inputs that lead to the same output for the UHs. This also only happens with negligible probability of $2^{-s'}$.
(iii) that $\mathcal{S}_A$ first commits to random output labels in C_A and then simulates their opening. But due to the hiding property of the trapdoor commitments, this is indistinguishable.

$\mathcal{S}_A$ also behaves exactly like the Arbiter during resolutions.

5.2 Security against Colluding Arbiter - Malicious Protocol

Theorem 5.2. *If the committed-OT protocol used is secure against malicious adversaries [18], the garbling scheme is secure and provides output authenticity [13], the (labeled) public key encryption scheme employed is semantically secure [67], the hash function used is second-preimage resistant [62], the trapdoor commitment scheme is hiding and binding [23, 52], and the signature scheme is existentially unforgeable [25], then our construction is secure (with abort) against malicious Bob and malicious Alice, even when the Arbiter acts maliciously and colludes with one of them.*

5.2.1 Proof

Note that by security here, we mean the usual unfair definition that allows the adversary to abort after receiving his output but before the honest party receives her output. The corresponding security definition was already given. We prove for Alice and Bob separately, assuming the Arbiter colludes with the malicious one. This means, the simulator is *not* allowed to act on behalf of the Arbiter in its simulation.

Security against *malicious* Bob and colluding Arbiter: For any adversary $\mathcal{B}$ corrupting Bob in the real protocol, we describe a simulator $\mathcal{S}_B$ in the ideal world such that the joint outputs of both parties in the two worlds are indistinguishable.

1. $\mathcal{S}_B$ generates a key pair on behalf of Alice and sends the signature verification key to $\mathcal{B}$. She obtains the Arbiter's public key from $\mathcal{B}$.
2. $\mathcal{S}_B$ picks $x_A^C = pad_B \| r_p \| x'_A \| pad_A \| r_x$ randomly, and emulates honest Alice all the way upto the oblivious transfer for Bob's inputs. In these OTs, $\mathcal{S}_B$ plays the ideal trusted party $\mathcal{U}_{COT}$ in the committed oblivious transfer to receive $\mathcal{B}$'s input x'_B. She continues as an honest Alice would.
3. $\mathcal{S}_B$ sends x'_B to the ideal trusted party $\mathcal{U}_f$ for f, and receives $out'_B = f_B(x_A, x'_B)$ as output.
4. $\mathcal{S}_B$ plays the honest Alice in the coin-toss and learns the evaluation sets $\mathbb{E}$ and $\mathbb{E}'$ as a result. She generates and sends the circuits, commitments, encryptions all as an honest Alice would.
5. Until the output exchange phase, $\mathcal{S}_B$ simulates honest Alice. If $\mathcal{B}$ aborts, $\mathcal{S}_B$ sends ABORT to the ideal trusted party $\mathcal{U}_f$ for f and simulates honest Alice aborting.
6. $\mathcal{S}_B$ finally receives the openings for C_A.
   (a) If the openings are proper, $\mathcal{S}_B$ then responds to Bob with the openings of c_i^B for $i \in \mathbb{E}$. Note that these were already prepared according to the output of Bob, so there is no problem. She sends CONTINUE to U_f.
   (b) If the openings are problematic (or were not received within message timeout), $\mathcal{S}_B$ waits until the resolution timeout and then contacts the Arbiter. If the resolution is successful, she sends CONTINUE to U_f, else she sends ABORT to U_f and halts.

Indistinguishability: $\mathcal{S}_B$ is indistinguishable from real Alice, as she acts exactly the same with one exception: She uses random inputs for Alice in the garbled circuits. But due to security properties of the garbling scheme this remains indistinguishable to Bob. Hence, outputs in the two worlds would be identical (*i.e.* either abort or the correct output).

Security against *malicious* Alice and colluding Arbiter: For any adversary $\mathcal{A}$ corrupting Alice in the real protocol, we describe a simulator $\mathcal{S}_A$ in the ideal world such that the joint outputs of both parties in the two worlds are indistinguishable.

