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

Payman Mohassel[1] and Ben Riva[2,⋆]

[1] University of Calgary, Canada
[2] Tel Aviv University, Israel

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

- We design an efficient fully-secure 2PC protocol for two-output functions that only requires $O(t|C|)$ symmetric-key operations (with small constant factors, and ignoring factors that are independent of the circuit in use) in the Random Oracle Model, where $|C|$ is the circuit size and t is a statistical security parameter. This is essentially the *optimal* complexity for protocols based on cut-and-choose, resolving a main question left open by the previous work on the subject.
  Our protocol utilizes novel techniques for enforcing *garbler's input consistency* and handling *two-output functions* that are more efficient than all prior solutions.
- Motivated by the goal of eliminating the *all-or-nothing* nature of 2PC with covert security (that privacy and correctness are fully compromised if the adversary is not caught in the challenge phase), we propose a new security definition for 2PC that strengthens the guarantees provided by the standard covert model, and offers a smoother security vs. efficiency tradeoff to protocol designers in choosing the *right deterrence factor*. In our new notion, correctness is always guaranteed, privacy is fully guaranteed with probability $(1-\epsilon)$, and with probability 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 covert 2PC protocols based on cut-and-choose.

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

⋆ Research supported by the Check Point Institute for Information Security and an ISF grant.

R. Canetti and J.A. Garay (Eds.): CRYPTO 2013, Part II, LNCS 8043, pp. 36–53, 2013.

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* [21,12]: 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 [9], reducing communication [4,18], and designing tailored circuits [5]).

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. [8,16] 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.

The Cut-and-Choose Approach. A slightly more explored direction is based on using the cut-and-choose method for checking the garbled circuit. (E.g., see implementations by [18,19,10].) 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 [13,19] that with high probability the majority of the evaluated garbled circuits are properly constructed.

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

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

To address the first issue, i.e. how to make sure Alice is using the same input in all circuits, [14,11] present two methods that require $O(t^2 \cdot n_1)$ inexpensive cryptographic operations (commitments), where n_1 is the length of Alice's input, and t is the number of circuits we use in the cut-and-choose. ([20] shows how to reduce this asymptotic overhead, but with large constants even for small security parameters.) [14,13,19] show alternative methods that require $O(t \cdot n_1)$ expensive cryptographic operations (i.e. exponentiations). These consistency-checking mechanisms can lead to significant overhead. Recall that garbling of a single gate requires a constant number of symmetric encryptions, where the constant is 4 in most implementations. Thus, e.g. for $t = 130$, the price of checking consistency for a single input bit is roughly equivalent to the price of garbling several tens of additional gates in each circuit in the first method, and even more in the second. Moreover, the first method has a large communication overhead (e.g., for input size $n_1 = 500$ and $t = 130$, it requires several millions of commitments, with a total communication overhead of hundreds of megabytes).

To address the second issue, i.e. making sure Alice is using the same labels in her OT answers and the garbled circuits, [11] presents a method that requires $O(t \cdot \max(4n_2, 8t))$ expensive cryptographic operations (specifically, oblivious transfers), where n_2 is the length of Bob's input. [13,19] introduce alternative methods that require $O(t \cdot n_2)$ expensive cryptographic operations.

To address the last issue, of verifying the computation output, [11] proposes to apply a *one time MAC* to the output and XOR the result with a random input to hide the outcome (both are done as part of the circuit). However, this solution increases Alice's input with additional $q_1 + 2t$ input bits and increases the circuit size by $O(t \cdot q_1)$ gates, where q_1 is Alice's output length (i.e. overall overhead of $O(t^2 \cdot q_1)$ inexpensive operations). [19] suggests a solution that requires the use of digital signatures and a witness-indistinguishable proof, resulting in a total overhead of $O(t \cdot q_1)$ expensive operations.

In the covert setting [1] the techniques are similar, although the issue of the garbler's input consistency is not always relevant [4,1].

