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

Payman Mohassel* Ben Riva†

February 3, 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 definitions that are more suitable for use in practice.

- We design an efficient fully-secure malicious 2PC protocol for two-output functions that only requires $O(t|C|)$ symmetric-key operations (with small constant factors) 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 ϵ (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 2PC based on cut-and-choose. E.g., the price of strengthening a covert 2PC to satisfy our notion (to obtain full correctness and maximum leakage of a single bit), is only $\frac{1}{\epsilon}$ additional garbled circuits.

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. These techniques may be of an independent interest.

*University of Calgary, Canada.
†Tel Aviv University, Israel.

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).

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* [Yao86, LP09]: 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 [KS08], reducing communication [GMS08, PSSW09], and designing tailored circuits [HEKM11]).

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. [JS07, NO09] 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 the cut-and-choose method. (E.g., see implementations by [PSSW09, SS11, KSS12].) 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 [LP11, SS11] 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, how to make sure Alice is using the same input in all circuits, [MF06, LP07] present two methods that require $O(n_1 \cdot t^2)$ 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. ([Woo07] shows how to reduce this asymptotic overhead, but with large constants even for small security parameters.) [MF06, LP11, SS11] show alternative methods that require $O(n_1 \cdot t)$ 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, [LP07] presents a method that requires $O(\max(4n_2, 8t) \cdot t)$ expensive cryptographic operations (specifically, oblivious transfer), where n_2 is the length of Bob's input. [LP11, SS11] introduce alternative methods that require $O(n_2 \cdot t)$ expensive cryptographic operations.

To address the last issue, of verifying the computation output, [LP07] 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 $O(q_1 \cdot t)$ gates, where q_1 is Alice's output length (i.e. overall overhead of $O(q_1 \cdot t^2)$ inexpensive operations). [SS11] suggests a solution that requires the use of digital signatures and a witness-indistinguishable proof, resulting in a total overhead of $O(q_1 \cdot t)$ expensive operations.

In the covert case, the techniques are similar, although usually the issue of garbler's input consistency is not relevant since there is only one circuit to evaluate [GMS08, AL10].

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.

[MF06] 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. More importantly, the resulting protocol is much more efficient than the best known protocols with full security against malicious players. (See [HKE12] for an optimized variant of the protocol of [MF06] and its performance.)

The idea behind the protocols of [MF06, HKE12] 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, learns the actual output bit of the computation b, and since she knows the labels for the output-wires of gc_2, she 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, with only two garbled circuit executions and the associated oblivious transfers. 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 check. We call this scenario *Input-Dependent Abort* (IDA).

	P_1's input	P_2's input	Two-output Overhead
[LP07]	inexpensive$(t^2 n_1)$	expensive$(\max(4n_2, 8t))+$ inexpensive$(t \cdot \max(4n_2, 8t))$	inexpensive$(t^2 q_1)$
[LP11, SS11]	expensive(tn_1)	expensive(tn_2)	expensive(tq_1)
Our protocol	inexpensive(tn_1)	inexpensive$(t \cdot \max(4n_2, 8t))$	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. The number of base OTs in the OT extension is omitted as it is independent of the circuit and input sizes.

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.

1.1.1 Optimal Fully-Secure 2PC Based on Cut-and-Choose

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 [MF06, LP07], and only requires inexpensive operations in contrast to the solution of [SS11]. Second, we show how to combine the efficient OT extension of [NNOB12] with the technique of [LP07], to get an efficient realization of OT in which the sender is committed to his inputs. This primitive is used to solve the evaluator's input consistency issue with complexity of $O(t \cdot \max(4n_2, 8t))$ 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 [SS11] 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 [MF06, LP07, LP11, SS11] (except for [Woo07], 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 highly depends on the efficient OT extension of [NNOB12], 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 [NNOB12] is in the Random Oracle Model (ROM) and therefore our construction inherit this assumption as well.

We remark that our proposed solutions can be modified to work with any of the existing garbled-circuit optimization techniques of [KS08, GMS08, PSSW09, HEKM11, KSS12]. Furthermore, in Appendix C.2 we show how to transform the protocol to be *universally composable* secure [Can01] by adding only t oblivious transfers.

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 those 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.) Without loss of generality, assume P_1 needs to learn a single output bit. Extending our solution to any number of output bits is straightforward.

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 label. (Note that since the opening step is not performed yet, P_2 cannot guess the unknown output-wire label 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 label. 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, 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.

The above solution only requires a single commitment (per output wire), and can be applied to all 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 [HEKM11, KSS12]. In Section 3.2, we describe a second variant of this protocol that is compatible with those techniques. The overhead of this variant is only $t \cdot q_1$ additional commitments.

XOR-GADGETS AND GARBLER'S INPUT CONSISTENCY. Recall that 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 input 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 the garbled circuits of C_2. These garbled XOR-gadgets will be evaluated by P_1 and on his own inputs. 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) We 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) We 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) We check that $r_i^1 = r_i^2$ in the majority of the evaluated 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 the subset of garbled circuits chosen by P_2. 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 used 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 is only part of our techniques, and in particular, does not guarantee protection against a malicious P_2.

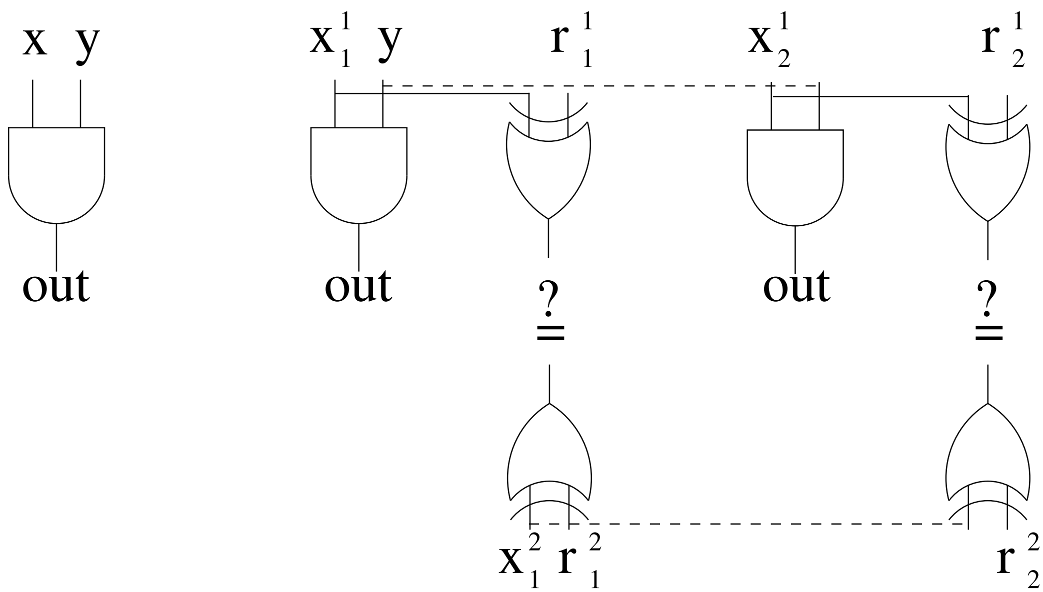

Figure 1: Example of garbling the simple AND circuit on the left that computes the AND between P_1's bit x and P_2's bit y. P_1 garbles the upper circuits and P_2 the lower ones. Specifically, P_1 garbled two AND circuits (i.e., $t = 2$) and 2 XOR-gates, and P_2 garbled two XOR-gates. P_2's input is 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 the Presence of Covert Adversaries

We propose a new security definition that naturally combines security with input-dependent abort of [IKO+11a] (alternatively, security with limited leakage of [MF06, HKE12]), with security against covert adversaries [AL10]. The resulting security guarantee, denoted by ϵ-CovIDA, is a *strict strengthening of covert security* (hence more desirable in practice): In covert security, with probability ϵ both correctness and privacy are gone!

