Security Against Covert Adversaries: Efficient Protocols for Realistic Adversaries*

Yonatan Aumann† Yehuda Lindell†

February 18, 2007

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.

*An extended abstract of this work appeared in the 4th *Theory of Cryptography Conference* (TCC), 2007.

†Department of Computer Science, Bar-Ilan University, Israel. email: {aumann,lindell}@cs.biu.ac.il. Work supported in part by an Infrastructures grant from the Ministry of Science, Israel.

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 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 the *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 adversarial behavior 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 by 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 by presenting a generic protocol for secure two-party computation in our model 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 (this follows the folklore cut-and-choose methodology for boosting the security of Yao's protocol for adversaries that may not be semi-honest). 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 yield $\epsilon = 9/10$.)

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 Preliminaries and Standard Definitions

2.1 Preliminaries

A function $\mu(\cdot)$ is negligible in n, or just negligible, if for every positive polynomial $p(\cdot)$ and all sufficiently large n's it holds that $\mu(n) < 1/p(n)$. A probability ensemble $X = \{X(a,n)\}_{a\in\{0,1\}^*;n\in\mathbb{N}}$ is an infinite sequence of random variables indexed by a and $n \in \mathbb{N}$. (The value a will represent the parties' inputs and n the security parameter.) Two distribution ensembles $X = \{X(a,n)\}_{n\in\mathbb{N}}$ and $Y = \{Y(a,n)\}_{n\in\mathbb{N}}$ are said to be computationally indistinguishable, denoted $X \stackrel{c}{\equiv} Y$, if for every non-uniform polynomial-time algorithm D there exists a negligible function $\mu(\cdot)$ such that for every $a \in \{0,1\}^*$,

$$|\Pr[D(X(a,n)) = 1] - \Pr[D(Y(a,n)) = 1]| \leq \mu(n)$$

All parties are assumed to run in time that is polynomial in the security parameter. (Formally, each party has a security parameter tape upon which that value 1^n is written. Then the party is polynomial in the input on this tape.)

2.2 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).

Adversarial behavior. Loosely speaking, the aim of a secure multiparty protocol is to protect honest parties against dishonest behavior by other parties. In this section, we present the definition for *malicious adversaries* who control some subset of the parties and may instruct them to arbitrarily deviate from the specified protocol. We also consider *static corruptions*, meaning that the set of corrupted parties is fixed at the onset.

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. 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. One technical detail that arises when considering the setting of no honest majority is that it is impossible to achieve fairness or guaranteed output delivery. That is, it is possible for the adversary to prevent the honest parties from receiving outputs. Furthermore, it may even be possible for the adversary to receive output while the honest parties do not. We consider malicious adversaries and static corruptions in all of our definitions in this paper.

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. 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.

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 (by replacing the input x_i with a special abort_i message), 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 receives an input of the form abort_i for some $i \in I$, it sends abort_i to all parties 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 $\mathsf{continue}$ or abort_i to the trusted party (for some $i \in I$). If it sends $\mathsf{continue}$, 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 abort_i, the trusted party sends abort_i 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.

This ideal model is different from that of [10] in that in the case of an "abort", the honest parties output abort_i and not a $\perp$ symbol. This means that the honest parties *know* the identity of the corrupted party who causes the abort. This is achieved by most multiparty protocols, including that of [12], but not all (e.g., the protocol of [14] does not meet this requirement).

Let $f : (\{0,1\}^*)^m \rightarrow (\{0,1\}^*)^m$ 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 $\mathsf{ideal\ execution\ of}$ 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 $\mathsf{real\ execution\ of}$ 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. Loosely speaking, the definition asserts that a secure party protocol (in the real model) emulates the ideal model (in which a trusted party exists). This is formulated by saying that adversaries in the ideal model are able to simulate executions of the real-model protocol. 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 2.1 (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 $z \in \{0,1\}^*$:*

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

We note that the above definition assumes that the parties (and adversary) know the input lengths (this can be seen from the requirement that $\overline{x}$ is balanced and so all the inputs in the vector of inputs are of the same length).[1] We remark that some restriction on the input lengths is unavoidable, see [10, Section 7.1] for discussion.

