An Efficient Protocol for Secure Two-Party Computation in the Presence of Malicious Adversaries[*]

Yehuda Lindell[†] Benny Pinkas[‡]

Abstract

We show an efficient secure two-party protocol, based on Yao's construction, which provides security against malicious adversaries. Yao's original protocol is only secure in the presence of semi-honest adversaries, and can be transformed into a protocol that achieves security against malicious adversaries by applying the compiler of Goldreich, Micali and Wigderson (the "GMW compiler"). However, this approach does not seem to be very practical as it requires using generic zero-knowledge proofs.

Our construction is based on applying cut-and-choose techniques to the original circuit and inputs. Security is proved according to the ideal/real simulation paradigm, and the proof is in the standard model (with no random oracle model or common reference string assumptions). The resulting protocol is computationally efficient: the only usage of asymmetric cryptography is for running $O(1)$ oblivious transfers for each input bit (or for each bit of a statistical security parameter, whichever is larger). Our protocol combines techniques from folklore (like cut-and-choose) along with new techniques for efficiently proving consistency of inputs. We remark that a naive implementation of the cut-and-choose technique with Yao's protocol does *not* yield a secure protocol. This is the first paper to show how to properly implement these techniques, and to provide a full proof of security.

Our protocol can also be interpreted as a constant-round black-box reduction of secure two-party computation to oblivious transfer and perfectly-hiding commitments, or a black-box reduction of secure two-party computation to oblivious transfer alone, with a number of rounds which is linear in a statistical security parameter. These two reductions are comparable to Kilian's reduction, which uses OT alone but incurs a number of rounds which is linear in the depth of the circuit [19].

Key words: secure two-party computation, Yao's protocol, real/ideal simulation paradigm, security against malicious adversaries.

[*]An extended abstract of this work was published in the proceedings of EUROCRYPT 2007.

[†]Dept. of Computer Science, Bar-Ilan University, Israel. Research supported in part by an Infrastructures grant from the Ministry of Science, Israel. Email: `lindell@cs.biu.ac.il`.

[‡]Dept. of Computer Science, University of Haifa, Israel. Research supported in part by the Israel Science Foundation (grant number 860/06). Email: `benny@pinkas.net`.

1 Introduction

Secure two-party computation. In the setting of two-party computation, two parties with respective private inputs x and y, wish to jointly compute a functionality $f(x,y) = (f_1(x,y), f_2(x,y))$, such that the first party receives $f_1(x,y)$ and the second party receives $f_2(x,y)$. Loosely speaking, the security requirements are that nothing is learned from the protocol other than the output (*privacy*), and that the output is distributed according to the prescribed functionality (*correctness*). The actual definition follows the simulation paradigm and blends the above two requirements. Of course, security must be guaranteed even when one of the parties is adversarial. Such an adversary may be *semi-honest* (or passive), in which case it correctly follows the protocol specification, yet attempts to learn additional information by analyzing the transcript of messages received during the execution. In contrast, the adversary may be *malicious* (or active), in which case it can arbitrarily deviate from the protocol specification.

The first general solutions for the problem of secure computation were presented by Yao [31] for the two-party case (with security against semi-honest adversaries) and Goldreich, Micali and Wigderson [12] for the multi-party case (with security even against malicious adversaries). Thus, the results of [31] and [12] constitute important and powerful feasibility results for secure two-party and multi-party computation.

Yao's protocol. In [31], Yao presented a *constant-round* protocol for securely computing any functionality in the presence of semi-honest adversaries. Denote party P_1 and P_2's respective inputs by x and y and let f be the functionality that they wish to compute (for simplicity, assume that both parties wish to receive $f(x,y)$). Loosely speaking, Yao's protocol works by having one of the parties (say party P_1) first generate a "garbled" (or encrypted) circuit computing $f(x,\cdot)$ and then send it to P_2. The circuit is such that it reveals nothing in its encrypted form and therefore P_2 learns nothing from this stage. However, P_2 can obtain the output $f(x,y)$ by "decrypting" the circuit. In order to ensure that P_2 learns nothing more than the output itself, this decryption must be "partial" and must reveal $f(x,y)$ only. Without going into unnecessary details, this is accomplished by P_2 obtaining a series of keys corresponding to its input y, such that given these keys and the circuit, the output value $f(x,y)$, and only this value, may be obtained. Of course, P_2 must somehow receive these keys without revealing anything about y to P_1. This can be accomplished by running $|y|$ instances of a secure 1-out-of-2 Oblivious Transfer protocol [29, 8]. Yao's generic protocol is highly efficient, and even practical, for functionalities that have relatively small circuits. An actual implementation of the protocol was presented in [23], with very reasonable performance.

Security against malicious behavior. Yao's protocol is only secure in the presence of relatively weak semi-honest adversaries. Thus, an important question is how to "convert" the protocol into one that is secure in the presence of malicious adversaries, while preserving the efficiency of the original protocol to the greatest extent possible. Of course, one possibility is to use the compiler of Goldreich, Micali and Wigderson [12]. This compiler converts any protocol that is secure for semi-honest adversaries into one that is secure for malicious adversaries, and as such is a powerful tool for demonstrating feasibility. However, it is based on reducing the statement that needs to be proved (in our case, the honesty of the parties' behavior) to an NP-complete problem, and using generic zero-knowledge proofs to prove this statement. The resulting secure protocol therefore runs in polynomial time but is rather inefficient. (For more details on existing methods for proving security against malicious behavior see the section on related work below.)

Malicious behavior and cut-and-choose. Consider for a moment what happens if party P_1 is malicious. In such a case, it can construct a garbled circuit that computes a function that is different to the one that P_1 and P_2 agreed to compute. A folklore solution to this problem uses the "cut-and-choose" technique. According to this technique, P_1 first constructs *many* garbled circuits and sends them to P_2. Then, P_2 asks P_1 to "open" half of them (namely, reveal the decryption keys corresponding to these circuits). P_1 opens the requested half, and P_2 checks that they were constructed correctly. If they were, then P_2 evaluates the rest of the circuits and derives the output from them. The idea behind this methodology is that if a malicious P_1 constructs the circuits incorrectly, then P_2 will detect this with high probability. Clearly, this solution solves the problem of P_1 constructing the circuit incorrectly. However, it does not suffice. First, it creates new problems within itself. Most outstandingly, once the parties now evaluate a number of circuits, some mechanism must be employed to make sure that they use the same input when evaluating each circuit (otherwise, as we show below, an adversarial party could learn more information than allowed). Second, in order to present a proof of security based on *simulation*, there are additional requirements that are not dealt with by just employing cut-and-choose (e.g., input extraction). Third, the folklore description of cut-and-choose is very vague and there are a number of details that are crucial when implementing it. For example, if P_2 evaluates many circuits, then the protocol must specify what P_2 should do if it does not receive the same output in every circuit. If the protocol requires P_2 to abort in this case (because it detected cheating from P_1), then this behavior actually yields a concrete attack in which P_1 can always learn a specified bit of P_2's input. It can be shown that P_2 must take the majority output and proceed, even if it knows that P_1 has attempted to cheat. This is just one example of a subtlety that must be dealt with. Another example relates to the fact that P_1 may be able to construct a circuit that can be opened with two different sets of keys: the first set opens the circuit correctly and the second incorrectly. In such a case, an adversarial P_1 can pass the basic cut-and-choose test by opening the circuits to be checked correctly. However, it can also supply incorrect keys to the circuits to be computed and thus cause the output of the honest party to be incorrect.

Our contributions. This paper provides several contributions:

- *Efficient protocol for malicious parties:* We present an implementation of Yao's protocol with the cut-and-choose methodology, which is secure in the presence of malicious adversaries and is computationally efficient: the protocol does not use public-key operations, except for performing oblivious transfers for every input bit of P_2. For n-bit inputs and a statistical security parameter s the protocol uses $O(\max(s, n))$ oblivious transfers. Thus, when the input size is of the same order as the security parameter, only $O(1)$ oblivious transfers are needed per input bit. Due to the use of cut-and-choose, we incur a multiplicative increase by a factor of s in the communication complexity of our protocol, which is $O(s|C| + s^2 n)$.

  Beyond carefully implementing the cut-and-choose technique on the circuits in order to ensure that the garbled circuits are constructed correctly, we present a new method for enforcing the parties to use the same input in every circuit. This method involves "consistency checks" that are based on cut-and-choose tests which are applied to *sets of commitments* to the garbled values associated with the input wires of the circuit, rather than to the circuits themselves.

  In actuality, we combine the cut-and-choose test over the circuits together with the cut-and-choose test over the commitments in order to obtain a secure solution. The test is rather complex conceptually, but is exceedingly simple to implement. Specifically, P_1 just needs to generate a number of commitments to the garbled values associated with the input wires, and then open them based on cut-and-choose queries from P_2. (Actually, these cut-and-choose queries are chosen jointly by the parties using a simple coin-tossing protocol; this is necessary for achieving simulation.) We note that in this work we have emphasized providing a clear and full proof of the protocol, rather than fully optimizing its overhead at the expense of complicating the proof.

- *Simulation based proof:* We present a rigorous proof of the security of the protocol, based on the real/ideal-model simulation paradigm [5, 10]. The proof is in the standard model, with no random oracle model or common random string assumptions. The protocol was designed to support such a proof, rather than make do with separate proofs of privacy and correctness. (It is well-known that it is strictly harder to obtain a simulation based proof rather than security under such definitions.) One important advantage of simulation based proofs is that they enable the use of the protocol as a building block in more complicated protocols, while proving the security of the latter using general composition theorems like those of [5, 10]. (For example, the secure protocol of [1] for finding the k^{th} ranked element is based on invoking several secure computations of simpler functions, and provides simulation based security against malicious adversaries if the invoked computations have a simulation based proof. However, prior to our work there was no known way, except for the GMW compiler, of efficiently implementing these computations with this level of security, and as a result there was no efficient way of implementing the protocols of [1] with security against malicious adversaries.) See [5, 10] for more discussion on the importance of simulation-based definitions.

  We stress that although a number of the techniques employed here have appeared previously in the literature, to the best of our knowledge this is the first paper to present a rigorous proof of security for the resulting protocol. A straightforward implementation of the folklore techniques does not yield a secure protocol, and great care needs to be taken in order to enable the proof to go through.

- *A black-box reduction:* Our protocol can be interpreted as a constant-round black-box reduction of secure two-party computation to oblivious transfer and perfectly-hiding commitments. The perfectly-hiding commitments are only used for conducting constant-round joint coin-tossing of a string of length s, where s is a statistical security parameter. This coin-tossing can be carried out sequentially (bit by bit), without using perfectly-hiding commitments. We therefore also obtain an $O(s)$ round black-box reduction of secure two-party computation to oblivious transfer alone. These two reductions are comparable to Kilian's reduction, which uses OT alone but incurs a number of rounds which is linear in the depth of the circuit [19]. In addition, our reduction is much more efficient than that of [19].

Related work. As we have mentioned, this paper presents a protocol which **(1)** has a proof of security against malicious adversaries in the standard model, according to the real/ideal model simulation definition, **(2)** has essentially the same computational overhead as Yao's original protocol (which is only secure against semi-honest adversaries), and **(3)** has a somewhat larger communication overhead, which depends on a statistical security parameter s.

We compare this result to other methods for securing Yao's protocol against malicious parties. There are several possible approaches to this task:

- The parties can reduce the statement about the honesty of their behavior to a statement which has a well-known zero-knowledge proof, and then prove this statement. This is the approach taken by the GMW compiler [12]. The resulting secure protocol is not black-box, and is rather inefficient.

- Another approach is to apply a cut-and-choose modification to Yao's protocol. Mohassel and Franklin [25] show such a protocol which has about the same overhead as ours, namely a communication overhead of $O(|C|s + n^2s)$ for a circuit C with n inputs, and a statistical security parameter s. The protocol of [25] provides output to the circuit evaluator alone. It enables, however, the circuit constructor to carry out the following attack: it can corrupt its OT input which corresponds to a 0 value of the first input bit of the circuit evaluator, while not corrupting the OT input for the 1 value. Other than that it follows the protocol. This behavior forces the circuit evaluator to abort if its first input bit is 0, while if its first input bit is 1 it does not learn anything at all about the attack. If the evaluator complains, then the circuit constructor can conclude that its first input bit is 0, and therefore the evaluator cannot complain if it wants to preserve its privacy. (This attack is similar to the attack we describe in Section 3.2 where we discuss the encoding of P_2's input.) The protocol therefore does not provide security according to a standard definition. (We note however that this attack can be prevented using the methods we describe in Section 3.2 for encoding P_2's input.) Another protocol which is based on cut-and-choose is described in [20]. This protocol uses committed OT to address attacks similar to the one described above. We stress that both of these papers ([25, 20]) lack a full proof of security, and to our best judgment they need considerable changes in order to support security according to a simulation based definition.
- The construction of [25] was improved by Woodruff [30], who described how to reduce the communication to $O(s|C| + sn) = O(s|C|)$, using expanders. It seems that this approach can also be applied to our construction, but we have not incorporated it into this work.
- Jarecki and Shmatikov [16] designed a protocol in which the parties efficiently prove, gate by gate, that their behavior is correct. The protocol runs in a constant number of rounds, and is based on the use of a special homomorphic encryption system, which is used to encode the tables of each gate of the circuit (compared to the use of symmetric encryption in Yao's original protocol and in our paper). The protocol is secure in a universally composable way under the decisional composite residuosity and the strong RSA assumptions, assuming a common reference string. Compared to our protocol, this protocol has a greater computational overhead ($O(|C|)$ rather than $O(n)$ public key operations), but a smaller communication overhead ($O(|C|)$ rather than $O(s|C| + s^2n)$. In addition, our protocol can be based on general assumptions.

In this paper, we construct an efficient protocol for *general* secure computation. Thus, we do not (and cannot) compete with protocols that are constructed for specific tasks, like voting, auctions, etcetera. We also do not discuss here the large body of work that considers the efficiency of secure *multi-party* computation.

Organization. We present standard definitions of security for secure two-party computation in Section 2.1. Then, in Section 2.2 we show that a functionality that provides outputs to both parties can be securely reduced to one which provides output for a single party, and therefore we can focus on the latter case. In Section 3 we describe our protocol and in Section 4 prove its security. In Section 5 we focus on the issue of efficiency and begin by analyzing the complexity of our protocol. Then, in Section 5.1 we discuss efficient implementations of the primitives that are used in our protocol. The basic protocol we describe increases the number of inputs, and therefore the number of OT invocations. In Section 5.2 we show how to reduce this number of OT invocations in order to improve efficiency. We remark that in Appendix A we provide a description of Yao's basic protocol for two-party that is based on [22].

2 Preliminaries

2.1 Definitions – Secure Computation

In this section we present the definition for secure two-party computation. The following description and definition is based on [10, Chapter 7], which in turn follows [13, 24, 4, 5].

Two-party computation. A two-party protocol problem is cast by specifying a random process that maps pairs of inputs to pairs of outputs (one for each party). We refer to such a process as a functionality and denote it $f : \{0,1\}^* \times \{0,1\}^* \rightarrow \{0,1\}^* \times \{0,1\}^*$, where $f = (f_1, f_2)$. That is, for every pair of inputs (x, y), the output-pair is a random variable $(f_1(x, y), f_2(x, y))$ ranging over pairs of strings. The first party (with input x) wishes to obtain $f_1(x, y)$ and the second party (with input y) wishes to obtain $f_2(x, y)$.[1]

Adversarial behavior. Loosely speaking, the aim of a secure two-party protocol is to protect an honest party against dishonest behavior by the other party. In this paper, we consider *malicious adversaries* who may arbitrarily deviate from the specified protocol. When considering malicious adversaries, there are certain undesirable actions that cannot be prevented. Specifically, a party may refuse to participate in the protocol, may substitute its local input (and use instead a different input) and may abort the protocol prematurely. One ramification of the adversary's ability to abort, is that it is impossible to achieve "fairness". That is, the adversary may obtain its output while the honest party does not. As is standard for two-party computation, in this work we consider a static corruption model, where one of the parties is adversarial and the other is honest, and this is fixed before the execution begins.

