Security Against Covert Adversaries: Efficient Protocols for Realistic Adversaries*

Yonatan Aumann and Yehuda Lindell

Department of Computer Science
Bar-Ilan University, Israel
{aumann,lindell}@cs.biu.ac.il

Abstract. In the setting of secure multiparty computation, a set of mutually distrustful parties wish to securely compute some joint function of their private inputs. The computation should be carried out in a secure way, meaning that no coalition of corrupted parties should be able to learn more than specified or somehow cause the result to be "incorrect". Typically, corrupted parties are either assumed to be semi-honest (meaning that they follow the protocol specification) or malicious (meaning that they may deviate arbitrarily from the protocol). However, in many settings, the assumption regarding semi-honest behavior does not suffice and security in the presence of malicious adversaries is excessive and expensive to achieve.

In this paper, we introduce the notion of *covert adversaries*, which we believe faithfully models the adversarial behavior in many commercial, political, and social settings. Covert adversaries have the property that they may deviate arbitrarily from the protocol specification in an attempt to cheat, but do not wish to be "caught" doing so. We provide a definition of security for covert adversaries and show that it is possible to obtain highly efficient protocols that are secure against such adversaries. We stress that in our definition, we quantify over all (possibly malicious) adversaries and do not assume that the adversary behaves in any particular way. Rather, we guarantee that if an adversary deviates from the protocol in a way that would enable it to "cheat", then the honest parties are guaranteed to detect this cheating with good probability. We argue that this level of security is sufficient in many settings.

1 Introduction

1.1 Background

In the setting of secure multiparty computation, a set of parties with private inputs wish to jointly compute some functionality of their inputs. Loosely speaking, the security requirements of such a computation are that (i) nothing is learned from the protocol other than the output (privacy), (ii) the output is distributed according to the prescribed functionality (correctness), and (iii) parties cannot

* Work supported in part by an Infrastructures grant from the Ministry of Science, Israel. The full version of this paper is available from the *Cryptology ePrint Archive*.

make their inputs depend on other parties' inputs. Secure multiparty computation forms the basis for a multitude of tasks, including those as simple as coin-tossing and agreement, and as complex as electronic voting, electronic auctions, electronic cash schemes, anonymous transactions, remote game playing (a.k.a. "mental poker"), and privacy-preserving data mining.

The security requirements in the setting of multiparty computation must hold even when some of the participating parties are adversarial. It has been shown that, with the aid of suitable cryptographic tools, *any* two-party or multiparty function can be securely computed [23, 12, 10, 3, 6], even in the presence of very strong adversarial behavior. However, the efficiency of the computation depends dramatically on the adversarial model considered. Classically, two main categories of adversaries have been considered:

1. *Malicious adversaries:* these adversaries may behave arbitrarily and are not bound in any way to following the instructions of the specified protocol. Protocols that are secure in the malicious model provide a very strong security guarantee, as honest parties are "protected" irrespective of the adversarial behavior of the corrupted parties.
2. *Semi-honest adversaries:* these adversaries correctly follow the protocol specification, yet may attempt to learn additional information by analyzing the transcript of messages received during the execution. Security in the presence of semi-honest adversaries provides only a weak security guarantee, and is not sufficient in many settings. Semi-honest adversarial behavior primarily models inadvertent leakage of information, and is suitable only where participating parties essentially trust each other, but may have other concerns.

Secure computation in the semi-honest adversary model can be carried out very efficiently, but, as mentioned, provides weak security guarantees. Regarding malicious adversaries, it has been shown that, under suitable cryptographic assumptions, *any* multiparty probabilistic polynomial-time functionality *can* be securely computed for any number of *malicious* corrupted parties [12, 10]. However, this comes at a price. These feasibility results of secure computation typically do not yield protocols that are efficient enough to actually be implemented and used in practice (particularly if standard *simulation-based security* is required). Their importance is more in telling us that it is perhaps worthwhile searching for other efficient protocols, because we at least know that a solution exists in principle. However, the unfortunate state of affairs today – many years after these feasibility results were obtained – is that very few truly efficient protocols exist for the setting of malicious adversaries. Thus, we believe that some middle ground is called for: an adversary model that accurately models adversarial behavior in the real world, on the one hand, but for which efficient, secure protocols can be obtained, on the other.

1.2 Our Work – Covert Adversaries

In this work, we introduce a new adversary model that lies between the semi-honest and malicious models. The motivation behind the definition is that in many real-world settings, adversaries are willing to actively cheat (and as such are not semi-honest), but only if they are not caught (and as such they are not arbitrarily malicious). This, we believe, is the case in many business, financial, political and diplomatic settings, where honest behavior cannot be assumed, but where the companies, institutions and individuals involved cannot afford the embarrassment, loss of reputation, and negative press associated with being *caught* cheating. It is also the case, unfortunately, in many social settings, e.g. elections for a president of the country-club. Finally, in remote game playing, players may also be willing to actively cheat, but would try to avoid being caught, or else they may be thrown out of the game. In all, we believe that this type of *covert* adversarial behavior accurately models many real-world situations. Clearly, with such adversaries, it may be the case that the risk of being caught is weighed against the benefits of cheating, and it cannot be assumed that players would avoid being caught at any price and under all circumstances. Accordingly, our definition explicitly models the probability of catching adversarial behavior; a probability that can be tuned to the specific circumstances of the problem. In particular, we do not assume that adversaries are only willing to risk being caught with negligible probability, but rather allow for much higher probabilities.

The definition. Our definition of security is based on the classical *ideal/real simulation paradigm*. Loosely speaking, our definition provides the following guarantee. Let $0 < \epsilon \leq 1$ be a value (called the *deterrence factor*). Then, any attempt to cheat by an adversary is detected by the honest parties with probability at least ϵ. Thus, provided that ϵ is sufficiently large, an adversary that wishes not to be caught cheating, will refrain from *attempting* to cheat, lest it be caught doing so. Clearly, the higher the value of ϵ, the greater the probability that the adversary is caught and thus the greater the *deterrent* to cheat. We therefore call our notion security in the presence of covert adversaries with ϵ-deterrent. Note that the security guarantee does not preclude successful cheating. Indeed, if the adversary decides to cheat then it may gain access to the other parties' private information or bias the result of the computation. The only guarantee is that if it attempts to cheat, then there is a fair chance that it will be caught doing so. This is in contrast to standard definitions, where absolute privacy and security are guaranteed, for the given type of adversary. We remark that by setting $\epsilon = 1$, our definition can be used to capture a requirement that cheating parties are always caught.

When attempting to translate the above described basic approach into a formal definition, we obtain three different possible formulations, which form a hierarchy of security guarantees. In Section 3 we present the three formulations, and discuss the relationships between them and between the standard definitions of security for semi-honest and malicious adversaries. We also present *modular sequential composition* theorems (like that of [4]) for all of our definitions. Such composition theorems are important as security goals within themselves and as tools for proving the security of protocols.