2.3 Functionalities that Provide Output to a Single Party

In the standard definition of secure computation, both parties receive output and these outputs may be *different*. However, the presentation of our two-party protocol is far simpler if we assume that only party P_2 receives output. We will show now that this suffices for the general case. That is, we claim that any protocol that can be used to 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 $f_1(x_1, x_2)$ and party P_2 receives $f_2(x_1, x_2)$. For simplicity, we will assume that the length of the output of $f_1(x_1, x_2)$ is at most n, where n is the security parameter. This can be achieved by simply taking n to be larger in case it is necessary.

Let $f = (f_1, f_2)$ be a functionality. We wish to construct a secure protocol in which P_1 receives $f_1(x_1, x_2)$ and P_2 receives $f_2(x_1, x_2)$. As a building block we use a protocol for computing any efficient functionality with the limitation that only P_2 receives output. Let $r, a, b \in_R \{0,1\}^n$ be randomly chosen strings. Then, in addition to x_1, party P_1's input includes the elements r, a and b. Furthermore, define a functionality g (that has only a single output) as follows:

$$g((r, a, b, x_1), x_2) = (\alpha, \beta, f_2(x_1, x_2))$$

where $\alpha = r + f_1(x_1, x_2)$, $\beta = a \cdot \alpha + b$, and the arithmetic operations are defined over $GF[2^n]$. 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 $\alpha - r$, and otherwise it outputs abort_2.

[1] In the case that no parties are corrupted, we assume that the adversary receives the length of the inputs as part of its auxiliary input z.

It is easy to see that P_2 learns nothing about P_1's output $f_1(x_1, x_2)$, and that it cannot alter the output that P_1 will receive (beyond causing it to abort), except with probability 2^{-n}. We remark that it is also straightforward to construct a simulator for the above protocol. Applying the composition theorem of [4] (for standard security) or Theorems 4.1 and 4.2 (for covert adversaries – to be defined below), we have the following proposition:

Proposition 2.2 *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. This holds also for security in the presence of covert adversaries for any of Definitions 3.2, 3.4 and 3.5.*

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 as is the case for our protocol below).

3 Definitions – Secure Computation with Covert Adversaries

3.1 Motivation

The standard definition of security (see Definition 2.1) is such that all possible (polynomial-time) adversarial behavior is simulatable. Here, in contrast, we wish to model the situation that parties may *successfully* 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).[2] A number of problems arise with this definition.

- 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 can detect this in any specific execution. Consider for example a coin-tossing protocol where the honest distribution gives even probabilities to 0 and 1, while the adversary gives double the probability to the 1 outcome. Clearly, the distributions differ. However, in any given execution, even an outcome of 1 does not provide the honest players with sufficient evidence of any wrong-doing. Thus, it is not sufficient that the *distributions* differ. Rather, one needs to be able to detect cheating in each adversarial execution.

- The fact that the distributions differ does not necessarily imply that the honest parties have an efficient distinguisher. Furthermore, in order to guarantee that the honest parties detect the cheating, they would have 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).

[2] 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.

- 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 risk being caught in one run in a million.

Second attempt. To solve the aforementioned problems, we first 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). But $\epsilon = 1/2$ 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 2.1), 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. We stress that in the when the adversary chooses to cheat, it 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 doing so.

As it turns out, the above intuition can be formalized in three different ways, which form a hierarchy of security guarantees. In practice, the implementor 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, and 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 3.1 *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 3.2 (covert security – failed simulation formulation): *Let f and π be as in Definition 2.1, 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,*

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

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 3.3 *A protocol π is* non-halting detection accurate *if it is detection accurate as in Definition 3.1 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 may be 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 3.2 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 that are achieved more explicit. Second, it makes it easy to prove a sequential composition theorem (see below).

We modify the ideal model in the following way. Let $\epsilon : \mathbb{N} \rightarrow [0,1]$ 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 2.1, 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.[3] 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 3.4 (covert security – explicit cheat formulation): *Let f, π and ϵ be as in Definition 3.2. Protocol π is said to* $\mathsf{securely\ compute}$ f $\mathsf{in\ the\ presence\ of\ covert\ adversaries\ with}$ ϵ-$\mathsf{deterrent}$ *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 $z \in \{0,1\}^*$:*

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

Definition 3.4 and detection accuracy. We note that in Definition 3.4 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 3.3), then this should be explicitly added to the definition.