Security of protocols (informal). The security of a protocol is analyzed by comparing what an adversary can do in the protocol to what it can do in an ideal scenario that is secure by definition. This is formalized by considering an *ideal* computation involving an incorruptible *trusted third party* to whom the parties send their inputs. The trusted party computes the functionality on the inputs and returns to each party its respective output. Loosely speaking, a protocol is secure if any adversary interacting in the real protocol (where no trusted third party exists) can do no more harm than if it was involved in the above-described ideal computation.

Execution in the ideal model. As we have mentioned, some malicious behavior cannot be prevented (for example, early aborting). This behavior is therefore incorporated into the ideal model. An ideal execution proceeds as follows:

Inputs: Each party obtains an input, denoted w ($w = x$ for P_1, and $w = y$ for P_2).

Send inputs to trusted party: An honest party always sends w to the trusted party. A malicious party may, depending on w, either abort or send some $w' \in \{0,1\}^{|w|}$ to the trusted party.

Trusted party answers first party: In case it has obtained an input pair (x, y), the trusted party first replies to the first party with $f_1(x, y)$. Otherwise (i.e., in case it receives only one valid input), the trusted party replies to both parties with a special symbol $\perp$.

[1] Another way of defining f is as a deterministic function $f : \{0,1\}^* \times \{0,1\}^* \times \{0,1\}^* \rightarrow \{0,1\}^* \times \{0,1\}^*$, where the third input is uniformly chosen. That is, first a uniformly distributed string r is chosen, and then the first and second parties receive $f_1(x, y, r)$ and $f_2(x, y, r)$, respectively.

Trusted party answers second party: In case the first party is malicious it may, depending on its input and the trusted party's answer, decide to *stop* the trusted party by sending it $\perp$ after receiving its output. In this case the trusted party sends $\perp$ to the second party. Otherwise (i.e., if not stopped), the trusted party sends $f_2(x,y)$ to the second party.

Outputs: An honest party always outputs the message it has obtained from the trusted party. A malicious party may output an arbitrary (probabilistic polynomial-time computable) function of its initial input and the message obtained from the trusted party.

Let $f : \{0,1\}^* \times \{0,1\}^* \to \{0,1\}^* \times \{0,1\}^*$ be a functionality, where $f = (f_1, f_2)$, and let $\overline{M} = (M_1, M_2)$ be a pair of non-uniform probabilistic *expected* polynomial-time machines (representing parties in the ideal model). Such a pair is admissible if for at least one $i \in \{1,2\}$ we have that M_i is honest (i.e., follows the honest party instructions in the above-described ideal execution). Then, the joint execution of f under $\overline{M}$ in the ideal model (on input pair (x,y)), denoted $\text{IDEAL}_{f,\overline{M}}(x,y)$, is defined as the output pair of M_1 and M_2 from the above ideal execution.

Execution in the real model. We next consider the real model in which a real (two-party) protocol is executed (and there exists no trusted third party). In this case, a malicious party may follow an arbitrary feasible strategy; that is, any strategy implementable by non-uniform probabilistic polynomial-time machines. In particular, the malicious party may abort the execution at any point in time (and when this happens prematurely, the other party is left with no output).

Let f be as above and let Π be a two-party protocol for computing f. Furthermore, let $\overline{M} = (M_1, M_2)$ be a pair of non-uniform probabilistic polynomial-time machines (representing parties in the real model). Such a pair is admissible if for at least one $i \in \{1,2\}$ we have that M_i is honest (i.e., follows the strategy specified by Π). Then, the joint execution of Π under $\overline{M}$ in the real model (on input pair (x,y)), denoted $\text{REAL}_{\Pi,\overline{M}}(x,y)$, is defined as the output pair of M_1 and M_2 resulting from the protocol interaction.

Security as emulation of a real execution in the ideal model. Having defined the ideal and real models, we can now define security of protocols. Loosely speaking, the definition asserts that a secure two-party protocol (in the real model) emulates the ideal model (in which a trusted party exists). This is formulated by saying that admissible pairs in the ideal model are able to simulate admissible pairs in an execution of a secure real-model protocol.

Definition 1 (secure two-party computation): *Let f and Π be as above. Protocol Π is said to* securely compute f (in the malicious model) *if for every pair of admissible non-uniform probabilistic polynomial-time machines $\overline{A} = (A_1, A_2)$ for the real model, there exists a pair of admissible non-uniform probabilistic expected polynomial-time machines $\overline{B} = (B_1, B_2)$ for the ideal model, such that*

$$\left\{\text{IDEAL}_{f,\overline{B}}(x,y)\right\}_{x,y \text{ s.t. } |x|=|y|} \overset{c}{\equiv} \left\{\text{REAL}_{\Pi,\overline{A}}(x,y)\right\}_{x,y \text{ s.t. } |x|=|y|}$$

Namely, the two distributions are computationally indistinguishable.

We note that the above definition assumes that the parties know the input lengths (this can be seen from the requirement that $|x| = |y|$). Some restriction on the input lengths is unavoidable, see [10, Section 7.1] for discussion. We also note that we allow the ideal adversary/simulator to run in expected (rather than strict) polynomial-time. This is essential for achieving constant-round protocols; see [3].

We denote the security parameter by n and, for the sake of simplicity, unify it with the length of the inputs (thus we consider security for "all sufficiently long inputs"). Everything in the paper remains the same if a separate security parameter n is used, and we consider security for inputs of all lengths. We will also use a statistical security parameter s; see the beginning of Section 3.1 for an explanation of the use of this separate parameter.

The hybrid model. Our protocol uses a secure oblivious transfer protocol as a subprotocol. It has been shown in [5] that it suffices to analyze the security of such a protocol in a hybrid model in which the parties interact with each other *and* have access to a trusted party that computes the oblivious transfer protocol for them. This model is a hybrid of the real and ideal models: on the one hand, the parties send regular messages to each other, like in the real model; on the other hand, the parties have access to a trusted party, like in the ideal model. We remark that the composition theorem of [5] holds for the case that the subprotocol executions are all run sequentially (and the messages of the protocol calling the subprotocol do not overlap with any execution). We also remark that if the oblivious transfer subprotocol is secure under parallel composition, then it is straightforward to extend [5] so that the subprotocols may be run in parallel (again, as long as the messages of the protocol calling the subprotocol do not overlap with any execution).

2.2 Functionalities that Provide Output to a Single Party

In the definition above, we have considered the case that both parties receive output, and these outputs may be *different*. However, the presentation of our protocol is far simpler for the case that only party P_2 receives output. We will show now that this suffices for the general case. That is, any protocol that can securely compute *any* efficient functionality $f(x, y)$ where only P_2 receives output, can be used to securely compute *any* efficient functionality $f = (f_1, f_2)$ where party P_1 receives $f_1(x, y)$ and party P_2 receives $f_2(x, y)$.

Let $f = (f_1, f_2)$ be a functionality. We wish to construct a secure protocol in which P_1 receives $f_1(x, y)$ and P_2 receives $f_2(x, y)$; as a building block we use a protocol for computing any efficient functionality with the limitation that only P_2 receives output. Let $\mathcal{F}$ be a field that contains the range of values $\{f_1(x, y)\}_{x,y\in\{0,1\}^n}$, and let p, a, b be randomly chosen elements in $\mathcal{F}$. Then, in addition to x, party P_1's input includes the elements p, a, b. Furthermore, define a functionality g (that has only a single output) as follows:

$$g((p, a, b, x), y) = (\alpha, \beta, f_2(x, y))$$

where $\alpha = p + f_1(x, y)$, $\beta = a \cdot \alpha + b$, and the arithmetic operations are defined in $\mathcal{F}$. Note that α is a one-time pad encryption of P_1's output $f_1(x, y)$, and β is an information-theoretic message authentication tag of α (specifically, $a\alpha + b$ is a pairwise-independent hash of α). Now, the parties compute the functionality g, using a secure protocol in which only P_2 receives output. Following this, P_2 sends the pair (α, β) to P_1. Party P_1 checks that $\beta = a \cdot \alpha + b$; if yes, it outputs $\alpha - p$, and otherwise it outputs $\perp$.

It is easy to see that P_2 learns nothing about P_1's output $f_1(x, y)$, and that it cannot alter the output that P_1 will receive (beyond causing it to abort), except with probability $1/|\mathcal{F}|$. (We assume that $1/|\mathcal{F}$ is the required probability for detecting attempts to alter the output. If it is required instead that any change by P_2 to P_1's output is detected with probability 2^{-s}, then the parameters a, b and the computation of $\beta = a \cdot \alpha + b$ can be defined in a field whose representation is s bits long.) We remark that it is also straightforward to construct a simulator for the above protocol.[2] We therefore conclude with the following proposition:

Proposition 2 *Assume that there exists a protocol for securely computing any functionality in which only a single party receives output. Then, there exists a protocol for securely computing any functionality in which both parties receive output.*

We remark that the circuit for computing g is only mildly larger than that for computing f. Thus, the construction above is also efficient and has only a mild effect on the complexity of the secure protocol (assuming that the complexity of the original protocol, where only P_2 receives output, is proportional to the size of the circuit computing f).

3 The Protocol

Our protocol is based upon Yao's garbled circuit construction, which is secure in the presence of semi-honest adversaries [31]. That protocol has two parties: P_1 (who is the the *sender*, or circuit *constructor*), and P_2 (who is the *receiver*, or the circuit *evaluator*). The protocol is described in Appendix A, and a full description and proof of this protocol is available in [22]. Our presentation from here on assumes full familiarity with Yao's basic protocol.

There are a number of issues that must be dealt with when attempting to make Yao's protocol secure against malicious adversaries rather than just semi-honest ones (beyond the trivial observation that the oblivious transfer subprotocol must now be secure in the presence of malicious adversaries).

First and foremost, a malicious P_1 must be forced to construct the garbled circuit correctly so that it indeed computes the desired function. The method that is typically referred to for this task is called *cut-and-choose.* According to this methodology, P_1 constructs many independent copies of the garbled circuit and sends them to P_2. Party P_2 then asks P_1 to open half of them (chosen randomly). After P_1 does so, and party P_2 checks that the opened circuits are correct, P_2 is convinced that most of the remaining (unopened) garbled circuits are also constructed correctly. (If there are many incorrectly constructed circuits, then with high probability, one of those circuits will be in the set that P_2 asks to open.) The parties can then evaluate the remaining unopened garbled circuits as in the original protocol for semi-honest adversaries, and take the majority output-value.[3]

The cut-and-choose technique described above indeed solves the problem of a malicious P_1 constructing incorrect circuits. However, it also generates new problems! The primary problem that arises is that since there are now many circuits being evaluated, we must make sure that both

[2]Note that in order to meet Definition 1, one must actually switch the roles of P_1 and P_2 above. This is due to the fact that by our definition of the ideal model, a corrupted P_1 is given the capability to cause P_2 to abort, even after P_1 has received its own output. In contrast, a corrupted P_2 is not given this capability. In the protocol described, P_2 receives output first and can cause P_1 to abort, rather than the reverse. This is "fixed" by just switching the roles.

[3]The reason for taking the majority value as the output is that the aforementioned test only reveals a single incorrectly constructed circuit with probability 1/2. Therefore, if P_1 generates a single or constant number of "bad" circuits, there is a reasonable chance that it will not be caught. In contrast, there is only an exponentially small probability that the test reveals no corrupt circuit *and* at the same time a majority of the circuits that are not checked are incorrect. Consequently, with overwhelming probability it holds that if the test succeeds and P_2 takes the majority result of the remaining circuits, the result is correct. We remark that the alternative of aborting in case not all the outputs are the same (namely, where cheating is detected) is not secure and actually yields a concrete attack. The attack works as follows. Assume that P_1 is corrupted and that it constructs all of the circuits correctly except for one. The "incorrect circuit" is constructed so that it computes the exclusive-or of the desired function f with the first bit of P_2's input. Now, if P_2 policy is to abort as soon as two outputs are not the same then P_1 knows that P_2 aborts if, and only if, the first bit of its input is 1.

P_1 and P_2 use the same inputs in each circuit; we call these *consistency checks.* Consistency checks are important since if the parties were able to provide different inputs to different copies of the circuit, then they can learn information that is different from the desired output of the function. It is obvious that P_2 can do so, since it observes the outputs of all circuits, but in fact even P_1, who only gets to see the majority output, can learn additional information: information: Suppose, for example, that the protocol computes n invocations of a circuit computing the inner-product between n bit inputs. A malicious P_2 could provide the inputs $\langle 10\cdots 0\rangle$, $\langle 010\cdots 0\rangle$,...,$\langle 0\cdots 01\rangle$, and learn all of P_1's input. If, on the other hand, P_1 is malicious, it could also provide the inputs $\langle 10\cdots 0\rangle$, $\langle 010\cdots 0\rangle$,...,$\langle 0\cdots 01\rangle$. In this case, P_2 sends it the value which is output by the majority of the circuits, and which is equal to the majority value of P_2's input bits.

Another problem that arises when proving security is that the simulator must be able to fool P_2 and give it incorrect circuits (even though P_2 runs a cut-and-choose test). This is solved using rather standard techniques, like choosing the circuits to be opened via a coin-tossing protocol (to our knowledge, this issue has gone unnoticed in all previous applications of cut-and-choose to Yao's protocol). Yet another problem is that P_1 might provide corrupt inputs to some of P_2's possible choices in the OT protocols. P_1 might then learn P_2's input based on whether or not P_2 aborts the protocol.

We begin by presenting a high-level overview of the protocol. We then proceed to describe the consistency checks, and finally the full protocol.

3.1 High-Level Overview

We work with two security parameters. The parameter n is the security parameter for the commitment schemes, encryption, and the oblivious transfer protocol. The parameter s is a statistical security parameter which specifies how many garbled circuits are used. The difference between these parameters is due to the fact that the value of n depends on computational assumptions, whereas the value of s reflects the possible error probability that is incurred by the cut-and-choose technique and as such is a "statistical" security parameter. Although it is possible to use a single parameter n, it may be possible to take s to be much smaller than n. Recall that for simplicity, and in order to reduce the number of parameters, we denote the length of the input by n as well.

Protocol 1 (high-level overview): *Parties P_1 and P_2 have respective inputs x and y, and wish to compute the output $f(x, y)$ for P_2.*

0. *The parties decide on a circuit computing f. They then change the circuit by replacing each input wire of P_2 by a gate whose input consists of s new input wires of P_2 and whose output is the exclusive-or of these wires (such an s-bit exclusive-or gate can be implemented using $s-1$ two-bit exclusive-or gates). Consequently, the number of input wires of P_2 increases by a factor of s. This is depicted in Figure 1. (In Section 5.2, we show how to reduce the number of inputs.)*

1. *P_1 commits to s different garbled circuits computing f, where s is a statistical security parameter. (See Appendix A for a description of the garbled-circuit construction.) P_1 also generates additional commitments to the garbled values corresponding to the input wires of the circuits. These commitments are constructed in a special way in order to enable consistency checks.*

2. *For every input bit of P_2, parties P_1 and P_2 run a 1-out-of-2 oblivious transfer protocol in which P_2 learns the garbled values of input wires corresponding to its input.*

3. *P_1 sends to P_2 all the commitments of Step 1.*

4. *P_1 and P_2 run a coin-tossing protocol in order to choose a random string that defines which commitments and which garbled circuits will be opened.*

5. *P_1 opens the garbled circuits and committed input values that were chosen in the previous step. P_2 verifies the correctness of the opened circuits and runs consistency checks based on the decommitted input values.*

6. *P_1 sends P_2 the garbled values corresponding to P_1's input wires in the unopened circuits. P_2 runs consistency checks on these values as well.*

7. *Assuming that all of the checks pass, P_2 evaluates the unopened circuits and takes the majority value as its output.*

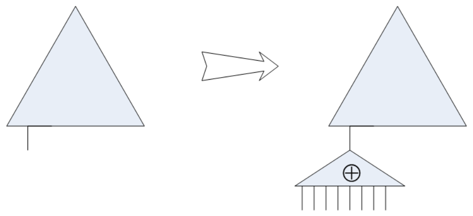


Figure 1: Transforming one of P_2's input wires (Step 0 of Protocol 1).