	Overall Complexity	Security
[GMS08]	$\mathsf{inexpensive}(\frac{1}{\epsilon}(\lvert C\rvert + n_2))$	Covert with ϵ-deterrent
[HKE12]	$\mathsf{inexpensive}(2\lvert C\rvert)$	IDA
Section 4.2	$\mathsf{inexpensive}(\frac{1}{\epsilon}(2\lvert C\rvert + n_1 + n_2))$	ϵ-CovIDA
Appendix D.2	$\mathsf{inexpensive}(\log(\frac{1}{\epsilon})(2\lvert C\rvert + n_1 + n_2 + q))$	ϵ-CovIDA

Table 2: Comparison with covert and IDA protocols. n_i the length of P_i's input and q is the output length.

Our definition always guarantees correctness, and with probability ϵ, 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 [MF06, HKE12] 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 the above protocol guarantees correctness similar to [MF06, HKE12], the protocol does not satisfy our security definition. One main problem is that 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 outcome of the final equality check. Note that this attack is successful even if *all* the circuits (including the two being evaluated) 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 $\frac{n}{\epsilon}$ additional XOR gates, where n is the length of the input. We emphasize that compared to the protocol of [HKE12], where the adversary can *always* learn one bit of information, our protocol leaks one bit only with probability ϵ.

The first construction is sufficient for large values of ϵ 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. See Table 2 for a complexity comparison of our protocols with those of the input-dependent abort model and the standard covert model.

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 checks). 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 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 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 $q = q_1 + q_2$.

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

COMMITMENT. Denote by $\mathsf{Com}(m, r)$ the commitment on message m using randomness r. The decommitment of $\mathsf{Com}(m, r)$ is m and r.

We will also use a special type of commitment called *trapdoor commitments*. These can be constructed efficiently from a variety of assumptions such as DDH and RSA (see [Fis01]). Denote by $\mathsf{Com}_{ck}(m, r)$ the commitment on message m using randomness r and commitment key ck, and by m and r the decommitment of $\mathsf{Com}_{ck}(m, r)$. A party who knows the trapdoor ct can commit to $\mathsf{Com}_{ck}(m, r)$, and later on decommit to whatever message m' it wants. Without the knowledge of trapdoor, the commitment scheme functions as a normal commitment scheme with the standard hiding and binding properties.

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 described our protocols. We refer the reader to [LP09, BHR12] for different approaches to creating the garbled circuits.

Denote by $\mathsf{Enc}(sk, m)$ the encryption of message m under secret key sk. Given a boolean circuit C, the garbled circuit consists of the following: For each gate g with input-wires u, v and output-wire w, for each $c_0, c_1 \in \{0, 1\}$, the encryption

$$\mathsf{Enc}^{g,c_0,c_1}(k^u_{c0\oplus r^u} \circ k^v_{c1\oplus r^v}, k^w_c \circ c)$$

where for each wire j, k^j_0, k^j_1 are random labels of length l, and r^j is a random bit (all chosen by the circuit garbler), $c = g(c_0 \oplus r^u, c_1 \oplus r^v) \oplus r^w$, and $\circ$ is the concatenation operator. For simplicity, we require that for any of the circuit's output-wires, r^j is zero (thus, revealing their actual output bits).

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).

Given a garbled circuit gc, we denote by $\mathsf{label}(gc, j, b)$ the label of wire j corresponding to bit value b (i.e. k^j_b of that garbled circuit). 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).

UNDENIABLE OBLIVIOUS TRANSFER. Here, sender S has n sets, each of m pairs of inputs, and receiver R has a vector of input bits $\bar{b} = (b_1, \cdots, b_n)$. An undeniable OT has two stages: The first is similar to the standard OT for many inputs, where R learns the outputs according to his input bits; In the second, both parties request the same subset $I \subseteq [m]$ and R learns the j-th pair for all $j \in I$, in all n sets. We formally define this functionality denoted by $\mathcal{F}^{l,m,n}_{UOT}$ in Figure 4. See a diagram explaining the functionality in Figure 5 of Appendix B. Also in Appendix B, we present realizations of it with different complexities. In our protocols we will specifically use two implementations of UOT which have different properties: The first, UOT_2, works for any I and requires $\mathsf{expensive}(s) + \mathsf{inexpensive}(\max(4n, 8t) \cdot m)$ operations, and the second, UOT_3, works for $I = [m]$ and requires $\mathsf{expensive}(s) + \mathsf{inexpensive}(nm)$ operations.

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 $\mathcal{F}^l_{2SET}$ is split into two stages in order to emulate a commitment on the inputs before revealing the result (we will use this property in 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. In Appendix B we formally define this functionality and present realizations of it with different complexities.

First Stage

Inputs: S inputs n sets, each of m pairs $\{(x_0^{j,z}, x_1^{j,z})\}$ of strings of length l, for $j = 1 \ldots n, z = 1 \ldots m$. R inputs a binary vector $b \in \{0,1\}^n$.

Outputs: For all j and z, R obtains $x_{b_j}^{j,z}$.

Second Stage

Inputs: Both S and R input $\mathsf{Reveal}\ I$.

Outputs: For all j and for each $z \in I$, R obtains $(x_0^{j,z}, x_1^{j,z})$.

Figure 2: $\mathcal{F}_{UOT}^{l,m,n}$.

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

In this section, we describe an efficient 2PC protocol with full security against malicious adversaries. In Section 3.1 we review the main steps of the protocol and highlight the new techniques, considering the case where only P_2 needs to learn the output. In Section 3.2, we show how to extend the ideas in order to handle two-output functions. Due to lack of space, a detailed description of the protocol and the proof of the following theorem appear in Appendix C. (We use the standard security definition of 2PC with malicious players as defined in [Gol04].)

Theorem 3.1. *The protocol from Figure 7 is a secure two-party computation against malicious players. In the hybrid F_{UOT} model, the complexity of the protocol is $O(t \cdot (|C| + n_1))$ inexpensive operations. The number of calls to $\mathcal{F}_{UOT}$ is 3 with $O(t \cdot n)$ inputs overall.*

When realized using UOT_2 and UOT_3, the overall cost is $O(t \cdot (|C| + n))$ inexpensive operations and $O(s)$ expensive ones.

3.1 Overview of the Construction

We recall the high-level description from Section 1.1. Consistency of the Garbler's input is done using the XOR-gadgets. Consistency of the Evaluator's input is taken care of by working with UOT: when the players open a subset of the garbled circuits for checking, P_1 also reveals his inputs to the UOT and P_2 verifies that they are consistent with the opened garbled circuits. By the definition of UOT, P_1 is committed to those inputs, thus his only option to cheat is to call the UOT with values that are different than the ones used for garbling. However, since P_2 checks that option, we are assured that with high probability (in terms of t), most of the evaluated garbled circuits and their corresponding OTs are consistent.

We now describe the main steps of the protocol.

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 $xgc_i = \mathsf{Garb}(C_2, z_i')$, 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 call the UOT functionality 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 the UOT_2 protocol where P_1 acts as the sender and P_2 acts as the receiver. P_1's inputs for UOT are the input labels of input-wire

k 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. Second, they call UOT_3 where P_2 acts as the sender and P_1 acts as the receiver: P_2's inputs are the labels of the input-wires in his XOR-gadgets 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 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 the second UOT can be realized using a regular OT and OT-extension as described in Appendix B.

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.

(In the protocol itself, these UOT calls are done before the garbled circuits are exchanged because this order is needed for the simulation. However, the intuition stays the same.)

Cut-and-Choose (first stage): After the OT-s, 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 simply showing the labels he learned in the second UOT), and then they ask the first UOT functionality to reveal all the inputs it received for the opened circuits. P_2 checks the correctness of the opened circuits/gadgets and verifies that the same r_j-s were used in both gc_j and xg_j.

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 asking the second UOT to reveal all his inputs to it for those gadgets. P_1 checks that all the XOR-gadgets he received were properly constructed, and that the labels are consistent with the UOT answers. If so, P_1 decommits the output-wire labels of the XOR-gadgets to P_2.

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.