All-or-Nothing Security vs. Security with Input-Dependent Abort. All the cut-and-choose protocols discussed above provide an *all-or-nothing* guarantee, which means that both correctness and privacy are preserved with the same probability (the probability of getting caught in case of cheating), and are completely compromised if cheating is not detected. For example, in case of a protocol with covert security and deterrence factor of 1/2, there is a 50% chance that the protocol reveals the honest party's input and provides him with an incorrect output.

This can become an obstacle to using covert security, in some practical scenarios. For example, the participants of an MPC protocol may not be able to afford the lack of correctness or privacy (even if only with a constant probability), due to the high financial/legal cost, or the loss of reputation.

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

The idea behind the protocols of [14] is as follows: Alice garbles a circuit gc_1 and sends it to Bob, along with the labels of Alice's input-wires. They execute a fully-secure oblivious transfer protocol in which Bob learns the labels for his input-wires. Then, they run the same steps in the other direction, where Bob garbles gc_2 and Alice is the receiver. Next, each player evaluates the garbled circuit he or she received, resulting in output-wire label out_i (we require that the output-wire labels are the actual outputs concatenated with random labels). Last, each player computes the *supposed to be* concatenation $out_1 \circ out_2$. (Alice gets out_1 from her evaluation, and can determine the value of out_2 by herself. Bob does the same.) Now they run a protocol for securely testing whether their values $out_1 \circ out_2$ are the same. If they are indeed the same, they output b. Otherwise, they abort.

The resulting protocol is highly efficient, requiring only two garbled circuits and the associated oblivious transfers. (See [6] for an optimized variant of the protocol and its performance.) Since one of the players is honest, the result from his garbled circuit will be correct. Thus, if the honest party does not abort, the output is indeed correct. On the other hand, if one of the players is malicious, he can *always* learn one bit of information by observing whether the honest party aborts or not in the final equality test. We call this scenario *Input-Dependent Abort* (IDA) (following [7]).

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.

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

First, we show how to use garbled *XOR-gates* to efficiently enforce the garbler's input consistency, while requiring only $O(t \cdot n_1)$ inexpensive operations. This approach asymptotically improves the solutions in [14,11], and only requires inexpensive operations in contrast to the solution of [19]. Second, we observe that the solution of [11] to the evaluator's input consistency issue can be improved by combining it with the OT extension of [15] and the Free-XOR technique of [9]. The resulting protocol requires only $O(t \cdot \max(4n_2, 8t))$ inexpensive operations. Third, we show how to use garbled *identity-gates* to efficiently solve the two-output function problem, while requiring only $O(t \cdot q_1)$ inexpensive operations, where q_1 is the garbler's output length, improving on the recent construction of [19] 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 [14,11,13,19] (except for [20], which is impractical due to large constants). In Table 1, we compare the protocol's complexity with previous constructions. We stress that the efficiency of our protocol relies on the efficient OT extension of [15], 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 [15] is in the Random Oracle Model (ROM) and our construction inherits the same weakness. (Besides using ROM for the OT-extension of [15], in some of our techniques we show two alternatives: A more efficient instantiation in the ROM, and one without the ROM requirement, which still is more efficient than current techniques.)

We remark that our proposed solutions can be modified to work with any of the existing garbled-circuit optimization techniques of [9,4,18,5,10].

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

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

	P_1's input	P_2's input	Two-output Overhead
[11]	$\mathsf{inexpensive}(t^2 n_1)$	$\mathsf{expensive}(\max(4n_2, 8t))+$ $\mathsf{inexpensive}(t \cdot \max(4n_2, 8t))$	$\mathsf{inexpensive}(t^2 q_1)$
[13,19]	$\mathsf{expensive}(tn_1)$	$\mathsf{expensive}(tn_2)$	$\mathsf{expensive}(tq_1)$
Our protocol	$\mathsf{inexpensive}(tn_1)$	$\mathsf{inexpensive}(t \cdot \max(4n_2, 8t))$	$\mathsf{inexpensive}(tq_1)$

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

Multi-Stage Cut-and-Choose and Handling Two-Output Functions. From now on we denote by P_1 the garbler (Alice), and by P_2 the evaluator (Bob). Note that the main difficulty here is to convince the garbler, P_1, that the output he receives is correct. (Privacy of the output is easily achieved by xoring the output with a random string.)