Protocol constructions. As mentioned, the aim of this work is to provide a definition of security for which it is possible to construct highly efficient protocols. We demonstrate this fact by presenting a generic protocol for secure two-party computation that is only mildly less efficient than the protocol of Yao [23], which is secure only for semi-honest adversaries. The first step of our construction is a protocol for oblivious transfer that is based on homomorphic encryption schemes. Highly efficient protocols under this assumption are known [1, 17]. However, these protocols do not achieve *simulation-based* security. Rather, only privacy is guaranteed (with the plus that privacy is preserved even in the presence of fully malicious adversaries). Having constructed an oblivious transfer protocol that meets our definition, we use it in the protocol of Yao [23]. We modify Yao's protocol so that two garbled circuits are sent, and then a random one is opened in order to check that it was constructed correctly. Our basic protocol achieves deterrent $\epsilon = 1/2$, but can be extended to greater values of ϵ at a moderate expense in efficiency. (For example, 10 copies of the circuit yields $\epsilon = 9/10$.)

Protocol efficiency. The protocol we present offers a great improvement in efficiency, when compared to the best known results for the malicious adversary model. The exact efficiency depends on the variant used in the definition of covert adversary security. For the weakest variant, our protocol requires only *twice* the amount of work and twice the bandwidth of the basic protocol of [23] for semi-honest adversaries. Specifically, it requires only a constant number of rounds, a single oblivious transfer for each input bit, and has communication complexity $O(n|C|)$ where n is the security parameter and $|C|$ is the size of the circuit being computed. For the intermediate variant, the complexity is slightly higher, requiring twice the number of oblivious transfers than in the weakest variant. For the strongest variant, the complexity increases to n oblivious transfers for each input bit. This is still much more efficient than any known protocol for the case of malicious adversaries. We view this as a "proof of concept" that highly efficient protocols are achievable in this model, and leave the construction of such protocols for specific tasks of interest for future work.

1.3 Related Work

The idea of allowing the adversary to cheat as long as it will be detected was first considered by [9] who defined a property called *t-detectability;* loosely speaking, a protocol fulfilling this property provides the guarantee that no coalition of t parties can cheat without being caught. The work of [9] differs to ours in that (a) they consider the setting of an honest majority, and (b) their definition is not simulation based. Another closely related work to ours is that of [5] that considers *honest-looking adversaries*. Such adversaries may deviate arbitrarily from the protocol specification, but only if this deviation cannot be detected. Our definition differs from that of [5] in a number of important ways. First, we quantify over *all* adversaries, and not only over adversaries that behave in a certain way. Second, our definition provides guarantees even for adversaries that may be willing to risk being caught cheating with non-negligible (or even constant) probability. Third, we place the onus of detecting any cheating by an adversary on the protocol, and not on the chance that the honest parties will analyze the distribution of the messages generated by the corrupted parties. (See Section 3 for more discussion on why these differences are important.) Finally, we remark that [5] considered a more stringent setting where all parties are either malicious or honest-looking. In contrast, we consider a *relaxation* of the adversary model (where parties are either fully honest or covert).

We remark that the idea of allowing an adversary to cheat with non-negligible probability as long as it will be caught with good probability has been mentioned many times in the literature; see [15, 20] for just two examples. We stress, however, that none of these works formalized this idea. Furthermore, our experience in proving our protocol secure is that simple applications of cut-and-choose do not meet our definition (and there are actual attacks that can be carried out on the cut-and-choose technique used in [20], for example).

Our work studies a weaker definition of security than the standard one. Weaker definitions have been used before in order to construct efficient protocols for specific problems. However, in the past these relaxed definitions typically have not followed the simulation paradigm, but rather have considered privacy via indistinguishability (and sometimes correctness); see [7] for one example. Our work takes a completely different approach.

2 Secure Multiparty Computation – Standard Definition

In this section we briefly present the standard definition for secure multiparty computation and refer to [10, Chapter 7] for more details and motivating discussion. The following description and definition is based on [10], which in turn follows [13, 21, 2, 4].

Multiparty computation. A multiparty protocol problem is cast by specifying a random process that maps sets of inputs to sets of outputs (one for each party). We refer to such a process as a functionality and denote it $f : (\{0,1\}^*)^m \to (\{0,1\}^*)^m$, where $f = (f_1, \ldots, f_m)$. That is, for every vector of inputs $\overline{x} = (x_1, \ldots, x_m)$, the output-vector is a random variable $\overline{y} = (f_1(\overline{x}), \ldots, f_m(\overline{x}))$ ranging over vectors of strings. The i^{th} party P_i, with input x_i, wishes to obtain $f_i(\overline{x})$. We sometimes denote such a functionality by $(\overline{x}) \mapsto (f_1(\overline{x}), \ldots, f_m(\overline{x}))$. Thus, for example, the oblivious transfer functionality is denoted by $((x_0, x_1), \sigma) \mapsto (\lambda, x_\sigma)$, where (x_0, x_1) is the first party's input, σ is the second party's input, and λ denotes the empty string (meaning that the first party has no output).

Security of protocols (informal). The security of a protocol is analyzed by comparing what an adversary can do in a real protocol execution 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 (in order to model the possibility of early aborting, the adversary receives its outputs first and then can decide if the honest parties also receive 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. We consider malicious adversaries and static corruptions in all of our definitions in this paper.

Execution in the ideal model. Let the set of parties be $P_1, \ldots, P_m$ and let $I \subseteq [m]$ denote the indices of the corrupted parties, controlled by an adversary $\mathcal{A}$. An ideal execution proceeds as follows:

Inputs: Each party obtains an input; the i^{th} party's input is denoted x_i. The adversary $\mathcal{A}$ receives an auxiliary input denoted z (and we assume that it knows the length of all inputs).

Send inputs to trusted party: Any honest party P_j sends its received input x_j to the trusted party. The corrupted parties controlled by $\mathcal{A}$ may either abort, send their received input, or send some other input of the same length to the trusted party. This decision is made by $\mathcal{A}$ and may depend on the values x_i for $i \in I$ and its auxiliary input z. Denote the vector of inputs sent to the trusted party by $\overline{w}$ (note that $\overline{w}$ does not necessarily equal $\overline{x}$).
If the trusted party does not receive m valid inputs (including the case that one of the inputs equals $\perp$), it replies to all parties with a special symbol $\perp$ and the ideal execution terminates. Otherwise, the execution proceeds to the next step.

Trusted party sends outputs to adversary: The trusted party computes $(f_1(\overline{w}), \ldots, f_m(\overline{w}))$ and sends $f_i(\overline{w})$ to party P_i, for all $i \in I$ (i.e., to all corrupted parties).

Adversary instructs trusted party to continue or halt: $\mathcal{A}$ sends either CONTINUE or HALT to the trusted party. If it sends CONTINUE, the trusted party sends $f_j(\overline{w})$ to party P_j, for all $j \notin I$ (i.e., to all honest parties). Otherwise, if it sends HALT, the trusted party sends $\perp$ to all parties P_j for $j \notin I$.

Outputs: An honest party always outputs the message it obtained from the trusted party. The corrupted parties output nothing. The adversary $\mathcal{A}$ outputs any arbitrary (probabilistic polynomial-time computable) function of the initial inputs $\{x_i\}_{i \in I}$ and the messages $\{f_i(\overline{w})\}_{i \in I}$ obtained from the trusted party.

Let $f : (\{0,1\}^*)^m \to (\{0,1\}^*)^m$ be an m-party functionality, where $f = (f_1, \ldots, f_m)$, let $\mathcal{A}$ be a non-uniform probabilistic polynomial-time machine, and let $I \subseteq [m]$ be the set of corrupted parties. Then, the ideal execution of f on inputs $\overline{x}$, auxiliary input z to $\mathcal{A}$ and security parameter n, denoted $\text{IDEAL}_{f,\mathcal{A}(z),I}(\overline{x}, n)$, is defined as the output vector of the honest parties and the adversary $\mathcal{A}$ from the above ideal execution.