3.2 Checks for Correctness and Consistency

As can be seen from the above overview, P_1 and P_2 run a number of checks, with the aim of forcing a potentially malicious P_1 to construct the circuits correctly and use the same inputs in (most of) the evaluated circuits. This section describes these checks. Unfortunately, we are unable to present the protocol, nor prove its security, in a modular fashion. Rather, the correctness and consistency checks are closely intertwined with the other parts of the protocol. We will therefore describe the correctness and consistency checks here, and describe the full protocol is Section 3.3. We hope that this improves the readability of the actual protocol.

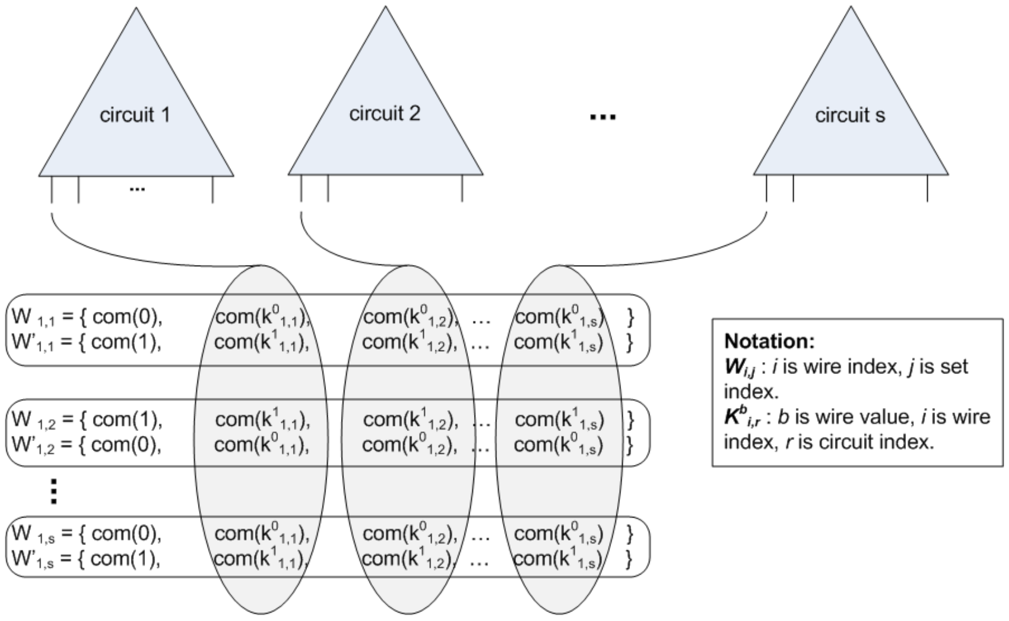
Encoding P_2's input: As mentioned above, a malicious P_1 may provide corrupt input to one of P_2's possible inputs in an OT protocol. If P_2 chooses to learn this input it will not be able to decode the garbled tables which use this value, and it will therefore have to abort. If P_2 chooses to learn the other input associated with this wire then it will not notice that the first input is corrupt. P_1 can therefore learn P_2's input based on whether or not P_2 aborts. (Note that checking that the circuit is well-formed will not help in thwarting this attack, since the attack is based on changing P_1's input to the OT protocol.) The attack is prevented by the parties replacing each input bit of P_2 with s new input bits whose exclusive-or is used instead of the original input (this step was described as Step 0 of Protocol 1, and is analyzed in Lemma 5). P_2 therefore has 2^{s-1} ways to encode a 0 input, and 2^{s-1} ways to encode a 1, and given its input it chooses an encoding with uniform probability. The parties then execute the protocol with the new circuit, and P_2 uses oblivious transfer to learn the garbled values of its new inputs. As we will show, if P_1 supplies incorrect values as garbled values that are associated with P_2's input, the probability of P_2 detecting this cheating is *almost independent* (up to a bias of 2^{-s+1}) of P_2's actual input. This is not true if P_2's inputs are not "split" in the way described above. The encoding presented here increases the number of P_2's input bits and, respectively, the number of OTs, from n to ns. In Section 5.2 we show how to reduce the number of new inputs for P_2 (and thus OTs) to a total of only $O(\max(s, n))$.

An unsatisfactory method for proving consistency of P_1's input: Consider the following idea for forcing P_1 to provide the same input to all circuits. Let s be a security parameter and assume that there are s garbled copies of the circuit. Then, P_1 generates two ordered sets of commitments for every wire of the circuit. Each set contains s commitments: the "0 set" contains commitments to the garbled encodings of 0 for this wire in every circuit, and the "1 set" contains commitments to the garbled encodings of 1 for this wire in every circuit. P_2 receives these commitments from P_1 and then chooses a random subset of the circuits, which will be defined as check-circuits. These circuits will never be evaluated and are used only for checking correctness and consistency. Specifically, P_2 asks P_1 to de-garble all of the check-circuits and to open the values that correspond to the check-circuits in *both* commitment sets. (That is, if circuit i is a check-circuit, then P_1 decommits to both the 0 encoding and 1 encoding of all the input wires in circuit i.) Upon receiving the decommitments, P_2 verifies that all opened commitments from the "0 set" correspond to garbled values of 0, and that a similar property holds for commitments from the "1 set".

It now remains for P_2 to evaluate the remaining circuits. In order to do this, P_1 provides (for each of its input wires) the garbled values that are associated with the wire in all of the remaining circuits. Then, P_1 must prove that all of these values come from the same set, without revealing whether the set that they come from is the "0 set" or the "1 set" (otherwise, P_2 will know P_1's input). In this way, on the one hand, P_2 does not learn the input of P_1, and on the other hand, it is guaranteed that all of the values come from the same set, and so P_1 is forced into using the same input in all circuits. This proof can be carried out using, for example, the proofs of partial knowledge of [6]. However, this would require n proofs, each for s values, thereby incurring $O(ns)$ costly asymmetric operations which we want to avoid.

Proving consistency of P_1's input: P_1 can prove consistency of its inputs without using public-key operations. The proof is based on a cut-and-choose test for the consistency of the commitment sets, which is combined with the cut-and-choose test for the correctness of the circuits. (Note that in the previous proposal, there is only one cut-and-choose test, and it is for the correctness of the circuits.) We start by providing a high level description of the proof of consistency: The proof is based on P_1 constructing, *for each of its input wires*, s pairs of sets of commitments. One set in every pair contains commitments to the 0 values of this wire in *all* circuits, and the other set is the same with respect to 1. The protocol chooses a random subset of these pairs, and a random subset of the circuits, and checks that these sets provide consistent inputs for these circuits. Then the protocol evaluates the *remaining* circuits, and asks P_1 to open, in each of the *remaining* pairs, and only in one set in every pair, its garbled values for all evaluated circuits. (In this way, P_2 does not learn whether these garbled values correspond to a 0 or to a 1.) In order for the committed sets and circuits to pass P_2's checks, there must be large subsets C and S, of the circuits and commitment sets, respectively, such that every choice of a circuit from C and a commitment set from S results in a circuit and garbled values which compute the desired function f. P_2 accepts the verification stage only if all the circuits and sets it chooses to check are from C and S, respectively. This means that if P_2 does not abort then circuits which are not from C are likely to be a minority of the evaluated circuits, and a similar argument holds for S. Therefore the majority result of the evaluation stage is correct. The exact construction is as follows:

Stage 1 – Commitments: P_1 generates s garbled versions of the circuit. Furthermore, it generates commitments to the garbled values of the wires corresponding to P_2's input in each circuit. These commitments are generated in ordered pairs so that the first item in a pair corresponds to the 0 value and the second to the 1 value. The procedure regarding the input bits of P_1 is more complicated (see Figure 2 for a diagram explaining this construction). P_1 generates s pairs of sets of committed values for each of its input wires. Specifically, for every input wire i of P_1, it generates s sets of the form $\{W_{i,j}, W'_{i,j}\}_{j=1}^{s}$; we call these commitment sets. Before describing the content of these sets, denote by $k_{i,r}^{b}$ the garbled value that is assigned to the value $b \in \{0,1\}$ in wire i of circuit r. Then, the sets $W_{i,j}$ and $W'_{i,j}$ both contain $s+1$ commitments and are defined as follows. Let $b \in_R \{0,1\}$ be a random bit, chosen independently for every $\{W_{i,j}, W'_{i,j}\}$ pair. Define $W_{i,j}$ to contain a commitment to b, as well as commitments to the garbled value corresponding to b in wire i in all of the s circuits, and define $W'_{i,j}$ similarly, but with respect to $1-b$. In other words, $W_{i,j} = \{\mathsf{com}(b), \mathsf{com}(k_{i,1}^{b}), \ldots, \mathsf{com}(k_{i,s}^{b})\}$ and $W'_{i,j} = \{\mathsf{com}(1-b), \mathsf{com}(k_{i,1}^{1-b}, \ldots, \mathsf{com}(k_{i,s}^{1-b})\}$. The fact that b is chosen randomly means that with probability 1/2 the set $W_{i,j}$ contains the commitments to values corresponding to 0, and with probability 1/2 it contains the commitments to values corresponding to 1. We stress that in each of the pairs $(W_{i,1}, W'_{i,1}), \ldots, (W_{i,s}, W'_{i,s})$, the values that are committed to are the same. The only difference is that independent randomness is used in each pair for choosing b and constructing the commitments. We denote the first bit committed to in a commitment set as the indicator bit.



circuit 1
circuit 2
...
circuit s
W 1,1 = { com(0), com(k^0 1,1), com(k^0 1,2), ... com(k^0 1,s) }
W'1,1 = { com(1), com(k^1 1,1), com(k^1 1,2), ... com(k^1 1,s) }
W 1,2 = { com(1), com(k^1 1,1), com(k^1 1,2), ... com(k^1 1,s) }
W'1,2 = { com(0), com(k^0 1,1), com(k^0 1,2), ... com(k^0 1,s) }
W 1,s = { com(0), com(k^0 1,1), com(k^0 1,2), ... com(k^0 1,s) }
W'1,s = { com(1), com(k^1 1,1), com(k^1 1,2), ... com(k^1 1,s) }
Notation:
W_{i,j} : i is wire index, j is set index.
K^b_{i,r} : b is wire value, i is wire index, r is circuit index.

Figure 2: The commitment sets corresponding to P_1's **first** input wire.

After constructing these circuits and commitment sets, party P_1 sends to P_2 all of the s garbled circuits (i.e., the garbled gate-tables and output-tables, but *not* the garbled values corresponding to the input wires), and all the commitment sets. Note that if P_1's input is of length n, then there are sn pairs of commitment *sets*; and a total of $sn(2s+2) = O(s^2 n)$ commitments.

Stage 2 – Challenge: Two random strings $\rho, \rho' \in_R \{0,1\}^s$ are chosen and sent to P_1 (in the actual protocol, these strings are determined via a simple coin-tossing protocol). The string ρ is a challenge indicating which garbled circuits to open, and the string ρ' is a challenge indicating which commitment sets to open. We call the opened circuits check-circuits and the unopened ones evaluation-circuits. Likewise, we call the opened sets check-sets and the unopened ones evaluation-sets. A circuit (resp., commitment set) is defined to be a check-circuit (resp., check-set) if the corresponding bit in ρ (resp., ρ') equals 1; otherwise, it is defined to be an evaluation-circuit (resp., evaluation-set).

STAGE 3 – OPENING: First, party P_1 opens all the commitments corresponding to P_2's input wires in all of the check-circuits. Second, in all of the *check-sets* P_1 opens the commitments that correspond to *check-circuits*. That is, if circuit r is a check circuit, then P_1 decommits to all of the values $\mathsf{com}(k^0_{i,r}), \mathsf{com}(k^1_{i,r})$ in check-sets, where i is any of P_1's input bits. Finally, for every check-set, P_1 opens the commitment to the indicator bit, the initial value in each of the sets $W_{i,j}, W'_{i,j}$. See Figure 3 for a diagram in which the values which are opened are highlighted (the diagram refers to only one of P_1's input wires in the circuit).

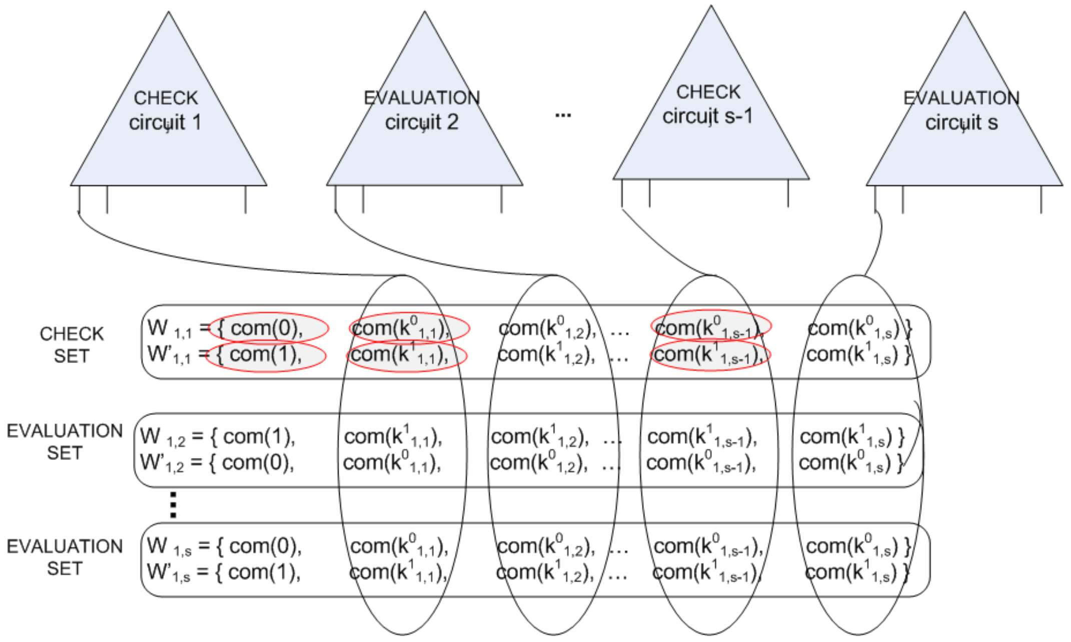

CHECK circuit 1
EVALUATION circuit 2
...
CHECK circuit s-1
EVALUATION circuit s
CHECK SET
W 1,1 = { com(0), com(k0 1,1), com(k0 1,2), … com(k0 1,s-1), com(k0 1,s) }
W'1,1 = { com(1), com(k1 1,1), com(k1 1,2), … com(k1 1,s-1), com(k1 1,s) }
EVALUATION SET
W 1,2 = { com(1), com(k1 1,1), com(k1 1,2), … com(k1 1,s-1), com(k1 1,s) }
W'1,2 = { com(0), com(k0 1,1), com(k0 1,2), … com(k0 1,s-1), com(k0 1,s) }
EVALUATION SET
W 1,s = { com(0), com(k0 1,1), com(k0 1,2), … com(k0 1,s-1), com(k0 1,s) }
W'1,s = { com(1), com(k1 1,1), com(k1 1,2), … com(k1 1,s-1), com(k1 1,s) }

Figure 3: In every check-set, the commitment to the indicator bit, and the commitments corresponding to check-circuits are all opened.

STAGE 4 – VERIFICATION: In this step, party P_2 verifies that all of the check-circuits were correctly constructed. In addition, it verifies that, with regards to P_1's inputs, all the opened commitments in sets whose first item is a commitment to 0 are to garbled encodings of 0; likewise for 1. These checks are carried out as follows. First, in all of the check-circuits, P_2 receives the decommitments to the garbled values corresponding to its own input, and by the order of the commitments P_2 knows which value corresponds to 0 and which value corresponds to 1. Second, for every check-circuit, P_2 receives decommitments to the garbled input values of P_1 in all the check-sets, along with a bit indicating whether these garbled values correspond to 0 or to 1. It first checks that for every wire, the garbled values of 0 (resp., of 1) are all equal. Then, the above decommitments enable the complete opening of the garbled circuits (i.e., the decryption of all of the garbled tables). Once this has been carried out, it is possible to simply check that the check-circuits are all correctly constructed. Namely, that they agree with a specific and agreed-upon circuit computing f.