[3] 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 3.5 (covert security – strong explicit cheat formulation): *Let f, π and ϵ be as in Definition 3.2. 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, any potentially successful cheating attempt is 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. 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.

Proposition 3.6 *Let π be a protocol that securely computes some functionality f in the presence of covert adversaries with ϵ-deterrent by Definition 3.4. Then, π securely computes f in the presence of covert adversaries with ϵ-deterrent by Definition 3.2.*

Proof: Let f, π and ϵ be as in the proposition. Then, we first claim that π is detection accurate. This is due to the fact that in the ideal model of Definition 3.4, honest parties only output $\mathsf{corrupted}_i$ for $i \in I$. Therefore, this must hold also in the real model, except with negligible probability (as required by Definition 3.1). Now, let $\mathcal{A}$ be an adversary and let $\mathcal{S}$ be the simulator that is guaranteed to exist for IDEALC by Definition 3.4. We claim that the simulator $\mathcal{S}$ also works for Definition 3.2. In order to see this, let Δ be the probability that $\mathcal{S}$ sends $\mathsf{corrupted}_i$ or cheat_i for input for some $i \in I$ (this probability depends only on $\mathcal{A}$, the corrupted parties' inputs and the auxiliary input z). Now, when $\mathcal{S}$ sends input $\mathsf{corrupted}_i$, the honest parties all output $\mathsf{corrupted}_i$ with probability 1. In addition, when $\mathcal{S}$ sends input cheat_i, the honest parties all output $\mathsf{corrupted}_i$ with probability ϵ in the ideal model. It follows that the honest parties output $\mathsf{corrupted}_i$ with probability *at least* $\epsilon \cdot \Delta$. It remains, therefore, to show that the IDEAL and REAL distributions can be distinguished with probability at most Δ (because then the probability that the adversary is caught cheating is

at least ϵ times the maximum distinguishing "gap" between the IDEAL and REAL distributions). However, this follows immediately from the fact that if $\mathcal{S}$ does *not* send any input of the form $\mathsf{corrupted}_i$ or cheat_i, then the ideal execution is the same as in the standard definitions (and so the same as in Definition 3.2). Thus, in the event that $\mathcal{S}$ does not send $\mathsf{corrupted}_i$ or cheat_i, the IDEAL and REAL of Definition 3.2 are computationally indistinguishable. Since $\mathcal{S}$ sends $\mathsf{corrupted}_i$ or cheat_i with probability Δ, we obtain that the IDEAL distribution can be distinguished from the REAL one with probability at most $\Delta + \mu(n)$ as desired. ■

The following proposition is straightforward and is therefore stated without a proof:

Proposition 3.7 *Let π be a protocol that securely computes some functionality f in the presence of covert adversaries with ϵ-deterrent by Definition 3.5. Then, π securely computes f in the presence of covert adversaries with ϵ-deterrent by Definition 3.4.*

So far we have shown that our three definitions form a hierarchy. The fact that the hierarchy is *strict* is not difficult to show. In fact, the protocols that we present in Sections 6 (Protocol 6.1 and its extensions in Section 6.2) demonstrate that the hierarchy is strict.

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 3.8 *Let π be a protocol that securely computes some functionality f with abort in the presence of malicious adversaries, as in Definition 2.1. Then, π securely computes f in the presence of covert adversaries with ϵ-deterrent, for any of the three formulations (Definitions 3.2, 3.4, and 3.5) and for every $0 \leq \epsilon \leq 1$.*

This proposition follows from the simple observation that according to Definition 2.1, there exists a simulator that always succeeds in its simulation. Thus, Definition 3.2 holds even if the probability of detecting cheating is 0. Likewise, for Definitions 3.4 and 3.5 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 3.9 *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 $\epsilon \geq 1/\text{poly}(n)$. 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 $\epsilon \geq 1/\text{poly}(n)$ this implies that the REAL and IDEAL distributions can be distinguished with at most negligible probability, as required. 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. If $1/\text{poly}(n) \leq \epsilon \leq 1 - 1/\text{poly}(n)$ then it can be shown that the definition of security for covert adversaries is strictly different to the semi-honest and malicious models. We remark that for Definitions 3.2 and 3.4 this holds for any $\epsilon \geq 1/\text{poly}(n)$. For Definition 3.5 and the case of $\epsilon = 1 - \mu(n)$, see below.