Execution in the real model. We next consider the real model in which a real m-party protocol π is executed (and there exists no trusted third party). In this case, the adversary $\mathcal{A}$ sends all messages in place of the corrupted parties, and may follow an arbitrary polynomial-time strategy. In contrast, the honest parties follow the instructions of π.

Let f be as above and let π be an m-party protocol for computing f. Furthermore, let $\mathcal{A}$ be a non-uniform probabilistic polynomial-time machine and let I be the set of corrupted parties. Then, the real execution of π on inputs $\overline{x}$, auxiliary input z to $\mathcal{A}$ and security parameter n, denoted $\text{REAL}_{\pi,\mathcal{A}(z),I}(\overline{x}, n)$, is defined as the output vector of the honest parties and the adversary $\mathcal{A}$ from the real execution of π.

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. We will consider executions where all inputs are of the same length (see discussion in [10]), and will therefore say that a vector $\overline{x} = (x_1, \ldots, x_m)$ is balanced if for every i and j it holds that $|x_i| = |x_j|$.

Definition 1 (secure multiparty computation): *Let f and π be as above. Protocol π is said to* securely compute f with abort in the presence of malicious adversaries *if for every non-uniform probabilistic polynomial-time adversary $\mathcal{A}$ for the real model, there exists a non-uniform probabilistic polynomial-time adversary $\mathcal{S}$ for the ideal model, such that for every $I \subseteq [m]$, every balanced vector $\overline{x} \in (\{0,1\}^*)^m$, and every auxiliary input $z \in \{0,1\}^*$:*

$$\left\{\text{IDEAL}_{f,\mathcal{S}(z),I}(\overline{x}, n)\right\}_{n\in\mathbb{N}} \stackrel{c}{\equiv} \left\{\text{REAL}_{\pi,\mathcal{A}(z),I}(\overline{x}, n)\right\}_{n\in\mathbb{N}}$$

where $\stackrel{c}{\equiv}$ indicates computational indistinguishability.

3 Definitions – Security with Covert Adversaries

3.1 Motivation

The standard definition of security (see Definition 1) is such that all possible (polynomial-time) adversarial behavior is simulatable. In contrast, as we have mentioned, here we wish to model the situation that parties may cheat. However, if they do so, they are likely to be caught. There are a number of ways of defining this notion. In order to motivate ours, we begin with a somewhat naive implementation of the notion, and show its shortcoming.

First attempt: Define an adversary to be covert if the distribution over the messages that it sends during an execution is computationally indistinguishable from the distribution over the messages that an honest party would send. Then quantify over all covert adversaries $\mathcal{A}$ for the real world (rather than all adversaries).[1] A number of problems arise with this definition. First, the fact that the distribution generated by the adversary can be distinguished from the distribution generated by honest parties does not mean that the honest parties indeed detect this. This is due to the fact that the honest parties may not have an efficient distinguisher; it is only guaranteed that there exists one. Furthermore, in order to guarantee that the honest parties detect the cheating, they would have

[1] We remark that this is the conceptual approach taken by [5], and that there are important choices that arise when attempting to formalize the approach. In any case, as we have mentioned, the work of [5] differs greatly because their aim was to model all parties as somewhat adversarial.

to analyze all traffic during an execution. However, this analysis *cannot* be part of the protocol because then the distinguishers used by the honest parties would be known (and potentially bypassed). Another problem is that, as mentioned in the introduction, adversaries may be willing to risk being caught with more than negligible probability, say 10^{-6}. With such an adversary, the definition would provide no security guarantee. In particular, the adversary may be able to always learn all parties' inputs, and only risk being caught in one run in a million.

Second attempt. To solve the aforementioned problems, we first we require that the protocol itself be responsible for detecting cheating. Specifically, in the case that a party P_i attempts to cheat, the protocol may instruct the honest parties to output a message saying that "party P_i has cheated" (we require that this only happens if P_i indeed cheated). This solves the first problem. To solve the second problem, we explicitly quantify the probability that an adversary is caught cheating. Roughly, given a parameter ϵ, a protocol is said to be secure against covert adversaries with ϵ-deterrent if any cheating adversary will necessarily be caught with probability at least ϵ.

This definition captures the spirit of what we want, but is still problematic. To illustrate the problem, consider an adversary that plays honestly with probability 0.99, and cheats otherwise. Such an adversary can only ever be caught with probability 0.01 (because otherwise it is honest). If $\epsilon = 1/2$ for example, then such an adversary must be caught with probability 0.5, which is impossible. We therefore conclude that an *absolute* parameter cannot be used, and the probability of catching the adversary must be related to the probability that it cheats.

Final definition. We thus arrive at the following approach. First, as mentioned, we require that the protocol itself be responsible for detecting cheating. That is, if a party P_i successfully cheats, then with good probability (ϵ), the honest parties in the protocol will all receive a message that "P_i cheated". Second, we do not quantify only over adversaries that are covert (i.e., those that are not detected cheating by the protocol). Rather, we allow all possible adversaries, even completely malicious ones. Then, we require either that this malicious behavior can be successfully simulated (as in Definition 1), or that the honest parties will receive a message that cheating has been detected, and this happens with probability at least ϵ times the probability that successful cheating takes place. In other words, when an adversarial attack is carried out, we are guaranteed that one of the following two happens:

1. *The attack fails:* this event is represented by the fact that the adversary can simulate the interaction on its own, and so the attack cannot yield any more than what is possible in the ideal model.
2. *The attack succeeds:* in this case we are guaranteed that with good probability (and this probability is a parameter in the definition), the adversarial parties will be caught.

We stress that in the second case, the adversary may actually learn secret information or cause some other damage. However, since it is guaranteed that such a strategy will likely be caught, there is strong motivation to refrain from carrying it out.

As it turns out, the above intuition can be formalized in three different ways, which form a hierarchy of security guarantees. Since we view the definitional part of this work as of no less importance than the protocol constructions, we present all three formulations. In practice, the practitioner should choose the formulation that best suites her needs, and for which sufficiently efficient protocols exists. All three definitions are based on the ideal/real simulation paradigm, as presented in Section 2. We now present the definitions in order of security, starting with the weakest (least secure) one.

3.2 Version 1: Failed Simulation Formulation

The first formulation we present is based on allowing the simulator to fail sometimes, where by "fail" we mean that its output distribution is not indistinguishable from the real one. This corresponds to an event of successful cheating. However, we guarantee that the probability that the adversary is caught cheating is at least ϵ times the probability that the simulator fails. The details follow.