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

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

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

The above solution can be applied to most previous 2PC protocols based on cut-and-choose to obtain their two-output variants. But, since the circuit checking is done after the circuit evaluation, the above solution falls short when combined with circuit streaming or parallelized garbling techniques [5,10]. In the full version of this paper we describe a second variant of this protocol that is compatible with those techniques. The cost of this variant is only additional $t \cdot q_1$ commitments.

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

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

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

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

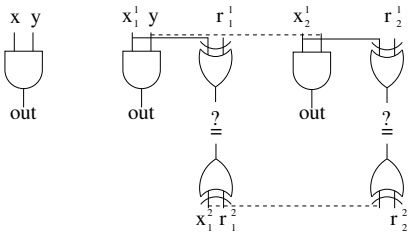

Fig. 1. Example of garbling the simple AND circuit on the left that computes the AND between P_1's bit x and P_2's bit y. P_1 garbles the upper circuits and P_2 the lower ones. Specifically, P_1 garbles two AND circuits (i.e., $t = 2$) and two XOR-gates, and P_2 garbles two XOR-gates. P_2's input is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

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.

Security with Input-Dependent Abort in Presence of Covert Adversaries. We propose a new security notion that naturally combines security with input-dependent abort of [7] (alternatively, security with limited leakage of [14,6]), with security against covert adversaries [1]. The resulting security guarantee, denoted by ϵ-CovIDA, is a *strict strengthening of covert security*: In covert security, with probability ϵ both correctness and privacy are gone! 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 [14,6] with the cut-and-choose method is *not* secure under our definition. Say that instead of garbling a single circuit, each player P_i garbles t circuits $gc_1^i, \ldots, gc_t^i$ and sends them to the other player. Players pick a random value $e \in [t]$, *open* all the circuits $gc_{j \neq e}^i$ (i.e., reveal the randomness used to generate them), and verify that they were constructed properly. This assures that with probability $1 - 1/t$, the remaining two circuits (one circuit from each player) is properly constructed. Parties then engage in the dual-execution protocol discussed above using these two garbled

circuits. Although this protocol guarantees correctness similar to [14,6], it does not satisfy our security definition. One problem is that a malicious player can use different inputs for the two evaluated circuits, and learn whether their outputs are the same or not based on the final outcome. This attack is successful even if *all* the circuits are constructed correctly.

We show two constructions that do achieve our definition. Both constructions require a constant number of rounds. In our first construction, each player garbles only $\frac{1}{\epsilon}$ circuits and $\frac{n+2q}{\epsilon}$ additional XOR gates, where n is the length of the input and q is the length of the output. We emphasize that compared to the protocols of [14,6], 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.

The costs of both constructions are roughly the costs of running their covert counterparts in both directions. E.g. the second protocol requires $O(2\log(\frac{1}{\epsilon})(|C| + n + q))$ inexpensive operations and $O(\log(\frac{1}{\epsilon})(n + q))$ expensive ones, while the covert protocol of [13] requires $O(\log(\frac{1}{\epsilon})|C|)$ inexpensive operations and $O(\log(\frac{1}{\epsilon})n)$ expensive ones.

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 $q = q_1 + q_2$.

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

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

Yao's Garbling. For the sake of simplicity and generality, we do not go into the details of the garbling mechanism and only introduce the notations we need to describe our protocols. We refer the reader to [12,2] for different approaches to creating the garbled circuits.