Stage 5 – Evaluation and Verification: Party P_1 reveals the garbled values corresponding to its input: If i is a wire that corresponds to a bit of P_1's input and r is an evaluation-circuit, then P_1 decommits to the commitments $k^b_{i,r}$ in all of the *evaluation-sets*, where b is the value of its input bit. This is depicted in Figure 4. Finally, P_2 verifies that (1) for every input wire, all the opened commitments that were opened in evaluation-sets contain the same garbled value, and (2) for every i, j P_1 opened commitments of evaluated circuits in exactly one of $W_{i,j}$ or $W'_{i,j}$. If these checks pass, it continues to evaluate the circuit.

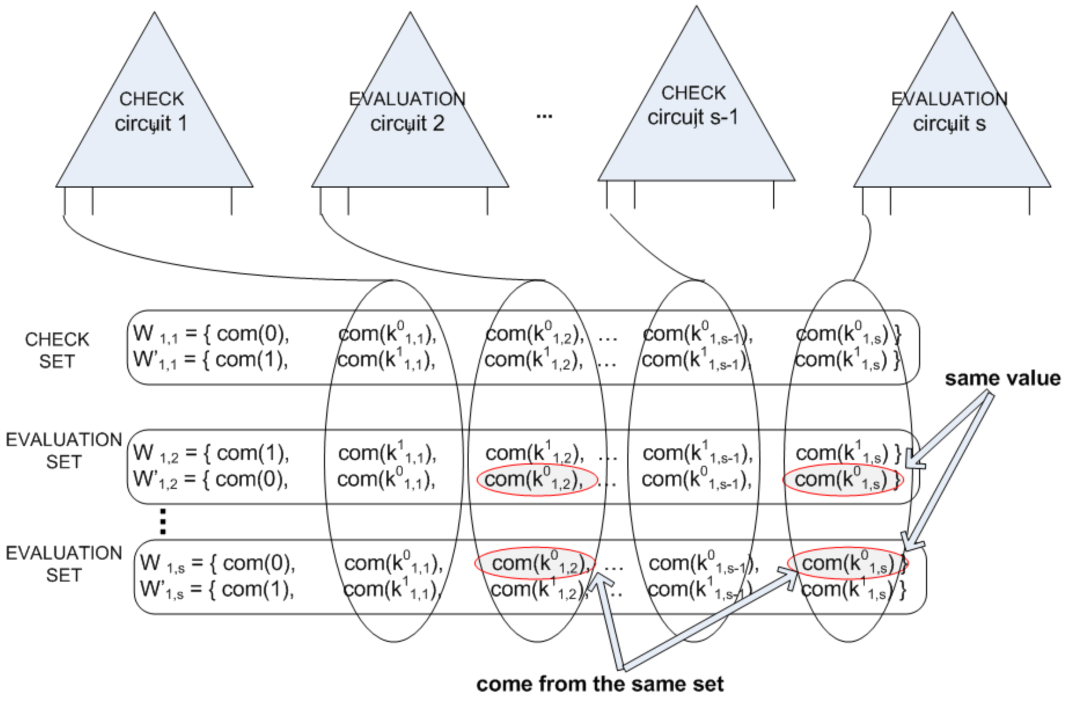

CHECK circuit 1
EVALUATION circuit 2
...
CHECK circuit s-1
EVALUATION circuit s
CHECK SET
W 1,1 = { com(0), com(k0 1,1), com(k0 1,2), … com(k0 1,s-1), com(k0 1,s) }
W'1,1 = { com(1), com(k1 1,1), com(k1 1,2), … com(k1 1,s-1), com(k1 1,s) }
EVALUATION SET
W 1,2 = { com(1), com(k1 1,1), com(k1 1,2), … com(k1 1,s-1), com(k1 1,s) }
W'1,2 = { com(0), com(k0 1,1), com(k0 1,2), … com(k0 1,s-1), com(k0 1,s) }
EVALUATION SET
W 1,s = { com(0), com(k0 1,1), com(k0 1,2), … com(k0 1,s-1), com(k0 1,s) }
W'1,s = { com(1), com(k1 1,1), com(k1 1,2), … com(k1 1,s-1), com(k1 1,s) }
same value
come from the same set

Figure 4: P_1 opens in the evaluation-sets, the commitments that correspond to its input. In every evaluation-set these commitments come from the same item in the pair.

Intuition. Having described the mechanism for checking consistency, we now provide some intuition as to why it is correct. A simple cut-and-choose check verifies that most of the evaluated circuits are correctly constructed. The main remaining issue is ensuring that P_1's inputs to most circuits are consistent. If P_1 wants to provide different inputs to a certain wire in two circuits, then *all* the $W_{i,j}$ (or $W'_{i,j}$) sets it opens in *evaluation-sets* must contain a commitment to 0 in the first circuit and a commitment to 1 in the other circuit. However, if any of these sets is chosen to be checked, and the circuits are among the checked circuits, then P_2 aborts. This means that if P_1 attempts to provide different inputs to two circuits and they are checked, it is almost surely caught. Now, since P_2 outputs the majority output of the evaluated circuits, the result is affected by P_1 providing different inputs only if these inputs affect a constant fraction of the circuits. But since all of these circuits must not be checked, P_1's probability of success is exponentially small in s.

3.3 The Full Protocol

We now describe the full protocol in detail. We use the notation com_b to refer to a perfectly binding commitment scheme, and com_h to refer to a perfectly hiding commitment scheme (See [9] for definitions).

Protocol 2 (protocol for computing $f(x, y)$):

- **Input:** *P_1 has input $x \in \{0,1\}^n$ and P_2 has input $y \in \{0,1\}^n$.*
- **Auxiliary input:** *a statistical security parameter s and the description of a circuit C^0 such that $C^0(x, y) = f(x, y)$.*
- **Specified output:** *party P_2 should receive $f(x, y)$ and party P_1 should receive no output. (Recall that this suffices for the general case where both parties receive possibly different outputs; see Section 2.2.)*
- **The protocol:**

  0. Circuit construction: *The parties replace C^0 with a circuit C which is constructed by replacing each input wire of P_2 by the result of an exclusive-or of s new input wires of P_2, as depicted in Figure 1. (We show in Section 5.2 how the number of new input bits can be reduced.) The number of input wires of P_2 is increased from $|y| = n$ to sn. Let the bit-wise representation of P_2's original input be $y = y_1 \ldots y_n$. Denote its new input as $\hat{y} = \hat{y}_1, \ldots, \hat{y}_{ns}$. P_2 chooses its new input at random subject to the constraint $y_i = \hat{y}_{(i-1)\cdot s+1} \oplus \cdots \oplus \hat{y}_{i \cdot s}$.*

  1. Commitment construction: P_1 constructs the circuits and commits to them, as follows:

  (a) P_1 constructs s independent copies of a garbled circuit of C, denoted $GC_1, \ldots, GC_s$.

  (b) P_1 commits to the garbled values of the wires corresponding to P_2's input to each circuit. That is, for every input wire i corresponding to an input bit of P_2, and for every circuit GC_r, P_1 computes the ordered pair $(\mathsf{com}_b(k_{i,r}^0), \mathsf{com}_b(k_{i,r}^1))$, where $k_{i,r}^b$ is the garbled value associated with b on input wire i in circuit GC_r.

  (c) P_1 computes commitment-sets for the garbled values that correspond to its own inputs to the circuits. That is, for every wire i that corresponds to an input bit of P_1, it generates s pairs of commitment sets $\{W_{i,j}, W'_{i,j}\}_{j=1}^s$, in the following way:
  Denote by $k_{i,r}^b$ the garbled value that was assigned by P_1 to the value $b \in \{0,1\}$ of wire i in GC_r. Then, P_1 chooses $b \in_R \{0,1\}$ and computes

$$\begin{aligned} W_{i,j} &= \langle \mathsf{com}_b(b), \mathsf{com}_b(k_{i,1}^b), \ldots, com_b(k_{i,s}^b) \rangle, \qquad \text{and} \\ W'_{i,j} &= \langle \mathsf{com}_b(1-b), \mathsf{com}_b(k_{i,1}^{1-b}), \ldots, \mathsf{com}_b(k_{i,s}^{1-b}) \rangle. \end{aligned}$$

  For each i, j, the sets are constructed using independent randomness, and in particular the value of b is chosen independently for every $j = 1 \ldots s$. There is a total of ns commitment-sets. We divide them into s supersets, *where superset S_j is defined to be the set containing the j^{th} commitment set for all wires. Namely, it is defined as $S_j = \{(W_{1,j}, W'_{1,j}), \ldots, (W_{n,j}, W'_{n,j})\}$.*

  2. Oblivious transfers: *For every input bit of P_2, parties P_1 and P_2 run a 1-out-of-2 oblivious transfer protocol in which P_2 receives the garbled values for the wires that correspond to its input bit (in every circuit). That is, let $c_{i,r}^b$ denote the commitment to the garbled value $k_{i,r}^b$ and let $dc_{i,r}^b$ denote the decommitment value for $c_{i,r}^b$. Furthermore, let $i_1, \ldots, i_{ns}$ be the input wires that correspond to P_2's input.*

Then, for every $j = 1, \ldots, ns$, parties P_1 and P_2 run a 1-out-of-2 OT protocol in which:

(a) P_1's input is the pair of vectors $([dc^0_{i_j,1}, \ldots, dc^0_{i_j,s}], [dc^1_{i_j,1}, \ldots, dc^1_{i_j,s}])$.

(b) P_2's input is its j^{th} input bit $\hat{y}_j$ (and its output should thus be $[dc^{\hat{y}_j}_{i_j,1}, \ldots, dc^{\hat{y}_j}_{i_j,s}]$).

If the oblivious transfer protocol provides security for parallel execution, then these executions are run in parallel. Otherwise, they are run sequentially.

3. Send circuits and commitments: *P_1 sends to P_2 the garbled circuits (i.e., the gate and output tables), as well as all of the commitments that it prepared above.*

4. Prepare challenge strings:

   (a) P_2 chooses a random string $\rho_2 \in_R \{0,1\}^s$ and sends $\mathsf{com}_h(\rho_2)$ to P_1.

   (b) P_1 chooses a random string $\rho_1 \in \{0,1\}^s$ and sends $\mathsf{com}_b(\rho_1)$ to P_2.

   (c) P_2 decommits, revealing ρ_2.

   (d) P_1 decommits, revealing ρ_1.

   (e) P_1 and P_2 set $\rho = \rho_1 \oplus \rho_2$.

   The above steps are run a second time, defining an additional string ρ'.[4]

5. Decommitment phase for check-circuits: *From here on, we refer to the circuits for which the corresponding bit in ρ is 1 as* check-circuits, *and we refer to the other circuits as* evaluation-circuits. *Likewise, if the j^{th} bit of ρ' equals 1, then all commitments sets in superset $S_j = \{(W_{i,j}, W'_{i,j})\}_{i=1\ldots n}$ are referred to as* check-sets*; otherwise, they are referred to as* evaluation-sets.

   For every check-circuit *GC_r, party P_1 operates in the following way:*

   (a) For every input wire i corresponding to an input bit of P_2, party P_1 decommits to the pair $(\mathsf{com}(k^0_{i,r}), \mathsf{com}(k^1_{i,r}))$ (namely to both of P_2's inputs).

   (b) For every input wire i corresponding to an input bit of P_1, party P_1 decommits to the appropriate values in the check-sets *$\{W_{i,j}, W'_{i,j}\}$. Specifically, P_1 decommits to the $\mathsf{com}(k^0_{i,r})$ and $\mathsf{com}(k^1_{i,r})$ values in $(W_{i,j}, W'_{i,j})$, for every check-set S_j (see Figure 3). In addition, P_1 decommits to the indicator bits of these sets (i.e., to the first committed value in each set) .*

   For every pair of check-sets *$(W_{i,j}, W'_{i,j})$, party P_1 decommits to the first value in each set (i.e., to the value that is supposed to be a commitment to the indicator bit, $\mathsf{com}(0)$ or $\mathsf{com}(1)$).*

6. Decommitment phase for P_1's input in evaluation-circuits: *P_1 decommits to the garbled values that correspond to its inputs in evaluation-circuits. Let i be the index of an input wire that corresponds to P_1's input (the following procedure is applied to all such wires). Let b be the binary value that P_1 assigns to input wire i. In every evaluation-set $(W_{i,j}, W'_{i,j})$, P_1 chooses the set (out of $(W_{i,j}, W'_{,j})$), which corresponds to the value b. It then opens in this set the commitments that correspond to evaluation-circuits. Namely, to the values $k^b_{i,r}$, where r is an index of an evaluation circuit (see Figure 4).*

7. Correctness and consistency checks: *P_2 performs the following checks; if any of them fails it aborts.*

[4]Recall that ρ and ρ' are used to ensure that P_1 constructs the circuits correctly and uses consistent input in each circuit. Thus, it may seem strange that they are generated via a coin-tossing protocol, and not just chosen singlehandedly by P_2. Indeed, in order to prove the security of the protocol when P_1 is corrupted, there is no need for a coin-tossing protocol here. However, having P_2 choose ρ and ρ' singlehandedly creates a problem for the simulation in the case that P_2 is corrupted. We therefore use a coin-tossing protocol instead.

(a) Checking correctness of the check-circuits: *P_2 verifies that each check-circuit GC_i is a garbled version of C. This check is carried out by P_2 first constructing the input tables that associate every garbled value of an input wire to a binary value. The input tables for P_2's inputs are constructed by checking that the decommitments in Step 5a to the pairs $(\mathsf{com}(k_{i,r}^0), \mathsf{com}(k_{i,r}^1))$ (where i is a wire index and r is a circuit index) are valid, and then interpreting the first value to be associated with 0 and the second value to be associated with 1.*

Next, P_2 checks the decommitments to P_1's inputs. This check involves first checking that the decommitment values of Step 5b are valid. Then, P_2 verifies that in each pair of check-sets, one of $(W_{i,j}, W'_{i,j})$ begins with a commitment to 0 (henceforth the 0-tuple), and the other begins with a commitment to 1 (henceforth the 1-tuple). Then P_2 checks that for every wire, the values that are decommitted to in the 0-tuples in all check-sets are all equal, and that a similar property holds for the 1-tuples. P_2 then assigns the logical value of 0 to all of the opened commitments in the 0-tuples, and the logical value of 1 to the opened commitments in the 1-tuples.

Finally, given all the garbled values to the input wires and their associated binary values, P_2 decrypts the circuit and compares it to the circuit C.

(b) Verifying P_2's input in the check-circuits: *P_2 verifies that P_1's decommitments to the wires corresponding to P_2's input values in the check-circuits are correct, and agree with the logical values of these wires (the indicator bits). P_2 also checks that the inputs it learned in the oblivious transfer stage for the check-circuits correspond to its actual input. Specifically, it checks that the decommitment values that it received in the oblivious transfer stage open the committed values that correspond to the garbled values of its logical input (namely, that it received the first value in the pair if the input bit is 0 and the second value if it is 1).*[5]

(c) Checking P_1's input to evaluation-circuits: *Finally, P_2 verifies that for every input wire i of P_1 the following two properties hold:*

i. In every evaluation-set, P_1 chose one of the two sets and decommitted to all the commitments in it which correspond to evaluation-circuits.

ii. For every evaluation-circuit, all of the commitments that P_1 opened in evaluation-sets commit to the same garbled value.

8. Circuit evaluation: *If any of the above checks fails, P_2 aborts and outputs $\perp$. Otherwise, P_2 evaluates the evaluation circuits (in the same way as for the semi-honest protocol of Yao). It might be that in certain circuits the garbled values provided for P_1's inputs, or the garbled values learned by P_2 in the OT stage, do not match the tables and so decryption of the circuit fails. In this case P_2 also aborts and outputs $\perp$. Otherwise, P_2 takes the output that appears in most circuits, and outputs it (the proof shows that this value is well defined).*

4 Proof of Security

The security of Protocol 2 is stated in the following theorem.

