# Garbled Circuits Checking Garbled Circuits: More Efficient and Secure Two-Party Computation

Payman Mohassel* Ben Riva†

June 20, 2013

**Abstract**

Applying cut-and-choose techniques to Yao's garbled circuit protocol has been a promising approach for designing efficient Two-Party Computation (2PC) with malicious and covert security, as is evident from various optimizations and software implementations in the recent years. We revisit the security and efficiency properties of this popular approach and propose alternative constructions and a new definition that are more suitable for use in practice.

- We design an efficient fully-secure 2PC protocol for two-output functions that only requires $O(t|C|)$ symmetric-key operations (with small constant factors, and ignoring factors that are independent of the circuit in use) in the Random Oracle Model, where $|C|$ is the circuit size and $t$ is a statistical security parameter. This is essentially the *optimal* complexity for protocols based on cut-and-choose, resolving a main question left open by the previous work on the subject.

  Our protocol utilizes novel techniques for enforcing *garbler's input consistency* and handling *two-output functions* that are more efficient than all prior solutions.
- Motivated by the goal of eliminating the *all-or-nothing* nature of 2PC with covert security (that privacy and correctness are fully compromised if the adversary is not caught in the challenge phase), we propose a new security definition for 2PC that strengthens the guarantees provided by the standard covert model, and offers a smoother security vs. efficiency tradeoff to protocol designers in choosing the *right deterrence factor*. In our new notion, correctness is always guaranteed, privacy is fully guaranteed with probability $(1 - \epsilon)$, and with probability $\epsilon$ (i.e. the event of undetected cheating), privacy is only "partially compromised" with at most a *single bit* of information leaked, in *case of an abort*.

  We present two efficient 2PC constructions achieving our new notion. Both protocols are competitive with the previous covert 2PC protocols based on cut-and-choose.

A distinct feature of the techniques we use in all our constructions is to check consistency of inputs and outputs using new gadgets that are themselves *garbled circuits*, and to verify validity of these gadgets using *multi-stage* cut-and-choose openings.

## 1 Introduction

Informally, a secure two-party protocol for a known function $f(\cdot, \cdot)$ is a protocol between Alice and Bob with private inputs $x$ and $y$ that satisfies the following two requirements: (1) *Correctness*: If at least one of the players is honest then the result should be the correct output of $f(x, y)$; (2) *Privacy*: No player learns any information about the other player's input, except for the function output.

Security is defined with respect to an adversary, who is *semi-honest* if the corrupted players always follow the protocol, is *malicious* if the players can arbitrarily deviate, and is *covert* in case a cheating player has an incentive not to be caught (or more specifically, any deviation can be detected with a constant probability).

*University of Calgary, Canada.

†Tel Aviv University, Israel. Research supported by the Check Point Institute for Information Security and an ISF grant.

A classical solution for the case of semi-honest players (i.e., players who do not deviate from the protocol) is to use a *garbled circuit* and *oblivious transfer* [28, 18]: The resulting protocol is fairly efficient since computing each gate requires a constant number of symmetric-key encryptions. Furthermore, recent results show how to improve both the computation and communication cost of the garbling process (e.g., getting XOR gates for free [14], reducing communication [5, 25], and designing tailored circuits [6]).

The case of malicious players is more complicated and less efficient. A classical solution is to use zero-knowledge proofs to verify that the players follow the protocol. However, the proofs in this case are rather inefficient. [11, 22] show how to garble a circuit in such a way that these proofs can be instantiated more efficiently. Still, these constructions require a constant number of exponentiations per gate, making them inefficient for large circuits.

See Appendix A for other approaches we do not discuss here.

THE CUT-AND-CHOOSE APPROACH. A slightly more explored direction is based on using the cut-and-choose method for checking the garbled circuit. (E.g., see implementations by [25, 26, 15].) Instead of sending only one (and possibly not properly constructed) garbled circuit, Alice sends $t$ garbled circuits. Then, Bob asks her to *open* a constant fraction of them. For those circuits, Alice sends all the randomness she used in the garbling process. Bob can check that the opened circuits were indeed correctly garbled. If that is not the case, Bob knows that Alice has cheated and aborts. Otherwise, Bob evaluates the remaining garbled circuits and computes the majority output. It is shown in [19, 26] that with high probability the majority of the evaluated garbled circuits are properly constructed.

However, the above cut-and-choose of the circuits is not sufficient to obtain a fully-secure 2PC. There are three well-known issues to resolve: (1) *Garbler's input consistency:* Since Bob evaluates many circuits, he needs assurance that Alice uses the same input in all of them. (2) *Evaluator's input consistency:* Alice can use different input labels in the oblivious transfers and in creation of the garbled circuits, in such a way that reveals Bob's input. (E.g., she can use invalid labels for the input bit 0 in the oblivious transfer, but valid ones for 1, causing Bob to abort if his input bit is 0.) (3) *Two-output functions:* There are cases in which the players want to securely compute two different functions $f_1, f_2$ where each party only learns his *own* output and is assured he has obtained the correct result.

When addressing these issues, the deciding efficiency factors are both the number and the type of additional cryptographic operations required. By *expensive operations*, we refer to cryptographic primitives that require exponentiations (e.g. oblivious transfer, or public-key encryption), and by *inexpensive operations* we mean the use of primitives that do not require exponentiations (e.g. symmetric-key encryption, commitments, or hashing). To simplify the exposition, from now on we omit small constants and complexities that are independent of the computation size or input length, unless said otherwise.

To address the first issue, i.e. how to make sure Alice is using the same input in all circuits, [20, 17] present two methods that require $O(t^2 \cdot n_1)$ inexpensive cryptographic operations (commitments), where $n_1$ is the length of Alice's input, and $t$ is the number of circuits we use in the cut-and-choose. ([27] shows how to reduce this asymptotic overhead, but with large constants even for small security parameters.) [20, 19, 26] show alternative methods that require $O(t \cdot n_1)$ expensive cryptographic operations (i.e. exponentiations). These consistency-checking mechanisms can lead to significant overhead. Recall that garbling of a single gate requires a constant number of symmetric encryptions, where the constant is $4$ in most implementations. Thus, e.g. for $t = 130$, the price of checking consistency for a single input bit is roughly equivalent to the price of garbling several tens of additional gates in each circuit in the first method, and even more in the second. Moreover, the first method has a large communication overhead (e.g., for input size $n_1 = 500$ and $t = 130$, it requires several millions of commitments, with a total communication overhead of hundreds of megabytes).

To address the second issue, i.e. making sure Alice is using the same labels in her OT answers and the garbled circuits, [17] presents a method that requires $O(t \cdot \max(4n_2, 8t))$ expensive cryptographic operations (specifically, oblivious transfers), where $n_2$ is the length of Bob's input. [19, 26] introduce alternative methods that require $O(t \cdot n_2)$ expensive cryptographic operations.

To address the last issue, of verifying the computation output, [17] proposes to apply a *one time MAC* to

the output and XOR the result with a random input to hide the outcome (both are done as part of the circuit). However, this solution increases Alice's input with additional $q_1 + 2t$ input bits and increases the circuit size by $O(t \cdot q_1)$ gates, where $q_1$ is Alice's output length (i.e. overall overhead of $O(t^2 \cdot q_1)$ inexpensive operations). [26] suggests a solution that requires the use of digital signatures and a witness-indistinguishable proof, resulting in a total overhead of $O(t \cdot q_1)$ expensive operations.

In the covert setting [1] the techniques are similar, although the issue of the garbler's input consistency is not always relevant [5, 1].

ALL-OR-NOTHING SECURITY VS. SECURITY WITH INPUT-DEPENDENT ABORT. All the cut-and-choose protocols discussed above provide an *all-or-nothing* guarantee, which means that both correctness and privacy are preserved with the same probability (the probability of getting caught in case of cheating), and are completely compromised if cheating is not detected. For example, in case of a protocol with covert security and deterrence factor of $1/2$, there is a $50\%$ chance that the protocol reveals the honest party's input and provides him with an incorrect output. This can become an obstacle to using covert security, in some practical scenarios. For example, the participants of an MPC protocol may not be able to afford the lack of correctness or privacy (even if only with a constant probability), due to the high financial/legal cost, or the loss of reputation.

[20] suggests an alternative to the all-or-nothing approach and designs a secure two-party protocol that always guarantees correctness but may leak one bit of information to a malicious party. While this security guarantee is weaker than the standard definition of security against covert/malicious adversaries, it ensures correctness and "partial privacy" even in case of successful cheating, making it a reasonable relaxation in some scenarios.

The idea behind the protocols of [20] is as follows: Alice garbles a circuit $gc_1$ and sends it to Bob, along with the labels of Alice's input-wires. They execute a fully-secure oblivious transfer protocol in which Bob learns the labels for his input-wires. Then, they run the same steps in the other direction, where Bob garbles $gc_2$ and Alice is the receiver. Next, each player evaluates the garbled circuit he or she received, resulting in output-wire label $out_i$ (we require that the output-wire labels are the actual outputs concatenated with random labels). Last, each player computes the *supposed to be* concatenation $out_1 \circ out_2$. (Alice gets $out_1$ from her evaluation, and can determine the value of $out_2$ by herself. Bob does the same.) Now they run a protocol for securely testing whether their values $out_1 \circ out_2$ are the same. If they are indeed the same, they output $b$. Otherwise, they abort.

The resulting protocol is highly efficient, requiring only two garbled circuits and the associated oblivious transfers. (See [7] for an optimized variant of the protocol and its performance.) Since one of the players is honest, the result from his garbled circuit will be correct. Thus, if the honest party does not abort, the output is indeed correct. On the other hand, if one of the players is malicious, he can *always* learn one bit of information by observing whether the honest party aborts or not in the final equality test. We call this scenario *Input-Dependent Abort* (IDA) (following [9]).

## 1.1 Our Contributions

Given the discussion above, we put forth and answer the following two questions: (1) Can we improve on the efficiency of the existing solutions for checking input-consistency and handling two-output functions, to the extent that they are no longer considered a major computation/communication overhead? (2) Can we design cut-and-choose protocols that do not suffer from the all-or-nothing limitation of standard constructions but that provide better security guarantees than those of 2PC with input-dependent abort?

In the process of answering these questions, we introduce a set of new techniques to enforce consistency of inputs and outputs in garbled circuits. Interestingly, these techniques themselves employ *specially-designed garbled circuits* (gadgets) correctness of which is checked as part of a modified cut-and-choose process containing multiple opening stages.

| | $P_1$'s input | $P_2$'s input | Two-output Overhead |
|---|---|---|---|
| [17] | $\mathsf{inexpensive}(t^2 n_1)$ | $\mathsf{expensive}(\max(4n_2, 8t))+$ $\mathsf{inexpensive}(t \cdot \max(4n_2, 8t))$ | $\mathsf{inexpensive}(t^2 q_1)$ |
| [19, 26] | $\mathsf{expensive}(tn_1)$ | $\mathsf{expensive}(tn_2)$ | $\mathsf{expensive}(tq_1)$ |
| Our protocol | $\mathsf{inexpensive}(tn_1)$ | $\mathsf{inexpensive}(t \cdot \max(4n_2, 8t))$ | $\mathsf{inexpensive}(tq_1)$ |

Table 1: Comparison of different fully secure 2PC protocols. $n_i$ is the length of $P_i$'s input, $q_1$ is the length of $P_1$'s output, and $t$ is a statistical security parameter (where $t$ garbled circuits are used in the cut-and-choose). The number of base OTs in the OT extension is omitted as it is independent of the circuit and input sizes.

### 1.1.1 Fully-Secure 2PC Based on Cut-and-Choose with Small Overheads.

Towards answering the first question, we propose new and efficient solutions for the three problems of (1) Garbler's input consistency (2) Evaluator's input consistency and (3) Handling two-output functions, that asymptotically and concretely improve on all previous solutions.

First, we show how to use garbled *XOR-gates* to efficiently enforce the garbler's input consistency, while requiring only $O(t \cdot n_1)$ inexpensive operations. This approach asymptotically improves the solutions in [20, 17], and only requires inexpensive operations in contrast to the solution of [26]. Second, we observe that the solution of [17] to the evaluator's input consistency issue can be improved by combining it with the OT extension of [21] and the Free-XOR technique of [14]. The resulting protocol requires only $O(t \cdot \max(4n_2, 8t))$ inexpensive operations. Third, we show how to use garbled *identity-gates* to efficiently solve the two-output function problem, while requiring only $O(t \cdot q_1)$ inexpensive operations, where $q_1$ is the garbler's output length, improving on the recent construction of [26] which requires the same number of expensive operations. The resulting 2PC protocol is constant round and asymptotically better than all previous constructions based on the cut-and-choose method [20, 17, 19, 26] (except for [27], which is impractical due to large constants). In Table 1, we compare the protocol's complexity with previous constructions. We stress that the efficiency of our protocol relies on the efficient OT extension of [21], which allows one to efficiently extend a small number of OTs to $n$ OTs with the price of only $O(n)$ invocations of a hash function. The protocol of [21] is in the Random Oracle Model (ROM) and our construction inherits the same weakness. (Besides using ROM for the OT-extension of [21], in some of our techniques we show two alternatives: A more efficient instantiation in the ROM, and one without the ROM requirement, which still is more efficient than current techniques.)

We remark that our proposed solutions can be modified to work with any of the existing garbled-circuit optimization techniques of [14, 5, 25, 6, 15].

Furthermore, in Appendix E we describe how to use our techniques to construct a fully-secure 2PC protocol for the case where $y$ is not private, using only a single garbled circuit. This scenario which we call *authenticated computation with private inputs* naturally arises in applications such as anonymous credentials or targeted advertising.

Our main contributions are the new techniques we use for solving the Garbler's input consistency issue and handling two-output functions. Next, to give a flavor of our techniques, we present the ideas behind our solutions.

MULTI-STAGE CUT-AND-CHOOSE AND HANDLING TWO-OUTPUT FUNCTIONS. From now on we denote by $P_1$ the garbler (Alice), and by $P_2$ the evaluator (Bob). Note that the main difficulty here is to convince the garbler, $P_1$, that the output he receives is correct. (Privacy of the output is easily achieved by xoring the output with a random string.)