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 UOT 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 gc_j and xg_j. 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 second OT is a regular one (i.e. we use UOT_3), thus we can use OT-extension which results in $O(t \cdot |\mathrm{INP}_1|)$ inexpensive operations for checking P_1's input consistency. This is in contrast to the previous (efficient) constructions, which require either $O(t^2 \cdot |\mathrm{INP}_1|)$ inexpensive operations [MF06, LP07], or $O(t \cdot |\mathrm{INP}_1|)$ expensive ones [MF06, LP11, SS11]. (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* input consistency checks is now reduced to the cost of performing a UOT. Previous constructions [LP07, LP11, SS11] 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.

Moreover, as we show in Appendix B, there are several efficient candidates for UOT. More specifically, when the input size of one of the players is small, running the above protocol with that player as the evaluator, and using the UOT of [SS11] results in a very efficient protocol. In cases where both players have long inputs, the UOT construction based on [LP07] could give the most efficient protocol. (Note that using the later option is actually asymptotically optimal, since it needs only a constant number of inexpensive operations per input bit.)

3.2 Handling Two-Output functions

As discussed in the introduction, 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 [KSS12]).

The additional cost over the above protocol is small as summarized in the following theorem.

Theorem 3.2. *Extending the protocol from Figure 7 with the below method results in a secure two-party computation against malicious players for two-output functions. The overhead over the protocol from Figure 7 is $O(t \cdot q_1)$ inexpensive operations.*

We augment the garbling stage as follows. Let 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 final 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 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 ok, P_2 decommits his commitment and P_1 checks that the labels are legal outputs.

The above protocol provides authenticity of the output. In case privacy of the output is also needed, we can modify the circuit being evaluated in a standard way: For each output-wire we add one input bit and one XOR gate. P_1 will pick a bit at random, and xor it with the actual output bit. The above protocol would then be run for this new circuit. The overhead of the resulting protocol is only $O(t \cdot |\mathrm{OUT}_1|)$ inexpensive operations.

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

4.1 The Model

Following [LP07, AL10, GMS08, HKE12], we use the ideal/real paradigm for our security definitions. Loosely speaking, the execution of the protocol in the real model, in which the adversary controls several parties, is compared to an execution in an ideal model in which a trusted third-party computes the output of the function. We say that a protocol is secure if there exists a simulator S such that the view of the adversary in the real model is indistinguishable from a simulated view generated by S in the ideal model.

REAL-MODEL EXECUTION. The real-model execution of protocol Π takes place between players (P_1, P_2), at most one of whom is corrupted by a non-uniform 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 $\mathrm{REAL}_{\Pi,\mathcal{A}(aux)}(x_1, x_2)$ be the output vector of the honest party and the adversary $\mathcal{A}$ from the real execution of Π, 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 $x'_i = \perp$, then the trusted party returns 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 ϵ',
  - The trusted party sends $\mathsf{undetected}$ and $f(x'_1, x'_2)$ to the adversary $\mathcal{A}$.
  - $\mathcal{A}$ responds with an arbitrary boolean function g.
  - The trusted party computes $g(x'_1, x'_2)$. If the result is 0 then the trusted party sends abort to all parties and the adversary $\mathcal{A}$ and halts.

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

Trusted party answers honest parties: If the adversary sends abort in response, the trusted party sends 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* Π *is* secure with input-dependent abort in the presence of covert adversaries with ϵ-deterrent (ϵ-CovIDA) *if for any probabilistic polynomial-time adversary $\mathcal{A}$ in the real model, there exists a non-uniform probabilistic polynomial time adversary $\mathcal{S}$ in the ideal model such that*

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

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 ϵ, 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 ϵ is the deterrence factor). Note that this is not a relaxation in security since the adversary can only increase the probability of itself getting caught. We believe that the way we let the adversary to cheat in the ideal-model with probability smaller than ϵ is of independent interest. Specifically, it could give a different definition of covert security that is more convenient to use in simulation-based proofs. (To get the definition of covert security, replace the steps that are done with probability ϵ' with the steps: (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 the function output.)

A REMARK ON ADAPTIVENESS. 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 protocol and highlight the new techniques. Due to lack of space, a detailed description of the protocol and the proof of the following theorem appear in Appendix D.1.

Theorem 4.2. *The protocol from Figure 9 is ϵ-CovIDA. In the hybrid $(\mathcal{F}_{UOT}, \mathcal{F}_{2SET})$-model, the complexity of the protocol is $O(\frac{2}{\epsilon} \cdot (|C| + n))$ inexpensive operations. The number of calls to $\mathcal{F}_{UOT}$ is 4 with $O(\frac{2}{\epsilon} \cdot n)$ inputs overall.*

Also in Appendix D.2, we show how to modify the protocol to work with a smaller number of garbled circuits (i.e., logarithmic in $\frac{1}{\epsilon}$ instead of linear).

As discussed in the introduction, in the dual-execution protocol of [MF06, HKE12] 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 $t = \frac{1}{\epsilon}$ instead of ϵ since t would be the number of circuits each party garbles. (This is a statistical parameter.)

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. Instead of garbling a single circuit, each player garbles t circuits and sends them to the other player. Denote the circuits generated by player P_i by $gc^i_1, \ldots, gc^i_t$, and denote by the pair (gc^1_j, gc^2_j) a *circuit-pair*. 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 (one circuit from each player) 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. The first issue is that a malicious player can execute a selective-OT attack to learn a bit of information about the other player's input with probability greater than $1/t$. But this can be solved using a UOT with $1/t$ privacy (see Appendix B).

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). In particular, 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 outcome of the Equality Test. Note that this attack is successful even if all the circuits (including the two being evaluated) are constructed correctly. We prevent this attack using the XOR-gadget techniques we discussed earlier along with some enhancements. We discuss the details next:

XOR-gadgets. Based on C, 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) $C'_2(x, r_1) = x \oplus r_1$ evaluate 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 four C_i-s.)

Instead of garbling C, each player P_i generates and sends t garbled circuits for C_i: $gc^i_1, \ldots, gc^i_t$ and t garbled circuits of C'_i: $xg^i_1, \ldots, xg^i_t$. Note that here, in contrast to protocol of Section 3.1, the XOR-gadgets include XOR-gates for the inputs of both players.

After sending the sets of garbled circuits, for each $j \in [t]$, player P_i picks at random a string r^i_j and sends the input-wire labels that correspond to r^i_j in gc^i_j.

OTs for input labels. Then, they call the UOT functionality in order to learn the input-wire labels for both their actual inputs and the r^i_j-s in their counterpart's circuits. More specifically, first they run the UOT_2 protocol where P_1 acts as the sender and P_2 acts as the receiver. P_1's inputs for UOT are the input-wire labels of input-wire k in all gc_j^1-s (i.e., the input pairs are $\mathsf{label}(gc_j^1, k, 0), \mathsf{label}(gc_j^1, k, 1)$ and $\mathsf{label}(xg_j^1, k, 0), \mathsf{label}(xg_j^1, k, 1)$ for $k \in \mathrm{INP}_2$ and $j \in [t]$). P_2's input is his actual input. They also call UOT_3 with the labels for the rest of the input-wires of xg_j^1 (i.e., $\mathsf{label}(xg_j^1, \alpha(k), 0)$ and $\mathsf{label}(xg_j^1, \alpha(k), 1)$ for $k \in \mathrm{INP}_2$ 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.

(In the protocol itself, these UOT calls are done before the garbled circuits are exchanged because this order is needed for the simulation. However, the intuition stays the same.)

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 simply showing the labels they learned in the UOTs), and then ask the UOT functionality to reveal all the inputs it received 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 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 they ask both UOT functionalities to reveal all the inputs they received from P_{3-i} for the inputs of the XOR-gates (i.e., $\mathsf{label}(xg_e^{3-i}, k, 0), \mathsf{label}(xg_e^{3-i}, k, 1)$ in the first UOT, and $\mathsf{label}(xg_e^{3-i}, \alpha(k), 0), \mathsf{label}(xg_e^{3-i}, \alpha(k), 1)$ in the second, both for $k \in \mathrm{INP}_i$). (We stress that only the XOR-gates of wires INP_i are opened, and that those were generated using random labels independently of the garbled circuits. The XOR-gadgets of wires INP_{3-i} are checked as part of the previous phase.) P_i verifies that these XOR-gates were generated properly and that the UOT inputs were consistent with the XOR-gates. If everything is ok 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} confirms 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-check. If there is no abort, the players call the Equality Testing functionality as before to obtain their final output.