Given a garbled circuit gc, we denote by $\mathsf{label}(gc, j, b)$ the label of wire j corresponding to bit value b. Also, we denote by $\mathsf{Garb}(C, r)$ the (deterministic) garbling of circuit C using randomness r. (In practice, r would be a short seed

for a pseudo-random function). For simplicity, we assume that the labels of the circuit's output-wires include also the actual output bits (thus, allowing the evaluator to learn the output).

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

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

[19] shows an implementation of BCOT with a cost of $O(mn)$ expensive operations. Combining their protocol with the OT-extension of [15] in the Random Oracle Model results in an alternative protocol that requires only $O(s)$ expensive operations and $O(nm)$ inexpensive ones. However, in the latter construction, the commitments on the sender's inputs cannot be opened separately and one needs to decommit all the inputs at once (we use both instantiations in our protocols). See the full version for more details.We denote the first protocol by BCOT1 and the second by BCOT2.

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

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

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

Consistency of the evaluator's input is taken care of by combining the technique of [11] with the OT-extension of [15] and the Free-XOR technique [9]. In a nutshell, P_2's input is encoded using $\max(4n_2, 8t)$ bits in a way that any leakage of less than t of the bits does not reveal meaningful information about P_2's input. During the cut-and-choose, P_2 asks P_1 to reveal all his inputs to

the OTs. If some of the inputs are not consistent with the one P_2 has learned from the OTs, P_2 aborts. This abort leaks information only in case P_1 guessed successfully more than t bits in P_2's encoded input. However, this can happen with only a negligible probability by the way the encoding is done.

As we discussed in Section 1.1, consistency of the garbler's input is addressed using the XOR-gadgets. In the following we describe the main steps of that part.

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 $k \in \text{INP}_1$ 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 $i = 1 \dots t$. In addition, P_2 picks a random string z'_i and generates a garbled circuit $xg_i = \mathsf{Garb}(C_2, z'_i)$, for $i = 1 \dots t$. Both players send the garbled circuits they created to each other. Next, P_1 picks r_j at random for $j \in [t]$ and sends to P_2 the labels that correspond to r_j in gc_j.

OTs for Input Labels. Parties execute OTs and BCOTs in order for each to learn the input-wire labels for his inputs in the circuits/gadgets created by his counterpart. More specifically, first they run any simulatable OT protocol with the OT-extension of [15], where P_1 acts as the sender and P_2 acts as the receiver. They use the technique of [11] for protecting against inconsistent inputs as described earlier. P_1's inputs are the labels of P_2's input-wires in all gc_j (i.e., the inputs are $\mathsf{label}(gc_j, k, 0)$ and $\mathsf{label}(gc_j, k, 1)$ for $k \in \text{INP}_2$ and $j \in [t]$). P_2's input is his actual input. (We ignore here the details of encoding P_2's input.) Second, they execute BCOT2 twice where P_2 acts as the sender and P_1 acts as the receiver: (1) P_2's inputs are the labels of the input-wires in his XOR-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 (2) P_2's inputs are $\mathsf{label}(xg_j, \alpha(k), 0)$ and $\mathsf{label}(xg_j, \alpha(k), 1)$ while P_1's inputs are the bits of r_j. Note that in the first BCOT2, P_1 inputs a single bit for each input bit and receives t input-wire labels. That restricts him to use the same input in all the XOR-gadgets.).

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

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

Cut-and-Choose (first stage). After the OTs/BCOTs, P_1 opens a constant fraction of his garbled circuits/gadgets. In particular, P_1 opens the garbled circuit gc_j for all $j \notin E$, where E is chosen randomly using a joint coin-tossing

protocol. (A joint coin-tossing protocol is needed for the simulation to work.) Moreover, P_1 reveals the random strings r_j-s he used in the opened circuits (by showing the labels he learned from BCOT2), and all his inputs to the OTs for the opened circuits. P_2 checks the correctness of the opened circuits and verifies that the same r_j was used in both gc_j and xg_j for all $j \notin E$. (He also verifies that the values he has received in the OTs for his inputs are consistent with what P_1 revealed, following the technique of [11].)