A common method for authenticating the output of a garbled circuit is to send the random labels resulted from the evaluation of the garbled circuit. However, when we use the cut-and-choose method, many circuits are being evaluated, and sending the labels for all the garbled circuits can leak secret information (e.g., $P_1$ can create a single bad circuit that simply outputs $P_2$'s input, and not get caught with high probability). We can fix this issue by using the same output-wire labels in all the garbled circuits, but then we would lose our authenticity

guarantee since $P_2$ learns all the output-wire labels from the opened circuits and can use that information to tamper with the output of the evaluated circuits.

We propose a workaround that allows us to simultaneously use the same output-wire labels in all circuits, and preserve the authenticity guarantee, in cut-and-choose 2PC. We separate the "cut" step from the "opening" step (this is a recurring idea in all our constructions). After $P_1$ sends the $t$ garbled circuits, $P_2$ picks a random subset $S$ which he wants to check and sends it to $P_1$. Then, instead of opening the garbled circuits in $S$, they proceed to the evaluation of the rest of the garbled circuits. I.e., $P_1$ sends the labels of his input-wires for the garbled circuits not in $S$; $P_2$ evaluates all of them and takes the majority; he then commits to the output along with the corresponding output-wire labels. (Note that since the opening step is not performed yet, $P_2$ cannot guess the unknown output-wire labels and commit to the wrong output). Now, they complete the cut-and-choose and do the opening step: $P_1$ sends the randomness he used for all the garbled circuits in $S$, and $P_2$ verifies that everything was done correctly. If so, $P_2$ decommits the output and reveals to $P_1$ the actual output and its output-wire labels. To summarize, since $P_1$ learns the output only after $P_2$ has verified the garbled circuits, he cannot cheat in this new cut-and-choose strategy, any differently than he could in regular cut-and-choose. On the other hand, since $P_2$ is committed to his output before the opening, he cannot change the output after he sees the opened circuits.[1]

The above solution can be applied to most previous 2PC protocols based on cut-and-choose to obtain their two-output variants. But, since the circuit checking is done after the circuit evaluation, the above solution falls short when combined with circuit streaming or parallelized garbling techniques [6, 15].

In Appendix C.2 we describe a second variant of this protocol that is compatible with those techniques. The cost of this variant is only additional $t \cdot q_1$ commitments.

XOR-GADGETS AND GARBLER'S INPUT CONSISTENCY. Here, our goal is to make sure $P_1$ uses the same input in all (or at least most of) the evaluated garbled circuits. Observe that we do not have the same issue with $P_2$'s input since for each specific input bit, $P_2$ learns the $t$ corresponding input-wire labels using a single OT. But, since $P_1$ does not use OT to learn the labels for his input-wires, the same approach does not work here.

First, we augment the circuit $C$ being computed with a small circuit we call an *XOR-gadget*. Say we want to compute the circuit $C(x, y)$ where $x$ is $P_1$'s input, and $y$ is $P_2$'s. Instead of working with $C$, the players work with a circuit that computes $C_1(x, y, r) = (C(x, y), x \oplus r)$, where $r$ is a random input string of length $|x|$ generated by $P_1$. Note that $x$ is kept private from $P_2$ if $r$ is chosen randomly. Denote $P_1$'s inputs to the $t$ garbled circuits of $C_1$ by $x_1^1, x_2^1, \ldots, x_t^1$ and $r_1^1, r_2^1, \ldots, r_t^1$. If $P_1$ is honest, the $r_i^1$-s are chosen independently at random while all the $x_i^1$-s are equal to $x$.

Let $C_2(x, r) = x \oplus r$, where $x$ and $r$ are $P_1$'s inputs of the same length. (Note that $y$ is not an input here.) In addition to $P_1$'s garbled circuits, $P_2$ also generates $t$ XOR-gadgets, which are garbled circuits of $C_2$. These garbled XOR-gadgets will be evaluated by $P_1$ and on his own inputs. (For simplicity, we assume for now that $P_2$ is semi-honest.) Denote $P_1$'s inputs to these $t$ garbled circuits by $x_1^2, x_2^2, \ldots, x_t^2$ and $r_1^2, r_2^2, \ldots, r_t^2$. If $P_1$ is honest, then $r_i^1 = r_i^2$ for all $i$, and all the $x_i^2$-s are equal to $P_1$'s actual input $x$.

We enforce that $x_i^1$-s are the same in the majority of the evaluated circuits, using a combination of *three different checks*: (1) Check that $P_1$ uses the same value $x'$ for all $x_i^2$-s. We can easily enforce this since $P_1$ learns the input-wire label for each bit using a single OT. (E.g., if the first bit of $x'$ is zero, $P_1$ will learn $t$ concatenated labels that correspond to the bit zero in the $t$ XOR-gadgets $P_2$ prepared.) (2) Check that $(x_i^2 + r_i^2) = (x_i^1 + r_i^1)$ in all the evaluated circuits. We enforce this by evaluating the two XOR-gadgets corresponding to the $i$-th garbled circuit (one created by $P_1$ and one created by $P_2$), and checking the equality of their outputs (see Section 3 for subtleties that need to be addressed when doing so). (3) Check that $r_i^1 = r_i^2$ in the majority of the evaluated

[1] We note that the above solution is not enough. First, the commitment in use must be non-malleable with respect to the garbled circuits being opened. E.g., consider a garbling scheme that outputs also commitments of the possible output-wire labels; $P_2$ could use one of those commitments as his commitment and later use the information he learned from the opening to decommit successfully. Second, the commitment has to be equivocal to allow us to later simulate $P_2$'s message. Both requirements can be solved in the plain model by using trapdoor commitments [3] and efficient Zero-Knowledge Proof of Knowledge (ZKPoK), or in the Random Oracle Model, by committing using a hash function. The first solution requires $O(q_1)$ expensive operations while the second requires only one call to the hash function.

circuits. We enforce this as part of the cut-and-choose: When $P_1$ sends his garbled circuits, he also sends the labels that correspond to all $r_i^1$-s. After $P_1$ learns the labels for $r_i^2$-s (from the OTs), they do the opening phase and $P_1$ opens a subset of the garbled circuits. In addition, for each opened circuit, $P_1$ reveals the labels of the $r_i^2$-s he learned, and $P_2$ verifies that $r_i^1 = r_i^2$. (Note that once $P_1$ sends the labels of $r_i^1$ and the garbled circuit, he cannot change $r_i^1$. On the other hand, $P_1$ cannot fake a valid label for $r_i^2$ that is different from the one he learned in the OTs.) As a result, $P_2$ knows that with high probability (in terms of $t$) $r_i^1 = r_i^2$ in the majority of the evaluated circuits.

It is easy to see that the above three checks imply (with high probability) that $x_i^1$-s are the same in the majority of the evaluated circuits. Since $P_2$ outputs the majority result, this is sufficient for our needs.

Figure 1 shows an example of the above technique for the circuit that computes AND and $t = 2$.

We stress that the above-mentioned is only part of our techniques, and in particular, does not guarantee protection against a malicious $P_2$.

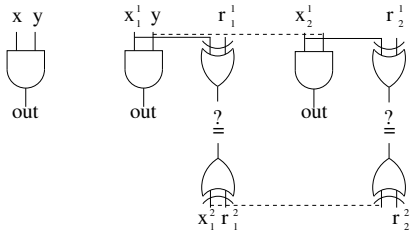


Figure 1: Example of garbling the simple AND circuit on the left that computes the AND between $P_1$'s bit $x$ and $P_2$'s bit $y$. $P_1$ garbles the upper circuits and $P_2$ the lower ones. Specifically, $P_1$ garbles two AND circuits (i.e., $t = 2$) and two XOR-gates, and $P_2$ garbles two XOR-gates. $P_2$'s input is the same for all garbled circuits because of the OT (the top dashed line). Recall that the first input $P_1$ learns in all of $P_2$'s XOR-gates is the same since $P_1$ learns the corresponding input-wire labels from the OT (the lower dashed line). Also, that the equality of $r_i^1$ and $r_i^2$, $i = 1, 2$, is checked in the cut-and-choose (e.g., by $P_1$ revealing the labels of $r_1^1$ and $r_1^2$ if $P_2$ picked to check the first set) and hence holds with high probability. Combining these two observations with the fact that $P_2$ compares the outputs of the XOR-gates, $P_2$ gets the assurance that $x_1^1 = x_2^1$.

### 1.1.2 Security with Input-Dependent Abort in Presence of Covert Adversaries.

We propose a new security notion that naturally combines security with input-dependent abort of [9] (alternatively, security with limited leakage of [20, 7]), with security against covert adversaries [1]. The resulting security guarantee, denoted by $\epsilon$-CovIDA, is a *strict strengthening of covert security*: In covert security, with probability $\epsilon$ both correctness and privacy are gone! Our definition always guarantees correctness, and with probability $\epsilon$, privacy is only "slightly compromised", i.e. only a *single* bit of information may be leaked in case of an abort.

We stress that simply combining the protocols of [20, 7] with the cut-and-choose method is *not* secure under our definition. Say that instead of garbling a single circuit, each player $P_i$ garbles $t$ circuits $gc_1^i, \ldots, gc_t^i$ and sends them to the other player. Players pick a random value $e \in [t]$, *open* all the circuits $gc_{j\neq e}^i$ (i.e., reveal the randomness used to generate them), and verify that they were constructed properly. This assures that with probability $1 - 1/t$, the remaining two circuits (one circuit from each player) is properly constructed. Parties then engage in the dual-execution protocol discussed above using these two garbled circuits. Although this protocol guarantees correctness similar to [20, 7], it does not satisfy our security definition. One problem is that a malicious player can use different inputs for the two evaluated circuits, and learn whether their outputs are the same or not based on the final outcome. This attack is successful even if *all* the circuits are constructed correctly.

We show two constructions that do achieve our definition. Both constructions require a constant number of rounds. In our first construction, each player garbles only $\frac{1}{\epsilon}$ circuits and $\frac{n+2q}{\epsilon}$ additional XOR gates, where $n$ is

the length of the input and $q$ is the length of the output. We emphasize that compared to the protocols of [20, 7], where the adversary can *always* learn one bit of information, our protocol leaks one bit only with probability $\epsilon$.

The first construction is sufficient for large values of $\epsilon$ but fails to scale for the smaller ones. For example, if one aims for a probability of leakage of less than $2^{-10}$, the first protocol would require the exchange of a thousand garbled circuits. A more desirable goal is a protocol with a cost that grows only logarithmically in $\frac{1}{\epsilon}$. We achieve this in our second protocol.

The costs of both constructions are roughly the costs of running their covert counterparts in both directions. E.g. the second protocol requires $O(2\log(\frac{1}{\epsilon})(|C| + n + q))$ inexpensive operations and $O(\log(\frac{1}{\epsilon})(n + q))$ expensive ones, while the covert protocol of [19] requires $O(\log(\frac{1}{\epsilon})|C|)$ inexpensive operations and $O(\log(\frac{1}{\epsilon})n)$ expensive ones.

DIFFICULTIES AND OUR TECHNIQUES. Both protocols use techniques that are similar to those used in our fully-malicious 2PC protocol. We now briefly discuss the difficulties that arise and how we solve them using those techniques. In our first protocol, each player prepares $t$ garbled circuits and opens all but one of them. The main difficulty is to make sure each player uses the same input in the evaluation of the circuit generated by himself and in the one by his counterpart.

In the second protocol, each player opens a constant fraction of his garbled circuits, and thus, the issue of Garbler's input consistency must also be addressed. Here, however, this issue is relevant for both players. Somewhat surprisingly, we show that our XOR-gadget technique can be used to solve both issues by forcing each player to use the *same* input not only in the garbled circuits generated by himself, but also in the ones generated by his counterpart.

An additional difficulty is in the last step of the protocol, wherein the players need to check the equality of the outputs they receive from each others' evaluation(s). The correctness of this step relies on the authenticity of the outputs (i.e., that forged outputs cannot be used in the equality test). But which output should the players use when they evaluate more than one circuit? Interestingly, this is closely related to the issue we needed to address in standard 2PC for two-output functions: in both cases, a player who evaluates a set of circuits wishes to learn the output along with an unforgeable authentication of that output. We show how the same techniques can be used here as well. See Sections 4.2 and D.2 for the details of the two constructions.

## 2 Preliminaries

Throughout this work we denote by $t$ a statistical security parameter and by $s$ a computational security parameter. For a fixed circuit in use, we denote by $\mathrm{INP}_i$ the set of indexes of $P_i$'s input-wires to the circuit, by INP the set $\mathrm{INP}_1 \cup \mathrm{INP}_2$, by $\mathrm{OUT}_i$ the set of indexes of $P_i$'s output-wires, and by OUT the set $\mathrm{OUT}_1 \cup \mathrm{OUT}_2$. For shortening, we sometimes refer to $|\mathrm{INP}_i|$ by $n_i$, to $|\mathrm{OUT}_i|$ by $q_i$, and set $n = n_1 + n_2$ and $q = q_1 + q_2$.

Denote by $\mathsf{Enc}(sk, m)$ the encryption of message $m$ under secret key $sk$, by $\mathsf{PRG}(s, l)$ the $l$-bit string generated by a pseudo-random generator with seed $s$, and by $\mathsf{Com}(m, r)$ the commitment on message $m$ using randomness $r$. The decommitment of $\mathsf{Com}(m, r)$ is $m$ and $r$. (In some cases we use the abbreviations $\mathsf{PRG}(s)$ and $\mathsf{Com}(m)$.)

We also use the following notation for the next cryptographic primitives and functionalities.

YAO'S GARBLING. For the sake of simplicity and generality, we do not go into the details of the garbling mechanism and only introduce the notations we need to describe our protocols. We refer the reader to [18, 2] for different approaches to creating the garbled circuits.

Given a garbled circuit $gc$, we denote by $\mathsf{label}(gc, j, b)$ the label of wire $j$ corresponding to bit value $b$. Also, we denote by $\mathsf{Garb}(C, r)$ the (deterministic) garbling of circuit $C$ using randomness $r$. (In practice, $r$ would be a short seed for a pseudo-random function). For simplicity, we assume that the labels of the circuit's output-wires include also the actual output bits (thus, allowing the evaluator to learn the output).

We require the garbling scheme to be *private* and *authenticated*, meaning that given a garbled circuit and input labels of a specific input, nothing is revealed except for the output of the circuit, and, that the output-wire labels authenticate the actual output (thus, the actual output cannot be forged). Also, we require that given a

garbled circuit and an input label, one can verify whether the input label is a valid input label. (This can be achieved by using *elusive double encryption* [18] or by adding commitments on the possible input-wire labels in a random order and decommitting when a verification is needed.)