Recall that we call a vector balanced if all of its items are of the same length. In addition, we denote the output vector of the honest parties and adversary $\mathcal{A}$ in an ideal execution of f by $\text{IDEAL}_{f,\mathcal{A}(z),I}(\overline{x}, n)$, where $\overline{x}$ is the vector of inputs, z is the auxiliary input to $\mathcal{A}$, I is the set of corrupted parties, and n is the security parameter. Finally, we denote the analogous outputs in a real execution of π by $\text{REAL}_{\pi,\mathcal{A}(z),I}(\overline{x}, n)$. We begin by defining what it means to "detect cheating":

Definition 2 *Let π be an m-party protocol, let $\mathcal{A}$ be an adversary, and let I be the index set of the corrupted parties. A party P_j is said to* detect cheating in π *if its output in π is* $\mathsf{corrupted}_i$*; this event is denoted* $\text{OUTPUT}_j(\text{REAL}_{\pi,\mathcal{A}(z),I}(\overline{x})) = \mathsf{corrupted}_i$. *The protocol π is called* detection accurate *if for every $j, k \notin I$, the probability that P_j outputs* $\mathsf{corrupted}_k$ *is negligible.*

We require that all protocols be detection accurate (meaning that only corrupted parties can be "caught cheating"). This is crucial because otherwise a party that is detected cheating can just claim that it is due to a protocol anomaly and not because it really cheated. The definition follows:

Definition 3 (security – failed simulation formulation): *Let f and π be as in Definition 1, and let $\epsilon : \mathbb{N} \to [0, 1]$ be a function. Protocol π is said to* securely compute f in the presence of covert adversaries with ϵ-deterrent *if it is* detection accurate *and if for every non-uniform probabilistic polynomial-time adversary $\mathcal{A}$ for the real model, there exists a non-uniform probabilistic polynomial-time adversary $\mathcal{S}$ for the ideal model such that for every $I \subseteq [m]$, every balanced vector $\overline{x} \in (\{0, 1\}^*)^m$, every auxiliary input $z \in \{0, 1\}^*$, and every non-uniform polynomial-time distinguisher D, there exists a negligible function $\mu(\cdot)$ such that,*

$$\Pr\left[\exists i \in I \ \forall j \notin I \ : \ \text{OUTPUT}_j(\text{REAL}_{\pi,\mathcal{A}(z),I}(\overline{x}, n)) = \mathsf{corrupted}_i\right]$$
$$\geq \epsilon(n) \cdot \left|\Pr\left[D(\text{IDEAL}_{f,\mathcal{S}(z),I}(\overline{x}, n)) = 1\right] - \Pr\left[D(\text{REAL}_{\pi,\mathcal{A}(z),I}(\overline{x}, n)) = 1\right]\right| - \mu(n)$$

The parameter ϵ indicates the probability that successful adversarial behavior is detected (observe that when such a detection occurs, *all* honest parties must detect the same corrupted party). Clearly, the closer ϵ is to one, the higher the deterrence to cheat, and hence the level of security, assuming covert adversaries. Note that the adversary can decide to never be detected cheating, in which case the IDEAL and REAL distributions are guaranteed to be *computationally indistinguishable*, as in the standard definition of security. In contrast, it can choose to cheat with some noticeable probability, in which case the IDEAL and REAL output distribution may be distinguishable (while guaranteeing that the adversary is caught with good probability). This idea of allowing the ideal and real models to not be fully indistinguishable in order to model "allowed cheating" was used in [11].

We stress that the definition does not *require* the simulator to "fail" with some probability. Rather, it is *allowed* to fail with a probability that is at most $1/\epsilon$ times the probability that the adversary is caught cheating. As we shall see, this is what enables us to construct highly efficient protocols. We also remark that due to the required detection accuracy, the simulator cannot fail when the adversary behaves in a fully honest-looking manner (because in such a case, no honest party will output $\mathsf{corrupted}_i$). Thus, security is always preserved in the presence of adversaries that are willing to cheat arbitrarily, as long as their cheating is not detected.

Cheating and aborting. It is important to note that according to the above definition, a party that halts mid-way through the computation may be considered a "cheat". Arguably, this may be undesirable due to the fact that an honest party's computer may crash (such unfortunate events may not even be that rare). Nevertheless, we argue that as a basic definition it suffices. This is due to the fact that it is possible for all parties to work by storing their input and random-tape on disk before they begin the execution. Then, before sending any message, the incoming messages that preceded it are also written to disk. The result of this is that if a party's machine crashes, it can easily reboot and return to its previous state. (In the worst case the party will need to request a retransmit of the last message if the crash occurred before it was written.) We therefore believe that honest parties cannot truly hide behind the excuse that their machine crashed (it would be highly suspicious that someone's machine crashed in an irreversible way that also destroyed their disk at the critical point of a secure protocol execution).

Despite the above, it is possible to modify the definition so that honest halting is never considered cheating. This modification only needs to be made to the notion of "detection accuracy" and uses the notion of a fail-stop party who acts semi-honestly, except that it may halt early.

Definition 4 *A protocol π is* non-halting detection accurate *if it is detection accurate as in Definition 2 and if for every honest party P_j and fail-stop party P_k, the probability that P_j outputs* $\mathsf{corrupted}_k$ *is negligible.*

The definition of security in the presence of covert adversaries can then be modified by requiring non-halting detection accuracy. We remark that although this strengthening is highly desirable, it may also be prohibitive. For example, we are able to modify our main protocol so that it meets this stronger definition. However, in order to do so, we need to assume fully secure oblivious transfer, for which highly efficient (fully simulatable) protocols are not really known.

3.3 Version 2: Explicit Cheat Formulation

The drawback of Definition 3 is that it allows the adversary to decide whether to cheat as a function of the honest parties' inputs or of the output. This is undesirable since there may be honest parties' inputs for which it is more "worthwhile" for the adversary to risk being caught. We therefore wish to force the adversary to make its decision about whether to cheat *obliviously* of the honest parties' inputs. This brings us to an alternate definition, which is based on redefining the ideal functionality so as to explicitly include the option of cheating. Aside from overcoming the input dependency problem this alternate formulation has two additional advantages. First, it makes the security guarantees more explicit. Second, it makes it easy to prove a sequential composition theorem.

We modify the ideal model in the following way. Let $\epsilon : \mathbb{N} \to [0,1]$ be a function. Then, the ideal execution with ϵ proceeds as follows:

Inputs: Each party obtains an input; the i^{th} party's input is denoted by x_i; we assume that all inputs are of the same length, denoted n. The adversary receives an auxiliary-input z.

Send inputs to trusted party: Any honest party P_j sends its received input x_j to the trusted party. The corrupted parties, controlled by $\mathcal{A}$, may either send their received input, or send some other input of the same length to the trusted party. This decision is made by $\mathcal{A}$ and may depend on the values x_i for $i \in I$ and the auxiliary input z. Denote the vector of inputs sent to the trusted party by $\overline{w}$.

Abort options: If a corrupted party sends $w_i = \mathsf{abort}_i$ to the trusted party as its input, then the trusted party sends abort_i to all of the honest parties and halts. If a corrupted party sends $w_i = \mathsf{corrupted}_i$ to the trusted party as its input, then the trusted party sends $\mathsf{corrupted}_i$ to all of the honest parties and halts.

Attempted cheat option: If a corrupted party sends $w_i = \mathsf{cheat}_i$ to the trusted party as its input, then the trusted party sends to the adversary all of the honest parties' inputs $\{x_j\}_{j \notin I}$. Furthermore, it asks the adversary for outputs $\{y_j\}_{j \notin I}$ for the honest parties. In addition,

1. With probability ϵ, the trusted party sends $\mathsf{corrupted}_i$ to the adversary and all of the honest parties.
2. With probability $1-\epsilon$, the trusted party sends $\mathsf{undetected}$ to the adversary and the outputs $\{y_j\}_{j \notin I}$ to the honest parties (i.e., for every $j \notin I$, the trusted party sends y_j to P_j).