Theorem 3 *Let $f : \{0,1\}^* \times \{0,1\}^* \to \{0,1\}^*$ be any probabilistic polynomial-time two-party functionality and consider the instantiation of Protocol 2 for functionality f. Assume that the*

[5]This check is crucial and thus the order of first running the oblivious transfer and then sending the circuits and commitments is not at all arbitrary.

oblivious transfer protocol is secure, that com_b *is a perfectly-binding commitment scheme, that* com_h *is a perfectly-hiding commitment scheme, and that the garbled circuits are constructed as in* [22]*. Then, Protocol 2 securely computes* f*.*

The theorem is proved in two stages: first for the case that P_1 is corrupted and next for the case that P_2 is corrupted.

4.1 Security against a Malicious P_1

Intuition. The proof constructs an ideal-model adversary/simulator which has access to P_1 and to the trusted party, and can simulate the view of an actual run of the protocol. It uses the fact that the strings ρ, ρ', which choose the circuits and commitment sets that are checked, are uniformly distributed even if P_1 is malicious. The simulator runs the protocol until P_1 opens the commitments of the checked circuits and checked commitment sets, and then rewinds the execution and runs it again with new random ρ, ρ' values. We expect that about one quarter of the circuits will be checked in the first execution *and* evaluated in the second execution. For these circuits, in the first execution the simulator learns the translation between the garbled values of P_1's input wires and the actual values of these wires, and in the second execution it learns the garbled values that are associated with P_1's input (this association is learned from the garbled values that P_1 sends to P_2). Combining the two, it learns P_1's input x, which can then be sent to the trusted party. The trusted party answers with $f(x, y)$, which we use to define P_2's output and complete the simulation.

When examining the detailed proof, first note that the strings ρ, ρ' computed in Step 4 of Protocol 2, are uniformly distributed even in the presence of a malicious P_1. This is due to the perfect hiding of P_2's commitments in the coin-tossing subprotocol. We say that a circuit GC_i and a superset S_j *agree* if the checks in Step 7 of the protocol succeed when considering only the check-circuit GC_i and the superset of check sets S_j. In particular, this means that GC_i computes the required function when the garbled values of P_1's input wires are taken from S_j, and that these garbled values agree with the indicator bit of the sets in S_j. This also means that the committed values of the garbled values of P_2's input wires in GC_i are correctly constructed. (Some circuits might not agree with any set S_j, e.g., if they do not compute f. Other circuits might agree with some supersets and not agree with others.)

We begin by proving two lemmas that will be used in analyzing the simulation (described below). We say that a circuit is *ε-bad* if more than εs of the supersets disagree with it. The following lemma shows that P_2 aborts (with high probability) if more than εs of the circuits are *ε-bad*. We can therefore concentrate on the case that εs or less of the circuits are bad.

Lemma 4 *If at least* εs *of the circuits are* ε-bad*, then* P_2 *aborts with probability of at least* $1 - 2 \cdot 2^{-\varepsilon s}$*.*

Proof: As a warmup, suppose that there is a single *ε-bad* circuit. Then the probability of P_2 not aborting is at most $1/2 + 1/2 \cdot 2^{-\varepsilon s}$; i.e., the probability that the bad circuit is not chosen as a check-circuit plus the probability that it is a check circuit but all check-sets agree with it (since the circuits are chosen independently, we can just multiply the probabilities in the latter case). Suppose now that there are j different *ε-bad* circuits. Then the probability of P_2 not aborting is at most $2^{-j} + (1 - 2^{-j})2^{-\varepsilon s} \leq 2^{-j} + 2^{-\varepsilon s}$. Setting $j = \varepsilon s$ yields the lemma. ∎

The following lemma shows that P_2 aborting does not reveal information to P_1 about P_2's input.

Lemma 5 *For any two different inputs* y *and* y' *of* P_2 *for the function* f*, the difference between the probability that* P_2 *aborts Protocol 2 when its input is* y *and when its input is* y' *is at most* $n2^{-s+1}$*.*

Proof: P_2 may abort Protocol 2 in Step 7(a) while checking the correctness of the check circuits and the check sets. In this case, the decision to abort is based on P_1's construction of the sets and circuits, and on the random inputs of the parties, and is independent of P_2's input. The same is true of Step 7(c) where P_2 checks P_1's input to the evaluation circuits. In Step 7(b), however, P_2 aborts based on whether the values it learned in the oblivious transfer invocations open P_1's commitments to the garbled values of P_2's input. This case must be examined in detail.

Consider a specific input bit of P_2. In Step 0 of Protocol 2 the circuit is changed so that this bit is computed as the exclusive-or of s new input bits of P_2. Consider the s new inputs which replace a single input wire of the original circuit. Suppose that P_1 provides in the OT protocol corrupt values to both garbled values of one of P_2's (new) input wires. Then P_2 aborts with probability 1 regardless of its input. If P_1 provides a corrupt OT value to exactly one of the two possible OT choices, of $1 \leq j < s$ new wires, then P_2 aborts with probability $1 - 2^{-j}$, again regardless of the actual value of its original input. This holds because the values assigned by P_2 to any *proper subset* of the s bits are independent of P_2's actual input. Assume now that P_1 corrupts one OT value for each of the s new wires (say all '1' values). Then P_2 aborts with probability 1 if its original input had one value ('1' in this example), and aborts with probability $1 - 2^{-s+1}$ if its original input had the other value (in this example, P_2 does not abort if its input is '0' and it chose only '0' inputs in the s OT invocations). Therefore, for any two different inputs y and y' of P_2 of length n-bits each, the probability that P_2 aborts the protocol differs by at most $n2^{-s+1}$, as required. ■

We are now ready to prove the security of the protocol under simulation-based definitions.

Lemma 6 *Assume that the oblivious transfer protocol is secure, that the commitment scheme* com_h *is perfectly-hiding, and that the commitment scheme* com_b *is perfectly-binding. Then, Protocol 2 is secure in the case that* P_1 *is corrupted. (We say that a protocol is secure in the case that* P_1 *is corrupted if Definition 1 holds when* $\overline{A} = (A_1, A_2)$ *is such that* A_2 *is honest.)*

Proof: Let A_1 be an adversary corrupting P_1; we construct an ideal-model adversary/simulator B_1. Since we assume that the oblivious transfer protocol is secure, we analyze the security of Protocol 2 in the hybrid model with a trusted party computing the oblivious transfer functionality.

The simulator. The simulator B_1 chooses a random input y' for P_2 and uses it in all but the last stage of the simulation. B_1 receives all of the garbled circuits and commitments from A_1. Simulator B_1 then runs the coin-tossing phase (for preparing the challenge strings) as P_2 would, and receives all of the required decommitments from A_1, including the garbled values that supposedly correspond to its input. B_1 runs all of the checks that P_2 would run. If any of the checks fail, B_1 sends an abort message to A_1, sends $\perp$ to the trusted party and halts, outputting whatever A_1 outputs. Otherwise, B_1 rewinds A_1 and returns to the coin-tossing phase. Once again B_1 runs this phase as P_2 would (but with new randomness) and runs all of the checks that P_2 would run. B_1 continues this until all of the checks pass for a second time. Let α be the output of A_1 in this second successful execution (note that an honest P_1 has no output).

Denote by ρ, ρ' the (uniformly distributed) challenge strings from the first execution of B_1 with A_1, and denote by $\hat{\rho}, \hat{\rho}'$ the challenge strings from the second execution. Furthermore, denote $\rho = \rho^1 \cdots \rho^s$ and $\hat{\rho} = \hat{\rho}^1 \cdots \hat{\rho}^s$. Now, if there are less than $s/8$ indices i for which $\rho_i = 1$ and $\hat{\rho}_i = 0$, then B_1 outputs fail_1. Otherwise, let I be a subset of indices of size exactly $s/8$ for which $\rho_i = 1$ and $\hat{\rho}^i = 0$ (it is easier to work with a fixed number of i's that have this property, so we choose them here). Then, for every $i \in I$, we have that in the first execution GC_i is a check circuit and in the second execution it is an evaluation circuit. Thus, B_1 obtains all of the decommitments of GC_i

in the first execution (including the association of the garbled values corresponding to P_1's input – i.e., the decommitment to $\mathsf{com}(b)$ in the commitment-sets), and in the second execution it obtains the garbled values corresponding to P_1's input that P_1 sends to P_2. For each such i, it is possible to define P_1's input in circuit GC_i by associating the indicator bit obtained when GC_i was a check circuit with the garbled value sent by P_1 when GC_i was an evaluation circuit. Thus, B_1 obtains $s/8$ possible n-bit input vectors for P_1. If no input value appears more than $s/16$ times, then B_1 outputs fail_2. Otherwise, B_1 sets x to be the value that appears more than $s/16$ times and sends it to the trusted party. B_1 then outputs α (the output of A_1 in the second execution) and halts.

Analysis. We claim that the view of A_1 in the simulation with B_1 is *statistically close* to its view in a hybrid execution of Protocol 2 with a trusted party computing the oblivious transfer protocol. We first claim that the difference between the probability that P_2 receives "abort" (i.e., $\perp$) in an ideal execution with B_1 and the probability that P_2 outputs "abort" (i.e. $\perp$) in a real execution with A_1 is at most negligible. Observe that in the simulation, B_1 uses a random input for its emulation of P_2 instead of the real y that P_2 holds. This makes a difference when B_1 checks the decommitments for the wires that are associated with P_2's input. (Notice that in the real protocol P_2 also uses its input in oblivious transfer subprotocols. Nevertheless, in the hybrid model that we are analyzing here, A_1 learns nothing about P_2's input in the oblivious transfer because it is ideal.) Nevertheless, by Lemma 5 we know that the probability of abort is at most negligibly different between the case that P_2 has a random input and the case that it has a specific input y. From here on, we therefore consider the case that P_2 does not abort the protocol (i.e., does not output $\perp$). We now prove that B_1 outputs fail_1 or fail_2 with at most negligible probability. The proof that fail_1 occurs with negligible probability follows from the Chernoff bound, as follows. Denote an index i as good if $\rho_i = 1$ and $\hat{\rho}_i = 0$. The probability of this event is 1/4, independently of other indices. Event fail_1 happens if less than $s/8$ of the indices are good. Let $X_i = 1$ if and only if index i is good. Then, $\Pr[X_i = 1] = 1/4$ and the Chernoff bound implies that

$$\Pr\left[\sum_{i=1}^{s} X_i < \frac{s}{8}\right] = \Pr\left[\frac{\sum_{i=1}^{s} X_i}{s} < \frac{1}{8}\right] \le \Pr\left[\left|\frac{\sum_{i=1}^{s} X_i}{s} - \frac{1}{4}\right| > \frac{1}{8}\right] < 2 \cdot e^{-\frac{(1/8)^2}{2\cdot(1/4)\cdot(3/4)}\cdot s} = 2 \cdot e^{\frac{-s}{24}} < 2 \cdot 2^{\frac{-s}{17}}$$

Bounding fail_2. We now show that the event fail_2 occurs with negligible probability. Let $\varepsilon = 1/16$ and denote by B the event that at least εs of the circuits are ε-*bad* (i.e., the event that $s/16$ of the circuits are 1/16-*bad*). Denote by A the event that B_1 sends $\perp$ to the trusted party. Lemma 4 shows that $\Pr[\bar{A}|B] \le 2 \cdot 2^{-s/16}$.

We begin by analyzing the probability that fail_2 occurs given $\bar{B}$; i.e., given the event that less than $s/16$ of the circuits are 1/16-*bad*. Consider the set of $s/8$ circuits GC_i with $i \in I$. The definition of $\bar{B}$ implies that a majority of the $s/8$ circuits in I are *not* 1/16-bad. The circuits which are not 1/16-bad agree with at least $15s/16$ of the commitment sets. The probability that any of these circuits does not agree with a majority of the evaluation sets is negligible: this event only happens if the number of evaluation sets is less than $s/8$, and the probability of this event happening can be bounded (using the Chernoff bound) by $(s/8) \cdot 2 \cdot e^{-\frac{(3/8)^2}{2\cdot(1/2)^2}\cdot s} = (s/8) \cdot e^{-\frac{9s}{32}} < 2^{\frac{-s}{2.5}}$. If a circuit agrees with a majority of the evaluation sets then the committed values of these sets open the circuit correctly. In the evaluation step, for each of its input wires P_1 opens the values for all evaluation circuits, taken from the same commitment set. P_2 and B_1 check that the values opened for a wire in all sets are equal. For the good circuits in I these values agree with the same logical value (the indicator bit of the set). Therefore in this case a majority of the circuits in I obtain the same logical input, and fail_2 does not occur.

When $\varepsilon = 1/16$, the previous argument shows that $\Pr[\mathsf{fail}_2|\bar{B}] < 2^{-s/2.5}$, and Lemma 4 shows that $\Pr[\bar{A}|B] < 2 \cdot 2^{-s/16}$. We are interested in $\Pr[\mathsf{fail}_2]$, which we bound as follows:

$$\Pr[\mathsf{fail}_2] = \Pr[\mathsf{fail}_2 \wedge A] + \Pr[\mathsf{fail}_2 \wedge \bar{A}] = \Pr[\mathsf{fail}_2 \wedge \bar{A}]$$

where the last equality is due to the fact that in the event of fail_2 the simulator B_1 does not send $\perp$ (and so $\bar{A}$ does not occur) and vice versa. Thus, $\Pr[\mathsf{fail}_2 \wedge A] = 0$. Now,

$$\Pr[\mathsf{fail}_2 \wedge \bar{A}] = \Pr[\mathsf{fail}_2 \wedge \bar{A} \wedge B] + \Pr[\mathsf{fail}_2 \wedge \bar{A} \wedge \bar{B}] \leq \Pr[\bar{A} \wedge B] + \Pr[\mathsf{fail}_2 \wedge \bar{B}]$$

Combining the above and using the fact that for all two events X and Y it holds that $\Pr[X \wedge Y] \leq \Pr[X|Y]$ we conclude that

$$\Pr[\mathsf{fail}_2] \leq \Pr[\bar{A}|B] + \Pr[\mathsf{fail}_2|\bar{B}] < 2 \cdot 2^{-s/16} + 2^{-s/2.5} < 3 \cdot 2^{-s/16}$$

Completing the proof. We now show that conditioned on B_1 not outputting any fail message, the view of A_1 in the simulation is *statistically close* to its view in an execution of Protocol 2. First note that the probability of abort in the real and ideal executions is at most negligibly far apart (this follows from Lemma 5 and the fact that B_1 uses a random input instead of the one that the honest P_2 has). Next, consider the case that abort does not occur. Recall that B_1 just runs the honest P_2's instructions. The only difference is that in the event that all of B_1's checks pass in the first execution (which is the event of no abort that we are considering here), it rewinds the second execution until this event occurs again. The final view of A_1 is then the view that appears in this second execution in which this occurs. Since B_1 uses independent random coins each time, and follows P_2's instructions each time, the above process results in a distribution that is identical to the view of A_1 in a real execution with P_2.

We now proceed to show that the *joint distribution* of B_1's output (which is just A_1's output α) and the honest B_2's output, is computationally indistinguishable from the joint distribution of A_1 and P_2's output in an execution of Protocol 2 (where an ideal oblivious transfer is used instead of the OT subprotocol). We will actually show statistical closeness. (This does not mean, however, that the overall protocol gives statistical security because our analysis is in the hybrid model for an oblivious transfer functionality and it depends on the security of the actual oblivious transfer subprotocol used.) In order to prove this, we show that if the real P_2 would have received the set of evaluation-circuits and decommitments that A_1 sent in the second execution with B_1, and it has input y, then it would compute $f(x, y)$ in a majority of the circuits (where x is the input value that B_1 sent to the trusted party computing f). This follows from the same argument that was used to show above that fail_2 occurs with negligible probability: with all but negligible probability, most of the evaluation circuits are not ε-bad and they each agree with a majority of the evaluation sets. Denote these circuits as *good* (or ε-good) circuits. In particular, the committed values provided in these sets for P_1's inputs in these circuits correctly decrypt the circuit according to their association with the indicator bit. P_2 also checks that each of P_1's input wires receives the same garbled value in all sets. Therefore, the evaluation step is aborted unless P_1 opens garbled values for the good circuits that agree with the same logical value (the indicator bit of the set). The fact that these circuits are good also implies that P_2 obtains garbled values in the OT stage that agree with its input. As a result, a majority of the evaluation circuits obtain the same logical input (x, y) and compute $f(x, y)$.