BATCH COMMITTING OBLIVIOUS TRANSFER (BCOT). Here, sender $S$ has $n$ sets, each of $m$ pairs of inputs, $\{(x_0^{j,z}, x_1^{j,z})\}_{j=1\ldots n, z=1\ldots m}$, and receiver $R$ has a vector of input bits $\bar{b} = (b_1, \cdots, b_n)$. The receiver $R$ learns the outputs according to his input bits, $x_{b_j}^{j,z}$ for all $j$ and $z$. In addition, $R$ learns commitments on *all* the sender's inputs.

[26] shows an implementation of BCOT with a cost of $O(mn)$ expensive operations. Using their construction to implement the seed OTs in the OT-extension of [21] in the Random Oracle Model results in an alternative protocol that requires only $O(s)$ expensive operations and $O(nm)$ inexpensive ones. However, in the latter construction, the commitments on the sender's inputs cannot be opened separately and one needs to decommit all the inputs at once. (We use both instantiations at different places in our protocols.) See Appendix B for more details. We denote the first protocol by BCOT1 and the second by BCOT2.

TWO-STAGE EQUALITY TESTING. In this protocol, player $P_1$ has input $x_1$ and player $P_2$ has input $x_2$. They want to test whether $x_1 = x_2$. The functionality is split into two stages in order to emulate a commitment on the inputs before revealing the result (we will use this property in one of our constructions). I.e., in the first stage players submit their inputs and learn nothing, and in the second stage, only if they both ask for the output, they receive the result. This functionality can be realized using ElGamal encryption and ZKPoKs. In Appendix B we formally define this functionality and discuss different realizations.

# 3 An Efficient 2PC for Two-output Functions with Full Security

In this section, we review the main ideas behind our efficient 2PC protocol with full security against malicious adversaries, considering the case where only $P_2$ needs to learn the output. In Appendix C.2 we show how to extend the ideas in order to handle two-output functions. A detailed description of the protocol and the proof of its security appear in Appendix C.

Consistency of the evaluator's input is taken care of by combining the technique of [17] with the OT-extension of [21] and the Free-XOR technique [14]. In a nutshell, $P_2$'s input is encoded using $\max(4n_2, 8t)$ bits in a way that any leakage of less than $t$ of the bits does not reveal meaningful information about $P_2$'s input. During the cut-and-choose, $P_2$ asks $P_1$ to reveal all his inputs to the OTs. If some of the inputs are not consistent with the one $P_2$ has learned from the OTs, $P_2$ aborts. This abort leaks information only in case $P_1$ guessed successfully more than $t$ bits in $P_2$'s encoded input. However, this can happen with only a negligible probability given how the encoding is done.

As we discussed in Section 1.1, consistency of the garbler's input is addressed using the XOR-gadgets. In the following we describe the main steps of the protocol with a focus on this component.

**Garbling stage and the XOR-gadgets.** Say the players want to compute $C(x, y)$, where $x$ is $P_1$'s input and $y$ is $P_2$'s input. Based on $C$, we define the following two circuits: (1) $C_1(x, y, r)$, which computes $(C(x, y), x \oplus r)$ where $r$ is a random input string of length $|x|$ selected by $P_1$; (2) $C_2(x, r)$, which computes $x \oplus r$, where $x$ and $r$ are $P_1$'s inputs and are of the same length. In both circuits we assume the indexes of the input-wires are the same as in $C$ and we define the function $\alpha(k)$ to be the function that given $k \in \mathrm{INP}_1$ returns the index of the input-wire of the random bit that is xored with input-wire $k$. (For simplicity, we assume the same function is applicable for both $C_1$ and $C_2$.)

$P_1$ picks a random string $z_i$ and generates a garbled circuit $gc_i = \mathsf{Garb}(C_1, z_i)$, for $i = 1 \ldots t$. In addition, $P_2$ picks a random string $z_i'$ and generates a garbled circuit $xg_i = \mathsf{Garb}(C_2, z_i')$, for $i = 1 \ldots t$. Both players send the garbled circuits they created to each other. Next, $P_1$ picks $r_j$ at random for $j \in [t]$ and sends to $P_2$ the labels that correspond to $r_j$ in $gc_j$.

**OTs for input labels.** Parties execute OTs and BCOTs in order for each to learn the input-wire labels for his inputs in the circuits/gadgets created by his counterpart. More specifically, first they run any simulatable OT

protocol with the OT-extension of [21], where $P_1$ acts as the sender and $P_2$ acts as the receiver. They use the technique of [17] for protecting against inconsistent inputs as described earlier. $P_1$'s inputs are the labels of $P_2$'s input-wires in all $gc_j$ (i.e., the inputs are $\mathsf{label}(gc_j, k, 0)$ and $\mathsf{label}(gc_j, k, 1)$ for $k \in \mathrm{INP}_2$ and $j \in [t]$). $P_2$'s input is his actual input. (We ignore here the details of encoding $P_2$'s input.) Second, they execute BCOT2 twice where $P_2$ acts as the sender and $P_1$ acts as the receiver: (1) $P_2$'s inputs are the labels of the input-wires in his XOR-gadget $xg_j$, and $P_1$'s inputs are his random input and actual input to the gadget (i.e., $P_2$ inputs are $\mathsf{label}(xg_j, k, 0)$ and $\mathsf{label}(xg_j, k, 1)$ while $P_1$'s inputs are his actual input bits, and (2) $P_2$'s inputs are $\mathsf{label}(xg_j, \alpha(k), 0)$ and $\mathsf{label}(xg_j, \alpha(k), 1)$ while $P_1$'s inputs are the bits of $r_j$. Note that in the first BCOT2, $P_1$ inputs a single bit for each input bit and receives $t$ input-wire labels. That restricts him to use the same input in all the XOR-gadgets.).

(In the detailed protocol, the players execute the OTs before sending the garbled circuits. Still, the intuition is similar.)

We note that $P_1$ is yet to send the labels for his input wires in the circuits he garbled himself, i.e. $gc_i$-s.

**Cut-and-Choose (first stage).** After the OTs/BCOTs, $P_1$ opens a constant fraction of his garbled circuits/gadgets. In particular, $P_1$ opens the garbled circuit $gc_j$ for all $j \notin E$, where $E$ is chosen randomly using a joint coin-tossing protocol. (A joint coin-tossing protocol is needed for the simulation to work.) Moreover, $P_1$ reveals the random strings $r_j$-s he used in the opened circuits (by showing the labels he learned from BCOT2), and all his inputs to the OTs for the opened circuits. $P_2$ checks the correctness of the opened circuits and verifies that the same $r_j$ was used in both $gc_j$ and $xg_j$ for all $j \notin E$. (He also verifies that the values he has received in the OTs for his inputs are consistent with what $P_1$ revealed, following the technique of [17].)

**Cut-and-Choose (second stage).** $P_1$ evaluates all the XOR-gadgets he received from $P_2$, and sends a commitment on all the output-wire labels he obtained to $P_2$. $P_2$ answers with opening *all* the XOR-gadgets $xg_j$ for $j \in E$, and by decommitting all his inputs to BCOT2. $P_1$ checks that all the XOR-gadgets he received were properly constructed, and that the labels are consistent with the decommitments. If so, $P_1$ decommits the output-wire labels of the XOR-gadgets to $P_2$.

In general, the last step is not sound for all commitments since $P_1$ can send a commitment for which he does not know the corresponding message and later be able to decommit once $P_2$ opens the XOR-gadgets (A similar issue was discussed earlier in Footnote 1). There are several ways to overcome this issue. One option is to require $P_1$ to *prove* that he *knows* how to construct this commitment, or more formally, $P_1$ commits on the output labels with $\mathsf{Com}(labels, r)$ and proves using a ZKPoK that he knows $labels$ and $r$. This step can be implemented efficiently for Pedersen's commitment [23], requiring only a small constant number of exponentiations. (When $labels$ is longer than the commitment input length, $P_1$ picks a random seed $seed$, sends $\mathsf{Com}(seed, r), \mathsf{PRG}(seed) \oplus labels$ and ZKPoK that he knows $seed$ and $r$.) A more efficient option is to implement $\mathsf{Com}(labels, r)$ in the Random Oracle model using $\mathrm{H}(key \circ labels \circ r)$, where the commitment key $key$ is chosen at random by the receiver (i.e., $P_2$ in our case). The complexity in this case is only a single call to the random oracle.

**Circuit Evaluation.** $P_1$ sends to $P_2$ the labels of his inputs for the remaining garbled circuits and XOR-gadgets. $P_2$ uses them to evaluate all his remaining circuits and gadgets. He checks that the output-wires of the XOR-gadgets are the same as the values $P_1$ sent him. If so, he takes the majority of the outputs to be his output.

**Summary.** Note that now, with high probability, not only do we know that the majority of the circuits being evaluated are correct, but also that $P_1$ used the same $r_j$-s in the XOR-gadget pairs (Check 3 from introduction). Also, recall that in the BCOT for XOR-gadgets created by $P_2$, $P_1$ can learn the labels for exactly one possible value of $x$. Thus, his $x$ is the same for all the $t$ XOR-gadgets $P_2$ generated (Check 1). Combined with the fact that $P_2$ checks equality of the output of the XOR-gadget pairs (Check 2), he is ensured that the same input bits are being used in majority of the $gc_j$-s. See Figure 1 for a diagram explaining the above intuition.

ADVANTAGES OVER PREVIOUS WORK. The resulting protocol has two main advantages over previous constructions: (1) The BCOT we use requires only $O(t \cdot |\mathrm{INP}_1|)$ inexpensive operations for checking $P_1$'s input consistency. (When realized for concrete parameters with the OT-extension protocol of [21], the constant is fairly small, about 16 inexpensive operations per input bit per garbled circuit (approximately). This cost is sim-

ilar to the cost of garbling additional $4-6$ AND gates, depending on the garbling scheme.) This is in contrast to the previous (efficient) constructions, which require either $O(t^2 \cdot |\text{INP}_1|)$ inexpensive operations [20, 17], or $O(t \cdot |\text{INP}_1|)$ expensive ones [20, 19, 26]. (2) Even if we do not use OT extension (e.g. to avoid making less standard assumptions about the hash function) the overhead of *both* (evaluator and generator) input consistency checks is now reduced to the cost of performing BCOT. (i.e. BCOT1 can be used to solve both the garbler's and the evaluator's input consistency issue.) Previous constructions [17, 19, 26] use different techniques for checking consistency of $P_1$'s and $P_2$'s inputs, that are incomparable and with difficulties that look unrelated. Having one concrete primitive to focus on is a cleaner approach for improving efficiency.

# 4 Security with Input-Dependent Abort in the Presence of Covert Adversaries

## 4.1 The Model

Following [17, 1, 7], we use the ideal/real paradigm for our security definition.

**Real-model execution.** The real-model execution of protocol $\Pi$ takes place between players $(P_1, P_2)$, at most one of whom is corrupted by a probabilistic polynomial-time machine adversary $\mathcal{A}$. At the beginning of the execution, each party $P_i$ receives its input $x_i$. The adversary $\mathcal{A}$ receives an auxiliary information $aux$ and an index that indicates which party it corrupts. For that party, $\mathcal{A}$ receives its input and sends messages on its behalf. Honest parties follow the protocol.

Let $\text{REAL}_{\Pi,\mathcal{A}(aux)}(x_1, x_2)$ be the output vector of the honest party and the adversary $\mathcal{A}$ from the real execution of $\Pi$, where $aux$ is an auxiliary information and $x_i$ is player $P_i$'s input.

**Ideal-model execution.** Let $f : (\{0,1\}^*)^2 \to \{0,1\}^*$ be a two-party functionality. In the ideal-model execution, all the parties interact with a trusted party that evaluates $f$. As in the real-model execution, the ideal execution begins with each party $P_i$ receiving its input $x_i$, and $\mathcal{A}$ receives the auxiliary information $aux$. The ideal execution proceeds as follows:

**Send inputs to trusted party:** Each party $P_1, P_2$ sends $x'_i$ to the trusted party, where $x'_i = x_i$ if $P_i$ is honest and $x'_i$ is an arbitrary value if $P_i$ is controlled by $\mathcal{A}$.

**Abort option:** If any $x'_i = \mathsf{abort}$, then the trusted party returns $\mathsf{abort}$ to all parties and halts.

**Attempted cheat option:** If $P_i$ sends $\mathsf{cheat}_i(\epsilon')$, then:

- If $\epsilon' > \epsilon$, the trusted party sends $\mathsf{corrupted}_i$ to all parties and the adversary $\mathcal{A}$, and halts.
- Else, with probability $1-\epsilon'$ the trusted party sends $\mathsf{corrupted}_i$ to all parties and the adversary $\mathcal{A}$ and halts.
- With probability $\epsilon'$,
  - The trusted party sends $\mathsf{undetected}$ and $f(x'_1, x'_2)$ to the adversary $\mathcal{A}$.
  - $\mathcal{A}$ responds with an arbitrary boolean (polynomial) function $g$.
  - The trusted party computes $g(x'_1, x'_2)$. If the result is 0 then the trusted party sends $\mathsf{abort}$ to all parties and the adversary $\mathcal{A}$ and halts. (i.e. $\mathcal{A}$ can learn $g(x'_1, x'_2)$ by observing whether the trusted party aborts or not.)

Otherwise, the trusted party sends $f(x'_1, x'_2)$ to the adversary.

**Second abort option:** The adversary sends either $\mathsf{abort}$ or $\mathsf{continue}$. In the first case, the trusted party sends $\mathsf{abort}$ to all parties. Else, it sends $f(x'_1, x'_2)$.

**Outputs:** The honest parties output whatever they are sent by the trusted party. $\mathcal{A}$ outputs an arbitrary function of its view.

Let $\text{IDEAL}^{\epsilon}_{f,\mathcal{A}(aux)}(x_1, x_2)$ be the output vector of the honest party and the adversary $\mathcal{A}$ from the execution in the ideal model.