The ideal execution then ends at this point.

If no w_i equals abort_i, $\mathsf{corrupted}_i$ or cheat_i, the ideal execution continues below.

Trusted party answers adversary: The trusted party computes $(f_1(\overline{w}), \ldots, f_m(\overline{w}))$ and sends $f_i(\overline{w})$ to $\mathcal{A}$, for all $i \in I$.

Trusted party answers honest parties: After receiving its outputs, the adversary sends either abort_i for some $i \in I$, or $\mathsf{continue}$ to the trusted party. If the trusted party receives $\mathsf{continue}$ then it sends $f_j(\overline{w})$ to all honest parties P_j $(j \notin I)$. Otherwise, if it receives abort_i for some $i \in I$, it sends abort_i to all honest parties.

Outputs: An honest party always outputs the message it obtained from the trusted party. The corrupted parties output nothing. The adversary $\mathcal{A}$ outputs any arbitrary (probabilistic polynomial-time computable) function of the initial inputs $\{x_i\}_{i \in I}$ and the messages obtained from the trusted party.

The output of the honest parties and the adversary in an execution of the above ideal model is denoted by $\text{IDEALC}^{\epsilon}_{f,\mathcal{S}(z),I}(\overline{x}, n)$.

Notice that there are two types of "cheating" here. The first is the classic abort, except that unlike in Definition 1, the honest parties here are informed as to who caused the abort. Thus, although it is not possible to guarantee fairness here, we do achieve that an adversary who aborts after receiving its output is "punished" in the sense that its behavior is always detected.[2] The other type of cheating in this ideal model is more serious for two reasons: first, the ramifications of the cheat are greater (the adversary may learn all of the parties' inputs and may be able to determine their outputs), and second, the cheating is only guaranteed to be detected with probability ϵ. Nevertheless, if ϵ is high enough, this may serve as a deterrent. We stress that in the ideal model the adversary must decide whether to cheat obliviously of the honest-parties inputs and before it receives any output (and so it cannot use the output to help it decide whether or not it is "worthwhile" cheating). We define:

Definition 5 (security – explicit cheat formulation): *Let f, π and ϵ be as in Definition 3. Protocol π is said to* securely compute f in the presence of covert adversaries with ϵ-deterrent *if for every non-uniform probabilistic polynomial-time adversary $\mathcal{A}$ for the real model, there exists a non-uniform probabilistic polynomial-time adversary $\mathcal{S}$ for the ideal model such that for every $I \subseteq [m]$, every balanced vector $\overline{x} \in (\{0,1\}^*)^m$, and every auxiliary input $z \in \{0,1\}^*$:*

$$\left\{\text{IDEALC}^{\epsilon}_{f,\mathcal{S}(z),I}(\overline{x}, n)\right\}_{n \in \mathbb{N}} \stackrel{c}{\equiv} \left\{\text{REAL}_{\pi,\mathcal{A}(z),I}(\overline{x}, n)\right\}_{n \in \mathbb{N}}$$

Definition 5 and detection accuracy. We note that in Definition 5 it is not necessary to explicitly require that π be detection accurate because this is taken care of in the ideal model (in an ideal execution, only a corrupted party can send a cheat_i input). However, if *non-halting detection accuracy* is desired (as in Definition 4), then this should be explicitly added to the definition.

---

[2] Note also that there are two types of abort: in one the honest parties receive abort_i and in the second they receive $\mathsf{corrupted}_i$. This is included to model behavior by the real adversary that results in it being caught cheating with probability greater than ϵ (and not with probability exactly ϵ as when the ideal adversary sends a cheat_i message).

3.4 Version 3: Strong Explicit Cheat Formulation

The third, and strongest version follows the same structure and formulation of the previous version (Version 2). However, we make the following slight, but important change to the ideal model. In the case of an attempted cheat, if the trusted party sends $\mathsf{corrupted}_i$ to the honest parties and the adversary (an event which happens with probability ϵ), then the adversary does *not* obtain the honest parties' inputs. Thus, if cheating is detected, the adversary does not learn anything and the result is essentially the same as a regular abort. This is in contrast to Version 2, where a detected cheat may still be successful. (We stress that in the "undetected" case here, the adversary still learns the honest parties' private inputs and can set their outputs.) We denote the resultant ideal model by $\text{IDEALSC}^{\epsilon}_{f,\mathcal{S}(z),I}(\overline{x},n)$ and have the following definition:

Definition 6 (security – strong explicit cheat formulation): *Let f, π and ϵ be as in Definition 3. Protocol π is said to* securely compute f in the presence of covert adversaries with ϵ-deterrent *if for every non-uniform probabilistic polynomial-time adversary $\mathcal{A}$ for the real model, there exists a non-uniform probabilistic polynomial-time adversary $\mathcal{S}$ for the ideal model such that for every $I \subseteq [m]$, every balanced vector $\overline{x} \in (\{0,1\}^*)^m$, and every auxiliary input $z \in \{0,1\}^*$:*

$$\left\{\text{IDEALSC}^{\epsilon}_{f,\mathcal{S}(z),I}(\overline{x},n)\right\}_{n\in\mathbb{N}} \stackrel{\text{c}}{\equiv} \left\{\text{REAL}_{\pi,\mathcal{A}(z),I}(\overline{x},n)\right\}_{n\in\mathbb{N}}$$

The difference between the regular and strong explicit cheat formulations is perhaps best exemplified in the case that $\epsilon = 1$. In both versions, all potentially successful cheating attempt are detected. However, in the regular formulation, the adversary may learn the honest parties' private inputs (albeit, while being detected). In the strong formulation, in contrast, the adversary learns nothing when it is detected. Since it is always detected, this means that full security is achieved.

3.5 Relations Between Security Models

Relations between covert security definitions. It is not difficult to show that the three security definitions for covert adversaries constitute a strict hierarchy, with version 1 being strictly weaker than version 2, which is strictly weaker than version 3. We explicitly prove this in the full version of the paper.

Relation to the malicious and semi-honest models. As a sanity check regarding our definitions, we present two propositions that show the relation between security in the presence of covert adversaries and security in the presence of malicious and semi-honest adversaries.

Proposition 7 *Let π be a protocol that securely computes some functionality f with abort in the presence of malicious adversaries, as in Definition 1. Then, π securely computes f in the presence of covert adversaries with ϵ-deterrent, for any of the three formulations and for every $0 \le \epsilon \le 1$.*

This proposition follows from the simple observation that according to Definition 1, there exists a simulator that always succeeds in its simulation. Thus, Definition 3 holds even if the probability of detecting cheating is 0. Likewise, for Definitions 5 and 6 the same simulator works (there is simply no need to ever send a cheat input). Next, we consider the relation between covert and semi-honest adversaries.

Proposition 8 *Let π be a protocol that securely computes some functionality f in the presence of covert adversaries with ϵ-deterrent, for any of the three formulations and for $\epsilon \geq 1/\mathrm{poly}(n)$. Then, π securely computes f in the presence of semi-honest adversaries.*

This proposition follows from the fact that due to the requirement of detection accuracy, no party outputs $\mathsf{corrupted}_i$ when the adversary is semi-honest. Since $\epsilon \geq 1/\mathrm{poly}(n)$ this implies that the REAL and IDEAL distributions can be distinguished with at most negligible probability, as is required for semi-honest security. We stress that if $\epsilon = 0$ (or is negligible) then the definition of covert adversaries requires nothing, and so the proposition does not hold for this case.