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 3.10 *Let π be a protocol and μ a negligible function. Then π securely computes some functionality f in the presence of covert adversaries with $\epsilon(n) = 1 - \mu(n)$ under Definition 3.5 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 that is negligibly close to 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. We stress that Proposition 3.10 does not hold for Definitions 3.2 and 3.4 because in these definitions the adversary may learn the honest parties' private inputs even when it is caught (something that is not allowed in the malicious model).

4 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 basic idea behind these composition theorems is that it is possible to design a protocol that uses an ideal functionality as a subroutine, and then analyze the security of the protocol when a trusted party computes this functionality. For example, assume that a protocol is constructed that uses oblivious transfer as a subroutine. Then, first we construct a protocol for oblivious transfer and prove its security. Next, we prove the security of the protocol that uses oblivious transfer as a subroutine, in a model where the parties have access to a trusted party computing the oblivious transfer functionality. The composition theorem then states that when the "ideal calls" to the trusted party for the oblivious transfer functionality are replaced by real executions of a secure protocol computing this functionality, the protocol remains secure.

The f-hybrid model. We consider a *hybrid model* where parties both interact with each other (as in the real model) and use trusted help (as in the ideal model). Specifically, the parties run a protocol π that contains "ideal calls" to a trusted party computing a functionality f. These ideal calls are just instructions to send an input to the trusted party. Upon receiving the output back from the trusted party, the protocol π continues. We stress that honest parties do not send messages from π between the time that they send input to the trusted party and the time that they receive back output (this is because we consider *sequential composition* here). Of course, the trusted party may be used a number of times throughout the π-execution. However, each time is independent (i.e., the trusted party does not maintain any state between these calls). We call the regular messages of π that are sent amongst the parties standard messages and the messages that are sent between parties and the trusted party ideal messages.

Let f be a functionality and let π be an m-party protocol that uses ideal calls to a trusted party 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 f-hybrid execution of π on inputs $\overline{x}$, auxiliary input z to $\mathcal{A}$ and security parameter n, denoted $\text{HYBRID}^{f}_{\pi,\mathcal{A}(z),I}(\overline{x})$, is defined as the output vector of the honest parties and the adversary $\mathcal{A}$ from the hybrid execution of π with a trusted party computing f.

Sequential modular composition. Let f and π be as above, and let ρ be a protocol. Consider the real protocol π^{ρ} that is defined as follows. All standard messages of π are unchanged. When a

party P_i is instructed to send an ideal message x to the trusted party, it begins a real execution of ρ with input x instead. When this execution of ρ concludes with output y, party P_i continues with π as if y was the output received by the trusted party (i.e. as if it were running in the f-hybrid model). A special case of the composition theorem of [4] for malicious adversaries states that if ρ securely computes f, and π securely computes some functionality g in the f-hybrid model, then π^ρ securely computes g (in the real model). Here, we prove an analogous theorem for covert adversaries with ϵ-deterrent. Since our protocols here are for the two-party case, we prove the theorem only for this special case. We also consider the simplified scenario where π contains only a single call to f; the more general case can be proven in a similar way (with the addition of a standard hybrid argument). Finally, we assume that the lengths of the inputs to ρ can be derived given the input to π and the security parameter. All of these assumptions/simplifications are true for our protocols in this paper, and so suffice. A more general theorem can be derived in a straightforward manner.

We prove sequential modular composition theorems for all of our definitions of security in the presence of covert adversaries. The proof for Definition 3.2 is based on the ideas in the composition theorem of [4] while making necessary changes due to the difference in the models. The proofs for Definitions 3.4 and 3.5 are an almost direct corollary of the theorem of [4] (after casting the models of Definitions 3.4 and 3.5 in a different, yet equivalent, model). We present all theorems because, to the best of our knowledge, none can be used to derive another.

4.1 Composition for Definition 3.2

In this section we prove a modular sequential composition theorem for the (weaker) Definition 3.2.