**Definition 4.1.** *A two-party protocol $\Pi$ is* secure with input-dependent abort in the presence of covert adversaries with $\epsilon$-deterrent ($\epsilon$-CovIDA) *if for any probabilistic polynomial-time adversary $\mathcal{A}$ in the real model, there exists a probabilistic polynomial time adversary $\mathcal{S}$ in the ideal model such that*

$$\left\{\text{REAL}_{\Pi,\mathcal{A}(aux)}(x_1, x_2)\right\}_{x_1,x_2,aux\in\{0,1\}^*} \overset{c}{\approx} \left\{\text{IDEAL}^{\epsilon}_{f,\mathcal{S}(aux)}(x_1, x_2)\right\}_{x_1,x_2,aux\in\{0,1\}^*}$$

*for all $|x_1| = |x_2|$ and $aux$.*

COMPARISON WITH COVERT SECURITY. When we let $\epsilon = 1/t$ for any constant $t$, the above definition is strictly stronger than the standard definition of security against covert adversaries. In covert security, in case of undetected cheating which happens with probability $\epsilon$, the adversary learns *all* the honest parties' private inputs and is able to change the outcome of computation to *whatever* value it wishes (i.e. no privacy or correctness guarantee). In our definition, however, the adversary can learn at most a single bit of information (from the abort), and under no condition is able to change the output (full correctness).

In the above definition, in contrast to the standard covert security, the adversary can choose the exact probability he gets caught (i.e. $1 - \epsilon'$) as long as this probability is larger than $1 - \epsilon$ (where $\epsilon$ is the deterrence factor). Note that letting the adversary choose $1 - \epsilon'$ is not a relaxation in security since the adversary can only increase the probability of itself getting caught. We believe that this variant of the definition where the adversary can choose $\epsilon' > \epsilon$ with which it can get caught is of independent interest. Specifically, it yields an alternative definition for covert security that is more convenient to use in simulation-based proofs. (To obtain this alternative definition for covert security, replace the steps that are done with probability $\epsilon'$ with the following: (1) The trusted party sends $x'_1, x'_2$ to $\mathcal{A}$; (2) $\mathcal{A}$ sends the value $y$ to the trusted party, and the trusted party sends it to all parties as their output.)

A REMARK ON ADAPTIVENESS OF LEAKAGE FUNCTION. In the above definition, the leakage function $g$ can be chosen adaptively after seeing $f(x'_1, x'_2)$. Somewhat surprisingly, this does not give any extra power to the adversary compared to the non-adaptive case since even in the non-adaptive case, $g$ can be chosen to be a function that computes $f(x'_1, x'_2)$, emulates the adversary's computation given that value and evaluates the leakage function he would have chosen in the adaptive case.

## 4.2 An Efficient Protocol with $\frac{2}{\epsilon}$ Circuits

In this section, we review the main steps of our $\epsilon$-CovIDA protocol and highlight the new techniques. A detailed description of the protocol and how to reduce the number of circuits (from linear in $\frac{1}{\epsilon}$ to logarithmic) appear in Appendix D.1.

As discussed in the introduction, in the dual-execution protocol of [20, 7] parties engage in two different executions of the semi-honest Yao's garbled circuit protocol, and then run an equality testing protocol to confirm that the outputs of the two executions are the same before revealing the actual output values. We show how to extend this protocol to work in the presence of covert adversaries using the ideas presented in Section 3. For simplicity of the description, from now on we work with $t = \frac{1}{\epsilon}$ (a statistical security parameter) instead of $\epsilon$ since $t$ would be the number of circuits each party garbles.

**Dual-execution & cut-and-choose.** Our first step is to combine the dual-execution protocol with a standard cut-and-choose protocol for covert players. Each player $P_i$ garbles $t$ circuits $gc^i_1, \ldots, gc^i_t$ and sends them to the other player. Parties pick a random value $e \in [t]$, *open* all the circuits $gc^i_{j\neq e}$ and verify that they were constructed properly. This assures that with probability $1 - 1/t$, the remaining *circuit-pair* $(gc^1_j, gc^2_j)$ is properly constructed. As before, they send the garbler's input-wire labels for the $e$-th circuit, execute OTs for the respective evaluators to learn their input-wire labels, evaluate the circuits, call the Equality Testing functionality and output accordingly.

The above protocol would guarantee correctness similar to the dual-execution protocol, and it would ensure that the evaluated circuits are correct with probability $1 - 1/t$. However, the protocol does not satisfy our security definition. One issue is that a malicious player learns the output of the computation even if the other player catches him cheating (as a result of the equality test). We show how this can be avoided by masking the output of the computation with random strings, chosen by the two players, and revealing them at the end of the computation in order to unmask the actual output.

A more subtle attack to address is that a malicious player can learn one bit of information about an honest party's input with probability greater than $1/t$ (in fact with probability 1): a malicious player can use different inputs in each of the two evaluated circuits, and learn whether their outputs are the same or not based on the final outcome. This attack is successful even if *all* the circuits are constructed correctly. We prevent this attack using the XOR-gadget techniques discussed earlier, along with some enhancements. We discuss the details next:

**XOR-gadgets.** Define $C(x \circ m_1, y \circ m_2)$ to be the circuit that receives inputs $x, y$ and two masks $m_1, m_2$ and computes $f(x, y) \oplus m_1 \oplus m_2$. Based on $C$, let $P_1$'s input $x'$ be $x \circ m_1$ and $P_2$'s input $y'$ be $y \circ m_2$, where $m_i$ is a random string of length $q$ ($f$'s output length) selected by $P_i$. We define the following four circuits:

(1) $C_1(x', y', r_1) = (C(x', y'), x' \oplus r_1)$, where $r_1$ is a random input string of length $|x'|$ selected by $P_1$; (2) $C_2(x', y', r_2) = (C(x', y'), y' \oplus r_2)$ where $r_2$ is a random input string of length $|y'|$ selected by $P_2$; (3) $C'_1(y', r_2) = y' \oplus r_2$ evaluated by $P_2$ on his own inputs; (4) $C'_2(x', r_1) = x' \oplus r_1$ evaluated by $P_1$ on his own inputs; In all circuits we assume the indexes of the input-wires are the same as in $C$ and we define the function $\alpha(k)$ to be the function that given $k \in \text{INP}$ returns the index of the input-wire of the random bit input-wire that is xored with input-wire $k$. (For simplicity, we assume the same function is applicable for all $C_i$-s and $C'_i$-s.)

Instead of garbling $C$, each player $P_i$ generates and sends $t$ garbled circuits for $C_i$: $gc_1^i, \ldots, gc_t^i$ and $t$ garbled circuits for $C'_i$: $xg_1^i, \ldots, xg_t^i$. After sending the sets of garbled circuits, for each $j \in [t]$, player $P_i$ picks at random a string $r_j^i$ and sends the input-wire labels that correspond to $r_j^i$ in $gc_j^i$.

**OTs for input labels.** Then, they execute BCOTs in order to learn the input-wire labels for both their actual inputs and the $r_j^i$-s in their counterpart's circuits. More specifically, first they use BCOT1 where $P_1$ acts as the sender and $P_2$ acts as the receiver. $P_1$'s inputs are the input-wire labels of $P_2$'s input-wire $k$ in all $gc_j^1$-s and $xg_j^1$-s (i.e., the input pairs are $\big(\mathsf{label}(gc_j^1, k, 0), \mathsf{label}(gc_j^1, k, 1)\big)_{j \in [t]}$ and $\mathsf{label}(xg_1^1, k, 0) \circ \cdots \circ \mathsf{label}(xg_t^1, k, 0), \mathsf{label}(xg_1^1, k, 1) \circ \cdots \circ \mathsf{label}(xg_t^1, k, 1)$ for $k \in \text{INP}_2$). $P_2$'s input is his actual input. Second, they use BCOT2 with the labels for the rest of the input-wires of $xg_j^1$ (i.e., $\mathsf{label}(xg_j^1, \alpha(k), 0)$,$\mathsf{label}(xg_j^1, \alpha(k), 1)$ for $k \in \text{INP}_2$ and $j \in [t]$, where $P_2$'s inputs are the bits of $r_j^2$). The players run the same protocols in the opposite direction (switching roles). At the end, each player learns the labels for his input-wires of $gc_j^{3-i}$ and of $xg_j^{3-i}$. But we note that $P_i$ is yet to send the labels for his input wires in the circuits he garbled himself, i.e. $gc_j^i$ and $xg_j^i$.

**Cut-and-Choose Phase (first opening).** Next, as before, parties agree on a random $e \in [t]$ (using a joint coin-tossing protocol), and open the rest of the garbled circuits. In particular, they open the garbled circuit-pairs $(gc_j^1, gc_j^2)$ and the XOR-gadgets $(xg_j^1, xg_j^2)$ for all $j \neq e$. Moreover, for $j \neq e$, they reveal to each other the random strings $r_j^i$-s they used in the opened circuits (by showing the labels they learned in BCOT2), and then they decommit all the inputs they used as senders in BCOT1 for the opened circuits. The players check the correctness of the circuits and verify that the same $r_j^i$-s were used in both $gc_j^i$ and $xg_j^{3-i}$. (Note that at the end of the opening phase, the players know that with $1 - 1/t$ probability the remaining circuit-pair $(gc_e^1, gc_e^2)$ and the XOR gadget-pair $(xg_e^1, xg_e^2)$ are properly constructed, and, that the inputs $r_e^i$ used by the players in both $gc_e^i$, and $xg_e^{3-i}$ are the same.)

**Evaluation.** Each party sends to his counterpart the input-wire labels for his inputs in the unopened circuit-pair. Parties then evaluate the circuit-pair $(gc_e^1, gc_e^2)$ and the XOR-gadgets $(xg_e^1, xg_e^2)$. (i.e., $P_i$ evaluates $gc_e^{3-i}$, and $xg_e^{3-i}$.) $P_i$ sends a commitment (along with a ZKPoK, as in Section 3) on the concatenation of the output labels he obtained after evaluating $xg_e^{3-i}$ to $P_{3-i}$.

**Cut-and-Choose Phase (second opening).** $P_{3-i}$ now opens the remaining XOR-gadget $xg_e^{3-i}$, and decommits all his inputs as a sender to the BCOTs of the XOR-gates (i.e., $\mathsf{label}(xg_1^{3-i}, k, 0) \circ \cdots \circ \mathsf{label}(xg_t^{3-i}, k, 0)$,

$\mathsf{label}(xg_1^{3-i}, k, 1) \circ \cdots \circ \mathsf{label}(xg_t^{3-i}, k, 1)$ in BCOT1, and $\mathsf{label}(xg_e^{3-i}, \alpha(k), 0), \mathsf{label}(xg_e^{3-i}, \alpha(k), 1)$ in BCOT2, both for $k \in \mathrm{INP}_i$). (We stress that only the XOR-gates of wires $\mathrm{INP}_i$ are opened, and that those were generated using random labels independently of the garbled circuits. The XOR-gadgets of wires $\mathrm{INP}_{3-i}$ are checked as part of the previous phase.) $P_i$ verifies that these XOR-gates were generated properly and that the BCOTs inputs were consistent with the XOR-gates. If everything is correct he decommits his commitment, otherwise he outputs $\perp$ and aborts. (Note that $P_i$ reveals his output only *after* he verified that all the XOR-gates $P_{3-i}$ generated were properly constructed. Since the only secrets in these gates are $P_i$'s inputs, revealing them does not help $P_i$ learn any new information.) $P_{3-i}$ verifies that the decommitted values are valid output-wire labels, and compares the actual output with their output he obtains from evaluation of $xg_e^i$. If either check fails, $P_{3-i}$ outputs $\perp$.

**Equality-test.** If there is no abort, players call the Equality Testing functionality as before. Note that now, with probability $1 - 1/t$, not only we know that the circuits being evaluated are correct, but also that the players use the same $r_e^i$-s in the final XOR gadget-pair. Combined with the fact that the players check equality of the output of the final XOR gadget-pair, they are ensured (with probability $1 - 1/t$) that the same input strings are being used in $gc_e^1$ and $gc_e^2$ or else, $x \oplus r_e^i$ would be different.

**Output Unmasking.** If the Equality Testing functionality returns False, the players abort. Otherwise, they unmask the output. (Recall that at this stage, each player knows the value of $C(x', y') = f(x, y) \oplus m_1 \oplus m_2$.) Player $P_i$ sends the value of $m_i$ along with labels that correspond to $m_i$ in $gc_e^{3-i}$. These labels prove that $m_i$ is indeed the value that $P_i$ have used in the protocol.

Putting things together, correctness is always guaranteed due to the dual execution; full-privacy is guaranteed with probability $1 - 1/t$ due to the discussion above; and privacy with 1-bit leakage is guaranteed in the case that a cheating adversary is not caught, which only happens with probability $1/t$.

## Acknowledgements

We would like to thank Ran Canetti, Benny Pinkas and Yehuda Lindell for their comments and helpful discussions. We also thank Yehuda Lindell for referring us to the malleability issue discussed in Footnote 1.