We conclude that, as one may expect, security in the presence of covert adversaries with ϵ-deterrent lies in between security in the presence of malicious adversaries and security in the presence of semi-honest adversaries.

Strong explicit cheat formulation and the malicious model. The following proposition shows that the strong explicit cheat formulation converges to the malicious model as ϵ approaches 1.

Proposition 9 *Let π be a protocol. Then π securely computes some functionality f in the presence of covert adversaries with $\epsilon = 1$ under Definition 6 if and only if it securely computes f with abort in the presence of malicious adversaries.*

This is true since, by definition, either the adversary does not attempt cheating, in which case the ideal execution is the same as in the regular ideal model, or it attempts cheating, in which case it is caught with probability 1 and the protocol is aborted. In both cases, the adversary gains no advantage, and the outcome can be simulated in the standard ideal model. (There is one technicality here relating to whether the output of an honest party due to an abort is $\perp$, or $\mathsf{abort}/\mathsf{corrupted}$. In order for the proposition to go through, we actually have to modify the basic ideal model so that abort_i is received rather than $\perp$.)

3.6 Modular Sequential Composition

Sequential composition theorems for secure computation are important for two reasons. First, they constitute a security goal within themselves. Second, they are useful tools that help in writing proofs of security. As such, we believe that when presenting a new definition, it is of great importance to also prove an appropriate composition theorem for that definition. In our case, we obtain composition theorems that are analogous to that of [4] for all three of our definitions. The exact formulation of these theorems and the proofs appear in the full version.

4 Secure Two-Party Computation

In this section, we show how to securely compute any two-party functionality in the presence of covert adversaries. We have three different protocols, one for each of the three different security definitions. We first present the protocol for the strong explicit cheat formulation, which provides $\epsilon = 1/2$-deterrent. The variations for the other models are minor and will be presented later. In all cases, the deterrent can be boosted to $1 - 1/p(n)$ for any polynomial $p(\cdot)$, with an additional price in complexity, as will be explained later.

The protocol is based on Yao's protocol for semi-honest adversaries [23]. We will base our description on the write-up of [18] of this protocol, and due to lack of space we will assume familiarity with it. The protocol uses an oblivious transfer (OT) protocol that is secure in the presences of covert adversaries. In the full version, we prove the following theorem (via a highly efficient protocol):

Theorem 10 *Assume the existence of semantically secure homomorphic encryption schemes with errorless decryption. Then, for any* $k = \mathrm{poly}(n)$ *there exists a secure protocol for computing the parallel* string *oblivious transfer functionality* $((x_1^0, x_1^1), \ldots, (x_n^0, x_1^n), (\sigma_1, \ldots, \sigma_n)) \mapsto (\lambda, (x_1^{\sigma_1}, \ldots, x_n^{\sigma_n}))$ *in the presence of covert adversaries with* ϵ*-deterrent for* $\epsilon = 1 - \frac{1}{k}$*, under any of the three security definitions.*

4.1 The Protocol

The original protocol of Yao is not secure when the parties may be malicious. Intuitively, there are two main reasons for this. First, the circuit constructor P_1 may send P_2 a garbled circuit that computes a completely different function. Second, the oblivious transfer protocol that is used when the parties can be malicious must be secure for this case. The latter problem is solved here by using the protocol guaranteed by Theorem 10. The first problem is solved by having P_1 send P_2 two garbled circuits. Then, P_2 asks P_1 to open one of the circuits at random, in order to check that it is correctly constructed. (This takes place before P_1 sends the keys corresponding to its input, so nothing is revealed by opening one of the circuits.) The protocol then proceeds similarly to the semi-honest case. The main point here is that if the unopened circuit is correct, then this will constitute a secure execution that can be simulated. However, if it is not correct, then with probability 1/2 party P_1 will have been caught cheating and so P_2 will output $\mathsf{corrupted}_1$. While the above intuition forms the basis for our protocol, the actual construction of the appropriate simulator is somewhat delicate, and requires a careful construction of the protocol. We note some of these subtleties hereunder.

First, it is crucial that the oblivious transfers are run before the garbled circuit is sent by P_1 to P_2. This is due to the fact that the simulator sends a corrupted P_2 a fake garbled circuit that evaluates to the exact output received from the trusted party (and only this output), as described in [18]. However, in order for the simulator to receive the output from the trusted party, it must first send it the input used by the corrupted P_2. This is achieved by first running the oblivious transfers, from which the simulator is able to extract the corrupted P_2's input.

The second subtlety relates to an issue we believe may be a problem for many other implementations of Yao that use cut-and-choose. The problem is that the adversary can construct (at least in theory) a garbled circuit with two sets of keys, where one set of keys decrypt the circuit to the specified one and another set of keys decrypt the circuit to an incorrect one. This is a problem because the adversary can supply "correct keys" to the circuits that are opened and "incorrect keys" to the circuit (or circuits) that are computed. Such a strategy cannot be carried out without risk of detection for the keys that are associated with P_2's input because these keys are obtained by P_2 in the oblivious transfers *before* the garbled circuits are even sent (thus if incorrect keys are sent for one of the circuits, P_2 will detect this if that circuit is opened). However, it is possible for a corrupt P_1 to carry out this strategy for the input wires associated with its own input. We prevent this by having P_1 commit to these keys and send the commitments together with the garbled circuits. Then, instead of P_1 just sending the keys associated with its input, it sends the appropriate decommitments.

A third subtlety that arises is connected to the difference between Definitions 3 and 5 (where the latter is the stronger definition where the decision by the adversary to cheat is not allowed to depend on the honest parties' inputs or on the output). Consider a corrupted P_1 that behaves exactly like an honest P_1 except that in the oblivious transfers, it inputs an invalid key in the place of the key associated with 0 as the first bit of P_2. The result is that if the first bit of P_2's input is 1, then the protocol succeeds and no problem arises. However, if the first bit of P_2's input is 0, then the protocol will always fail and P_2 will always detect cheating. Thus, P_1's decision to cheat may depend on P_2's private input, something that is impossible in the ideal models of Definitions 5 and 6. In summary, this means that the protocol achieves Definition 3 (with $\epsilon = 1/2$) but not Definition 5. In order to solve this problem, we use a circuit that computes the function $g(x_1, x_2^1, \ldots, x_2^n) = f(x_1, \oplus_{i=1}^n x_2^i)$, instead of a circuit that directly computes f. Then, upon input x_2, party P_2 chooses random $x_2^1, \ldots, x_2^{n-1}$ and sets $x_2^n = (\oplus_{i=1}^{n-1} x_2^i) \oplus x_2$. This makes no difference to the result because $\oplus_{i=1}^n x_2^i = x_2$ and so $g(x_1, x_2^1, \ldots, x_2^n) = f(x_1, x_2)$. However, this modification makes every bit of P_2's input uniform when considering any proper subset of $x_2^1, \ldots, x_2^n$. This helps because as long as P_1 does not provide invalid keys for all n shares of x_2, the probability of failure is independent of P_2's actual input (because any set of $n-1$ shares is independent of x_2). If, on the other hand, P_2 attempts to provide invalid keys for all the n shares, then it is caught with probability almost 1. This method was previously used in [19]. We are now ready to describe the actual protocol.