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. (Recall that in the UOT for the XOR gadgets, each party can learn the labels for exactly one possible value of x. Thus his x is the same for all sets.)

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.

References

[AL10] Yonatan Aumann and Yehuda Lindell. Security against covert adversaries: Efficient protocols for realistic adversaries. *J. Cryptol.*, 23(2):281–343, April 2010.

[BHR12] Mihir Bellare, Viet Tung Hoang, and Phillip Rogaway. Foundations of garbled circuits. In *Proceedings of the 2012 ACM conference on Computer and communications security*, CCS '12, pages 784–796, New York, NY, USA, 2012. ACM.

[Can01] R. Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *Proceedings of the 42nd IEEE symposium on Foundations of Computer Science*, FOCS '01, pages 136–, Washington, DC, USA, 2001. IEEE Computer Society.

[CKKZ12] Seung Geol Choi, Jonathan Katz, Ranjit Kumaresan, and Hong-Sheng Zhou. On the security of the "free-XOR" technique. In Ronald Cramer, editor, *TCC*, volume 7194 of *Lecture Notes in Computer Science*, pages 39–53. Springer, 2012.

[Fis01] Marc Fischlin. *Trapdoor Commitment Schemes and Their Applications*. Ph.D. Thesis (Doktorarbeit), Department of Mathematics, Goethe-University, Frankfurt, Germany, 2001.

[FNP04] Michael J. Freedman, Kobbi Nissim, and Benny Pinkas. Efficient private matching and set intersection. In *Advances in Cryptology - EUROCRYPT 2004, International Conference on the Theory and Applications of Cryptographic Techniques, Interlaken, Switzerland, May 2-6, 2004, Proceedings*, pages 1–19, 2004.

[GMS08] Vipul Goyal, Payman Mohassel, and Adam Smith. Efficient two party and multi party computation against covert adversaries. In Nigel P. Smart, editor, *EUROCRYPT*, volume 4965 of *Lecture Notes in Computer Science*, pages 289–306. Springer, 2008.

[GMW87] O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game. In *Proceedings of the nineteenth annual ACM symposium on Theory of computing*, STOC '87, pages 218–229, New York, NY, USA, 1987. ACM.

[Gol04] Oded Goldreich. *Foundations of Cryptography: Volume 2, Basic Applications*. Cambridge University Press, New York, NY, USA, 2004.

[HEKM11] Yan Huang, David Evans, Jonathan Katz, and Lior Malka. Faster secure two-party computation using garbled circuits. In *Proceedings of the 20th USENIX conference on Security*, SEC'11, pages 35–35, Berkeley, CA, USA, 2011. USENIX Association.

[HKE12] Yan Huang, Jonathan Katz, and David Evans. Quid-pro-quo-tocols: Strengthening semi-honest protocols with dual execution. In *Proceedings of the 2012 IEEE Symposium on Security and Privacy*, SP '12, pages 272–284, Washington, DC, USA, 2012. IEEE Computer Society.

[IKNP03] Yuval Ishai, Joe Kilian, Kobbi Nissim, and Erez Petrank. Extending oblivious transfers efficiently. In Dan Boneh, editor, *CRYPTO*, volume 2729 of *Lecture Notes in Computer Science*, pages 145–161. Springer, 2003.

[IKO+11a] Yuval Ishai, Eyal Kushilevitz, Rafail Ostrovsky, Manoj Prabhakaran, and Amit Sahai. Efficient non-interactive secure computation. In Kenneth G. Paterson, editor, *EUROCRYPT*, volume 6632 of *Lecture Notes in Computer Science*, pages 406–425. Springer, 2011.

[IKO+11b] Yuval Ishai, Eyal Kushilevitz, Rafail Ostrovsky, Manoj Prabhakaran, and Amit Sahai. Efficient non-interactive secure computation. In *Proceedings of the 30th Annual international conference on Theory and applications of cryptographic techniques: advances in cryptology*, EUROCRYPT'11, pages 406–425, Berlin, Heidelberg, 2011. Springer-Verlag.

[IPS08] Yuval Ishai, Manoj Prabhakaran, and Amit Sahai. Founding cryptography on oblivious transfer — efficiently. In *Proceedings of the 28th Annual conference on Cryptology: Advances in Cryptology*, CRYPTO 2008, pages 572–591, Berlin, Heidelberg, 2008. Springer-Verlag.

[JS07] Stanislaw Jarecki and Vitaly Shmatikov. Efficient two-party secure computation on committed inputs. In *Proceedings of the 26th annual international conference on Advances in Cryptology*, EUROCRYPT '07, pages 97–114, Berlin, Heidelberg, 2007. Springer-Verlag.

[KS06] Mehmet S. Kiraz and Berry Schoenmakers. A protocol issue for the malicious case of Yaos garbled circuit construction. In *In Proceedings of 27th Symposium on Information Theory in the Benelux*, pages 283–290, 2006.

[KS08] Vladimir Kolesnikov and Thomas Schneider. Improved garbled circuit: Free xor gates and applications. In *Proceedings of the 35th international colloquium on Automata, Languages and Programming, Part II*, ICALP '08, pages 486–498, Berlin, Heidelberg, 2008. Springer-Verlag.

[KSS12] Benjamin Kreuter, Abhi Shelat, and Chih-Hao Shen. Billion-gate secure computation with malicious adversaries. In *Proceedings of the 21st USENIX conference on Security symposium*, Security'12, pages 14–14, Berkeley, CA, USA, 2012. USENIX Association.

[Lin11] Yehuda Lindell. Highly-efficient universally-composable commitments based on the ddh assumption. In *Proceedings of the 30th Annual international conference on Theory and applications of cryptographic techniques: advances in cryptology*, EUROCRYPT'11, pages 446–466, Berlin, Heidelberg, 2011. Springer-Verlag.

[LOP11] Yehuda Lindell, Eli Oxman, and Benny Pinkas. The ips compiler: Optimizations, variants and concrete efficiency. In Phillip Rogaway, editor, *Advances in Cryptology - CRYPTO 2011 - 31st Annual Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2011. Proceedings*, volume 6841 of *Lecture Notes in Computer Science*, pages 259–276. Springer, 2011.

[LP07] Yehuda Lindell and Benny Pinkas. An efficient protocol for secure two-party computation in the presence of malicious adversaries. In Moni Naor, editor, *EUROCRYPT*, volume 4515 of *Lecture Notes in Computer Science*, pages 52–78. Springer, 2007.

[LP09] Yehuda Lindell and Benny Pinkas. A proof of security of Yao's protocol for two-party computation. *J. Cryptol.*, 22(2):161–188, April 2009.

[LP11] Yehuda Lindell and Benny Pinkas. Secure two-party computation via cut-and-choose oblivious transfer. In Yuval Ishai, editor, *TCC*, volume 6597 of *Lecture Notes in Computer Science*, pages 329–346. Springer, 2011.

[MF06] Payman Mohassel and Matthew K. Franklin. Efficiency tradeoffs for malicious two-party computation. In Moti Yung, Yevgeniy Dodis, Aggelos Kiayias, and Tal Malkin, editors, *Public Key Cryptography*, volume 3958 of *Lecture Notes in Computer Science*, pages 458–473. Springer, 2006.

[NNOB12] Jesper Buus Nielsen, Peter Sebastian Nordholt, Claudio Orlandi, and Sai Sheshank Burra. A new approach to practical active-secure two-party computation. In *Advances in Cryptology - Crypto 2012*, volume 7417 of *Lecture Notes in Computer Science*, pages 681–700. Springer, 2012.