## References

[1] Aumann, Y., Lindell, Y.: Security against covert adversaries: Efficient protocols for realistic adversaries. J. Cryptol. 23(2), 281–343 (Apr 2010)

[2] Bellare, M., Hoang, V.T., Rogaway, P.: Foundations of garbled circuits. pp. 784–796. CCS 2012, ACM (2012)

[3] Fischlin, M.: Trapdoor Commitment Schemes and Their Applications. Ph.D. Thesis (Doktorarbeit), Department of Mathematics, Goethe-University, Frankfurt, Germany (2001)

[4] Goldreich, O., Micali, S., Wigderson, A.: How to play any mental game. pp. 218–229. STOC 1987, ACM (1987)

[5] Goyal, V., Mohassel, P., Smith, A.: Efficient two party and multi party computation against covert adversaries. pp. 289–306. EUROCRYPT 2008, Springer-Verlag (2008)

[6] Huang, Y., Evans, D., Katz, J., Malka, L.: Faster secure two-party computation using garbled circuits. pp. 35–35. Security 2011, USENIX Association (2011)

[7] Huang, Y., Katz, J., Evans, D.: Quid-pro-quo-tocols: Strengthening semi-honest protocols with dual execution. pp. 272–284. SP 2012, IEEE Computer Society (2012)

[8] Ishai, Y., Kilian, J., Nissim, K., Petrank, E.: Extending oblivious transfers efficiently. pp. 145–161. CRYPTO 2003, Springer (2003)

[9] Ishai, Y., Kushilevitz, E., Ostrovsky, R., Prabhakaran, M., Sahai, A.: Efficient non-interactive secure computation. pp. 406–425. EUROCRYPT 2011, Springer-Verlag (2011)

[10] Ishai, Y., Prabhakaran, M., Sahai, A.: Founding cryptography on oblivious transfer — efficiently. pp. 572–591. CRYPTO 2008, Springer-Verlag (2008)

[11] Jarecki, S., Shmatikov, V.: Efficient two-party secure computation on committed inputs. pp. 97–114. EUROCRYPT 2007, Springer-Verlag (2007)

[12] Jawurek, M., Kerschbaum, F., Orlandi, C.: Zero-knowledge using garbled circuits: How to prove non-algebraic statements efficiently. Cryptology ePrint Archive, Report 2013/073 (2013)

[13] Kiraz, M.S., Schoenmakers, B.: A protocol issue for the malicious case of Yaos garbled circuit construction. In: In Proceedings of 27th Symposium on Information Theory in the Benelux. pp. 283–290 (2006)

[14] Kolesnikov, V., Schneider, T.: Improved garbled circuit: Free xor gates and applications. pp. 486–498. ICALP 2008, Springer-Verlag (2008)

[15] Kreuter, B., Shelat, A., Shen, C.H.: Billion-gate secure computation with malicious adversaries. pp. 14–14. Security 2012, USENIX Association (2012)

[16] Lindell, Y., Oxman, E., Pinkas, B.: The IPS compiler: Optimizations, variants and concrete efficiency. pp. 259–276. CRYPTO 2011, Springer (2011)

[17] Lindell, Y., Pinkas, B.: An efficient protocol for secure two-party computation in the presence of malicious adversaries. pp. 52–78. EUROCRYPT 2007, Springer-Verlag (2007)

[18] Lindell, Y., Pinkas, B.: A proof of security of Yao's protocol for two-party computation. J. Cryptol. 22(2), 161–188 (Apr 2009)

[19] Lindell, Y., Pinkas, B.: Secure two-party computation via cut-and-choose oblivious transfer. pp. 329–346. TCC 2011, Springer-Verlag (2011)

[20] Mohassel, P., Franklin, M.: Efficiency tradeoffs for malicious two-party computation. pp. 458–473. PKC 2006, Springer-Verlag (2006)

[21] Nielsen, J.B., Nordholt, P.S., Orlandi, C., Burra, S.S.: A new approach to practical active-secure two-party computation. pp. 681–700. CRYPTO 2012, Springer (2012)

[22] Nielsen, J.B., Orlandi, C.: Lego for two-party secure computation. pp. 368–386. TCC 2009, Springer-Verlag (2009)

[23] Pedersen, T.P.: Non-interactive and information-theoretic secure verifiable secret sharing. pp. 129–140. CRYPTO 1991, Springer-Verlag (1992)

[24] Peikert, C., Vaikuntanathan, V., Waters, B.: A framework for efficient and composable oblivious transfer. pp. 554–571. CRYPTO 2008, Springer-Verlag (2008)

[25] Pinkas, B., Schneider, T., Smart, N.P., Williams, S.C.: Secure two-party computation is practical. pp. 250–267. ASIACRYPT 2009, Springer-Verlag (2009)

[26] Shelat, A., Shen, C.H.: Two-output secure computation with malicious adversaries. pp. 386–405. EUROCRYPT 2011, Springer-Verlag (2011)

[27] Woodruff, D.P.: Revisiting the efficiency of malicious two-party computation. pp. 79–96. EUROCRYPT 2007, Springer-Verlag (2007)

[28] Yao, A.C.C.: How to generate and exchange secrets. pp. 162–167. SFCS 1986, IEEE Computer Society (1986)

# A Other Related Work

[10, 16] show how to use semi-honest secure two-party, and honest-majority multi-party protocols to achieve security against malicious players. Although this approach is asymptotically efficient, the constant factors seem to be large and we are not aware of any working implementation that evaluates its efficiency in practice. [21] constructs a protocol in the Random Oracle Model, based on OT extension [8] and the classic GMW protocol [4]. However, this protocol requires a number of rounds that depends on the depth of the circuit. Still, for some computations [21] shows better performance than the previous cut-and-choose based protocols.

[9] considers non-interactive secure computation protocols. Their first construction, which is asymptotically very efficient, achieves similar guarantees to the protocol of [7] (though, in a single round of interaction). Combining that protocol with the cut-and-choose method can result in constructions that achieve similar guarantees to our $\epsilon$-CovIDA protocols. However, it is not clear what would the efficiency of these protocols in practice be.

# B Preliminaries

Here we describe the functionalities and previous techniques we need in our constructions.

## B.1 Oblivious Transfer (OT)

In this protocol, sender $S$ has two inputs $x_0, x_1 \in \{0,1\}^l$ and receiver $R$ has input bit $b$. At the end of the protocol, $R$ should learn $x_b$ and $S$ should learn nothing.

[24] shows an efficient construction of fully-secure universally-composable OT based on a variety of standard assumptions. When instantiated based on the DDH assumption, the protocol requires $O(1)$ exponentiations, $O(l)$ inexpensive operations and a constant number of rounds.

[8] presents how to extend $O(s)$ OTs of length $s$ strings to any number $n$ of semi-honest OTs of length $l$ strings, using only additional $O(n \cdot l)$ inexpensive operations. [21] extends their results to *fully-secure* OTs in the (amortized) price of only a (small) constant number of inexpensive operations per OT.[2] Note that the construction of [8] is secure assuming the hash function in use is *correlation-robust*, whereas the construction of [21] is in the Random Oracle Model.

Throughout this work we assume that the strings we transfer in the OT protocols are shorter than the output length of the hash function in use, which allows us to omit the factor $l$ from the complexities. (Specifically, we say that the amortized cost per OT when we use the OT extension protocol of [21] is a constant number of hashes.) When we concatenate several strings in one OT, we count the cost for each one separately.

In our protocols we are interested in large batches of OTs. Specifically, we say that in the batch OT protocol, sender $S$ has $n$ sets, each of $m$ pairs of inputs $\{(x_0^{j,z}, x_1^{j,z})\}_{j=1\ldots n, z=1\ldots m}$, and receiver $R$ has a vector of input bits $\bar{b} = (b_1, \cdots, b_n)$. $R$ learns the outputs according to his input bits, i.e., $x_{b_j}^{j,z}$ for all $j$ and $z$. (See the left side of Figure 2 for an example.) We denote by BOT the batch OT protocol that uses the OT-extension of [21] and the OT of [24] for the seed OTs. The cost of this protocol is $O(s)$ expensive operations and $O(nm)$ inexpensive ones.

[2] [8] also presents how to extend fully-secure OTs. However, their construction has an overhead of $O(t)$ inexpensive operations per OT.

## B.2 Committing Oblivious Transfer (COT)

Committing OT [13] is a variant of OT in which at the end of the protocol, the receiver $R$ learns also commitments on all the sender's inputs. This additional property allows $S$ to "decommit" his inputs later independently of $R$'s inputs. See Figure 2 for an example of a batch version of COT.

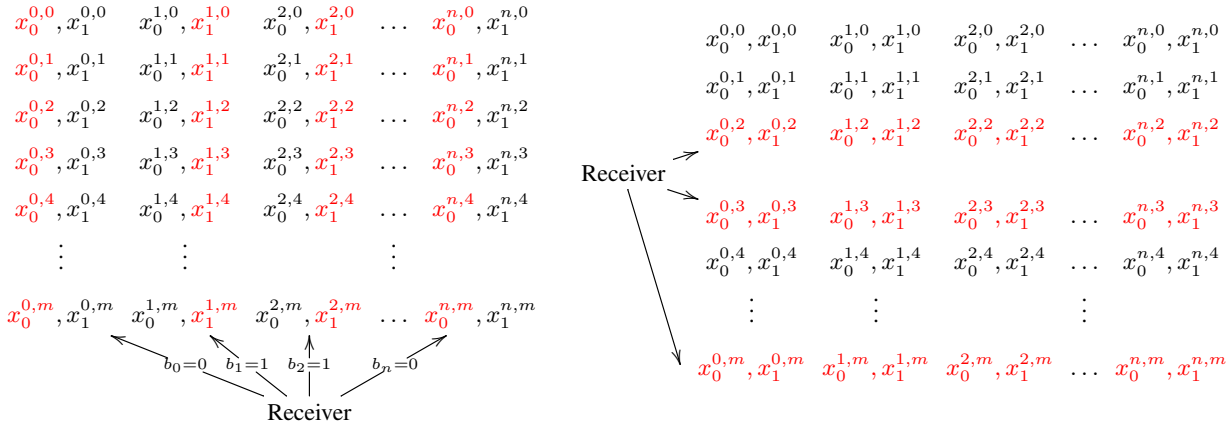


Figure 2: Batch COT Example. On the left is the OTs stage in which the receiver inputs his input bits $b_i$ and learns the red elements. (The commitments are omitted.) On the right is the second stage in which the receiver asks for all the elements in the selected rows to be decommitted, and once the sender reveals their inputs, the receiver can check if those are the right inputs using the commitments he received in the beginning.

[26] shows an efficient implementation of COT under the DDH assumption. Implementing a batch COT based on their protocol requires $O(mn)$ exponentiations. We denote this protocol by BCOT1. We observe that in the same protocol, the receiver is also committed to his inputs, and therefore, once it is combined with the OT-extension of [21] (i.e. using COT of [26] as the seed OTs in the construction of [21]), the sender who plays the role of the OT receiver in the protocol of [21] is committed to all his inputs in the larger protocol as well. The disadvantage of this combined protocol is that in order to "decommit" his inputs, the sender has to decommit *all* of them together, and not just a subset of them. (This happens since for decommitting his inputs, the sender has to reveal his inputs to the seed OTs, and knowing them reveals all his COT inputs.) Still, this combination provides a batch COT protocol with the cost of $O(s)$ expensive operations and $O(nm)$ inexpensive ones. We denote this protocol by BCOT2.

## B.3 The Technique of [17]

[17] shows how to use OT in a black-box way to solve the issue of the Evaluator's input consistency. We give here a brief description of their technique.

We start with some intuition of the construction. Say the receiver input bit is $b$ and the sender's inputs are $x_0, x_1$. Let $d = x_0 \oplus x_1$. Instead of running one OT with their actual inputs, the players execute $k$ OTs, where in the $i$-th OT, the receiver uses the bit $b_i$ and the sender uses the inputs $r_i, r_i \oplus d$. The $b_i$-s are chosen such that their xor equals $b$ and $r_i$ are chosen such that their xor equals $x_0$. Therefore, after executing the $k$ OTs, the receiver can xor the outputs he learned to get the actual output $x_b$. On the other hand, if after the execution of the OTs, the receiver asks the sender to reveal to him his inputs $x_0, x_1$, the sender sends him these values along with *all* the $r_i$-s as a proof (or the "decommitment"). The sender checks that these values are consistent with the outputs he received from the OTs. If the sender tries to cheat on $x_0, x_1$, he will be caught with probability that depends on $k$. Furthermore, the amount of information that the sender will learn about the receiver's input is negligible in $k$. (See [17] for complete details.) However, note that we increased the inputs by a factor of $k$, which is of course undesirable. In order to reduce this overhead, when we have more than one OT we can "share" the random bits among many input bits.

We now describe the more efficient construction in more detail (see [17] for concrete analysis of the parameters): For simplicity, let's assume $m = 1$. The extension to larger $m$ is straightforward. In the first stage, the parties do the following: The sender picks at random $4n$ strings $r_1, \ldots r_{4n}$ and a random string $d$, all of length $l$. The receiver picks at random $n$ random strings $z_1, \ldots z_n$ of length $4n$ and sends them to the sender. Then, he picks at random a string $b'$ of length $4n$ such that for each input bit $b_i$, $< b', z_i >= b_i$, where $< \cdot, \cdot >$ is the inner product operator. They execute an OT $4n$ times, where the sender's input pairs are $(r_j, r_j \oplus d)$ and the receiver's input is $b'_j$ for $j = 1, \ldots, 4n$. The receiver stores all the answers he received from the OT, and, for each $i$, the receiver computes the xor of the answers of the indecies in the set $\{j|$ the $j$-th bit of $z_i$ is $1\}$. These are his outputs from protocol.

In the second stage, the sender sends all the pairs $(r_j, r_j \oplus d)$ and $d$ (to "decommit"). The receiver compares these strings with the ones he received earlier and verifies that the xor of each pair is $d$. If there is a problem, he outputs $\perp$. This completes the description.

(The above description is an adaptation of the technique of [17] with the Free-XOR technique of [14], although the original protocol is the same but with garbled gates that compute the xor of the strings.)

For statistical security parameter $t$, $\max(4n, 8t)$ inputs are needed in order to obtain a negligible probability of failure against the selective-OT attack [17]. When implemented with BOT, the overall cost is $O(s)$ expensive operations and $O(\max(4n, 8t)m)$ inexpensive ones. For computations with large enough input (e.g. $n \geq 260$ for $t = 130$), this is rather efficient. However, for computations with short inputs, we can take the simpler approach of using $z_i$-s with Hamming Weight $t$, such that $< z_i, z_j >= 0$ for all $i \neq j$, and using $nt$ inputs in total.

### B.4 Two-Stage Equality Testing

In this protocol, player $P_1$ has input $x_1$ and player $P_2$ has input $x_2$, and they want to test whether $x_1 = x_2$. We define the functionality $\mathcal{F}^l_{2SET}$ to be:

**First Stage**

**Inputs:** $P_1$ inputs $x_1$ and $P_2$ inputs $x_2$ (both of length $l$).

**Outputs:** Both players receive Inputs Accepted.

**Second Stage**

**Inputs:** Both players input Reveal.

**Outputs:** Both players obtain $(x_1 = x_2)$.

Figure 3: $\mathcal{F}^l_{2SET}$.

A possible implementation of this functionality can be done using ElGamal encryption and efficient ZKPoKs for proving knowledge of discrete log and the exponents of a DH-tuple. (We require that both players prove that they know their inputs so it can be extracted by the simulator.) In the first stage, they compute the value of $(x_1 x_2^{-1})^{r_1 r_2}$ encrypted under a shared ElGamal secret key, where $r_i$ is $P_i$'s private input, chosen at random. In the second stage, they decrypt the result (along with ZKPoK for proving correctness of the decryption). If the result is 1, they output True, and False otherwise (details are omitted).