Protocol 11 (two-party computation of a function f):

- **Inputs:** *Party P_1 has input x_1 and party P_2 has input x_2, where $|x_1| = |x_2|$. In addition, both parties have a security parameter n. For simplicity, we will assume that the lengths of the inputs are n.*

- **Auxiliary input:** *Both parties have the description of a circuit C for inputs of length n that computes the function f. The input wires associated with x_1 are $w_1, \ldots, w_n$ and the input wires associated with x_2 are $w_{n+1}, \ldots, w_{2n}$.*
- **The protocol:**
  1. *Parties P_1 and P_2 define a new circuit C' that receives $n+1$ inputs $x_1, x_2^1, , \ldots, x_2^n$ each of length n, and computes the function $f(x_1, \oplus_{i=1}^n x_2^i)$. Note that C' has n^2+n input wires. Denote the input wires associated with x_1 by $w_1, \ldots, w_n$, and the input wires associated with x_2^i by $w_{in+1}, \ldots, w_{(i+1)n}$, for $i = 1, \ldots, n$.*
  2. *Party P_2 chooses $n-1$ random strings $x_2^1, \ldots, x_2^{n-1} \in_R \{0,1\}^n$ and defines $x_2^n = (\oplus_{i=1}^{n-1} x_2^i) \oplus x_2$, where x_2 is P_2's original input (note that $\oplus_{i=1}^n x_2^i = x_2$). The value $z_2 \stackrel{\text{def}}{=} x_2^1, \ldots, x_2^n$ serves as P_2's new input of length n^2 to C'.*
  3. *Party P_1 chooses two sets of $2n^2$ random keys by running $G(1^n)$, the key generator for the encryption scheme:*
  $$\begin{array}{ll} \hat{k}_{n+1}^0, \ldots, \hat{k}_{n^2+n}^0 & \tilde{k}_{n+1}^0, \ldots, \tilde{k}_{n^2+n}^0 \\ \hat{k}_{n+1}^1, \ldots, \hat{k}_{n^2+n}^1 & \tilde{k}_{n+1}^1, \ldots, \tilde{k}_{n^2+n}^1 \end{array}$$
  4. *P_1 and P_2 run n^2 executions of an oblivious transfer protocol, as follows. In the i^{th} execution, party P_1 inputs the pair $\left([\hat{k}_{n+i}^0, \tilde{k}_{n+i}^0], [\hat{k}_{n+i}^1, \tilde{k}_{n+i}^1]\right)$ and party P_2 inputs the bit z_2^i. (Note, P_2 receives for output the keys $\hat{k}_{n+i}^{z_2^i}$ and $\tilde{k}_{n+i}^{z_2^i}$.) The executions are run using a parallel oblivious transfer functionality, as in Theorem 10. If a party receives a $\mathsf{corrupted}_i$ or abort_i message as output from the oblivious transfer, it outputs it and halts.*
  5. *Party P_1 constructs two garbled circuits $G(C')_0$ and $G(C')_1$ using independent randomness. The keys to the input wires $w_{n+1}, \ldots, w_{n^2+n}$ in the garbled circuits are taken from above (i.e., in $G(C')_0$ they are $\hat{k}_{n+1}^0, \hat{k}_{n+1}^1, \ldots, \hat{k}_{n^2+n}^0, \hat{k}_{n^2+n}^1$, and in $G(C')_1$ they are $\tilde{k}_{n+1}^0, \tilde{k}_{n+1}^1, \ldots, \tilde{k}_{n^2+n}^0, \tilde{k}_{n^2+n}^1$). Let $\hat{k}_1^0, \hat{k}_1^1, \ldots, \hat{k}_n^0, \hat{k}_n^1$ be the keys associated with P_1's input in $G(C')_0$ and $\tilde{k}_1^0, \tilde{k}_1^1, \ldots, \tilde{k}_n^0, \tilde{k}_n^1$ the analogous keys in $G(C')_1$. Then, for every $i \in \{1, \ldots, n\}$ and $b \in \{0,1\}$, party P_1 computes $\hat{c}_i^b = \mathsf{Com}(\hat{k}_i^b; \hat{r}_i^b)$ and $\tilde{c}_i^b = \mathsf{Com}(\tilde{k}_i^b; \tilde{r}_i^b)$, where Com is a perfectly-binding commitment scheme and $\mathsf{Com}(x; r)$ denotes a commitment to x using randomness r. P_1 sends the garbled circuits to P_2 together with all of the above commitments. The commitments are sent as two vectors of pairs; in the first vector the i^{th} pair is $\{\hat{c}_i^0, \hat{c}_i^1\}$ in a random order, and in the second vector the i^{th} pair is $\{\tilde{c}_i^0, \tilde{c}_i^1\}$ in a random order.*
  6. *Party P_2 chooses a random bit $b \in_R \{0,1\}$ and sends b to P_1.*
  7. *P_1 sends P_2 all of the keys for the inputs wires $w_1, \ldots, w_{n^2+n}$ of the garbled circuit $G(C')_b$, together with the associated mappings and the decommitment values. (I.e. if $b = 0$, then party P_1 sends $(\hat{k}_1^0, 0), (\hat{k}_1^1, 1), \ldots, (\hat{k}_{n^2+n}^0, 0), (\hat{k}_{n^2+n}^1, 1)$ and $\hat{r}_1^0, \hat{r}_1^1, \ldots, \hat{r}_n^0, \hat{r}_n^1$ for the circuit $G(C')_0$.)*

8. *P_2 checks the decommitments to the keys associated with $w_1, \ldots, w_n$, decrypts the entire circuit (using the keys and mappings that it received) and checks that it is exactly the circuit C' derived from the auxiliary input circuit C. In addition, it checks that the keys that it received in the oblivious transfers match the correct keys that it received in the opening (i.e., if it received $(\hat{k}, \tilde{k})$ in the i^{th} oblivious transfer, then it checks that $\hat{k} = \hat{k}_{n+i}^{z_2^i}$ if $G(C')_0$ was opened, and $\tilde{k} = \tilde{k}_{n+i}^{z_2^i}$ if $G(C')_1$ was opened). If all the checks pass, it proceeds to the next step. If not, it outputs* $\mathsf{corrupted}_1$ *and halts. In addition, if P_2 does not receive this message at all, it outputs* $\mathsf{corrupted}_1$.
9. *P_1 sends decommitments to the input keys associated with its input for the unopened circuit. That is, if $b = 0$, then P_1 sends P_2 the keys and decommitment values $(\tilde{k}_1^{x_1^1}, \tilde{r}_1^{x_1^1}), \ldots, (\tilde{k}_n^{x_1^n}, \tilde{r}_n^{x_1^n})$ to P_2. Otherwise, if $b{=}1$, then P_2 sends the keys $(\hat{k}_1^{x_1^1}, \hat{r}_1^{x_1^1}), \ldots, (\hat{k}_n^{x_1^n}, \hat{r}_n^{x_1^n})$.*
10. *P_2 checks that the values received are valid decommitments to the commitments received above. If not, it outputs* abort_1. *If yes, it uses the keys to compute $C'(x_1, z_2) = C'(x_1, x_2^1, \ldots, x_2^n) = C(x_1, x_2)$, and outputs the result. If the keys are not correct (and so it is not possible to compute the circuit), or if P_2 doesn't receive this message at all, it outputs* abort_1.