It remains to show that B_1 runs in expected polynomial-time. In order to see this, notice that aside from the rewinding, all of B_1's work takes a strict polynomial number of steps. Furthermore, each rewinding attempt also takes a strict polynomial number of steps. Now, denote by p the probability that A_1 responds correctly and so B_1's checks all pass. Then, the probability that B_1 enters the rewinding phase equals p. Furthermore, the expected number of rewinding attempts equals exactly $1/p$ (notice that B_1 runs exactly the same strategy in each rewinding attempt). Thus, the overall expected running-time of B_1 equals $\text{poly}(n, s) + p \cdot 1/p \cdot \text{poly}(n, s) = \text{poly}(n, s)$. This completes the proof of Lemma 6 and thus the case that P_1 is corrupted.

(We note one important subtlety in this part of the proof: the sequential composition theorem of [5] was only proven for the case that the security of the subprotocol is proven via a simulator that runs in strict polynomial-time (see [18] for a full discussion of this issue). Thus, [5] does not cover the case that the simulator for the oblivious transfer subprotocol runs in expected polynomial-time. Despite this, we claim that this is no problem in our specific case. In order to see that B_1 runs in expected polynomial-time even if the oblivious transfer protocol is proven secure using expected polynomial-time simulation, note that we can divide A_1 into two parts. The first part runs up until the end of the oblivious transfer protocol and outputs state information; the second part takes the state information and continues until the end of the execution. Now, the simulator for the oblivious transfer protocol may result in an expected polynomial-time adversary for the first part of A_1. However, the second part of A_1 still runs in strict polynomial-time, and B_1 only rewinds this second part. Therefore, the overall running-time of B_1 – even after replacing the ideal oblivious transfer functionality with a real protocol that may use expected polynomial-time simulation – is expected polynomial-time, as required.) ■

4.2 Security against a Malicious P_2

Intuition. Intuitively, the security in this case is derived from the fact that: **(a)** the oblivious transfer protocol is secure, and so P_2 only learns a single set of keys (corresponding to a single input y) for decrypting the garbled circuits, and **(b)** the commitment schemes are hiding and so P_2 does not know what input corresponds to the garbled values that P_1 sends it for evaluating the circuit. Of course, in order to formally prove security we construct an ideal-model simulator B_2 working with an adversary A_2 that has corrupted P_2. The simulator first extracts A_2's input bits from the oblivious transfer protocol, and then sends the input y it obtained to the trusted party and receives back $z = f(x, y)$. Given the output, the simulator constructs the garbled circuits. However, rather than constructing them all correctly, for each circuit it tosses a coin and, based on the result, either constructs the circuit correctly, or constructs it to compute the constant function outputting z (the output is received from the trusted party). In order to make sure that the simulator is not caught cheating, it biases the coin-tossing phase so that all of the correctly-constructed garbled circuits are check-circuits, and all of the other circuits are evaluation-circuits (this is why the protocol uses joint coin-tossing rather than let P_2 alone choose the circuits to be opened). A_2 then checks the correctly-constructed circuits, and is satisfied with the result as if it were interacting with a legitimate P_1. A_2 therefore continues the execution with the circuits which always output z. The proof is based on the following lemma:

Lemma 7 *Assume that the oblivious transfer protocol is secure, that* com_h *is a perfectly-hiding commitment scheme, and that* com *and* com_b *are perfectly-binding commitment schemes. Then, Protocol 2 is secure in the case that* P_2 *is corrupted.*

Proof: As have described above, the simulator works by constructing some of the circuits correctly and some of them incorrectly. Before proceeding with the formal proof of the lemma, we show that it is possible to construct such "false circuits", so that A_2 cannot distinguish between them and correctly constructed circuits.

Claim 8 *Given a circuit C and an output value z (of the same length as the output of C) it is possible to construct a garbled circuit $\widetilde{GC}$ such that:*

1. *The output of $\widetilde{GC}$ is always z, regardless of the garbled values that are provided for P_1 and P_2's input wires, and*
2. *If $z = f(x, y)$, then no non-uniform probabilistic polynomial-time adversary $\mathcal{A}$ can distinguish between the distribution ensemble consisting of $\widetilde{GC}$ and a single arbitrary garbled value for every input wire, and the distribution ensemble consisting of a real garbled version of C, together with garbled values that correspond to x for P_1's input wires, and to y for P_2's input wires.*

Proof Sketch: The proof of this lemma is taken from [22] (it is not stated in this way there, but is proven). We sketch the construction of $\widetilde{GC}$ here for the sake of completeness, and refer the reader to [22] for a full description and proof. The first step in the construction of the fake circuit $\widetilde{GC}$ is to choose two random keys k_i and k'_i for every wire w_i in the circuit C. Next, the gate tables of C are computed: let g be a gate with input wires w_i, w_j and output wire w_ℓ. The table of gate g contains encryptions of the single key k_ℓ that is associated with wire w_ℓ, under *all four combinations* of the keys k_i, k'_i, k_j, k'_j that are associated with the input wires w_i and w_j to g. (This is in contrast to a real construction of the garbled circuit that involves encrypting both k_ℓ and k'_ℓ, depending on the function that the gate in question computes.) That is, the following values are computed:

$$\begin{aligned} c_{0,0} &= E_{k_i}(E_{k_j}(k_\ell)) \\ c_{0,1} &= E_{k_i}(E_{k'_j}(k_\ell)) \\ c_{1,0} &= E_{k'_i}(E_{k_j}(k_\ell)) \\ c_{1,1} &= E_{k'_i}(E_{k'_j}(k_\ell)) \end{aligned}$$

The gate table for g is then just a random ordering of the above four values. This process is carried out for all of the gates of the circuit. It remains to describe how the output decryption tables are constructed. Denote the n-bit output z by $z_1 \cdots z_n$, and denote the circuit-output wires by $w_{m-n+1}, \ldots, w_m$. In addition, for every $i = 1, \ldots, n$, let k_{m-n+i} be the (single) key encrypted in the gate whose output wire is w_{m-n+i}, and let k'_{m-n+i} be the other key (as described above). Then, the output decryption table for wire w_{m-n+i} is given by: $[(0, k_{m-n+i}), (1, k'_{m-n+i})]$ if $z_i = 0$, and $[(0, k'_{m-n+i}), (1, k_{m-n+i})]$ if $z_i = 1$. This completes the description of the construction of the fake garbled circuit $\tilde{G}C$.

Notice that by the above construction of the circuit, the output keys (or garbled values) obtained by P_2 for *any* set of input keys (or garbled values), equals $k_{m-n+1}, \ldots, k_m$. Furthermore, by the above construction of the output tables, these keys $k_{m-n+1}, \ldots, k_m$ decrypt to $z = z_1 \cdots z_n = z$ exactly. Thus, property (1) of the lemma trivially holds. The proof of property (2) follows from a hybrid argument in which the gate construction is changed one at a time from the real construction to the above fake one (indistinguishability follows from the indistinguishability of encryptions). The construction and proof of this hybrid are described in full in [22]. ■

We are now ready to begin with the formal proof of Lemma 7. We denote the number of input wires of P_2 as n' (P_2 had originally n input wires, but in Step 0 of the protocol they are expanded to $n' = ns$ wires, to prevent an attack by P_1). Let A_2 be an adversary controlling P_2. We construct a simulator B_2 as follows:

1. B_2 chooses garbled values for the input wires of P_2 in s garbled circuits. That is, it chooses $n' \cdot s$ pairs of garbled values k_i^0 and k_i^1, and constructs $2n'$ vectors of garbled values of length

s. Denote the vectors $v_1^0, v_1^1, \ldots, v_{n'}^0, v_{n'}^1$, where v_i^b contains the garbled values in all circuits that are associated with the bit b for the input wire associated with P_2's i^{th} input bit. Next, B_2 computes the commitment and decommitment values for these vectors. That is, let c_i^b be a vector of commitments, with the j^{th} element being a commitment to the j^{th} element of v_i^b. Likewise, let dc_i^b be a vector of decommitments, where the j^{th} element of dc_i^b is the decommitment of the j^{th} element of c_i^b.

2. B_2 invokes A_2 upon its initial input and obtains the inputs that A_2 sends to the trusted party computing the oblivious transfer functionality (recall that our analysis is in the hybrid model). Let y_i denote the bit sent by A_2 that corresponds to the i^{th} oblivious transfer, and let $y = y_1, \ldots, y_n$ (note that y is not necessarily the same as A_2 and B_2's initial input). B_2 hands A_2 the vector of decommitments to garbled values $dc_i^{y_i}$ as if they are the output for A_2 from the trusted party in the i^{th} computation of the oblivious transfer functionality.

3. B_2 *externally* sends y to the trusted party computing f and receives back $z = f(x, y)$.

4. B_2 chooses a random string $\rho \in_R \{0,1\}^s$ and constructs s garbled circuits $GC_1, \ldots GC_s$, as follows. Let $\rho = \rho_1, \ldots, \rho_s$. Then, for $i = 1, \ldots, s$, if $\rho_i = 1$ (and so GC_i will be a check-circuit), simulator B_2 constructs circuit GC_i correctly (exactly as described in Step 1 of Protocol 2). Otherwise, if $\rho_i = 0$ (and so GC_i will be an evaluation circuit), it constructs circuit $GC_i = \widetilde{GC}$ as described in Claim 8. That is, it constructs a garbled circuit whose output is always z, regardless of the inputs used. The above constructions use the garbled values chosen for the input wires above. That is, the garbled values from v_i^0 and v_i^1 are used to define the input values for the i^{th} wire in all of the s circuits (the j^{th} value in v_i^b defines the value in the j^{th} circuit).

   B_2 constructs the commitments and commitment sets as follows.

   - First, for every r such that $\rho_r = 1$ (and so GC_r will be a check-circuit), the commitment pairs $(\mathsf{com}(k_{i,r}^0), \mathsf{com}(k_{i,r}^1))$ that correspond to P_2's input wires in circuit GC_r are computed correctly (note that $k_{i,r}^b$ is the r^{th} value in v_i^b and $\mathsf{com}(k_{i,r}^b)$ is taken from c_i^b).
   - In contrast, for every j for which $\rho_r = 0$ (and so GC_r will be an evaluation-circuit), these commitment pairs are computed as follows. Assume that P_2's i^{th} input bit is associated with wire i. Then, B_2 sets $k_{i,r}^{y_i}$ to equal the r^{th} garbled value in the vector $v_i^{y_i}$, and sets $k_{i,r}^{1-y_i}$ to be the string of all zeros. B_2 then defines the commitment pair to be $(\mathsf{com}(k_{i,r}^0), \mathsf{com}(k_{i,r}^1))$.
   - Second, B_2 chooses a random string $\rho' \in_R \{0,1\}^s$ and constructs the commitment-sets $W_{i,j}$ and $W'_{i,j}$ (of P_1's inputs), as follows. For every input wire i and for every j such that $\rho'_j = 0$ (i.e., such that the sets $W_{i,j}$ and $W'_{i,j}$ are evaluation-sets), B_2 generates the commitment-set $W_{i,j}$ so that the first commitment is $\mathsf{com}(0)$ and the rest are "correct" (i.e., as instructed in the protocol). It then computes $W'_{i,j}$ *incorrectly*, committing to the exact same values as $W_{i,j}$ (we stress that the commitments are computed using fresh randomness, but they are commitments to the same values).
   - Finally, B_2 constructs the commitment-sets for the values of j such that $\rho'_j = 1$ (i.e., such that $W_{i,j}$ and $W'_{i,j}$ are check-sets). Recall that the commitment-set $W_{i,j}$ is made up of an initial indicator commitment (to 0 or 1) followed by s commitments, where the r^{th} commitment corresponds to the r^{th} circuit; denote the r^{th} commitment in $W_{i,j}$ by $W_{i,j}^r$. Now, for every input wire i and every j such that $\rho'_j = 1$:

- For every r such that $\rho_r = 1$ (corresponding to a check-circuit), simulator B_2 places the correct commitments in $W_{i,j}^r$ and $W'^r_{i,j}$.
- For every r such that $\rho_r = 0$ (corresponding to an evaluation-circuit), simulator B_2 places commitments to zeros. (These commitments are never opened; see Figures 3 and 4.)

B_2 internally hands A_2 the garbled circuits and commitments that it constructed. (Note that the commitments corresponding to P_1 and P_2's input wires in all of the evaluation circuits contain only a single garbled value from the pair associated with the wire. This will be important later on.)

5. B_2 simulates the coin-tossing ("prepare challenge strings") phase with A_2 so that the outcome of $\rho_1 \oplus \rho_2$ equals the string ρ that it chose above. If it fails, it outputs fail and halts. Likewise, the coin-tossing phase for the second challenge string is also simulated so that the outcome is ρ' as chosen above. Again, if it fails, it outputs fail and halts. We describe how this is achieved below.
6. B_2 opens the commitments for check-circuits and check-sets for A_2, exactly as described in Step 5 of Protocol 2.
7. B_2 internally hands A_2 decommitments for the garbled values for each of the input wires corresponding to P_1's input, in each of the evaluation-circuits. In order to do this, B_2 just chooses randomly between $W_{i,j}$ and $W'_{i,j}$ for each evaluation-set, and decommits to the garbled values that are associated with the evaluation-circuits.
8. B_2 outputs whatever A_2 outputs and halts.

If at any time during the simulation, A_2 aborts (either explicitly or by sending an invalid message that would cause the honest A_1 to abort), B_2 halts immediately and outputs whatever A_2 does.

Analysis. We now show that the view of A_2 in the above simulation by B_2 is computationally indistinguishable from its view in a real execution with A_1. We note that since only A_2 receives output in this protocol, it suffices to consider the view of A_2 only. Before demonstrating this, we show that the coin-tossing phases can be simulated so that B_2 outputs fail with at most negligible probability. Intuitively, the simulation of this phase (for ρ) is carried out as follows:

1. B_2 receives a perfectly-hiding commitment c from A_2.
2. B_2 generates a perfectly-binding commitment $\hat{c}$ to a random string $\hat{\rho}$ and internally hands it to A_2.
3. If A_2 aborts without decommitting, then B_2 halts the simulation immediately and outputs whatever A_2 outputs. Otherwise, let ρ_2 be the value decommitted to by A_2.
4. B_2 rewinds A_2 to after the point that it sends c, and sends it a new commitment $\tilde{c}$ to the string $\rho_1 = \rho \oplus \rho_2$ (where the result of the coin-tossing is supposed to be the string ρ).
5. If A_2 decommits to ρ_2, then B_2 has succeeded. Thus, it continues by decommitting to ρ_1, and the result of the coin-tossing is $\rho = \rho_1 \oplus \rho_2$.

   If A_2 decommits to some $\rho_2' \neq \rho_2$, then B_2 outputs ambiguous.

If A_2 does not decommit (but rather aborts), then B_2 continues by sending a new commitment to $\rho_1 = \rho_2 \oplus \rho$. Notice that B_2 sends a commitment to the same value ρ_1, but uses fresh randomness in generating the commitments and executes Step 5 of the simulation again.

Unfortunately, as was shown by [11], the above simulation strategy does not necessarily run in expected polynomial-time. Rather, it is necessary to first estimate the probability that A_2 decommits when it receives a commitment $\hat{c}$ to a random value $\hat{\rho}$. Then, the number of rewinding attempts, when A_2 is given a commitment to $\rho_1 = \rho \oplus \rho_2$, is truncated as some function of the estimate. In [11], it is shown that this strategy yields an expected polynomial-time simulation that fails with only negligible probability (including the probability of outputting ambiguous). Furthermore, the simulation has the property that the view of A_2 is computationally indistinguishable from its view in a real execution. The analysis here is *exactly the same* as that of [11] and is therefore not repeated. (Note that in the zero-knowledge protocol of [11] the verifier first sends a perfectly-hiding commitment, the prover then sends perfectly-binding commitments, and finally the parties decommit. Thus, the flow and structure of our protocol is identical to theirs.) Of course, the same strategy exactly is used for the simulation of the coin-tossing phase for ρ'.