Theorem 4.1 *Let f be a two-party probabilistic polynomial-time functionality and let ρ be a protocol that securely computes f in the presence of covert adversaries with ϵ_1-deterrent. Let g be a two-party functionality and let π be a protocol that securely computes g in the f-hybrid model (using a single call to f) in the presence of covert adversaries with ϵ_2-deterrent. Then, π^ρ securely computes g in the presence of covert adversaries with ϵ-deterrent, where $\epsilon = \mathsf{min}\{\epsilon_1, \epsilon_2\}$. The above all refer to Definition 3.2.*

Proof Sketch: By the assumption in the theorem, we have that for every non-uniform probabilistic polynomial-time adversary $\mathcal{A}_\rho$ attacking ρ in the real model, there exists a non-uniform probabilistic polynomial-time adversary $\mathcal{S}_\rho$ for the ideal model with f such that for every $I \subseteq [2]$, every balanced vector $\overline{x} \in (\{0,1\}^*)^2$, every $z \in \{0,1\}^*$ and every non-uniform polynomial-time D, there exists a negligible function μ such that:

$$\Pr\left[\exists i \in I \ \forall j \notin I \ : \ \text{OUTPUT}_j(\text{REAL}_{\rho,\mathcal{A}_\rho(z),I}(\overline{x},n)) = \mathsf{corrupted}_i\right]$$
$$\geq \epsilon_1(n) \cdot \left|\Pr[D(\text{IDEAL}_{f,\mathcal{S}_\rho(z),I}(\overline{x},n)) = 1] - \Pr[D(\text{REAL}_{\rho,\mathcal{A}_\rho(z),I}(\overline{x},n)) = 1]\right| - \mu(n) \quad (1)$$

Furthermore, for every non-uniform probabilistic polynomial-time adversary $\mathcal{A}_\pi$ attacking π in the f-hybrid model, there exists a non-uniform probabilistic polynomial-time adversary $\mathcal{S}_\pi$ for the ideal model with g such that for every $I \subseteq [2]$, every balanced vector $\overline{x} \in (\{0,1\}^*)^2$, every $z \in \{0,1\}^*$ and every non-uniform polynomial-time D, there exists a negligible function μ' such that:

$$\Pr\left[\exists i \in I \ \forall j \notin I \ : \ \text{OUTPUT}_j(\text{HYBRID}^f_{\pi,\mathcal{A}_\pi(z),I}(\overline{x},n)) = \mathsf{corrupted}_i\right]$$
$$\geq \epsilon_2(n) \cdot \left|\Pr[D(\text{IDEAL}_{g,\mathcal{S}_\pi(z),I}(\overline{x},n)) = 1] - \Pr[D(\text{HYBRID}^f_{\pi,\mathcal{A}_\pi(z),I}(\overline{x},n)) = 1]\right| - \mu'(n) \quad (2)$$

We need to show that under the above assumptions, the real protocol π^ρ securely computes g in the presence of covert adversaries with ϵ-deterrent, where $\epsilon = \mathsf{min}\{\epsilon_1, \epsilon_2\}$. In the case that $I = \phi$ or $I = \{1, 2\}$, the proof is straightforward (if $I = \phi$ then no parties are corrupted and the proof is like in the semi-honest case; if $I = \{1, 2\}$ then both parties are corrupted and nothing needs to be simulated). We will therefore focus on the case that $I = \{1\}$ or $I = \{2\}$. The proof is the same in both cases and so we will assume that $I = \{1\}$; we take this concrete case because it simplifies notation.

Let $\mathcal{A}$ be an adversary that controls party P_1 and attacks the real protocol π^ρ, and let x_1 be P_1's input and z the auxiliary input for $\mathcal{A}$. We begin by modifying $\mathcal{A}$ to $\mathcal{A}'$ in the following way:

1. Adversary $\mathcal{A}'$ invokes $\mathcal{A}$ on its own input and forwards all π-messages between $\mathcal{A}$ (controlling P_1) and the honest party P_2.

2. When $\mathcal{A}'$ reaches the point in the execution where ρ begins, it defines an adversary $\mathcal{A}_\rho$ who receives auxiliary-input z_ρ that consists of an internal state of a machine, and input 0^k where n is the length of the input to ρ in π. (Recall that we assume that the length of the inputs to ρ, or equivalently to f, are determined from the input to π and the security parameter.) The machine $\mathcal{A}_\rho$ runs the machine $\mathcal{A}$ from the initial state z_ρ and ignores its input entirely (the input is needed for a technicality that machines have input of the appropriate length). In addition, at the end of the execution of ρ, adversary $\mathcal{A}_\rho$ outputs the current state of $\mathcal{A}$.