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

In general, the last step is not sound for all commitments since P_1 can send a commitment for which he does not know the corresponding message and later be able to decommit once P_2 opens the XOR-gadgets (see Footnote 1). There are several ways to overcome this issue. One option is to require P_1 to *prove* that he *knows* how to construct this commitment, or more formally, P_1 commits on the output labels with $\mathsf{Com}(labels, r)$ and proves using a ZKPoK that he knows $labels$ and r. This step can be implemented efficiently for Pedersen's commitment [17], requiring only a small constant number of exponentiations. (When $labels$ is longer than the commitment input length, P_1 picks a random seed $seed$, sends $\mathsf{Com}(seed, r), \mathsf{PRG}(seed) \oplus labels$ and ZKPoK that he knows $seed$ and r.) A more efficient option is to implement $\mathsf{Com}(labels, r)$ in the Random Oracle model using $\mathrm{H}(key \circ labels \circ r)$, where the commitment key key is chosen at random by the receiver (i.e., P_2 in our case). The complexity in this case is only a single call to the random oracle.

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

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

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

4.1 The Model

Following [11,1,6], we use the ideal/real paradigm for our security definition.

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 probabilistic polynomial-time machine adversary $\mathcal{A}$. At the beginning of the execution, each party P_i receives its input x_i. The adversary $\mathcal{A}$ receives an auxiliary information aux and an index that indicates which party it corrupts. For that party, $\mathcal{A}$ receives its input and sends messages on its behalf. Honest parties follow the protocol.

Let $\text{REAL}_{\Pi,\mathcal{A}(aux)}(x_1, x_2)$ be the output vector of the honest party and the adversary $\mathcal{A}$ from the real execution of 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 $x'_i = \mathsf{abort}$, 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. (i.e. $\mathcal{A}$ can learn $g(x'_1, x'_2)$ by observing whether the trusted party aborts or not.)

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

Second abort option: The adversary sends either abort or $\mathsf{continue}$. In the first case, 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 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 probabilistic polynomial time adversary $\mathcal{S}$ in the ideal model such that*

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

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

Comparison with Covert Security. When we let $\epsilon = 1/t$ for any constant t, the above definition is strictly stronger than the standard definition of security against covert adversaries. In covert security, in case of undetected cheating which happens with probability 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 this variant of the definition where the adversary can choose $\epsilon' > \epsilon$ with which it can get caught is of independent interest. Specifically, it yields an alternative definition for covert security that is more convenient to use in simulation-based proofs. (To obtain this alternative definition for covert security, replace the steps that are done with probability ϵ' with the following: (1) The trusted party sends x'_1, x'_2 to $\mathcal{A}$; (2) $\mathcal{A}$ sends the value y to the trusted party, and the trusted party sends it to all parties as their output.)

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

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

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

As discussed in the introduction, in the dual-execution protocol of [14,6] parties engage in two different executions of the semi-honest Yao's garbled circuit protocol, and then run an equality testing protocol to confirm that the outputs of the two executions are the same before revealing the actual output values. We show how to extend this protocol to work in the presence of covert adversaries using the ideas presented in Section 3. For simplicity of the description, from now on we work with $t = \frac{1}{\epsilon}$ (a statistical security parameter) instead of ϵ since t would be the number of circuits each party garbles.

Dual-Execution & Cut-and-Choose. Our first step is to combine the dual-execution protocol with a standard cut-and-choose protocol for covert players. Each player P_i garbles t circuits $gc_1^i, \ldots, gc_t^i$ and sends them to the other player. Parties pick a random value $e \in [t]$, *open* all the circuits $gc_{j \neq e}^i$ and verify that they were constructed properly. This assures that with probability $1 - 1/t$, the remaining *circuit-pair* (gc_j^1, gc_j^2) is properly constructed. As before, they send the garbler's input-wire labels for the e-th circuit, execute OTs for the respective evaluators to learn their input-wire labels, evaluate the circuits, call the Equality Testing functionality and output accordingly.