We now continue with the analysis of the simulation. Intuitively, given that the above coin-tossing simulation succeeds, it follows that all of the check-circuits are correctly constructed, as in the protocol (because B_2 constructs all the circuits for which $\rho_i = 1$ correctly). Thus, the view of A_2 with respect to these circuits is the same as in a real execution with an honest A_1. Furthermore, the commitments for the evaluation circuits reveal only a single garbled value for each input wire. Thus, Claim 8 can be applied.

Formally, we prove indistinguishability in the following way. First, we modify B_2 into B_2' who works in exactly the same way as B_2 except for how it generates the circuits. Specifically, B_2' is given the honest B_1's input value x and constructs *all* of the circuits correctly. However, it only uses the garbled values corresponding to x in the commitment-sets. That is, if the value $k_{i,\ell}$ is used in all of the commitment sets $W_{i,j}$ and $W'_{i,j}$ with respect to circuit ℓ, then $k_{i,\ell}$ is the garbled value associated with x_i (i.e., the i^{th} bit of x) in circuit ℓ (the other garbled value associated with the wire is associated with $1-x_i$). Everything else remains the same. In order to see that A_2's view in an execution with B_2 is indistinguishable from its view in an execution with B_2', we apply Claim 8. In order to apply this claim, recall first that the evaluation-circuits with B_2 are all constructed according to $\widetilde{GC}$, yielding output $z = f(x, y)$ where y is the input obtained from A_2 and x is the honest party's input. In contrast, the evaluation-circuits with B_2' are all correctly constructed. Note also that the input y obtained by B_2' from A_2 is the same value as that obtained by B_2, that defines $z = f(x, y)$.[6] Finally, note that B_2' sends A_2 the garbled values that correspond to B_1's input x. Thus, by Claim 8, A_2's view with B_2 is indistinguishable from its view with B_2'. (The full reduction here works by an adversary obtaining the garbled circuits and values, and then running the simulation of B_2 or B_2'. Specifically, it generates all the check-circuits correctly and uses the garbled values it obtained to generate the evaluation-sets and the commitments in the evaluation-sets. However, it does not generate the evaluation-circuits itself, but uses the ones that it receives. If it receives real circuits then it will obtain the distribution of B_2', and if it receives fake garbled circuits then it will obtain the distribution of B_2. We therefore conclude by Claim 8 that these distributions are indistinguishable. We note that the full proof of this also requires a hybrid

[6]Notice that there is one oblivious transfer for each input bit. Furthermore, the input of A_1 into these executions is a pair of vectors of s garbled values so that in the i^{th} execution, the first vector contains all of the decommitments for garbled values that correspond to 0 for the i^{th} input wire of P_2, and the second vector contains all of the decommitments for garbled values that correspond to 1 for the i^{th} input wire of P_2. This means that in every circuit, A_2 receives garbled values that correspond to the same input y.

argument over the many evaluation circuits, versus the single circuit referred to in Claim 8.)

Next, we construct a simulator B_2'' that works in the same way as B_2' except that it generates all of the commitments correctly (i.e., as in the protocol specification). Notice that this only affects commitments that are never opened. Note also that B_2'' is given x and so it can do this. The indistinguishability between B_2' and B_2'' follows from the hiding property of the commitment scheme com. (The full reduction is straightforward and is therefore omitted.)

Finally, notice that the distribution generated by B_2'' is the same as the one generated by an honest A_1, except for the simulation of the coin-tossing phases. Since, as we have mentioned, the view of A_2 in the simulation of the coin-tossing is indistinguishable from its view of a real execution, we conclude that the view of A_2 in the simulation by B_2'' is indistinguishable from its view in a real execution with A_1. Combining the above steps, we conclude that A_2's view in the simulation with B_2 is indistinguishable from its view in a real execution with A_1. This completes the proof of Lemma 7 and thus the case that P_2 is corrupted. ■

Combining the cases. The proof of Theorem 3 is completed by combining Lemma 6 and Lemma 7.

5 Efficiency of the Protocol

We discuss below the efficient implementation of the different building blocks of the protocol (namely, encryption, commitment schemes, and oblivious transfer). The overhead of the protocol depends on a statistical security parameter s. The security proof shows that the adversary's cheating probability is exponentially small in s. We note that in this paper we preferred to present a full and clear proof, rather than overly optimize the construction at the cost of complicating the proof.

The computation overhead is dominated by the oblivious transfers, as all other primitives are implemented using symmetric operations. In Protocol 2 each input bit of P_2 is replaced by s new input bits and therefore $O(ns)$ OTs are required. In Section 5.2 we show how to use only $O(\max(n, s))$ new input bits, and consequently the number of OTs is reduced to $O(\max(n, s))$ (namely $O(1)$ OTs per input bit, assuming $n = \Omega(s)$).

The communication overhead of the protocol is dominated by sending s copies of the garbled circuit, and $2s(s+1)$ commitments for each of the n inputs of P_1. In the protocol, the original circuit C^0 is modified by replacing each of the n original input bits of P_2 with the exclusive-or of s of the new input bits, and therefore the size of the evaluated circuit C is $|C| = |C^0| + O(ns)$ gates. The communication overhead is therefore $O(s|C| + s^2 n) = O(s(|C^0| + ns) + s^2 n) = O(s|C^0| + s^2 n)$ times the length of the secret-keys (and ciphertexts) used to construct the garbled circuit. (Note that the improved construction in Section 5.2 reduces the size of the new circuit to $|C| = |C^0| + O(\max(n, s))$ and therefore only improves the communication overhead by a constant; the significance of the improvement is with respect to computation.)

5.1 Efficient Implementation of the Different Primitives

In this section, we describe efficient implementations of the different building blocks of the protocol.

Encryption scheme. Following [22], the construction uses a symmetric key encryption scheme that has indistinguishable encryptions for multiple messages and an elusive efficiently verifiable range. Informally, this means **(1)** that for any two (known) messages x and y, no polynomial-time adversary can distinguish between the encryptions of x and y, and **(2)** that there is a negligible probability that an encryption under one key falls into the range of encryptions under another key, and given a key k it is easy to verify whether a certain ciphertext is in the range of encryptions with k. See [22] for a detailed discussion of these properties, and for examples of easy implementations satisfying them. For example, the encryption scheme could be $E_k(m) = \langle r, f_k(r) \oplus m0^n \rangle$, where f_k is a pseudo-random function keyed by k whose output is $|m| + n$ bits long, and r is a randomly chosen value.

Commitment schemes. The protocol uses both unconditionally hiding and unconditionally binding commitments. Our goal should be, of course, to use the most efficient implementations of these primitives, and we therefore concentrate on schemes with $O(1)$ communication rounds (all commitment schemes we describe here have only two rounds). Efficient unconditionally hiding commitment schemes can be based on number theoretic assumptions, and use $O(1)$ exponentiations (see, e.g., [14, 28]). The most efficient implementation is probably the one due to Damgård, Pedersen, and Pfitzmann, which uses a collision-free hashing function and no other cryptographic primitive [7], see also [15]. Efficient unconditionally binding commitments can be constructed using the scheme of Naor [26], which has two rounds and is based on using a pseudo-random generator.

Oblivious transfer. The protocol needs to use an OT protocol which is secure according to the real/ideal model simulation definition. Candidate protocols can be the protocol of [8] compiled according to the GMW paradigm, or the two-round protocols of [27, 2, 17] with additional proofs of knowledge. Protocols of this latter type have been shown in [21].

5.2 Reducing the Number of Oblivious Transfers

Protocol 2 uses a construction which replaces each input bit of P_2 with s new input bits, providing P_2 with multiple options for encoding each of its inputs. This limits the information that P_1 can gain from corrupting OT inputs (and in particular, P_2 aborts with almost the same probability irrespective of its actual input). The construction increases the number of input wires of P_2 from n to ns. We describe here a probabilistic construction which reduces the number of input wires of P_2 to $\max(4n, 8s)$ (we also show how to use codes to construct an explicit construction with similar performance). The construction has a direct effect on the overhead of the protocol, since the number of OTs is equal to the number of input wires of P_2. The bottom line is that the number of OTs can be reduced to be in the order as the length of P_2's input and the security parameter.

We denote the original input bits as $w_1, \ldots, w_n$ and the new input bits as $w'_1, \ldots, w'_m$. Our goal is to minimize m. Each w_i is defined as the exclusive-or of a subset of the new input bits. We define the indicator vector z_i as an m-bit binary string whose jth bit is 1 iff w'_j is in the subset of new input bits whose exclusive-or is w_i. The construction described in Protocol 2 corresponds to indicator vectors $z_i = (\underbrace{0\ldots0}_{(i-1)s}\underbrace{1\ldots1}_{s}\underbrace{0\ldots0}_{(n-i)s})$. We describe here Protocol 3 which works with any set of indicator vectors z.

Protocol 3 (Protocol with a reduced number of input bits):

- **Input, auxiliary input and specified output:** *These are as in Protocol 2. In addition, there is a parameter m which defines the number of new input bits of P_2, and there are n linearly independent m-bit vectors $z_1, \ldots, z_m$, specifying the relations between the new and the original input bits of P_2.*

- **The protocol:**

  0. *Circuit construction: The parties replace C^0 with a circuit C in which P_2 has m input bits $w'_1, \ldots, w'_m$. The circuit is constructed by replacing each original input wire w_i of P_2 by the value $\oplus_{j=1\ldots m} z_{i,j} \cdot w'_j$, where $z_{i,j}$ is the jth bit in the vector z_i.*

     P_2 chooses its new input at random subject to the constraints $w_i = \oplus_{j=1\ldots m} z_{i,j} \cdot w'_j$.

  The rest of the protocol is as in Protocol 2.

P_2 chooses random values for the bits $w'_1, \ldots, w'_m$, subject to the constraint that the exclusive-or of any set of new bits corresponding to an original bit w_i is equal the original value of w_i. P_2 then runs an OT for each of its new input bits. If one of the answers it receives in these OTs is corrupt, it aborts the protocol. Our goal is to make sure that the decision to abort does not reveal information about P_2's original input (since this is the only place that it is used in the proof). It is clear that if teh Hamming weight of each z_i is at least 2, and P_1 corrupts the inputs of a single OT, then, since each input bit of P_2 is the exclusive-or of several new bits, the decision to abort does not reveal information about any specific input bit of P_2. This observation must be generalized for the case of P_1 corrupting more OT inputs, and hold with respect to any subset of P_2's inputs.

Warmup – reusing bits. In order to use less "new" than ns input bits, P_2 must reuse these bits. Assume that P_2 has two input wires w_1, w_2 and that we replace them with $s+1$ new wires, $w'_1, \ldots, w'_{s+1}$. The input values are defined as $w_1 = w'_1 \oplus \cdots \oplus w'_s$, and $w_2 = w'_2 \oplus \cdots \oplus w'_{s+1}$ (namely $z_1 = 11\cdots 10$ and $z_2 = 01\cdots 11$). In this case, as is shown in Section 3.2, any strategy used by a malicious P_1 to corrupt OT values gives it an advantage of at most 2^{-s+1} in identifying a *single* bit of P_2's original input (e.g., if P_1 corrupts the '1' inputs of $w'_1, \ldots, w'_s$, then if $w_1 = 1$ P_2 always aborts, whereas if $w_1 = 0$ there is a probability of 2^{-s+1} that P_2 does not abort). However, $w_1 \oplus w_2 = w'_1 \oplus w'_{s+1}$ (namely, $z_1 \oplus z_2 = 10\ldots01$) and therefore if P_1 corrupts the OT values of both w'_1 and w'_{s+1} it can obtain a non-negligible advantage in learning $w_1 \oplus w_2$. (For example, P_1 can corrupt the '1' inputs of w'_1 and w'_{s+1}. If P_2 does not abort P_1 can conclude that $w'_1 = w'_{s+1} = 0$ and therefore $w_1 \oplus w_2 = 0$.)

The attack presented above can be prevented if the exclusive-or of any subset of P_2's original bits contains at least s new input bits. Namely, if, in the general case, for every non-empty subset $L \subseteq \{1, \ldots, n\}$ it holds that the Hamming weight of $\oplus_{i \in L} z_i$ is at least s. The two lemmata stated below show that this requirement is sufficient to prove that, up to a negligible probability, P_2's decision to abort is independent of its input values.

Lemma 9 *Suppose that for all sets $L = \{i_1, \ldots, i_{|L|}\}$ (corresponding to a set $\{w_{i_1}, \ldots, w_{i_{|L|}}\}$ of original input wires) it holds that the Hamming weight of $z_{i_1} \oplus \cdots \oplus z_{i_{|L|}}$ is at least s. Fix the values of any subset of less than s new input wires arbitrarily, and choose the values of all other new input wires uniformly at random. Then for any set $L = \{i_1, \ldots, i_{|L|}\}$, it holds that the value of the vector $(w_{i_1}, \ldots, w_{i_{|L|}})$ is uniformly distributed.*

Proof: Denote the size of the set L as $\ell = |L|$. The lemma is proved by induction on ℓ. Any single original input bit is defined as the exclusive-or of s or more new inputs bits and is thus uniformly distributed even given the (fewer than s) bits corrupted by P_1. The claim therefore holds for $\ell = 1$.

Let us examine the induction step. For simplicity, let $L = \{w_1, w_2, \ldots, w_\ell\}$. Assume that the value of an arbitrary set of less than s new input bits are set. For a fixed vector $\bar{b} = b_1, \ldots, b_\ell$ denote by $P_{\bar{b}} = P_{b_1 b_2 \ldots b_\ell}$ the probability that $\forall\ 1 \le i \le \ell\ \ w_i = b_i$ (this probability is taken over the distribution of the new input bits whose values have not been set yet.) We need to show that for any

ℓ bit string $b_1 \dots b_\ell$ it holds that $P_{b_1 \dots b_\ell} = 2^{-\ell}$. Consider any string $\bar{b} = b_1 \dots b_\ell$, and assume that $P_{\bar{b}} = 2^{-\ell} + \varepsilon$ for some ε between $-1/2$ and $1/2$. Take any string $\bar{b}'$ whose Hamming difference from $\bar{b}$ is 1. Consider for example the string which differs from $\bar{b}$ in its last bit. Then $P_{\bar{b}} + P_{\bar{b}'} = P_{b_1 \dots b_{\ell-1}}$, and is equal, by the induction step, to $2^{-\ell+1}$. Therefore $P_{\bar{b}'} = 2^{-\ell} - \varepsilon$. By the same argument, this is also the probability of any other string whose Hamming difference from $\bar{b}$ is 1. Now consider any string $\bar{b}''$ whose Hamming difference from $\bar{b}$ is 2. There must be a string $\bar{b}'$ whose Hamming difference from both $\bar{b}$ and $\bar{b}''$ is 1. We know that $P_{\bar{b}'} = 2^{-\ell} - \varepsilon$, and therefore, by applying the above argument to the relation between $\bar{b}'$ and $\bar{b}''$, we get that $P_{\bar{b}''} = 2^{-\ell} + \varepsilon$. Continuing to apply this argument we get for any string $\bar{c}$, that $P_{\bar{c}} = 2^{-\ell} - \varepsilon$ if the Hamming difference between $\bar{c}$ and $\bar{b}$ is odd, and that $P_{\bar{c}} = 2^{-\ell} + \varepsilon$ if the Hamming difference is even. It remains to show that $\varepsilon = 0$.

Now, the value $w_1 \oplus w_2 \oplus \cdots \oplus w_\ell$ is equal to the exclusive-or of at least s new input wires, and is therefore uniformly distributed when the value of less than s of these wires is fixed. On the other hand, the probability that this value is equal to $b_1 \oplus \cdots \oplus b_\ell$ is $\sum_{d(\bar{b},\bar{c}) \text{ is even}} P_{\bar{c}} = 2^{\ell-1}(2^{-\ell} + \varepsilon)$ (where $d(\alpha, \beta)$ denotes the Hamming difference between strings α and β). The probability that the value of exclusive-or is equal to $1 \oplus b_1 \oplus \cdots \oplus b_\ell$ is $\sum_{d(\bar{b},\bar{c}) \text{ is odd}} P_{\bar{c}} = 2^{\ell-1}(2^{-\ell} - \varepsilon)$. Therefore it must hold that $\varepsilon = 0$, or otherwise we get a contradiction. ■

Recall that a corrupt OT value is one that P_2 receives in the circuit opening that is different to its corresponding value that it received in the oblivious transfers.