1. $\mathcal{S}_A$ obtains the signature public key and the Arbiter's public key from $\mathcal{A}$. He does not need the commitment trapdoor.
2. $\mathcal{S}_A$ runs $\mathcal{A}$ emulating an honest Bob with a random input x'_B all the way upto the oblivious transfer for Bob's inputs. In these OTs, $\mathcal{S}_A$ plays the role of the ideal trusted party $\mathcal{U}_{COT}$ in the committed OT in order to obtain $\mathcal{A}$'s input to the OT (all labels for Bob's input wires). If $\mathcal{A}$ aborts, $\mathcal{S}_A$ sends ABORT to the ideal trusted party $\mathcal{U}_f$ for f and simulates honest Bob aborting.
3. During committed OTs for cheating detection, $\mathcal{S}_A$ again acts as the ideal trusted party and learns Alice's input, including x'_A and pad'_A. He sends x'_A to $\mathcal{U}_f$ and receives $out'_A = f_A(x'_A, x_B)$ back.
4. $\mathcal{S}_A$ continues acting as the honest Bob until he sends the commitment C_A to Alice's output labels. At that point, he computes $o'_A = out'_A \oplus pad'_A$ and picks the output labels $\{W^{A,j}_{o_{A,j}}\}_{j=1}^{m}$ accordingly (using the evaluated ones, since at least one is correct with high probability, and he can check via GDec^A). He prepares the commitment C_A accordingly, and continues as honest Bob.
5. Upto the output exchange phase, $\mathcal{S}_A$ sends ABORT to the ideal trusted party $\mathcal{U}_f$ for f wherever honest Bob would have aborted. He opens C_A.

6. $\mathcal{S}_A$ then receives the openings for $c_i^B, i \in \mathbb{E}$. If the openings are problematic (or were not received in a reasonable amount of time –*e.g.* a few round-trip message delays–), he tries to resolve with the Arbiter. If he successfully obtains the correct openings of $c_i^B, i \in \mathbb{E}$ via the decryptions of $d_i^B, i \in \mathbb{E}$, he sends CONTINUE to U_f. Else, he sends ABORT to U_f.

Indistinguishability: $\mathcal{S}_A$ is indistinguishable from real Bob, as he behaves exactly the same, except:

(i) if all the evaluated circuits (or the associated commitments/inputs, etc.) are bad and $\mathcal{A}$ is not caught. But this only happens with negligible probability of 2^{-s}.

(ii) if $\mathcal{A}$ manages to find two different inputs that lead to the same output for the UHs. This also only happens with negligible probability of $2^{-s'}$.

Acknowledgements

We thank TÜBİTAK, the Scientific and Technological Research Council of Turkey, project 111E019, and European Union COST Action IC1306.

References

[1] M. Andrychowicz, S. Dziembowski, D. Malinowski, and Ł. Mazurek. Fair two-party computations via bitcoin deposits. In *FC*, 2014.

[2] M. Andrychowicz, S. Dziembowski, D. Malinowski, and L. Mazurek. Secure multiparty computations on bitcoin. In *IEEE Security and Privacy*, 2014.

[3] G. Asharov. Towards characterizing complete fairness in secure two-party computation. In *TCC*, 2014.

[4] G. Asharov and C. Orlandi. Calling out cheaters: Covert security with public verifiability. In X. Wang and K. Sako, editors, *ASIACRYPT 2012*, volume 7658 of *LNCS*, pages 681–698. Springer, Dec. 2012.

[5] N. Asokan, V. Shoup, and M. Waidner. Optimistic fair exchange of digital signatures. *IEEE Selected Areas in Communications*, 18:591–610, 2000.

[6] G. Ateniese. Efficient verifiable encryption (and fair exchange) of digital signatures. In *ACM CCS*, 1999.

[7] Y. Aumann and Y. Lindell. Security against covert adversaries: Efficient protocols for realistic adversaries. *Journal of Cryptology*, 23:281–343, 2010.

[8] G. Avoine and S. Vaudenay. Optimistic fair exchange based on publicly verifiable secret sharing. In *ACISP*, 2004.

[9] F. Bao, R. Deng, and W. Mao. Efficient and practical fair exchange protocols with off-line ttp. In *IEEE Security and Privacy*, 1998.