[NO09] Jesper Buus Nielsen and Claudio Orlandi. Lego for two-party secure computation. In *Proceedings of the 6th Theory of Cryptography Conference on Theory of Cryptography*, TCC '09, pages 368–386, Berlin, Heidelberg, 2009. Springer-Verlag.

[PSSW09] Benny Pinkas, Thomas Schneider, Nigel P. Smart, and Stephen C. Williams. Secure two-party computation is practical. In *Proceedings of the 15th International Conference on the Theory and Application of Cryptology and Information Security: Advances in Cryptology*, ASIACRYPT '09, pages 250–267, Berlin, Heidelberg, 2009. Springer-Verlag.

[PVW08] Chris Peikert, Vinod Vaikuntanathan, and Brent Waters. A framework for efficient and composable oblivious transfer. In David Wagner, editor, *CRYPTO*, volume 5157 of *Lecture Notes in Computer Science*, pages 554–571. Springer, 2008.

[SS11] Abhi Shelat and Chih-Hao Shen. Two-output secure computation with malicious adversaries. In Kenneth G. Paterson, editor, *EUROCRYPT*, volume 6632 of *Lecture Notes in Computer Science*, pages 386–405. Springer, 2011.

[Woo07] David P. Woodruff. Revisiting the efficiency of malicious two-party computation. In *Proceedings of the 26th annual international conference on Advances in Cryptology*, EUROCRYPT '07, pages 79–96, Berlin, Heidelberg, 2007. Springer-Verlag.

[Yao86] Andrew Chi-Chih Yao. How to generate and exchange secrets. In *Proceedings of the 27th Annual Symposium on Foundations of Computer Science*, SFCS '86, pages 162–167, Washington, DC, USA, 1986. IEEE Computer Society.

A Other Related Work

[IPS08, LOP11] show how to use semi-honest secure two-party, and honest-majority multi-party protocols to achieve security against malicious players. Although asymptotically this approach is very efficient, the constant factors seem to be large and we are not aware of any working implementation that evaluates its efficiency in practice. [NNOB12] constructs a protocol in the Random Oracle model, based on OT extension [IKNP03] and the classic GMW protocol [GMW87]. However, this protocol requires a number of rounds that depends on the depth of the circuit. Still, for some computations [NNOB12] shows better performance than the previous cut-and-choose based protocols.

[IKO$^+$11b] considers non-interactive secure computation protocols. Their first construction, which is asymptotically very efficient, achieves similar guarantees to the protocol of [HKE12] (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 ϵ-CovIDA protocols. However, it is not clear what would the efficiency of these protocols in practice be.

B Functionalities

Here we define all the functionalities we need in our constructions.

B.1 Oblivious Transfer

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. We define the functionality $\mathcal{F}_{OT}^l$ to be:

Inputs: S inputs $x_0, x_1 \in \{0,1\}^l$ and R inputs $b \in \{0,1\}$.

Outputs: R obtains x_b.

Figure 3: $\mathcal{F}_{OT}^l$.

[PVW08] shows an efficient construction of fully-secure universally-composable OT based on a variety of standard assumptions. When instantiating based on the DDH assumption, the protocol requires $O(1)$ exponentiations, $O(l)$ inexpensive operations and a constant number of rounds.

[IKNP03] presents how to extend $O(s)$ OT-s of length s strings to any number n of semi-honest OT-s of length l strings, using only additional $O(n \cdot l)$ inexpensive operations. [NNOB12] extends their results to *fully-secure* OT-s in the (amortized) price of only a (small) constant number of inexpensive operations per OT.[1] Note

[1] [IKNP03] also presents how to extend fully-secure OT-s. However, their construction has an overhead of $O(t)$ inexpensive operations per OT.

that the construction of [IKNP03] is secure assuming the hash function in use is *correlation-robust*, whereas the construction of [NNOB12] is proved secure in the Random Oracle Model. (See [NNOB12] for more details.)

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, and therefore we omit the factor l from the complexities. (Specifically, we say that the amortized cost per OT when we use the OT extension protocol of [NNOB12] is a constant number of hashes.) When we concatenate several strings in one OT, we count the cost of each of them separately.

B.2 Undeniable Oblivious Transfer

Here, sender S has n sets, each of m pairs of inputs, and receiver R has a vector of input bits $\bar{b} = (b_1, \cdots, b_n)$. An undeniable OT has two stages: The first is similar to the standard OT for many inputs, where R learns the outputs according to his input bits; In the second, both parties request the same subset $I \subseteq [m]$, and R learns the j-th pair for all $j \in I$, in all n sets. See Figure 4 for a formal definition of this functionality, denoted by $\mathcal{F}^{l,m,n}_{UOT}$, and see Figure 5 for an example of a UOT execution.

First Stage

Inputs: S inputs n sets, each of m pairs $\{(x_0^{j,z}, x_1^{j,z})\}$ of strings of length l, for $j = 1 \ldots n, z = 1 \ldots m$. R inputs a binary vector $b \in \{0,1\}^n$.

Outputs: For all j and z, R obtains $x_{b_j}^{j,z}$.

Second Stage

Inputs: Both S and R input Reveal I.

Outputs: For all j and for each $z \in I$, R obtains $(x_0^{j,z}, x_1^{j,z})$.

Figure 4: $\mathcal{F}^{l,m,n}_{UOT}$.

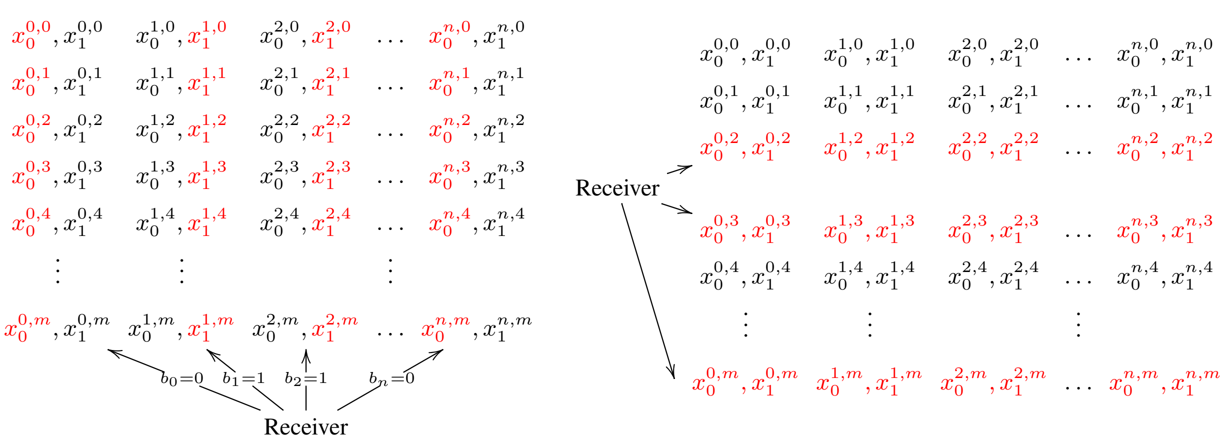

Figure 5: UOT Functionality Example. On the left is the first stage in which the receiver inputs his input bits b_i and learns the red elements. On the right is the second stage in which the receiver asks for all the elements in the selected rows to be revealed.

UOT FROM NUMBER-THEORETIC ASSUMPTIONS (UOT_1). This functionality is somewhat similar to the *Committing OT* functionality [KS06] and can be realized under the DDH assumption using $O(nm)$ exponentiations [SS11]. (We note that the *cut-and-choose OT* of [LP11] provides a similar functionality with similar complexity in a *single* stage.) The possibility of efficiently extending this functionality (*a la.* [IKNP03]) is an interesting open question.

UOT FROM STANDARD OT (UOT_2). Alternatively, a more efficient construction can be based on the technique of [LP07] which only uses standard OT in a black-box way. Combined with the OT-extension of [NNOB12], it results in complexity of $O(s)$ OT-s and $O(4nm)$ inexpensive operations. (A similar construction is implemented in [KSS12], though we are not aware of any description of its details, nor its complexity.)