Lemma 10 *Suppose that for all sets $L = \{i_1, \dots, i_{|L|}\}$ the Hamming weight of $z_{i_1} \oplus \cdots \oplus z_{i_{|L|}}$ is at least s. Then, for any two different inputs y and y' of P_2 for the function f, the difference between the probability that P_2 aborts the protocol as a result of corrupt OT values when its input is y and when its input is y' is at most 2^{-s+1}.*

Proof: If P_1 corrupts both its inputs to an OT instance then P_2 aborts with probability 1, and the lemma holds. We therefore assume that P_1 corrupts at most one of its pair of inputs any OT instance.

Assume that P_1 corrupts $s-1$ or less OT instances. We know that P_2 chooses its new input values independently of P_1's actions. We can assume that P_2 first chooses its input values for the new wires (OTs) which P_1 corrupts (and it chooses these values uniformly at random, as was argued above). P_2 then chooses its inputs to the other new wires. The probability of P_2 aborting depends only on its first step, whereas Lemma 9 shows that this step does not fix any of P_2's original inputs. The probability of abort is therefore independent of P_2's original input.

Assume now that P_1 corrupts s or more OT instances, and consider the first $s-1$ of these OTs. Lemma 9 implies that P_2's input values in these OTs are independent of its original input (since the original input values are not defined after fixing these $s-1$ values). Therefore, the distribution of P_2's inputs in these OTs, even given P_2's original input, is uniformly random, and consequently P_2 aborts with probability of at least $1 - 2^{-(s-1)}$ regardless of its input. It therefore holds, for any two input values y and y', that $2^{-(s-1)}$ is an upper bound for the difference between the probability that P_2 aborts the protocol (as a result of corrupt OT values) when its input is y and when its input is y'. ■

Following is a corollary of Lemma 10:

Corollary 11 *Running Protocol 3 with vectors z_i such that for any set $L = \{i_1, \dots, i_{|L|}\}$ the Hamming weight of $z_{i_1} \oplus \cdots \oplus z_{i_{|L|}}$ is at least s, ensures that OT corruptions by P_1 reveal only negligible information about P_2's input.*

We describe below an efficient randomized construction which achieves the desired property of the vectors z_i. As was pointed to us by David Woodruff, an explicit construction can be achieved using any explicit linear code from $\{0,1\}^s$ to $\{0,1\}^{O(s)}$, for which any two codewords have a distance of at least $\Omega(s)$ (Justesen codes are an example of such a code).

The randomized construction. We define $4n$ new input bits for P_2. Assume, without loss of generality, that $n > 2s$. (Otherwise add dummy input bits. Therefore the exact number of new input bits is $\max(4n, 8s)$.) The mapping between the n old input bits and the $4n$ new input bits is chosen randomly in the following way: each original input bit w_i is defined to be equal to the exclusive-or of a uniformly chosen subset of the new input bits (in other words, z_i is a uniformly distributed string of $4n$ bits).

We examine the probability that there is a subset $L \subseteq \{0,1\}^n$ for which the Hamming weight of $\oplus_{i\in L} z_i$ is less than s: Consider any subset L, then $\oplus_{i\in L} z_i$ is a uniformly distributed string with $4n > 8s$ bits, with an expected Hamming weight of $2n$. Let X_j be a random variable which is set to 1 if the jth bit in this string is 1. Note that $s/4n < 1/8$ by our assumption that $n > 2s$. We therefore have:

$$\Pr\left[\sum_{j=1}^{4n} X_j < s\right] = \Pr\left[\frac{\sum X_j}{4n} < \frac{s}{4n}\right] < \Pr\left[\frac{\sum X_j}{4n} < \frac{1}{8}\right] \le \Pr\left[\left|\frac{\sum X_j}{4n} - \frac{1}{2}\right| > \frac{3}{8}\right]$$

Applying the Chernoff bound, we have that

$$\Pr\left[\sum_{j=1}^{4n} X_j < s\right] =< 2e^{-\frac{(3/8)^2}{2(1/2)(1/2)}4n} = 2e^{-9n/8}$$

There are a total of 2^n subsets of the original input bits, and therefore the probability that any of them is equal to the exclusive-or of less than s new input bits is bounded by $2^n 2e^{-9n/8} \approx 2^{(1-9/8\log(e))n} \approx 2^{-0.6n} < 2^{-1.2s}$. Lemma 10 therefore implies that with probability $1 - 2^{-1.2s}$ the construction suffices for our proof of security.

Choosing the strings z_i. In order to use the above construction, the parties must construct a circuit that has $4n$ new input bits for P_2. Furthermore, the parties must define n random strings z_i of length $4n$ and then have the circuit map P_2's i^{th} input bit according to the string z_i (as described above). This can be done in two ways. One possibility is to choose the mapping once and for all and hardwire it into the protocol specification. This is problematic because then there is a negligible probability that the protocol is not secure (in any execution). Thus, the mapping should instead be chosen as part of the protocol execution (because negligible failure in any execution is allowed). Fortunately, P_2 can singlehandedly choose the strings $z_1, \ldots, z_n$ in the first step of the protocol and send them to P_1. The reason why this is fine is because this entire issue only arises in the proof of the case that P_1 is corrupted (indeed, for the case of a corrupted P_2 there is no need to split P_2's input bits at all).

6 Acknowledgments

We would like to thank Ivan Damgård, Yuval Ishai, Stas Jarecki, Moni Naor, Adam Smith and David Woodruff for helpful discussions about this work.

References

[1] G. Aggarwal, N. Mishra, and B. Pinkas. Secure Computation of the k-th Ranked Element. In *EUROCRYPT 2004*, Springer-Verlag (LNCS 3027), 40–55, 2004.

[2] B. Aiello, Y. Ishai, and O. Reingold. Priced Oblivious Transfer: How to Sell Digital Goods. In *EUROCRYPT 2001*, Springer-Verlag (LNCS 2045), 119–135, 2001.

[3] B. Barak and Y. Lindell. Strict Polynomial-Time in Simulation and Extraction. *SIAM Journal on Computing,* 33(4):783–818, 2004.

[4] D. Beaver. Foundations of Secure Interactive Computing. In *CRYPTO'91*, Springer-Verlag (LNCS 576), pages 377–391, 1991.

[5] R. Canetti. Security and Composition of Multiparty Cryptographic Protocols. *Journal of Cryptology*, 13(1):143–202, 2000.

[6] R. Cramer, I. Damgård, and B. Schoenmakers. Proofs of Partial Knowledge and Simplified Design of Witness Hiding Protocols. In *CRYPTO'94*, Springer-Verlag (LNCS 839), pages 174–187, 1994.

[7] I. Damgård, T.P. Pedersen and B. Pfitzmann. On the existence of statistically hiding bit commitment schemes and fail-stop signatures. In *CRYPTO'93*, Springer-Verlag (LNCS 773), pages 250–265, 1994.

[8] S. Even, O. Goldreich and A. Lempel. A Randomized Protocol for Signing Contracts. In *Communications of the ACM,* 28(6):637–647, 1985.

[9] O. Goldreich. *Foundations of Cryptography: Volume 1 – Basic Tools.* Cambridge Univ. Press, 2001.

[10] O. Goldreich. *Foundations of Cryptography: Volume 2 – Basic Applications.* Cambridge Univ. Press, 2004.

[11] O. Goldreich and A. Kahan. How To Construct Constant-Round Zero-Knowledge Proof Systems for NP. *Journal of Cryptology*, 9(3):167–190, 1996.

[12] O. Goldreich, S. Micali and A. Wigderson. How to Play any Mental Game – A Completeness Theorem for Protocols with Honest Majority. In 19*th STOC,* pages 218–229, 1987. For details see [10].

[13] S. Goldwasser and L. Levin. Fair Computation of General Functions in Presence of Immoral Majority. In *CRYPTO'90,* Springer-Verlag (LNCS 537), pages 77–93, 1990.

[14] S. Goldwasser, S. Micali and R.L. Rivest, A Digital Signature Scheme Secure Against Adaptive Chosen-Message Attacks. *SIAM J. Comput.* 17(2): 281-308, 1988.

[15] S. Halevi and S. Micali, Practical and Provably-Secure Commitment Schemes from Collision-Free Hashing, *CRYPTO 1996*, Springer-Verlag (LNCS 1109), pages 201-215, 1996.

[16] S. Jarecki and V. Shmatikov. Efficient Two-Party Secure Computation on Committed Inputs. In *Eurocrypt '07*, Springer-Verlag (LNCS 4515), pages 97–114, 2007.

[17] Y.T. Kalai. Smooth Projective Hashing and Two-Message Oblivious Transfer. In *EUROCRYPT 2005*, Springer-Verlag (LNCS 3494), pages 78–95, 2005.

[18] J. Katz and Y. Lindell. Handling Expected Polynomial-Time Strategies in Simulation-Based Security Proofs. In the *2nd Theory of Cryptography Conference* (TCC), Springer-Verlag (LNCS 3378), pp. 128–149, 2005.

[19] J. Kilian. Founding Cryptography on Oblivious Transfer. In *20th STOC*, pages 20–31, 1988.

[20] M. Kiraz and B. Schoenmakers. A Protocol Issue for the Malicious Case of Yao's Garbled Circuit Construction. In *Proceedings of 27th Symposium on Information Theory in the Benelux*, 283–290, 2006.

[21] Y. Lindell. Efficient Fully-Simulatable Oblivious Transfer. Manuscript 2007.

[22] Y. Lindell and B. Pinkas. A Proof of Yao's Protocol for Secure Two-Party Computation. To appear in the *Journal of Cryptology.* Also appeared as *Cryptology ePrint Archive,* Report 2004/175, 2004.

[23] D. Malkhi, N. Nisan, B. Pinkas and Y. Sella. Fairplay – A Secure Two-Party Computation System. In the *13th USENIX Security Symposium*, pages 287–302, 2004.

[24] S. Micali and P. Rogaway. Secure Computation. Unpublished manuscript, 1992. Preliminary version in *CRYPTO'91*, Springer-Verlag (LNCS 576), pages 392–404, 1991.

[25] P. Mohassel and M.K. Franklin. Efficiency Tradeoffs for Malicious Two-Party Computation. In the *9th PKC conference*, Springer-Verlag (LNCS 3958), pages 458–473, 2006.

[26] M. Naor. Bit Commitment Using Pseudorandomness. *Journal of Cryptology,* 4(2):151–158, 1991.

[27] M. Naor and B. Pinkas. Efficient Oblivious Transfer Protocols. In the *12th SODA*, pages 448-457, 2001.

[28] T.P. Pedersen. Non-Interactive and Information-Theoretic Secure Verifiable Secret Sharing. In *CRYPTO'91*, Springer-Verlag (LNCS 576), pages 129–140, 1992.

[29] M. Rabin. How to Exchange Secrets by Oblivious Transfer. Tech. Memo TR-81, Aiken Computation Laboratory, Harvard U., 1981.

[30] D. Woodruff. Revisiting the Efficiency of Malicious Two-Party Computation. In *Eurocrypt '07*, Springer-Verlag (LNCS 4515), pages 79–96, 2007.

[31] A. Yao. How to Generate and Exchange Secrets. In *27th FOCS*, pages 162–167, 1986.

A Two-Party Computation Secure against Semi-Honest Adversaries

We describe here the construction of secure two-party computation (for semi-honest adversaries) which is described in [22]. This construction is based on Yao construction. It is proved in [22] to be secure against semi-honest adversaries.

Let C be a Boolean circuit that receives two inputs $x, y \in \{0,1\}^n$ and outputs $C(x,y) \in \{0,1\}^n$ (for simplicity, we assume that the input length, output length and the security parameter are all of the same length n). We also assume that C has the property that if a circuit-output wire comes from a gate g, then gate g has no wires that are input to other gates.[7] (Likewise, if a circuit-input wire is itself also a circuit-output, then it is not input into any gate.)

We begin by describing the construction of a single garbled gate g in C. The circuit C is Boolean, and therefore any gate is represented by a function $g : \{0,1\} \times \{0,1\} \to \{0,1\}$. Now, let the two input wires to g be labelled w_1 and w_2, and let the output wire from g be labelled w_3. Furthermore, let $k_1^0, k_1^1, k_2^0, k_2^1, k_3^0, k_3^1$ be six keys obtained by independently invoking the key-generation algorithm $G(1^n)$; for simplicity, assume that these keys are also of length n. Intuitively, we wish to be able to compute $k_3^{g(\alpha,\beta)}$ from k_1^α and k_2^β, without revealing any of the other three values $k_3^{g(1-\alpha,\beta)}, k_3^{g(\alpha,1-\beta)}, k_3^{g(1-\alpha,1-\beta)}$. The gate g is defined by the following four values

$$\begin{aligned}
c_{0,0} &= E_{k_1^0}(E_{k_2^0}(k_3^{g(0,0)})) \\
c_{0,1} &= E_{k_1^0}(E_{k_2^1}(k_3^{g(0,1)})) \\
c_{1,0} &= E_{k_1^1}(E_{k_2^0}(k_3^{g(1,0)})) \\
c_{1,1} &= E_{k_1^1}(E_{k_2^1}(k_3^{g(1,1)}))
\end{aligned}$$

where E is from a private key encryption scheme (G, E, D) that has indistinguishable encryptions for multiple messages, and has an elusive efficiently verifiable range; see Section 5.1. The actual gate is defined by a *random permutation* of the above values, denoted as c_0, c_1, c_2, c_3; from here on we call them the garbled table of gate g. Notice that given k_1^α and k_2^β, and the values c_0, c_1, c_2, c_3, it is possible to compute the output of the gate $k_3^{g(\alpha,\beta)}$ as follows. For every i, compute $D_{k_2^\beta}(D_{k_1^\alpha}(c_i))$. If more than one decryption returns a non-$\perp$ value, then output abort. Otherwise, define k_3^γ to be the only non-$\perp$ value that is obtained. (Notice that if only a single non-$\perp$ value is obtained, then this will be $k_3^{g(\alpha,\beta)}$ because it is encrypted under the given keys k_1^α and k_2^β. Later we will show that except with negligible probability, only one non-$\perp$ value is indeed obtained.)

We are now ready to show how to construct the entire garbled circuit. Let m be the number of *wires* in the circuit C, and let $w_1, \ldots, w_m$ be labels of these wires. These labels are all chosen uniquely with the following exception: if w_i and w_j are both output wires from the same gate g, then $w_i = w_j$ (this occurs if the fan-out of g is greater than one). Likewise, if an input bit enters more than one gate, then all circuit-input wires associated with this bit will have the same label. Next, for every label w_i, choose two independent keys $k_i^0, k_i^1 \leftarrow G(1^n)$; we stress that all of these keys are chosen independently of the others. Now, given these keys, the four garbled values of each gate are computed as described above and the results are permuted randomly. Finally, the output or decryption tables of the garbled circuit are computed. These tables simply consist of the values $(0, k_i^0)$ and $(1, k_i^1)$ where w_i is a *circuit-output wire*. (Alternatively, output gates can just compute 0 or 1 directly. That is, in an output gate, one can define $c_{\alpha,\beta} = E_{k_1^\alpha}(E_{k_2^\beta}(g(\alpha,\beta)))$ for every $\alpha, \beta \in \{0,1\}$.)

The entire garbled circuit of C, denoted GC, consists of the garbled table for each gate and the output tables. We note that the structure of C is given, and the garbled version of C is simply defined by specifying the output tables and the garbled table that belongs to each gate. This completes the description of the garbled circuit.

[7] This requirement is due to our labelling of gates described below, that does not provide a unique label to each wire (see [22] for more discussion). We note that this assumption on C increases the number of gates by at most n.

Let $x = x_1 \cdots x_n$ and $y = y_1 \cdots y_n$ be two n-bit inputs for C. Furthermore, let $w_1, \ldots, w_n$ be the input labels corresponding to x, and let $w_{n+1}, \ldots, w_{2n}$ be the input labels corresponding to y. It is shown in [22] that given the garbled circuit GC and the strings $k_1^{x_1}, \ldots, k_n^{x_n}, k_{n+1}^{y_1}, \ldots, k_{2n}^{y_n}$, it is possible to compute $C(x, y)$, except with negligible probability.