In case the input size is larger than the encryption message length $l$, the players can do one of the following: (1) Execute the first stage of the protocol $O(|x_i|/l)$ times with different parts of the inputs, and multiply the resulting encryptions to get one encryption. (The second stage is done on that last encryption.) (2) Jointly pick a random key $k$, and use $\mathrm{H}(k \circ x_i)$ as their inputs to the protocol, where H is modeled as a random oracle. The simulator could record all the calls to the random oracle and "invert" $\mathrm{H}(k \circ x_i)$ for retrieving the input $x_i$. The advantage of this option is that only one invocation of the above protocol is needed in this case.

# C Detailed Construction and Proof of Our Fully-secure 2PC Construction

Figure 4 presents our protocol in detail. A simple example of the XOR-gadgets technique is given in Figure 1.

Before we prove security, we need to discuss the cut-and-choose step and its simulatability in more depth. Recall that in the cut-and-choose phase, we need to choose a random subset of $[t]$ of size $t \cdot c$, where $c$ is the constant fraction of the sets we use for evaluation. In particular, this step needs to be performed in a fashion that is simulatable in the proof. We note that a similar issue exists in previous 2PC constructions as well. In [17], this is resolved by generating a random bit for each set and decide whether to open or evaluate the set based on the bit. As shown in [17], this approach is efficiently simulatable but does not yield a previously agreed-on fraction $c$ (e.g. $c = 3/5$ for better security). To the best of our knowledge, the remaining 2PC protocols do not specify the exact procedure with which the random subset is chosen.

For the sake of completeness, we propose one such procedure that is also efficiently simulatable. The intuition is simple. In each iteration $1 \leq j \leq t \cdot c$, one element is sampled uniformly at random from the previously unchosen elements in $[t]$. It is easy to confirm that this yields a uniformly random subset of size $t \cdot c$. The element to be chosen is decided using a uniformly random integer $1 \leq v < (t - j + 1)$ generated by both parties using the following coin-tossing protocol:

- Parties initialize a boolean string $\rho$ of length $t$ to be all zeros.
- For $j = 1, \ldots, (t \cdot c)$, each player $P_i$ picks a random value $v_j^i \in [1..(t - j + 1)]$.
- $P_2$ sends a commitment $\mathsf{Com}(v_1^2 \circ v_2^2 \circ \cdots \circ v_{t \cdot c}^2)$.
- $P_1$ sends his values $v_1^1, \ldots, v_{t \cdot c}^1$.
- $P_2$ decommits and reveal $v_1^2, \ldots, v_{t \cdot c}^2$.
- For $j = 1, \ldots, (t \cdot c)$, let $v = ((v_j^1 + v_j^2) \mod (t - j + 1)) + 1$ and let $k$ be the $v$-th zero bit of $\rho$. Set $\rho_k = 1$.
- Let the set $E$ be $\{j | \rho_j = 1\}$. $E$ would be the set of indexes of the circuits that the players evaluate. (i.e. they open all sets with indexes not in $E$).

## C.1 Proof Sketch of Security of Protocol of Figure 4

We consider the two possible corruption scenarios separately.

$P_1$ IS CORRUPTED. We describe a simulator $\mathcal{S}$ that runs $\mathcal{A}$ internally and interacts with the trusted party that computes $f$. $\mathcal{S}$ does the following: It emulates an honest $P_2$ with random input until the end of stage Input-equality Check (see Figure 4). $\mathcal{S}$ extracts (as done in [26]) $\mathcal{A}$'s input to the BCOT2 (used for $P_1$ to learn the input-wire labels of his input). Denote by $x'$ this input. If the emulation of $P_2$ until the end of the protocol does not lead to an abort, $\mathcal{S}$ calls the trusted party with $x'$ and outputs whatever $\mathcal{A}$ does.

From the cut-and-choose we know that with probability $1 - neg(t)$ (see [17, 26]), at least for half of $j \in E$ it holds that: 1) $r_j$ are the same for both $gc_j, xg_j$; 2) $gc_j$ is properly constructed. Denote the by $E_g \subset E$ the indexes for which the two properties hold.

Denote by $x_j$ the input $P_1$ used for circuit $gc_j$. Observe that if all the XOR-gadgets that $P_1$ generated are correct, and $P_1$ uses the same $r_j$ for $gc_j, xg_j$, then if he uses even a single $x_i \neq x'$ then $P_2$ catches him (since $P_1$ learns only the labels for $x'$ in $xg_j$ for all $j \in E_g$). Therefore, from the cut-and-choose, $P_2$ is assured with probability $1 - neg(t)$ that $P_1$ used the same input for at least half of the sets, and for the same sets he garbled the circuit properly. Thus, the majority of the outputs are correct (and use $x'$ as $P_1$'s input) with probability $1 - neg(t)$.

**Garbling:**
Let $C_1, C_2, \alpha(\cdot)$ as defined in Section 3.
For $j = 1, \ldots, t$, player $P_1$ picks random strings $z_i$ (of length $s$) and $r_i$ (of length $|\mathrm{INP}_1|$), and computes the set $S_j$ containing:

1. A garbled circuit $gc_i = \mathsf{Garb}(C_1, z_i)$.
2. The input-wire labels corresponding to $r_j$ in $gc_j$.

For $j = 1, \ldots, t$, player $P_2$ picks at random a string $z'_i$ and computes XOR-gadget $xg_j = \mathsf{Garb}(C_2, z'_i)$, each includes $|\mathrm{INP}_1|$ XOR-gates.

**Oblivious Transfer:**
They execute the combination of [17] and BOT where $P_1$ is the sender: $P_1$'s input is $|\mathrm{INP}_2|$ sets of $t$ pairs $\big(\ \mathsf{label}(gc_j, k, 0), \mathsf{label}(gc_j, k, 1)\ \big)$ for $k \in \mathrm{INP}_2$ and $j \in [t]$, and $P_2$ uses his actual input bits. (Recall that the players use different inputs when applying the technique of [17]. To simplify the description here, we just refer to the actual inputs they use.)
They execute BCOT2 twice where $P_2$ is the sender: (We separate the two for simplifying the description. However, both protocols can be executed together to reduce the number of seed OTs.) In the first, $P_1$ inputs the bits of $r_j$ and $P_2$ inputs the pairs $(\ \mathsf{label}(xg^j, \alpha(k), 0), \mathsf{label}(xg_j, \alpha(k), 1)\ )$ for $k \in \mathrm{INP}_1$ and $j \in [t]$. In the second, $P_1$ inputs his actual input bits and $P_2$ inputs the pairs $(\ \mathsf{label}(xg_1, k, 0) \circ \mathsf{label}(xg_2, k, 0) \circ \cdots \circ \mathsf{label}(xg_t, k, 0), \mathsf{label}(xg_1, k, 1) \circ \mathsf{label}(xg_2, k, 1) \circ \cdots \circ \mathsf{label}(xg_t, k, 1)\ )$ for $k \in \mathrm{INP}_1$. (Note that in the last BCOT, $P_1$ gets the labels for *all* $t$ circuits together. Because of that, he cannot use inconsistent inputs for $P_2$'s XOR-gadgets.)

**Cut-and-choose:**
$P_1$ sends the sets $S_1, \ldots, S_t$ and $P_2$ sends the XOR-gadgets $xg_0, xg_1, \ldots, xg_t$.
They pick a random $E \subset [t]$ of size $t \cdot c$ in the following way:

1. They initialize a boolean string $\rho$ of length $t$ to be all zeros.
2. For $j = 1, \ldots, (t \cdot c)$, each player $P_i$ picks a random value $v_j^i \in [1..(t - j + 1)]$.
3. $P_2$ sends a commitment $\mathsf{Com}(v_1^2 \circ v_2^2 \circ \cdots \circ v_{t \cdot c}^2)$.
4. $P_1$ sends his values $v_1^1, \ldots, v_{t \cdot c}^1$.
5. $P_2$ decommits and reveal $v_1^2, \ldots, v_{t \cdot c}^2$.
6. For $j = 1, \ldots, (t \cdot c)$, let $v = ((v_j^1 + v_j^2) \mod (t - j + 1)) + 1$ and let $k$ be the $v$-th zero bit of $\rho$. Set $\rho_k = 1$.
7. Let the set $E$ be $\{j | \rho_j = 1\}$. $E$ would be the set of indexes in which the players will evaluate (and open all sets with indexes not in $E$).

**Checking Opened Circuits:**
For all $j \notin E$, $P_1$ sends: 1) $z_j$; 2) The labels he learned from BCOT2 for $r_j$.
For the opened sets, $P_2$ verifies that the circuits and gadgets were constructed properly, and that $P_1$ used the same $r_j$ for $xg_j$ and $gc_j$. Then, $P_1$ reveals all the inputs he used for the BOT in the opened sets and $P_2$ verifies that all the values are consistent with the opened circuits and with the values he received in the BOT.

**Input-equality check:**

1. $P_1$ evaluates the remaining XOR-gadgets he has. He sends a commitment $com$ on all the output-wire labels he got from the XOR-gadgets (or on a random value if there was a problem in the evaluation) along with a ZKPoK that he knows the decommitment of $com$. (If the ZKPoK is invalid, $P_2$ aborts.)[a]
2. $P_2$ opens all his XOR-gadgets in the set $E$ (by sending $z'_i$-s), and decommits all his inputs to BCOT2. $P_1$ verifies that the XOR-gadgets were constructed properly and consistent with the BCOT2 inputs. (If not, he aborts.)
3. $P_1$ decommits $com$ and reveals the output-wire labels he got from the XOR-gadgets. $P_2$ verifies that all labels are valid ones (i.e., generated by him).
4. $P_1$ sends the input-wire labels for his input in $S_j$ where $j \in E$. If some of them are invalid, $P_2$ aborts.
5. $P_2$ evaluates the XOR-gadgets in the sets $S_j$, $j \in E$ and compares the results to the outputs sent by $P_1$. If the outputs are not the same, $P_2$ aborts.

**Evaluation:** $P_2$ evaluates all the garbled circuits $gc_j$ where $j \in E$. He takes the majority to be his output.

[a] Recall that if the length of the concatenated labels is too long, $P_1$ can instead commit on a short seed as described in Section 3.

Figure 4: A Fully-secure 2PC Protocol.

We now need to argue that $P_1$'s view in the ideal and real executions are indistinguishable.[3] Except for the OT messages which are indistinguishable by the security of the OTs, the above simulated execution is distributed as the real one, except for two cases: (1) When $P_1$ successfully cheats (and then $P_2$'s output in the real execution would be different than in the ideal model). However, in order to cheat successfully $P_1$ must cheat in the majority of the evaluated sets, and as discussed above this happens with $1-neg(t)$ probability. (Indeed, $P_1$ can cheat also by guessing the output-wire labels of $P_2$'s XOR-gadgets, but that can work with probability $2^{-s}$ if the labels are of length $s$.) (2) When $P_1$ cheats in the OTs for $P_2$ to learn his inputs, in a way that is distinguishable from the simulated execution (since the simulator picks random inputs). However, by the analysis of [17], this can only happen with probability $neg(t)$ as well.

$P_2$ IS CORRUPTED. The simulator $\mathcal{S}$ does the following:

- Picks at random a subset $E$ and a random permutation of it $\pi(E)$.
- Emulates an honest $P_1$ until the end of stage Oblivious Transfer (See Figure 4). Learns $P_2$'s inputs to the BOT as done by the simulator of [24].
- Calls the trusted party with $P_2$'s input and receives the output $output$.
- Constructs the sets such that for $j \in E$, $gc_j$ outputs the constant $output$, and for $j \notin E$, $gc_j$ is a legal garbling.
- Emulates $P_1$ in the Cut-and-choose stage, until step 5. Learns $P_2$'s $v_j^2$-s.
- Rewinds to step 4 and picks $v_j^1$-s such that $\pi(E)_j = v_j^1 + v_j^2$.
- Emulates $P_1$ with a random input until the end.

Recall that if $P_2$ creates illegal XOR-gadgets, then $P_1$ always catches him since they always open all those gadgets and their corresponding OTs.

The only parts in which the simulation is different than the execution in the real model is where the simulator constructs the *fake* garbled circuits. However, this difference is indistinguishable by the results of [18, 2]. (This can be done, e.g., by setting the output gates to be constant gates of the output $output$. Then, by the security of the garbling scheme, this change is indistinguishable.) Also, the OT messages are indeed for different inputs, but the security of the OT implies that this difference is indistinguishable.

## C.2 Handling Two-Output functions

As discussed in Section 1.1, we have two (related) solutions for handling two-output functions. Here we describe the second one which allows circuit streaming and computation in parallel (e.g., as done in [15]).

We augment the garbling stage as follows. Let $\mathrm{OUT}_1$ be the set of $P_1$'s output wires. For each $k \in \mathrm{OUT}_1$, $P_1$ picks two random strings $w_{k,0}, w_{k,1}$. In addition to the garbled circuits, $P_1$ garbles $t \cdot |\mathrm{OUT}_1|$ identity-gates. The garbled identity-gate $ig_{j,k}$ for garbled circuit $gc_j$ and output-wire $k$ is the garbled version of an AND gate that receives the same input twice and has the output-wire labels $w_{k,0}, w_{k,1}$. (In practice, only two encryptions are needed: $\mathsf{Enc}(\mathsf{label}(gc_j,k,0), w_{k,0})$ and $\mathsf{Enc}(\mathsf{label}(gc_j,k,1), w_{k,1})$.) $P_1$ does not send those garbled identity-gates, but only sends $t$ commitments, one for each circuit committing to all its garbled identity-gates.

Now, the players execute the protocol from above. They follow the protocol upto the input equality check stage. Then, $P_1$ decommits the garbled identity-gates *only* for the circuits being evaluated. $P_2$ uses the output-wire labels from the evaluation stage to evaluate the identity-gates, takes the majority (taking into account the output-wire labels of $P_1$'s output) and sends a commitment on the output-wire labels for $k \in \mathrm{OUT}_1$ to $P_1$. $P_1$ decommits all the remaining garbled identity-gates, and $P_2$ verifies they were constructed properly (or otherwise

[3] Since we have two different security parameters, $t$ and $s$, the executions are actually $(s,t)$-indistinguishable [19].

aborts). Note that for the opened sets, $P_2$ has both labels, so essentially he concludes, again from the cut-and-choose, that the identity-gates are correct for the majority of the circuits. If everything was correct, $P_2$ decommits his commitment and $P_1$ checks that the labels are legal output labels.