3. After ρ concludes, adversary $\mathcal{A}'$ takes the state output by $\mathcal{A}_\rho$ and continues running $\mathcal{A}$ from this state by forwarding the messages of π unmodified between $\mathcal{A}$ and the honest P_2.

4. At the end of the execution of π^ρ, adversary $\mathcal{A}'$ outputs whatever $\mathcal{A}$ does.

It is immediate that the output distribution from a real execution of π^ρ with $\mathcal{A}'$ and an honest P_2, is identical to the output distribution of an execution of π^ρ with $\mathcal{A}$ and an honest P_2. It is also clear that $\mathcal{A}_\rho$ is a real adversary for the protocol ρ. Therefore, by the assumption in the theorem, there exists a simulator $\mathcal{S}_\rho$ as described above in Eq. (1).

We now use $\mathcal{A}'$ and $\mathcal{S}_\rho$ to construct an adversary $\mathcal{A}_\pi$ for the f-hybrid execution of π. Adversary $\mathcal{A}_\pi$ receives auxiliary-input z and input x_1 and invokes $\mathcal{A}'$ on these same inputs. However, when $\mathcal{A}'$ reaches the point that it invokes $\mathcal{A}_\rho$ upon auxiliary-input z_ρ, adversary $\mathcal{A}_\pi$ invokes the simulator $\mathcal{S}_\rho$ upon auxiliary-input z_ρ instead. Ideal messages that $\mathcal{S}_\rho$ wishes to send to the trusted party computing f are sent by $\mathcal{A}_\pi$ to its trusted party computing f. Likewise, outputs from the trusted party are handed by $\mathcal{A}_\pi$ back to $\mathcal{S}_\rho$. When $\mathcal{S}_\rho$ concludes its execution, its output is interpreted by $\mathcal{A}_\pi$ as an internal state of $\mathcal{A}$. The adversary $\mathcal{A}_\pi$ then continues by running $\mathcal{A}'$ from this internal state.

We claim that for every polynomial-time distinguisher D and every $\overline{x}$ and z, there exists a negligible function μ such that

$$\begin{aligned}&\Pr\left[\textsc{output}_2^\rho(\textsc{real}_{\pi^\rho,\mathcal{A}(z),I}(\overline{x},n)) = \mathsf{corrupted}_1\right] \\ &\quad \geq \epsilon_1(n) \cdot \left|\Pr[D(\textsc{hybrid}^f_{\pi,\mathcal{A}_\pi(z),I}(\overline{x},n)) = 1] - \Pr[D(\textsc{real}_{\pi^\rho,\mathcal{A}(z),I}(\overline{x},n)) = 1]\right| - \mu(n) \quad (3)\end{aligned}$$

where $\textsc{output}_2^\rho$ refers to the output of P_2 within the subprotocol ρ only (i.e., here we consider the probability that P_2 outputs $\mathsf{corrupted}_1$ within ρ). Eq. (3) holds because otherwise this contradicts the security of ρ. Specifically, the only difference between the real execution with $\mathcal{A}$ and the hybrid execution with $\mathcal{A}_\pi$ is that $\mathcal{A}$ runs $\mathcal{A}_\rho$ in the real execution of ρ whereas $\mathcal{A}_\pi$ runs $\mathcal{S}_\rho$ instead. Thus, if Eq. (3) does not hold, then $\mathcal{S}_\rho$ does not "simulate" for $\mathcal{A}_\rho$ as it should. More formally, if Eq. (3)