We start with some intuition of that 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 (k is even), 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. 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 [LP07] 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 [LP07] 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 call $\mathcal{F}_{OT}^{l}$ $4n$ 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. Moreover, 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 in the first stage of the UOT.

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

Note, however, that here the sender did not use specific inputs for the OT, and that all pairs were xor-ed with the same d. Since the inputs we use in our protocols are random labels for garbled circuits, the first issue is not problematic since we can simply use the random strings of the above protocol as the circuit's input-wire labels. As for security, since all pairs use the same d, we need to assume that the hash function in use is *circular 2-correlation robust* [CKKZ12]. (We remark that in the Random Oracle Model, the circular 2-correlation robustness is satisfied.)

For statistical security parameter t, $\max(4n, 8t)$ inputs are needed in order to obtain a negligible probability failure against the selective-OT attack [LP07]. Thus, 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 $n \cdot t$ inputs in total. (Another useful property of this approach is that it allows one to control the probability of potential leakage by adjusting t. In particular, we can choose a non-negligible probability of leakage in our covert protocol and gain better efficiency. We omit further details.)

Last, we note that the error probability of the cut-and-choose protocol is $2^{-0.311t}$ (according to [LP11]), while the above protocol has much smaller error probability for the same t. Still, for simplicity and since the difference is by a small constant in the exponent, we use the same parameter t for both. (Indeed, taking the optimal parameters for each of them will result in a more efficient protocol in practice.)

A SIMPLE UOT FOR THE CASE OF $I = [m]$ (UOT_3). Last, we note that in the OT construction of [PVW08] and the transformation of [NNOB12], the sender is *committed* to its inputs since the transcripts are essentially binding. This means that after the protocol ended, if the receiver asks the sender to reveal all his inputs for $I = [m]$ (namely, all his inputs in the protocol), the sender can reveal his inputs and the randomness he used in the protocol as a proof. Hence, this is another efficient realization of $\mathcal{F}_{UOT}^{l,m,n}$ for the case where $I = [m]$. When using OT extension, this requires only $O(s)$ OT-s and $O(mn)$ inexpensive operations.

We remark that the resulting protocol could be proven to be a secure realization of $\mathcal{F}_{UOT}^{l,m,n}$ assuming the hash function used in [NNOB12] is a programmable random oracle. However, in the simulation of our protocols, the simulator does not need to interact with the trusted party for $\mathcal{F}_{UOT}^{l,m,n}$ at all, thus the simulation in this case does not require the mentioned assumption.

The complexities of the different realizations is summarized in Table 3.

Based on	Possible I	Complexity
[SS11]	any	$\mathsf{expensive}(nm)$
[LP07], random z_i	any	$\mathsf{expensive}(s) + \mathsf{inexpensive}(\max(4n, 8t) \cdot m)$
[LP07], disjoint z_i	any	$\mathsf{expensive}(s) + \mathsf{inexpensive}(nmt)$
[PVW08, NNOB12]	$[m]$	$\mathsf{expensive}(s) + \mathsf{inexpensive}(nm)$

Table 3: Comparison of the four realizations of $\mathcal{F}_{UOT}^{l,m,n}$. s is a statistical security parameter.

B.3 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 $\mathcal{F}_{2SET}^{l}$ 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 6: $\mathcal{F}_{2SET}^{l}$.

[HKE12] REALIZATIONS. [HKE12] uses a similar functionality, that has only one stage in which the players learn if $x_1 = x_2$. For that functionality, they present two possible realizations: The first is to execute a fully-secure two party computation for this functionality. This option is quite efficient since we only need to compare the hashes of the strings, resulting in complexity that is independent of the circuit size or the input length. The second construction is based on [FNP04] in the random oracle model and requires an additively homomorphic encryption. However, the construction does not achieve simulation-based security, thus it is only conjectured to be secure in composition.

We note that both constructions can be modified to have two stages like we require. The idea is to replace the first step in which they send information that will reveal some information to the other player, with a commitment on the next message. Then, in the second stage, they decommit this message and continue the protocol. (E.g., in the fully-secure 2PC based on cut-and-choose we show in Section 3, P_1 sends a commitment on the labels of his inputs at the end of the protocol, before P_2 evaluates the garbled circuits.)

OTHER REALIZATIONS IN THE ROM. A simulation-based provable option is to slightly modify the fully-secure Private Equality Test of [FNP04], which requires t encryptions and hashes. Specifically, in case the equality test succeeds, the receiver sends to the sender the randomness he retrieved from the encryption. That proves also to the sender that indeed the two inputs are the same.

Another efficient option in case the inputs have high-entropy (as is the case in all our protocols) is the following: (1) P_2 generates trapdoor commitment keys ct, ck, sends ck and proves using ZK-PoK that he knows ct; (2) P_1 sends $\mathsf{Com}_{ck}(\mathrm{H}(x_1), r)$; (3) P_2 sends $\mathrm{H}(x_2)$; (4) P_1 decommits (by sending $\mathrm{H}(x_1), r$); (5) Both players check if $\mathrm{H}(x_1) = \mathrm{H}(x_2)$. Details of the (simulation-based) proof are omitted.

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

Figure 7 presents our protocol in detail. A simple example of the XOR-gadgets technique is given in Figure 1.

Before we prove Theorem 3.1, 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 [LP07], 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 [LP07], 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 ρ 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 ρ. 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).

C.1 Proof of Theorem 3.1

Proof. Let $\mathcal{A}$ be an adversary controlling P_1 in the execution of the protocol in the $\mathcal{F}_{UOT}^{l,t,n}$-hybrid model. 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. Note that $\mathcal{S}$ learned $\mathcal{A}$'s input to the OT-s (used for him to learn the input-wire labels of his actual input). Denote by x' this input. If P_2 did not abort, $\mathcal{S}$ calls the trusted party with x' and outputs its answer.

Garbling:
Let $C_1, C_2, \alpha(\cdot)$ as defined in Section 3.1.
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 UOT_2 protocol in which 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.
They execute two regular OT protocols (i.e. UOT_3) in which P_2 is the sender: (We separate the two for simplifying the description. However, both protocols can be executed together to reduce 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 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 OT, 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 ρ of length t to be all zeros.
2. For $j = 1, \ldots, (t \cdot c)$, each player P_i picks a random value $v^i_j \in [1..(t - j + 1)]$.
3. P_2 sends a commitment $\mathsf{Com}(v^2_1 \circ v^2_2 \circ \cdots \circ v^2_{t \cdot c})$.
4. P_1 sends his values $v^1_1, \ldots, v^1_{t \cdot c}$.
5. P_2 decommits and reveal $v^2_1, \ldots, v^2_{t \cdot c}$.
6. For $j = 1, \ldots, (t \cdot c)$, let $v = ((v^1_j + v^2_j) \mod (t - j + 1)) + 1$ and let k be the v-th zero bit of ρ. 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 the (first) OT 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, they ask the UOT functionality for all the inputs P_1 used in the opened sets and P_2 verifies that all the values are consistent with the opened circuits.

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).
2. P_2 opens all his XOR-gadgets in the set E (by sending z'_i-s), and reveals all the randomness he used in the regular OTs. P_1 verifies that the XOR-gadgets were constructed properly and consistent with the OT 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 $j \in E$.
5. P_2 evaluates the XOR-gadgets in the sets S_j, $j \in E$ and compares the results to the output-wire labels 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.

Figure 7: A Fully-secure 2PC Protocol.

From the cut-and-choose we know that with probability $1 - neg(t)$ (see [LP07, SS11]), 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 in which it holds.

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 answers is correct with probability $1 - neg(t)$.

We note that the simulation is distributed exactly as the execution in the real model, except when P_1 is able to cheat (and then P_2's output in the real model would be different than in the ideal model). However, from the analysis above, this can happen with $1 - neg(t)$ probability.