Note that steps 7–10 are actually a single step of P_1 sending a message to P_2, followed by P_2 carrying out a computation.

If any party fails to receive a message as expected during the execution, it outputs abort_i *(where P_i is the party who failed to send the message). This holds unless the party is explicitly instructed above to output* $\mathsf{corrupted}$ *instead (as in Step 8).*

We have the following theorem:

Theorem 12 *Let f be any probabilistic polynomial-time function. Assume that the encryption scheme used to generate the garbled circuits has indistinguishable encryptions under chosen-plaintext attacks (and has an elusive and efficiently verifiable range), and that the oblivious transfer protocol used is secure in the presence of covert adversaries with 1/2-deterrent by Definition 6. Then, Protocol 11 securely computes f in the presence of covert adversaries with 1/2-deterrent by Definition 6.*

The full proof of this theorem can be found in the full version.

4.2 Protocols for the Other Security Definitions

We present more efficient protocols for the two other security formulations (versions 1 and 2) which are more efficient. The protocols are essentially identical to the one described above, with the only difference being the number of shares used to split the inputs of P_2 in step 2:

– For the *failed-simulation formulation* (Version 1), we do not split the input of P_2 at all and use the original inputs (i.e., the original circuit C is used). This reduces the number of oblivious transfers from n^2 to n. The revised protocol provides security for covert adversaries in the failed simulation formulation with deterrence $1/2$.
– For the *explicit cheat formulation* (not strong) (Version 2), we split the input of P_2 into 2 shares, instead of n. Note again that this reduces the number of oblivious transfers from n^2 to $2n$. The revised protocol provides security for covert adversaries in the explicit cheat formulation with deterrence $1/4$.

4.3 Higher Deterrence Values

For all three versions, it is possible to boost the deterrence value to $1-1/\text{poly}(n)$, with an increased price in performance. Let $p(\cdot)$ be a polynomial. Then, Protocol 11 can be modified so that a deterrent of $1 - 1/p(n)$ is obtained, as follows. First, we use an oblivious transfer protocol that is secure in the presence of covert adversaries with deterrent $\epsilon = 1 - 1/p(n)$. Then, Protocol 11 is modified by having P_1 send $p(n)$ garbled circuits to P_2 and then P_2 randomly asking P_1 to open all circuits except one. Note that when doing so it is not necessary to increase the number of oblivious transfers, because the same oblivious transfer can be used for all circuits. This is important since the number of oblivious transfers is a dominant factor in the complexity. The modification yields a deterrent $\epsilon = 1 - 1/p(n)$ and thus can be used to obtain a high deterrent factor. For example, using 10 circuits the deterrence is $9/10$.

4.4 Non-Halting Detection Accuracy

It is possible to modify Protocol 11 so that it achieves *non-halting detection accuracy*; see Definition 4. Before describing how we do this, notice that the reason that we need to recognize a halting-abort as cheating in Protocol 11 is that if P_1 generates one faulty circuit, then it can always just refuse to continue (i.e., abort) in the case that P_2 asks it to open the faulty circuit. This means that if aborting is not considered cheating, then a corrupted P_1 can form a strategy whereby it is never detected cheating, but succeeds in actually cheating with probability $1/2$. In order to solve this problem, we construct a method whereby P_1 does not know if it will be caught or not. We do so by having P_2 receive the circuit opening via a fully secure oblivious transfer protocol, rather than having P_1 send it explicitly. This forces P_1 to either abort before learning anything, or to risk being caught with probability $1/2$. The details are provided in the full version. The price of this modification is that of one additional fully secure oblivious transfer and the replacement of all of the original oblivious transfer protocols with fully secure ones. (Of course, we could use an oblivious transfer protocol that is secure in the presence of covert adversaries with non-halting detection accuracy, but we do not know how to construct one.) Since fully-secure oblivious transfer is expensive, this is a considerable overhead.

References

1. W. Aiello, Y. Ishai and O. Reingold. Priced Oblivious Transfer: How to Sell Digital Goods. In *EUROCRYPT 2001*, Springer-Verlag (LNCS 2045), pages 119–135, 2001.
2. D. Beaver. Foundations of Secure Interactive Computing. In *CRYPTO'91*, Springer-Verlag (LNCS 576), pages 377–391, 1991.
3. M. Ben-Or, S. Goldwasser and A. Wigderson. Completeness Theorems for Non-Cryptographic Fault-Tolerant Distributed Computation. In 20*th STOC,* pages 1–10, 1988.
4. R. Canetti. Security and Composition of Multiparty Cryptographic Protocols. *Journal of Cryptology*, 13(1):143–202, 2000.
5. R. Canetti and R. Ostrovsky. Secure Computation with Honest-Looking Parties: What If Nobody Is Truly Honest? In 31*st STOC*, pages 255–264, 1999.
6. D. Chaum, C. Crépeau and I. Damgard. Multi-party Unconditionally Secure Protocols. In 20*th STOC*, pages 11–19, 1988.
7. B. Chor, O. Goldreich, E. Kushilevitz and M. Sudan. Private Information Retrieval. *Journal of the ACM,* 45(6):965–981, 1998.
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. M.K. Franklin and M. Yung. Communication Complexity of Secure Computation. In 24*th STOC*, 699–710, 1992.
10. O. Goldreich. *Foundations of Cryptography: Volume 2 – Basic Applications.* Cambridge University Press, 2004.
11. O. Goldreich and Y. Lindell. Session-Key Generation using Human Passwords Only. *Journal of Cryptology,* 19(3):241–340, 2006.
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.
13. S. Goldwasser and L. Levin. Fair Computation of General Functions in Presence of Immoral Majority. In *CRYPTO'90,* Springer-Verlag (LNCS 537), 77–93, 1990.
14. S. Goldwasser and Y. Lindell. Secure Computation Without Agreement. *Journal of Cryptology,* 18(3):247–287, 2005.
15. Y. Ishai, J. Kilian, K. Nissim and E. Petrank. Extending Oblivious Transfers Efficiently. In *CRYPTO 2003,* Springer-Verlag (LNCS 2729), pp. 145–161, 2003
16. Y. Ishai, E. Kushilevitz, Y. Lindell and E. Petrank. Black-Box Constructions for Secure Computation. In 38*th STOC,* pages 99–108, 2006.
17. Y.T. Kalai. Smooth Projective Hashing and Two-Message Oblivious Transfer. In *EUROCRYPT 2005*, Springer-Verlag (LNCS 3494) pages 78–95, 2005.
18. Y. Lindell and B. Pinkas. A Proof of Yao's Protocol for Secure Two-Party Computation. To appear in the *Journal of Cryptology. Cryptology ePrint Archive,* Report 2004/175, 2004.
19. Y. Lindell and B. Pinkas. An Efficient Protocol for Secure Two-Party Computation in the Presence of Malicious Adversaries. Manuscript, 2006.
20. D. Malkhi, N. Nisan, B. Pinkas and Y. Sella. Fairplay – A Secure Two-Party Computation System. In the 13*th USENIX*, pages 287–302, 2004.
21. S. Micali and P. Rogaway. Secure Computation. Unpublished manuscript, 1992.
22. M. Rabin. How to Exchange Secrets by Oblivious Transfer. Tech. Memo TR-81, Aiken Computation Laboratory, Harvard U., 1981.
23. A. Yao. How to Generate and Exchange Secrets. In 27*th FOCS*, pages 162–167, 1986.