does not hold when the probabilities are taken over the random tapes of the parties (and thus the entire execution), then there exists a prefix of the execution up until the point that ρ begins such that Eq. (3) does not hold, even conditioned on this prefix. Such an "execution prefix" is obtained by fixing the portion of the random-tape of P_2 that is used until this point in the protocol and by fixing the random-tape of $\mathcal{A}$. (Of course, these tapes can only be found non-uniformly.) Now, this prefix defines an auxiliary-input z_ρ for $\mathcal{A}_\rho$ in ρ that consists of $\mathcal{A}$'s internal state after this execution prefix. Furthermore, it defines the input x_2 of the honest party P_2 to ρ (the input x_1 of P_1 is implicitly defined by $\mathcal{A}$ and so can be set to an arbitrary value in order to obtain $\overline{x}$). Fix this input vector $\overline{x}$ and auxiliary input z_ρ. Next, we define a distinguisher D' that receives the outputs of the adversary and P_2 and tries to determine if the execution was IDEAL or REAL. D' works by internally simulating the execution until the end: it can do this because it has $\mathcal{A}'$'s state after the execution of ρ and its random-tape, and also has the input and output of P_2 (the initial input of P_2 needed for this internal simulation can be provided to D' as auxiliary input). Then, at the end of the internal simulation, D' applies D to the output of $\mathcal{A}'$ and P_2 and outputs whatever D outputs. It follows that if there exists a polynomial-time distinguisher D for which Eq. (3) does not hold, then there exists a polynomial-time distinguisher D' and inputs $\overline{x}$ and z_ρ such that for every negligible function $\mu(n)$

$$\begin{aligned}&\Pr\left[\text{OUTPUT}_2(\text{REAL}_{\rho,\mathcal{A}_\rho(z_\rho),I}(\overline{x},n)) = \mathsf{corrupted}_1\right] \\ &\quad \leq \epsilon_1(n)\cdot\left|\Pr[D(\text{IDEAL}_{f,\mathcal{S}_\rho(z_\rho),I}(\overline{x},n))=1] - \Pr[D(\text{REAL}_{\rho,\mathcal{A}_\rho(z_\rho),I}(\overline{x},n))=1]\right| - \mu(n)\end{aligned}$$

in contradiction to the security of ρ.

Next, we claim that by Eq. (2) it holds that for every polynomial-time distinguisher D and every $\overline{x}$ and z, there exists a negligible function μ' such that

$$\begin{aligned}&\Pr\left[\text{OUTPUT}_2(\text{HYBRID}^f_{\pi,\mathcal{A}_\pi(z),I}(\overline{x},n)) = \mathsf{corrupted}_1\right] \\ &\quad \geq \epsilon_2(n)\cdot\left|\Pr[D(\text{IDEAL}_{f,\mathcal{S}(z),I}(\overline{x},n))=1] - \Pr[D(\text{HYBRID}^f_{\pi,\mathcal{A}_\pi(z),I}(\overline{x},n))=1]\right| - \mu'(n) \quad (4)\end{aligned}$$

This follows directly from the fact that $\mathcal{A}_\pi$ is an f-hybrid adversary.

By combining Equations (3) and (4) we obtain that for every polynomial-time distinguisher D and every $\overline{x}$ and z,

$$\begin{aligned}&\Pr\left[\text{OUTPUT}_2(\text{REAL}_{\pi^\rho,\mathcal{A}(z),I}(\overline{x},n)) = \mathsf{corrupted}_1\right] \\ &= \Pr\left[\text{OUTPUT}_2(\text{HYBRID}^f_{\pi,\mathcal{A}_\pi(z),I}(\overline{x},n)) = \mathsf{corrupted}_1\right] \\ &\quad + \Pr\left[\text{OUTPUT}^\rho_2(\text{REAL}_{\pi^\rho,\mathcal{A}(z),I}(\overline{x},n)) = \mathsf{corrupted}_1\right] \\ &\geq \epsilon_2(n)\cdot\left|\Pr[D(\text{IDEAL}_{f,\mathcal{S}(z),I}(\overline{x},n))=1] - \Pr[D(\text{HYBRID}^f_{\pi,\mathcal{A}_\pi(z),I}(\overline{x},n))=1]\right| - \mu'(n) \\ &\quad + \epsilon_1(n)\cdot\left|\Pr[D(\text{HYBRID}^f_{\pi,\mathcal{A}_\pi(z),I}(\overline{x},n))=1] - \Pr[D(\text{REAL}_{\pi^\rho,\mathcal{A}(z),I}(\overline{x},n))=1]\right| - \mu(n) \\ &\geq \min\{\epsilon_1,\epsilon_2\}(n)\cdot\left|\Pr[D(\text{IDEAL}_{f,\mathcal{S}(z),I}(\overline{x},n))=1] - \Pr[D(\text{REAL}_{\pi^\rho,\mathcal{A}(z),I}(\overline{x},n))=1]\right| - \mu'(n) - \mu(n)\end{aligned}$$

where the last inequality is due to the fact that $|a-b|+|b-c| \geq |a-c|$, and replacing ϵ_1 and ϵ_2 with $\min\{\epsilon_1,\epsilon_2\}$ only makes the result smaller. We therefore conclude that π^ρ securely computes g in the presence of covert adversaries with ϵ-deterrent, where $\epsilon = \min\{\epsilon_1,\epsilon_2\}$. ■