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

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

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

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

with input-wire k. (For simplicity, we assume the same function is applicable for all C_i-s and C'_i-s.)

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

OTs for Input Labels. Then, they execute BCOTs in order to learn the input-wire labels for both their actual inputs and the r_j^i-s in their counterpart's circuits. More specifically, first they use BCOT1 where P_1 acts as the sender and P_2 acts as the receiver. P_1's inputs are the input-wire labels of P_2's input-wire k in all gc_j^1-s and xg_j^1-s (i.e., the input pairs are $\left(\mathsf{label}(gc_j^1, k, 0), \mathsf{label}(gc_j^1, k, 1)\right)_{j\in[t]}$ and $\mathsf{label}(xg_1^1, k, 0) \circ \cdots \circ \mathsf{label}(xg_t^1, k, 0), \mathsf{label}(xg_1^1, k, 1) \circ \cdots \circ \mathsf{label}(xg_t^1, k, 1)$ for $k \in \mathrm{INP}_2$). P_2's input is his actual input. Second, they use BCOT2 with the labels for the rest of the input-wires of xg_j^1 (i.e., $\mathsf{label}(xg_j^1, \alpha(k), 0)$,$\mathsf{label}(xg_j^1, \alpha(k), 1)$ for $k \in \mathrm{INP}_2$ 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.

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

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

Cut-and-Choose Phase (second opening). P_{3-i} now opens the remaining XOR-gadget xg_e^{3-i}, and decommits all his inputs as a sender to the BCOTs of the XOR-gates (i.e., $\mathsf{label}(xg_1^{3-i}, k, 0) \circ \cdots \circ \mathsf{label}(xg_t^{3-i}, k, 0), \mathsf{label}(xg_1^{3-i}, k, 1) \circ \cdots \circ \mathsf{label}(xg_t^{3-i}, k, 1)$ in BCOT1, and $\mathsf{label}(xg_e^{3-i}, \alpha(k), 0), \mathsf{label}(xg_e^{3-i}, \alpha(k), 1)$ in BCOT2, both for $k \in \mathrm{INP}_i$). (We stress that only the XOR-gates of wires 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 BCOTs inputs were consistent with the XOR-gates. If everything is correct he decommits his commitment, otherwise he outputs $\perp$ and aborts. (Note that P_i reveals his output only *after* he verified that all the XOR-gates P_{3-i} generated were properly constructed. Since the only secrets in these gates are P_i's inputs, revealing them does not help P_i learn any new information.) P_{3-i} verifies that the decommitted values are valid output-wire labels, and compares the actual output with their output he obtains from evaluation of xg_e^i. If either check fails, P_{3-i} outputs $\perp$.

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

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

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

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

References

1. Aumann, Y., Lindell, Y.: Security against covert adversaries: Efficient protocols for realistic adversaries. J. Cryptol. 23(2), 281–343 (2010)
2. Bellare, M., Hoang, V.T., Rogaway, P.: Foundations of garbled circuits. In: CCS 2012, pp. 784–796. ACM (2012)
3. Fischlin, M.: Trapdoor Commitment Schemes and Their Applications. Ph.D. Thesis (Doktorarbeit), Department of Mathematics, Goethe-University, Frankfurt, Germany (2001)
4. Goyal, V., Mohassel, P., Smith, A.: Efficient two party and multi party computation against covert adversaries. In: Smart, N.P. (ed.) EUROCRYPT 2008. LNCS, vol. 4965, pp. 289–306. Springer, Heidelberg (2008)