[10] A. Beimel, Y. Lindell, E. Omri, and I. Orlov. $1/p$-Secure multiparty computation without honest majority and the best of both worlds. In P. Rogaway, editor, *CRYPTO 2011*, volume 6841 of *LNCS*, pages 277–296. Springer, Aug. 2011.

[11] A. Beimel, E. Omri, and I. Orlov. Protocols for multiparty coin toss with dishonest majority. In T. Rabin, editor, *CRYPTO 2010*, volume 6223 of *LNCS*, pages 538–557. Springer, Aug. 2010.

[12] 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.

[13] M. Bellare, V. T. Hoang, and P. Rogaway. Foundations of garbled circuits. In T. Yu, G. Danezis, and V. D. Gligor, editors, *ACM CCS 12*, pages 784–796. ACM Press, Oct. 2012.

[14] M. Ben-Or, O. Goldreich, S. Micali, and R. L. Rivest. A fair protocol for signing contracts. *IEEE Transactions on Information Theory*, 36:40–46, 1990.

[15] I. Bentov and R. Kumaresan. How to use bitcoin to design fair protocols. In *CRYPTO*, 2014.

[16] F. Boudot, B. Schoenmakers, and J. Traoré. A fair and efficient solution to the socialist millionaires' problem. *Discrete Applied Mathematics*, 111(1-2):23–36, 2001.

[17] L. T. Brandão. Secure two-party computation with reusable bit-commitments, via a cut-and-choose with forge-and-lose technique. In *ASIACRYPT*, 2013.

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

[19] R. Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *FOCS*, 2001.

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

[21] Y. Dodis, P. J. Lee, and D. H. Yum. Optimistic fair exchange in a multi-user setting. In *PKC*, 2007.

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

[23] M. Fischlin. *Trapdoor Commitment Schemes and Their Applications.* PhD thesis, Goethe University Frankfurt, 2001.

[24] T. K. Frederiksen, T. P. Jakobsen, J. B. Nielsen, P. S. Nordholt, and C. Orlandi. Minilego: Efficient secure two-party computation from general assumptions. In *EUROCRYPT*, 2013.

[25] 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.

[26] S. Gordon and J. Katz. Partial fairness in secure two-party computation. *Journal of Cryptology*, 25(1):14–40, 2012.

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

[28] S. D. Gordon and J. Katz. Partial fairness in secure two-party computation. In H. Gilbert, editor, *EUROCRYPT 2010*, volume 6110 of *LNCS*, pages 157–176. Springer, May 2010.

[29] C. Hazay and Y. Lindell. *Efficient secure two-party protocols: Techniques and constructions.* Springer Science & Business Media, 2010.

[30] Y. Huang, J. Katz, and D. Evans. Efficient secure two-party computation using symmetric cut-and-choose. In *CRYPTO*, 2013.

[31] J. Katz. On achieving the best of both worlds in secure multiparty computation. In *STOC*, 2007.

[32] J. Katz, U. Maurer, B. Tackmann, and V. Zikas. Universally composable synchronous computation. In *TCC*, 2013.

[33] A. Kiayias, H.-S. Zhou, and V. Zikas. Fair and robust multi-party computation using a global transaction ledger. *Cryptology ePrint Archive, Report 2015/574*, 2015.

[34] H. Kılınç and A. Küpçü. Optimally efficient multi-party fair exchange and fair secure multi-party computation. In *CT-RSA*, 2015.

[35] H. Kılınç and A. Küpçü. Efficiently making secure two-party computation fair. In *FC*, 2016.

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

[37] M. S. Kiraz, B. Schoenmakers, and J. Villegas. Efficient committed oblivious transfer of bit strings. In J. A. Garay, A. K. Lenstra, M. Mambo, and R. Peralta, editors, *ISC 2007*, volume 4779 of *LNCS*, pages 130–144. Springer, Oct. 2007.