As discussed in Section 3, sending a commitment on the output labels does not suffice. Also, since in the simulation of $P_2$ the simulator has to commit and later decommit to output labels he never saw, the commitment has to be equivocal. For that, we can either use Pedersen's commitment as a trapdoor commitment [3] and require $P_2$ to prove that he knows the decommitment (See appendix D.1 for more references on how to instantiate such commitments) , which requires $O(q_1)$ expensive operations, or, in the Random Oracle model, to commit using one call to the hash function (that is modeled as the random oracle) without additional ZKPoKs.

The above protocol provides authenticity of the output. In case privacy of the output is also needed, one can modify the circuit being evaluated in the standard way of XORing the output with a random string.

# D More on Our $\epsilon$-CovIDA Constructions

## D.1 Detailed Construction and Proof of the Protocol from Section 4.2

Before we present the detailed construction (Figure 5), we note that a few interesting issues arise in the simulation-based proof of the protocol that do not exist in the previous standard 2PC constructions. For example, in the simulation-based proofs of previous 2PC constructions, the random challenge is used for checking only one player, the garbler. However, here we use the same challenge for checking both players. This prevents us from using regular commitments everywhere and constructing the simulation using the standard *commit, decommit and rewind* operations. Roughly speaking, the challenge is to construct two different simulators (for the two corruption cases) that can open the coin-toss to any challenge value.

To overcome these issues we use *trapdoor commitments* in some steps of the protocol (i.e. when $P_1$ commits to his coins and when he commits to his garbled sets). These special commitments have the property that given a trapdoor, a commitment can be decommitted to any message, or more formally, let $\mathsf{Com}_{ck}(m, r)$ be a commitment on message $m$ using randomness $r$ and commitment key $ck$. Then a party who knows the trapdoor $ct$ can successfully decommit $\mathsf{Com}_{ck}(m, r)$ to whatever message $m'$ it wants. Such commitments can be constructed efficiently from a variety of assumptions such as DDH and RSA (e.g. See [3]). The intuition is that each player generates a pair of a public key/trapdoor to a trapdoor commitment scheme, and proves using a zero-knowledge proof of knowledge protocol (ZKPoK) that he knows the trapdoor. Each player then uses the other player's public key to commit to his values. In the simulation, the simulator can extract the trapdoor and open the commitment to an appropriate value of its choice.

One option is to use DDH based trapdoor commitment and standard ZKPoK of discrete-log (see [3]). Then, the overhead introduced here is only a (small) constant number of exponentiations.

Figure 5 presents our protocol in detail.

### D.1.1 Proof Sketch of Security of Protocol of Figure 5

Let $\mathcal{A}$ be an adversary controlling $P_1$ in the execution of the protocol in the $\mathcal{F}_{2SET}^{l}$-hybrid world. We describe a simulator $\mathcal{S}$ that runs $\mathcal{A}$ internally and interacts with the trusted party that computes $f$. $\mathcal{S}$ does the following:

1. Invokes $\mathcal{A}$ and emulates honest $P_2$ with random inputs $y, m_2$ until the end of the stage "Committing to the sets and inputs". During the execution it extracts the $Z_j^1$ and $\mathrm{H}(S_j^1)$ from the ZKPoK and records $\mathcal{A}$'s inputs to BCOT1 and BCOT2 (as done in [26]). Also, it extracts the trapdoor $ct_1$ from $\mathcal{A}$.

2. Checks if some of the sets are problematic, which means that $Z_j^1$ is not consistent with $\mathrm{H}(S_j^1)$ or some of $P_1$'s inputs to the BCOTs (either for the BCOTs for $P_2$ to learn his input, or for $P_1$ to learn the inputs of $r_j$).

   If more than one set is incorrect,

**We describe $P_1$'s actions in the protocol. The protocol is symmetric, hence the same steps take place for $P_2$ as well.**

**Garbling:**
Let $C_1, C_2, C_1', C_2', \alpha(\cdot)$ as defined in Section 4.2, where $P_1$'s input is $x' = x \circ m_1$ and $P_2$'s is $y' = y \circ m_2$ and $m_1, m_2$ are chosen at random.
For $j = 1, \ldots, t$, player $P_1$ picks a random string $Z_j^1$ and uses it as a key for a PRF to generate $z_j^1, z_j^{1'}$ (of length $s$) and $r_j^1$ (of length $|\mathrm{INP}_1|$). Then $P_1$ computes the set $S_j^1$ containing:

1. Garbled circuits $gc_j^1 = \mathsf{Garb}(C_1, z_j^1)$ and $xg_j^1 = \mathsf{Garb}(C_1', z_j^{1'})$.
2. The input-wire labels corresponding to $r_j^1$ in $gc_j^1$.

**Oblivious Transfer:**
Players execute two batch committing-OT where $P_1$ is the sender, with all the input-wire labels for $gc_j^1$ and $xg_j^1$. More specifically, in the first execution, BCOT1 is used. $P_1$'s input is $|\mathrm{INP}_2|$ sets of $t+1$ pairs, i.e. $\big(\ \mathsf{label}(gc_j^1, k, 0), \mathsf{label}(gc_j^1, k, 1)\ \big)$ for $k \in \mathrm{INP}_2$ and $j \in [t]$, and $\big(\ \mathsf{label}(xg_1^1, k, 0) \circ \cdots \circ \mathsf{label}(xg_t^1, k, 0), \mathsf{label}(xg_1^1, k, 1) \circ \cdots \circ \mathsf{label}(xg_t^1, k, 0)\ \big)$ for $k \in \mathrm{INP}_2$, and $P_2$ uses his actual input bits. In the second execution, using BCOT2, $P_1$ inputs one set of $|\mathrm{INP}_2| \cdot t$ pairs $(\ \mathsf{label}(xg_j^1, \alpha(k), 0), \mathsf{label}(xg_j^1, \alpha(k), 1)\ )$ for $k \in \mathrm{INP}_2$, $j \in [t]$, and $P_2$ inputs the bits of his inputs $r_j^2$.
We note that $P_1$ is yet to send the labels for his input wires in $gc_j^1$.

**Committing to the sets and inputs:**

1. $P_1$ generates a key pair $(ck_1, ct_1)$ for a trapdoor commitment where $ct_1$ is the trapdoor and $ck_1$ is the public key. He sends $ck_1$ to $P_2$ and proves to him, using ZKPoK, that he knows the corresponding trapdoor. ($P_2$ does the same with $ck_2, ct_2$.)
2. For $j = 1, \ldots, t$, $P_1$ sends the commitments $\mathsf{Com}(Z_j^1), \mathsf{Com}_{ck_2}(\mathrm{H}(S_j^1))$ along with ZKPoK that he knows the corresponding messages $Z_j^1$ and $\mathrm{H}(S_j^1)$.

**Cut-and-choose:**
They pick a random $e \in [t]$ in the following way (this part is done only once):

1. They both toss $t$ coins.
2. $P_1$ sends a commitment on his coins $\mathsf{Com}_{ck_2}(coins_1)$.
3. $P_2$ sends his coins $coins_2$.
4. $P_1$ opens the decommitment and they both set $coins = coins_1 \oplus coins_2$.
5. They use $coins$ to pick (uniform) $e \in [t]$.

$P_1$ sends the $t$ sets $S_j^1$ to $P_2$, and decommits their $\mathsf{Com}_{ck_2}(\mathrm{H}(S_j^1))$.

**Checking Opened Circuits:**
For $(S_j^1, S_j^2)_{j \neq e}$, the players send to each other: 1) $Z_j^i$ and a decommitment of $\mathsf{Com}(Z_j^1)$ , 2) The labels they have learned from BCOT2 for $r_j^i$. For the opened sets, each player verifies that everything was generated properly from $Z_j^i$, and that the other player used the same $r_j^i$ for $gc_j^i$ and $xg_j^{3-i}$. Then, each player decommits all the inputs he has used as the sender in BCOT1 for all the opened sets and the other player verifies that all these values are consistent with the opened circuits. (I.e., $P_1$ decommits his inputs in BCOT1 for $gc_{j \neq e}^1$.) If some of the sets were not constructed properly, the players abort outputting $\perp$.

**Evaluation and Input-equality check:**
Each player sends his input-wire labels for the $e$-th circuit and they evaluate the circuits and the XOR-gadgets.
$P_1$ sends to $P_2$ a commitment on the concatenation of the labels he obtained from evaluation of $xg_e^2$ (or to a random string if there was a problem with the XOR-gates evaluation) along with a ZKPoK that he knows the decommitment. Next, $P_2$ sends the randomness he used for garbling the XOR-gadget $xg_e^2$, and decommits all his inputs as the sender to BCOT2, including the ones of $xg_j^2$, and his inputs $\big(\ \mathsf{label}(xg_1^1, k, 0) \circ \cdots \circ \mathsf{label}(xg_t^1, k, 0), \mathsf{label}(xg_1^1, k, 1) \circ \cdots \circ \mathsf{label}(xg_t^1, k, 0)\ \big)$ for $k \in \mathrm{INP}_2$ to BCOT1. $P_1$ verifies that the XOR-gadget was constructed properly and consistently with the BCOT2 inputs (otherwise, outputs $\perp$) and decommits his commitment to $P_2$.
$P_2$ checks that the output-wire labels he received are valid (i.e. generated by him for these gates) and compares them with the output-wire labels he got from his evaluation of the corresponding XOR-gates. If there is a problem, he outputs $\perp$.
(Recall that the same process goes in both directions, one for $P_1$'s inputs and one for $P_2$'s inputs.)

**Equality-Testing:**
They call the Equality Testing functionality with the output bits of $f(x, y) \oplus m_1 \oplus m_2$ of the $e$-th garbled circuits (including the labels as described earlier). If the answer is False, they abort.

**Output Unmasking:**
$P_1$ sends $m_1$ and the labels that correspond to $m_1$ in $gc_e^2$ and $P_2$ verifies that the labels indeed correspond to $m_1$. ($P_1$ does the same with $m_2$ and aborts if there is a problem.) Then, $P_1$ computes his output by XORing the output of $gc_e^2$ with $m_1 \oplus m_2$.

Figure 5: $1/t$-CovIDA Protocol.

- Sends $\mathsf{cheat}_1(1)$ to the trusted party (and since $1 > 1/t$, the trusted party would send $\mathsf{corrupted}_1$).
- Emulates honest $P_2$ until the end of the protocol. (Note that $P_2$ will abort.)

If all are correct,

- Calls the trusted party with $\mathcal{A}$'s input $x$ (learned from BCOT1) and receives the output $output$.
- Continues the protocol emulating honest $P_2$, but replaces $P_2$'s garbled circuit $gc_e^2$ with one that always outputs $output \oplus m_1 \oplus m_2$ (for fixed $output, m_1, m_2$) with labels that are consistent with previous steps. $\mathcal{S}$ computes the hash of $P_2$'s sets after this replacement and uses $ct_1$ to decommit successfully.
- Proceeds with the protocol emulating honest $P_2$. If $\mathcal{A}$'s input to the "Equality-Testing" is the right output of $gc_e^2$ and the corresponding labels in $gc_e^1$, the simulator returns True. Else, it returns False.
- Proceeds with emulating honest $P_2$, revealing $m_2$ at the end. Sends $\mathsf{abort}$ if $\mathcal{A}$ aborts or sends invalid labels for $m_1$.

Otherwise (i.e. if exactly one set is incorrect),

- Sends to the trusted party $\mathsf{cheat}_1(1/t)$.
- If the trusted party returns $\mathsf{corrupted}_1$, $\mathcal{S}$ picks random $coins_2$ such that $P_1$ will be caught later in the opening stage ($coins_1$ can be learned by rewinding $\mathcal{A}$). Emulates honest $P_2$ until the end. (Note that $P_2$ will abort.)
- If the trusted party returns $\mathsf{undetected}$ and the output $output$,
  - Picks random $coins_2$ such that $P_1$ will *not* be caught in the opening stage.
  - Continues the protocol emulating honest $P_2$, but replaces $P_2$'s garbled circuit $gc_e^2$ with one that always outputs $output \oplus m_1 \oplus m_2$ (for fixed $output, m_1, m_2$) with labels that are consistent with previous steps. $\mathcal{S}$ computes the hash of $P_2$'s sets after this replacement and uses $ct_1$ to decommit successfully.
  - Let $P_1$'s input to the "Equality-Testing" be $w_1$. $\mathcal{S}$ sends to the trusted party the function $g$ that has hardcoded the circuit $gc_e^1$, the input labels that $\mathcal{A}$ used in the BCOT1 for $P_2$'s inputs of $gc_e^1$, the labels that $\mathcal{A}$ sent for his inputs, the value $m_2$, the output-wire labels of $gc_e^2$, and $w_1$. The function evaluates the circuit with the real $P_2$'s input, using the corresponding input-wire labels, determines what string $w_2$ would honest $P_2$ in our protocol be using after obtaining the output of the evaluation, compares it with $w_1$ and outputs 1 if they are equal and 0 otherwise. If the trusted party sends $\mathsf{abort}$, $\mathcal{S}$ sends False to $\mathcal{A}$ and emulates $P_2$ aborting. If the trusted party did not abort, $\mathcal{S}$ sends True to $\mathcal{A}$ and proceeds emulating honest $P_2$ revealing $m_2$. $\mathcal{S}$ sends $\mathsf{abort}$ if $\mathcal{A}$ aborts or sends invalid labels for $m_1$.

When the $e$-th set is correct, the adversary cannot change his input since it is checked by the XOR-gadgets (since we know that the $r_e^1$ is the same for both evaluations), and he cannot reveal successfully at the end $m_1'$ that is different than $m_1$ because the corresponding labels are random. When the $e$-th set is incorrect, the adversary can indeed use different inputs for both evaluations or incorrect garbled circuit $gc_e^1$. The output he gets from the evaluation of $gc_e^2$ looks for him the same as the output in the real execution, and the only information he gets about the output of $gc_e^1$ is from the result of the Equality-Testing. However, since the simulator computes this output with the assistance of the trusted party (by evaluating $gc_e^1$ with the actual $P_2$'s input), the result of the Equality-Testing looks the same as in the real execution.

Inspecting the simulation shows that the only parts that are different than the real execution are (1) Where the simulator sends a fake garbled circuit with a fixed output. However, [18, 2] show that the two views are indistinguishable (under minor changes to the circuit in use); (2) $P_2$'s inputs to the BCOTs. However, these are indistinguishable by the security of the BCOTs.