Let $\mathcal{A}$ be an adversary controlling P_2 in the execution of the protocol in the $\mathcal{F}_{UOT}^{l,t,n}$-hybrid model. 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:

- Picks at random a subset E and a random permutation $\pi(E)$.
- Emulates an honest P_1 until the end of stage Oblivious Transfer. It learns P_2's input from the UOT.
- Calls the trusted party with P_2's input and receives the output z.
- Constructs the sets such that for $j \in E$, gc_j outputs the constant z, 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 OT-s.

Note that the only part in which the simulation is different than the execution in the real model is where the simulator constructs the *fake* garbled circuits. However, by the results of [LP09, BHR12], this difference is indistinguishable. (This can be done, e.g., by setting the output gates to be constant gates of the actual outputs. Then, by the security of the garbling scheme, this change is indistinguishable.)

■

C.2 Achieving UC Security

The only part in which the simulations of our 2PC protocol require rewinding, is for choosing the set of circuits for evaluation. (The rest of the protocol and its sub-protocols do not need rewinding.) Thus, all we need to change is the way this set is chosen.

We replace the cut-and-choose stage with the one described in Figure 8. Now, the simulator can extract ρ from P_2's OT queries, and be able to generate fake garbled circuits without rewinding. The rest of the simulation stays the same. The overhead over the stand-alone protocol is merely t OT-s. We omit a formal proof since the simulation for the environment is roughly the same.

Cut-and-choose:
They pick a random $E \subset [t]$ of size $t \cdot c$ in the following way:

1. P_1 picks t pairs of random strings $(a_j, b_j)_{j=1}^{t}$.
2. P_2 picks at random a boolean string ρ of length t with exactly $|E|$ non-zero elements.
3. They execute OT where P_1 inputs the t pairs (a_j, b_j), and P_2 inputs the bits ρ_j.
4. P_1 sends the t sets $S_1, \ldots, S_t$ to P_2, and P_2 sends the XOR-gadgets $xg_0, xg_1, \ldots, xg_t$.
5. For each $j \in [t]$, P_2 sends the j-th string it learned from the OT and ρ_j. P_1 verifies that the two are consistent with his pairs. If so, they set E be $\{j|\rho_j = 1\}$.

Figure 8: UC compatible cut-and-choose.

D More on Our ϵ-CovIDA Constructions

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

Before we present the detailed construction, 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 places in the protocol (i.e. when P_1 commits to his coins and when he commits to his garbled sets). The intuition is that each player generates a pair of a public key and a trapdoor to a trapdoor commitment scheme, and proves using a zero-knowledge proof of knowledge protocol (ZK-PoK) that he knows the trapdoor. Each player then uses the other player's public key to commit to his values. In the simulation, the simulators can rewind the ZK-PoK, extract the trapdoor, and open the commitment to an appropriate value of their choice.

One option is to use DDH based trapdoor commitment and standard ZK-PoK of discrete-log (see [Fis01]). Then, the overhead introduced here is only a (small) constant number of exponentiations. The total overhead is very small since the commitment scheme is only invoked $O(t)$ times in the entire protocol.

We remark that for the same purpose we can also use UC commitments [Lin11] and the simulation would be similar.

Figure 9 presents our protocol in detail.

D.1.1 Proof of Theorem 4.2

Proof. Let $\mathcal{A}$ be an adversary controlling P_1 in the execution of the protocol in the $(\mathcal{F}_{UOT}^{l,t,n}, \mathcal{F}_{2SET}^{l})$-hybrid model. 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 until the end of the stage Evaluation and Input-equality Check. During the execution, $\mathcal{S}$ records all the opened sets and $\mathcal{A}$'s inputs to UOT. Also, it extracts the trapdoor ct_1 from $\mathcal{A}$ (using the ZK-PoK extractor).

2. Rewinds $\mathcal{A}$ until the Cut-and-choose stage in order to pick a different e. Since $\mathcal{S}$ already saw $\mathcal{A}$'s coins, $\mathcal{S}$ simply picks the appropriate coins on behalf of P_2 to obtain the desired value for e. $\mathcal{S}$ emulates honest P_2 and continues again until the end of the stage Evaluation and Input-equality Check.

3. Repeats the above rewinding t times, until it has used all $e \in [t]$ (we assume this is done in a random ordering). This means that $\mathcal{S}$ now knows: 1) All the openings of the sets; 2) All of $\mathcal{A}$'s inputs to the

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.
For $j = 1, \ldots, t$, player P_1 picks random strings $z_j^1, z_j^{1'}$ (of length s) and r_j (of length $|\mathrm{INP}_1|$), and 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 in gc_j.

Oblivious Transfer:
Players execute two series of UOT protocols where P_1 is the sender, with all the input-wire labels for gc_j^1 and xg_j^1 as described before. More specifically, in the first execution P_1's input is $|\mathrm{INP}_2|$ sets of $2t$ pairs $\big(\ \mathsf{label}(gc_j^1, k, 0), \mathsf{label}(gc_j^1, k, 1)\ \big)$ and $\big(\ \mathsf{label}(xg_j^1, k, 0), \mathsf{label}(xg_j^1, k, 1)\ \big)$ for $k \in \mathrm{INP}_2$ and $j \in [t]$, and P_2 uses his actual input bits. In the second execution 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 input 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 ZK-PoK that he knows the corresponding trapdoor. (P_2 does the same.)
2. P_1 sends the commitment $\mathsf{Com}_{ck_2}(\mathrm{H}(S_1^1 \circ \ldots \circ S_t^1))$ to P_2. (I.e. a commitment to the hash of his sets, using P_2's commitment key.)
3. P_1 sends t (regular) commitments $c_1^1, \ldots, c_t^1$, where c_j^1 is a commitment to the labels that correspond to his inputs in gc_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 $\mathsf{Com}_{ck_2}(\mathrm{H}(S_1^1 \circ \ldots \circ S_t^1))$. ($P_2$ does the same for his sets and commitments, and P_1 verifies that they are consistent.)

Checking Opened Circuits:
For $(S_j^1, S_j^2)_{j \neq e}$, the players send to each other: 1) $z_j^i, z_j^{i'}$, 2) The labels they have learned from the UOT for r_j^i. For the opened sets, each player verifies that the circuits and gadgets were constructed properly, and that the other player used the same r_j^i for gc_j^i and xg_j^{3-i}. Then, they ask the first UOT functionality for all the inputs used in the opened sets and verify that all the values are consistent with the opened circuits.

Evaluation and Input-equality check:
Each player sends his input-wire labels for the e-th circuit, along with the decommitment of c_e^i. 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). Next, P_2 sends the randomness he used for garbling the XOR-gadget xg_e^2, they ask the first UOT functionality to reveal all corresponding inputs (i.e., the pairs $\big(\ \mathsf{label}(xg_e^1, k, 0), \mathsf{label}(xg_e^1, k, 1)\ \big)$ for $k \in \mathrm{INP}_2$) and the second UOT to reveal all P_2's inputs to it. P_1 verifies that the XOR-gadget was constructed properly and consistently with the UOT 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 outputs of the e-th garbled circuits (including the labels as before) and output accordingly.

Figure 9: $1/t$-CovIDA Protocol.

different sets; 3) $\mathcal{A}$'s inputs to the UOT.

4. Checks if some of the sets are problematic, which means that one (or more) of the following occurs: 1) The garbled circuits or the XOR gadgets are not generated correctly; 2) The labels of the garbled circuits and XOR gadgets are inconsistent with $\mathcal{A}$'s input to the UOT, or 3) The used inputs in the Evaluation and Input-equality Check stage are inconsistent with $\mathcal{A}$'s queries to the UOT.

   If all are correct,

   - Calls the trusted party with $\mathcal{A}$'s input and receives the output z.
   - Rewinds to step 3 in the Cut-and-choose stage. Plays the honest P_2 in order to generate a random $e \in [t]$. For the set e, $\mathcal{S}$ replaces P_2's garbled circuit with one that always outputs z 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.
   - Emulates honest P_2 (still with a random input) in the rest of the steps, but in the Equality-Testing step uses the output-wire labels that correspond to z (it knows the labels from the sets opening).