[38] V. Kolesnikov, P. Mohassel, and M. Rosulek. Flexor: Flexible garbling for XOR gates that beats free-XOR. In J. A. Garay and R. Gennaro, editors, *CRYPTO (2)*, volume 8617 of *Lecture Notes in Computer Science*, pages 440–457. Springer, 2014.

[39] V. Kolesnikov and T. Schneider. Improved garbled circuit: Free xor gates and applications. In *Automata, Languages and Programming*, pages 486–498. Springer, 2008.

[40] A. Kosba, A. Miller, E. Shi, Z. Wen, and C. Papamanthou. Hawk: The blockchain model of cryptography and privacy-preserving smart contracts. *Cryptology ePrint Archive, Report 2015/675*, 2015.

[41] A. Küpçü. *Efficient Cryptography for the Next Generation Secure Cloud.* PhD thesis, Brown University, 2010.

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

[43] A. Küpçü. Distributing trusted third parties. *ACM SIGACT News Distributed Computing Column*, 44:92–112, 2013.

[44] A. Küpçü and A. Lysyanskaya. Optimistic fair exchange with multiple arbiters. In *ESORICS*, 2010.

[45] A. Küpçü and A. Lysyanskaya. Usable optimistic fair exchange. In *CT-RSA*, 2010.

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

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

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

[49] Y. Lindell and B. Pinkas. A proof of yaos protocol for secure two-party computation. *ECCC*, 11:2004, 2004.

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

[51] Y. Lindell and B. Pinkas. Secure two-party computation via cut-and-choose oblivious transfer. In Y. Ishai, editor, *TCC*, volume 6597 of *Lecture Notes in Computer Science*, pages 329–346. Springer, 2011.

[52] P. MacKenzie and K. Yang. On simulation-sound trapdoor commitments. In *EUROCRYPT*, 2004.

[53] S. Micali. Simple and fast optimistic protocols for fair electronic exchange. In *PODC*, 2003.

[54] P. Mohassel and M. Franklin. Efficient polynomial operations in the shared-coefficients setting. In M. Yung, Y. Dodis, A. Kiayias, and T. Malkin, editors, *PKC 2006*, volume 3958 of *LNCS*, pages 44–57. Springer, Apr. 2006.

[55] P. Mohassel and M. K. Franklin. Efficiency tradeoffs for malicious two-party computation. In M. Yung, Y. Dodis, A. Kiayias, and T. Malkin, editors, *Public Key Cryptography*, volume 3958 of *Lecture Notes in Computer Science*, pages 458–473. Springer, 2006.

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

[57] T. Moran, M. Naor, and G. Segev. An optimally fair coin toss. In O. Reingold, editor, *TCC 2009*, volume 5444 of *LNCS*, pages 1–18. Springer, Mar. 2009.

[58] J. B. Nielsen, P. S. Nordholt, C. Orlandi, and S. S. Burra. A new approach to practical active-secure two-party computation. In *Advances in Cryptology–CRYPTO 2012*, pages 681–700. Springer, 2012.

[59] T. P. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In *CRYPTO*, 1991.

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

[61] M. O. Rabin. How to exchange secrets with oblivious transfer. *IACR Cryptology ePrint Archive*, 2005:187, 2005.

[62] P. Rogaway and T. Shrimpton. Cryptographic hash-function basics: Definitions, implications, and separations for preimage resistance, second-preimage resistance, and collision resistance. In *FSE*, 2004.

[63] 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.

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

[65] A. Shelat and C.-H. Shen. Two-output secure computation with malicious adversaries. In K. G. Paterson, editor, *EUROCRYPT 2011*, volume 6632 of *LNCS*, pages 386–405. Springer, May 2011.

[66] A. Shelat and C.-H. Shen. Fast two-party secure computation with minimal assumptions. In A.-R. Sadeghi, V. D. Gligor, and M. Yung, editors, *ACM CCS 13*, pages 523–534. ACM Press, Nov. 2013.

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

[68] A. C.-C. Yao. How to generate and exchange secrets. In *FOCS*, 1986.

[69] S. Zahur, M. Rosulek, and D. Evans. Two halves make a whole. In *Advances in Cryptology-EUROCRYPT 2015*, pages 220–250. Springer, 2015.