The simulation for the case where $\mathcal{A}$ controls $P_2$ is the same, except that for changing the coins $\mathcal{S}$ now needs to utilize the trapdoor $ct_2$.

## D.2 Reducing the Number of Circuits

A shortcoming of the previous protocol is that the probability of leakage decreases slowly with the number of circuits $t$. In particular, aiming for a probability of leakage of $1/1000$ would require the exchange of a thousand garbled circuits which is not practical. A more desirable goal is to make the leakage probability exponentially small in $t$ while the protocol cost still grows linearly in $t$.

The standard solution for reducing the probability of cheating in cut-and-choose protocols is to issue $t$ garbled circuits, open a constant fraction of them (e.g. half) and verify that they were constructed properly, and evaluate the rest. Using this method (ignoring the challenges in enforcing consistency of inputs and the OTs) we have that the majority of the evaluated circuits are correct, and thus the majority output is the correct output with all but negligible probability in $t$. (See [26] for a concrete analysis.)

However, if we try to combine this approach and dual execution, it is not clear how to perform the equality testing at the end, since now each player evaluates multiple circuits with different output-wire labels, some of which may encode the wrong result.

To overcome this issue we need a solution that ensures that the output labels are (1) the same for all the evaluated circuits and (2) unpredictable (i.e. hard to guess when not learned through evaluation), as is the case with output-wire labels in the standard garbled circuits. One possibility is to embed a carefully designed *one-time MAC* in the circuits being garbled and evaluated. The overhead of this solution, however, is too high to be of practical interest. Next we discuss an alternative and very efficient solution based on *identity-gates* and a *two-stage opening*.

**An efficient solution via identity-gates.** For each $k \in \mathrm{OUT}$, each player $P_i$ picks two random strings $w^i_{k,0}, w^i_{k,1}$. Note that these random strings are the same for all $t$ circuits. In addition to the garbled circuits and XOR-gadgets, for each set it also garbles $|\mathrm{OUT}|$ *identity-gates*. The garbled identity-gate $ig^i_{j,k}$ for garbled circuit $gc^i_j$ and output-wire $k$ is the encryptions $\mathsf{Enc}(\mathsf{label}(gc^i_j, k, 0), w^i_{k,0})$ and $\mathsf{Enc}(\mathsf{label}(gc^i_j, k, 1), w^i_{k,1})$. The players do not send those garbled identity-gates as part of the sets, but only send one commitment per set, committing to all the garbled-identity gates for that set.

Now, the players execute the protocol from Section 4.2, but open only a constant fraction of the sets (without opening the commitments on the identity-gates). Then, each player decommits the garbled identity-gates for the circuit-pairs being evaluated. Each player uses the output-wire labels from the circuit evaluations to evaluate the identity-gates, and then takes the majority to be his input to the Equality Testing functionality (or a random string if there is no majority). However, if the identity-gates were invalid, this step might reveal information. Thus, the players run only the first stage of the Equality Testing functionality (and essentially commit to their inputs). Then each player decommits all the remaining garbled identity-gates he generated and opens them, while the other player verifies they were constructed properly (or otherwise aborts). If everything was correct, they execute the second stage of the Equality Testing functionality and proceed accordingly.

The resulting protocol adds only $O(t \cdot |\mathrm{OUT}|)$ inexpensive operations since for each output-wire the players compute $t$ garbled identity-gates.

In addition to the above modifications, we require player $P_i$ to pick a random $Z'_i$ in the beginning of the protocol, and use it as a PRF key for generating the strings $w^i_{k,0}, w^i_{k,1}$ and the randomness used for committing on the identity-gates. Each player commits on his $Z'_i$ and proves, using ZKPoK, that he knows it. (This allows the simulator to extract $Z'_i$ and check if the commitments are consistent with the set of garbled circuits and the BCOTs.) When the players decommit all the garbled identity-gates, they also decommit $Z'_i$ and verify that the all commitments and identity-gates were generated correctly using this value.

The last modification is regarding the coin-tossing step. We replace the coin-tossing step of the protocol from Figure 5 with the one from Figure 4, and change it to use trapdoor commitments for the same reason explained in Appendix D.1. Specifically, the coin-tossing protocol we use is:

- Parties initialize a boolean string $\rho$ of length $t$ to be all zeros.
- For $j = 1, \ldots, (t \cdot c)$, each player $P_i$ picks a random value $v^i_j \in [1..(t - j + 1)]$.

- $P_1$ sends a commitment $\mathsf{Com}_{ck_2}(v_1^1 \circ v_2^1 \circ \cdots \circ v_{t \cdot c}^1)$.
- $P_2$ sends his values $v_1^2, \ldots, v_{t \cdot c}^2$.
- $P_1$ decommits and reveal $v_1^1, \ldots, v_{t \cdot c}^1$.
- For $j = 1, \ldots, (t \cdot c)$, let $v = ((v_j^1 + v_j^2) \mod (t - j + 1)) + 1$ and let $k$ be the $v$-th zero bit of $\rho$. Set $\rho_k = 1$.
- Let the set $E$ be $\{j | \rho_j = 1\}$. $E$ would be the set of indexes in which the players will evaluate (and open all sets with indexes not in $E$).

*Proof Sketch of Security.* The simulation is very similar to the one from Appendix D.1 except for some small changes.

Let $\mathcal{A}$ be an adversary controlling $P_1$ in the execution of the protocol in the $\mathcal{F}_{2SET}^l$-hybrid world. $\mathcal{S}$ does the following:

1. Invokes $\mathcal{A}$ and emulates honest $P_2$ with random inputs $y, m_2$ until the end of the stage "Committing to the sets and inputs". Extracts $Z_1', Z_j^1$ and $\mathrm{H}(S_j^1)$ from the ZKPoK and records $\mathcal{A}$'s inputs to BCOTs. Also, it extracts the trapdoor $ct_1$.
2. Checks if some of the sets are problematic, which means that if given $Z_1'$, $Z_j^1$ is not consistent with $\mathrm{H}(S_j^1)$ or some of $P_1$'s inputs to the BCOTs, or with the commitment on the identity-gates, then the set $j$ is problematic. Let $\mathcal{B} = \{j | \text{set } j \text{ is problematic}\}$.

   If all are correct,

   - Calls the trusted party with $\mathcal{A}$'s input $x$ (learned from the BCOT1) and receives the output $output$.
   - Continues the protocol emulating honest $P_2$, but replaces $P_2$'s garbled circuit $gc_e^2$ with one that always outputs $output \oplus m_1 \oplus m_2$ (for fixed $output, m_1, m_2$) with labels that are consistent with previous steps. $\mathcal{S}$ computes the hash of $P_2$'s sets after this replacement and uses $ct_1$ to decommit successfully.
   - Proceeds with the protocol emulating honest $P_2$. If $\mathcal{A}$'s input to the "Equality-Testing" is the right output of $gc_e^2$ and the corresponding labels in $gc_e^1$, the simulator returns True. Else, it returns False.
   - Proceeds with emulating honest $P_2$, revealing $m_2$ at the end. Sends $\mathsf{abort}$ if $\mathcal{A}$ aborts or sends invalid labels for $m_1$.

   If more than $|E|$ of the sets are incorrect ($|E| < |\mathcal{B}|$),

   - Sends $\mathsf{cheat}_1(1)$ to the trusted party (and since $1 > 1/t$, the trusted party would send $\mathsf{corrupted}_1$).
   - Emulates honest $P_2$ until the end of the protocol. (Note that $P_2$ will abort.)

   If less than $|E|$ of the sets are incorrect ($|\mathcal{B}| \leq |E|$),

   - Set $\epsilon' = \dfrac{\binom{t - |\mathcal{B}|}{t - |E|}}{\binom{t}{t - |E|}}$. (This is the probability of not being caught for the given set of problematic sets.)
   - Sends to the trusted party $\mathsf{cheat}_1(\epsilon')$.

- If the trusted party returns $\mathsf{corrupted}_1$, $\mathcal{S}$ makes sure that the subset $E$ will be chosen such that $P_1$ will be caught later. Emulates honest $P_2$ until the end. (Note that $P_2$ will abort.)

  We now describe how $E$ is chosen. Let $\rho$ be a binary string of length $t$, and let $c$ be the constant fraction of sets we evaluate (i.e., $c = |E|/t$). $\mathcal{S}$ chooses $\rho$ using the following strategy: Pick at random a binary string $\rho_{\mathcal{B}}$ of length $|\mathcal{B}|$ that has at least one zero element. Pick at random a binary string $\rho_{\mathrm{G}}$ of length $t - |\mathcal{B}|$ that has exactly $t \cdot c - \mathsf{HW}(\rho_{\mathcal{B}})$ non-zero elements. Choose $\rho$ such that $\rho : \mathcal{B} = \rho_{\mathcal{B}}$ and $\rho : [t] - \mathcal{B} = \rho_{\mathrm{G}}$, where $x : S$ denotes the substring of $x$ containing all indexes in set $S$.

  Set $E$ to be the set of indexes $\{i|\rho_i = 1\}$. Note that $E$ is uniform over all the challenges that reveal problematic sets.

  Let $\pi(E)$ be a random permutation of the indexes in $E$. In order to decide on $E$, for each round $j$ in the protocol from above, $\mathcal{S}$ does the following:

  - Receives $P_1$'s commitment.
  - Sends random $v_j^2$-s and receives $P_1$'s $v_1^1, \ldots, v_{t \cdot c}^1$.
  - Rewinds $\mathcal{A}$ and sends him $v_j^2 = \pi(E)_j - v_j^1 \mod (t - j + 1)) + 1$ for $j = 1, \ldots t \cdot c$.
- If the trusted party returns $\mathsf{undetected}$ and the output $output$,
  - Makes sure that all the malicious set/inputs are in $E$, and also, replaces $P_2$'s garbled circuits in the set $E$ with ones that always output a fake output $z \oplus m_1 \oplus m_2$. (Here we use the same process for picking $E$ as before, but instead we take $\rho_{\mathcal{B}}$ to be all ones.) $\mathcal{S}$ computes the hash of $P_2$'s sets after this replacement and uses $ct_1$ to decommit successfully.
  - If $|\mathcal{B}| < |E|/2$, $\mathcal{S}$ sends to the trusted party the function $g$ that always returns 1, proceeds with the protocol emulating honest $P_2$. If $\mathcal{A}$'s input to the "Equality-Testing" is the right output of $P_2$'s circuits and the corresponding labels, the simulator returns True. Else, it returns False. (Note that since the majority of the sets are good, $\mathcal{S}$ can extract the output labels from $P_1$'s sets.)
  - If $|E|/2 \leq |\mathcal{B}|$, let $\mathcal{A}$'s input to the "Equality-Testing" be $w_1$. $\mathcal{S}$ sends to the trusted party the function $g$ that has hardcoded the circuit $gc_e^1$ for all $e \in E$, the input labels that $\mathcal{A}$ used in BCOT1 for $P_2$'s inputs of $gc_e^1$, the labels that $\mathcal{A}$ sent for his inputs, the value $m_2$, the output-wire labels of $gc_e^2$, and $w_1$. The function evaluates the garbled circuits with the real $P_2$'s input, using the corresponding input-wire labels, computes what string $w_2$ would have honest $P_2$ in our protocol be using after he gets the outputs of these evaluations, compares it with $w_1$ and outputs 1 if they are equal and 0 otherwise. If the trusted party sends $\mathsf{abort}$, $\mathcal{S}$ sends False to $\mathcal{A}$ and emulates $P_2$ aborting. If the trusted party does not abort, $\mathcal{S}$ sends True to $\mathcal{A}$ and proceeds emulating honest $P_2$ revealing $m_2$. $\mathcal{S}$ sends $\mathsf{abort}$ if $\mathcal{A}$ aborts or sends invalid labels for $m_1$.

The rest of the proof is as in Appendix D.1.

■

# E Authenticated Computation with Private Input

In some cases, only one of the players' inputs should remain private. E.g., in anonymous credential protocols, if checking credentials can be done publicly then the only secret input to the protocol is the actual user's credentials. Other applications can be targeted-advertising, where only the client's preferences are secret, and any Zero-Knowledge Proof, in which only the prover's input (the witness) is private.[4]

Say $P_1$'s input $x$ should remain private, but the function in use $f$ is known to both players and $P_2$'s input $y$ can be revealed at the end of the protocol. The players wish to compute $f(x, y)$ while maintaining correctness

[4] We have learned that a result similar to ours was independently obtained by [12].

and privacy of $x$ even in the case one of the players is malicious. Indeed, realizing this functionality can be done using any fully-secure 2PC. However, there is no natural way to take advantage of the fact that $y$ can be revealed at the end of the protocol. (Note that we require that both inputs are independent of each other, as required implicitly by the standard 2PC security notion.)

A very efficient protocol for the above functionality, using the ideas presented earlier for handling two-output functions, is the following:

- $P_2$ picks a short seed $z$ for a pseudo-random function (PRF), and generates $gc = \mathsf{Garb}(C_f, z)$. (Recall that the output-wire labels include random strings and the actual bits.)
- $P_1$ and $P_2$ execute $|x|$ (fully-secure) OTs for $P_1$ to learn the labels that correspond to his input-wires. All the randomness $P_2$ uses in the OTs is derived from the PRF used with the seed $z$.
- $P_2$ sends $gc$ along with the labels that correspond to his input-wires (for $y$).
- $P_1$ evaluates $gc$ and gets the actual output bits (denote by $b_1, b_2, \ldots, b_q$) and their random labels (denote by $l_1, l_2, \ldots, l_q$).
- $P_1$ sends a commitment $\mathsf{Com}(b_1 \circ \cdots \circ b_q \circ l_1 \circ \cdots \circ l_q)$ and a ZKPoK that he knows its decommitment.
- $P_2$ sends $z$.
- $P_1$ verifies that $gc$ was garbled correctly and the OTs were consistent with $gc$. (This is done by emulating honest $P_2$ with the seed $z$.)
- If everything was properly constructed, $P_1$ decommits his commitment, $P_2$ checks that all labels $l_i$-s are indeed correct output-wire labels, and if so, they both output the $b_i$-s.

The above protocol requires only *a single* garbled circuit and a small constant number of rounds. When $\mathsf{Com}(\cdot)$ is implemented by a Random Oracle, only a single hash is needed for the commitment. Alternatively, a trapdoor commitment with ZKPoKs can be used with the cost of $O(q)$ expensive operations.