   If more than one set is incorrect,

   - Sends $\perp$ to the trusted party.
   - Emulates honest P_2 until the end of the protocol. (Note that P_2 will abort.)

   Otherwise,

   - Sends to the trusted party $\mathsf{cheat}_1(1/t)$.
   - If the trusted party returns $\mathsf{corrupted}_1$, $\mathcal{S}$ rewinds until step 3 in the Cut-and-choose stage, and sends instead $coins_2$ such that P_1 will be caught later. Emulates honest P_2 until the end. (Note that P_2 will abort.)
   - If the trusted party returns $\mathsf{undetected}$,
     - As before, $\mathcal{S}$ rewinds and makes sure that e is corresponding to the malicious set/inputs, and also, replaces P_2's e-th garbled circuit with one that always outputs z.
     - $\mathcal{S}$ emulates honest P_2 in the rest of the protocol until the Equality-Testing stage.
     - Receives $\mathcal{A}$'s input w to the Equality-Testing functionality, and sends to the trusted party the description of the following function g: Let g be the function that has hardcoded the circuit gc_e^1, the input labels that $\mathcal{A}$ used in the UOT for P_2's inputs of gc_e^1, the labels that $\mathcal{A}$ sent in the Evaluation and Input-equality Check stage for his inputs, the output-wire labels of gc_e^2 and w. The function evaluates the garbled circuit using the real input of P_2 and returns 0 if the input of an honest P_2 to the Equality Testing functionality after the evaluation does not equal to w.
   - The trusted party returns a bit and $\mathcal{S}$ sends it to $\mathcal{A}$.

5. Sends $\mathcal{A}$'s response to the trusted party (whether to abort or not).

Inspecting the simulation shows that it simulates the adversary perfectly except for two differences:

- The garbled circuit that P_1 evaluates is not a correct garbled circuit of the function they compute. However, [LP09, BHR12] show that the two views are indistinguishable (under minor changes to the circuit in use).

- The output of honest P_2 in the real model might be different than his output in the ideal model but this happens only in case the Equality Testing succeeds, which means that $\mathcal{A}$ guessed correctly the output-wire labels of gc_e^2 not corresponding to the output he received from the evaluation. However, this can only happen with probability negligible in l (where l is the length of the labels) unless the garbling procedure is not secure (this property is sometimes referred to as authenticity of the outputs of a garbling scheme).

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 [SS11] 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). They follow the protocol upto the final Equality-check step. 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 their secrets, while the other player verifies they were constructed properly (or otherwise aborts). If everything was ok, 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. Specifically, we prove the following theorem.

Theorem D.1. *The above protocol is ϵ-CovIDA secure. In the hybrid $(\mathcal{F}_{UOT}, \mathcal{F}_{2SET})$ model, the complexity of the protocol is $O(\log(\frac{1}{\epsilon}) \cdot (|C| + n + q))$ inexpensive operations. The number of calls to $\mathcal{F}_{UOT}$ is 4 with $O(\log(\frac{1}{\epsilon}) \cdot n)$ inputs overall.*

Before we present the proof, we describe the coin-tossing protocol we use in this protocol. We replace the coin-tossing step of the protocol from Figure 9 with the one from Figure 7, and modify 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 ρ 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_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 ρ. 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 of Theorem D.1. Let $\mathcal{A}$ be an adversary controlling P_1 in the execution of the protocol. The simulation is very similar to the one from Appendix D.1 except for some small changes.

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 until the end of stage Evaluation and Input-equality Check. During the execution, $\mathcal{S}$ records all the opened sets and $\mathcal{A}$'s inputs to UOT. Also, it extracts the trapdoor ct_1 from $\mathcal{A}$ (using the ZK-PoK extractor).

2. Rewinds $\mathcal{A}$ until the middle of the Cut-and-choose step in order to pick a different $E \subset [t]$. Since $\mathcal{S}$ already saw $\mathcal{A}$'s v_j^1-s, $\mathcal{S}$ simply picks the appropriate v_j^2-s on behalf of P_2 to obtain the desired subset E. $\mathcal{S}$ emulates honest P_2 and continues again until the end of the first stage of the equality testing where all the identity-gates are also opened.

3. Repeats the above rewinding enough times until the union of the chosen $[t] \backslash E$-s covers all of $[t]$. This means that $\mathcal{S}$ now knows: 1) All the openings of the sets; 2) All of $\mathcal{A}$'s inputs to the different sets; 3) $\mathcal{A}$'s inputs to the UOT.

4. Checks to see which sets are not generated correctly, are inconsistent with $\mathcal{A}$'s input to the UOT, or, use inputs in the Evaluation and Input-equality Check stage that are not consistent with $\mathcal{A}$'s input to the UOT. Let $\mathcal{B} = \{i | \text{set } i \text{ is problematic}\}$. If all are correct,

   - Calls the trusted party with $\mathcal{A}$'s input and receives the output z.
   - Rewinds to step 3 of the Cut-and-choose stage, plays the role of honest P_2 to generate a uniformly random subset E of $[t]$ (of size $t \cdot c$). For the sets of circuits/gadgets in E, $\mathcal{S}$ replaces P_2's garbled circuits with ones that always output z 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.
   - Emulates honest P_2 in the rest of the steps, but in the Equality-Testing step uses the output-wire labels of the identity-gate that correspond to z (it knows the labels from the opening phase).

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

   - Sends $\perp$ to the trusted party.
   - Emulates honest P_2 until the end of the protocol. (Note that P_2 will abort.)

   If more than $|E|/2$ but less than $|E|$ of the sets are incorrect ($\frac{|E|}{2} \leq |\mathcal{B}| \leq |E|$),

- Set $\epsilon' = \frac{\binom{t-|\mathcal{B}|}{t-|E|}}{\binom{t}{t-|E|}}$. (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 $\mathsf{corrupted}_1$, $\mathcal{S}$ rewinds until step 3 in the Cut-and-choose stage, and makes sure that a 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 ρ 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 ρ 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 ρ_{G} of length $t - |\mathcal{B}|$ that has exactly $t \cdot c - \mathsf{HW}(\rho_{\mathcal{B}})$ non-zero elements. Choose ρ 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 $\mathsf{undetected}$,
  - As before, $\mathcal{S}$ rewinds and 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. (Here we use the same process for picking E as before, but instead we take $\rho_{\mathcal{B}}$ to be all ones.)
  - $\mathcal{S}$ emulates honest P_2 in the rest of the protocol until the Equality-Testing stage.
  - Receives $\mathcal{A}$'s input w to the Equality-Testing functionality, and sends to the trusted party the description of the following function g: Let g be the function that has hardcoded the circuit gc_e^1 for all $e \in E$, the input labels that $\mathcal{A}$ used in the UOT for P_2's inputs of gc_e^1, the labels that $\mathcal{A}$ sent in the Evaluation and Input-equality Check step for his inputs, the output-wire labels of gc_e^2 and w. The function evaluates the garbled circuits using the real input of P_2, computes the majority output and returns 0 if the input of an honest P_2 to the Equality Testing functionality after the evaluation does not equal to w.
- The trusted party returns a bit and $\mathcal{S}$ sends it to $\mathcal{A}$.

If less than $|E|/2$ of the sets are incorrect ($|\mathcal{B}| < |E|/2$),

- Rewinds to step 3 of the Cut-and-choose stage, plays the role of honest P_2 to generate a uniformly random subset E of $[t]$ (of size $t \cdot c$).
- If any of the incorrect sets is in the $[t] - E$ opened ones, sends $\perp$ to the trusted party, simulates honest P_2 aborting and outputs what $\mathcal{A}$ does.
- If all the incorrect sets are in E, we still have that the majority of the sets in E are correct, i.e. the circuits are correct, and both parties inputs to them are the same (due to the XOR gadgets). $\mathcal{S}$ sends $\mathcal{A}$'s input in the good sets to the trusted party, receives the output z, replaces P_2's garbled circuits in E with ones that always output z, and simulates honest P_2 for rest of the simulation.

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

■