5. Huang, Y., Evans, D., Katz, J., Malka, L.: Faster secure two-party computation using garbled circuits. In: Security 2011, pp. 35–35. USENIX Association (2011)
6. Huang, Y., Katz, J., Evans, D.: Quid-pro-quo-tocols: Strengthening semi-honest protocols with dual execution. In: SP 2012, pp. 272–284. IEEE Computer Society (2012)
7. Ishai, Y., Kushilevitz, E., Ostrovsky, R., Prabhakaran, M., Sahai, A.: Efficient non-interactive secure computation. In: Paterson, K.G. (ed.) EUROCRYPT 2011. LNCS, vol. 6632, pp. 406–425. Springer, Heidelberg (2011)
8. Jarecki, S., Shmatikov, V.: Efficient two-party secure computation on committed inputs. In: Naor, M. (ed.) EUROCRYPT 2007. LNCS, vol. 4515, pp. 97–114. Springer, Heidelberg (2007)
9. Kolesnikov, V., Schneider, T.: Improved garbled circuit: Free XOR gates and applications. In: Aceto, L., Damgård, I., Goldberg, L.A., Halldórsson, M.M., Ingólfsdóttir, A., Walukiewicz, I. (eds.) ICALP 2008, Part II. LNCS, vol. 5126, pp. 486–498. Springer, Heidelberg (2008)
10. Kreuter, B., Shelat, A., Shen, C.H.: Billion-gate secure computation with malicious adversaries. In: Security 2012, p. 14. USENIX Association (2012)
11. Lindell, Y., Pinkas, B.: An efficient protocol for secure two-party computation in the presence of malicious adversaries. In: Naor, M. (ed.) EUROCRYPT 2007. LNCS, vol. 4515, pp. 52–78. Springer, Heidelberg (2007)
12. Lindell, Y., Pinkas, B.: A proof of security of Yao's protocol for two-party computation. J. Cryptol. 22(2), 161–188 (2009)
13. Lindell, Y., Pinkas, B.: Secure two-party computation via cut-and-choose oblivious transfer. In: Ishai, Y. (ed.) TCC 2011. LNCS, vol. 6597, pp. 329–346. Springer, Heidelberg (2011)
14. Mohassel, P., Franklin, M.: Efficiency tradeoffs for malicious two-party computation. In: Yung, M., Dodis, Y., Kiayias, A., Malkin, T. (eds.) PKC 2006. LNCS, vol. 3958, pp. 458–473. Springer, Heidelberg (2006)
15. Nielsen, J.B., Nordholt, P.S., Orlandi, C., Burra, S.S.: A new approach to practical active-secure two-party computation. In: Safavi-Naini, R. (ed.) CRYPTO 2012. LNCS, vol. 7417, pp. 681–700. Springer, Heidelberg (2012)
16. Nielsen, J.B., Orlandi, C.: LEGO for two-party secure computation. In: Reingold, O. (ed.) TCC 2009. LNCS, vol. 5444, pp. 368–386. Springer, Heidelberg (2009)
17. Pedersen, T.P.: Non-interactive and information-theoretic secure verifiable secret sharing. In: Feigenbaum, J. (ed.) CRYPTO 1991. LNCS, vol. 576, pp. 129–140. Springer, Heidelberg (1992)
18. Pinkas, B., Schneider, T., Smart, N.P., Williams, S.C.: Secure two-party computation is practical. In: Matsui, M. (ed.) ASIACRYPT 2009. LNCS, vol. 5912, pp. 250–267. Springer, Heidelberg (2009)
19. Shelat, A., Shen, C.-H.: Two-output secure computation with malicious adversaries. In: Paterson, K.G. (ed.) EUROCRYPT 2011. LNCS, vol. 6632, pp. 386–405. Springer, Heidelberg (2011)
20. Woodruff, D.P.: Revisiting the efficiency of malicious two-party computation. In: Naor, M. (ed.) EUROCRYPT 2007. LNCS, vol. 4515, pp. 79–96. Springer, Heidelberg (2007)
21. Yao, A.C.C.: How to generate and exchange secrets. In: SFCS 1986, pp. 162–167. IEEE Computer Society (1986)