4.2 Composition for Definitions 3.4 and 3.5

In this section, we prove an analogous modular sequential composition theorem for the stronger Definitions 3.4 and 3.5. Before doing so, we define an (f, ϵ)-hybrid model that is the same as the regular hybrid model except that the trusted party is as in IDEALC$^\epsilon$ (when considering Definition 3.4) or as in IDEALSC$^\epsilon$ (when considering Definition 3.5).

Theorem 4.2 *Let $\epsilon : \mathbb{N} \to [0,1]$ be a function, let f be a multiparty probabilistic polynomial-time functionality and let ρ be a secure protocol for computing f in the presence of covert adversaries with ϵ-deterrent. Furthermore, let g be a multiparty functionality and let π be a secure protocol for computing g in the (f, ϵ)-hybrid model in the presence of covert adversaries with ϵ-deterrent. Then, π^ρ is a secure protocol for computing g in the presence of covert adversaries with ϵ-deterrent. The above holds for Definitions 3.4 and 3.5 by taking the appropriate ideal model in each case.*

Proof Sketch: Theorem 4.2 can be derived as an almost immediate corollary from the composition theorem of [4] in the following way. First, define a special functionality interface that follows the instructions of the trusted party in Definition 3.4 (respectively, in Definition 3.5). That is, define a *reactive functionality* that receives inputs and write outputs (this functionality is modelled by an interactive Turing machine). The appropriate reactive functionality here acts exactly like the trusted party (e.g., if it receives a cheat_i message, then it tosses coins and with probability ϵ outputs $\mathsf{corrupted}_i$ to all parties and with probability $1-\epsilon$ gives the adversary all of the honest parties' inputs and lets it chooses their outputs). Next, consider the standard ideal model of Definition 2.1 with functionalities of the above form. It is easy to see that a protocol securely computes some functionality f under Definition 3.4 (resp., under Definition 3.5) *if and only if* it is securely computes the appropriately defined reactive functionality under Definition 2.1. This suffices because the composition theorem of [4] can be applied to Definition 2.1, yielding the result.[4]
■

We note that it is possible to generalize Theorem 4.2 so that ρ and π have different values of ϵ. However, π must be proven secure with the ϵ-value of ρ in mind. That is, we can state the following theorem: If ρ is a secure protocol for computing f in the presence of covert adversaries with ϵ'-deterrent, and π is a secure protocol for computing g in the (f, ϵ')-hybrid model in the presence of covert adversaries with ϵ-deterrent, then π^ρ is a secure protocol for computing g in the presence of covert adversaries with ϵ-deterrent.

5 Oblivious Transfer

In the oblivious transfer functionality [22, 8], a sender has two inputs (x_0, x_1) and a receiver has an input bit σ. The sender receives no output (and, in particular, learns nothing about the receiver's bit), while the receiver learns x_σ (but learns nothing about $x_{1-\sigma}$). This variant of oblivious transfer is often called 1-out-of-2 oblivious transfer.

In this section we will construct an efficient oblivious transfer protocol that is secure in the presence of covert adversaries with ϵ-deterrent. We will first present the basic scheme that considers a single oblivious transfer and $\epsilon = 1/2$. We will then extend this to enable the simultaneous parallel

[4] Two remarks are in place here. First, the composition theorem of [4] is formally proven for standard (non-reactive) functionalities and the case of an honest majority. Nevertheless, the proof can be extended to these cases in a straightforward way with almost no changes. Second, the composition theorem of [4] assumes a strict polynomial-time simulator. This is fine because we also required this in our definitions.

execution of many oblivious transfers and also higher values of ϵ. Our constructions all rely on the existence of secure homomorphic encryption schemes.

Homomorphic encryption. Let (G, E, D) be a public-key encryption scheme that has indistinguishable encryptions under chosen-plaintext attacks. We say that (G, E, D) is homomorphic if it has the following homomorphic property:

1. The plaintext is taken from a finite Abelian group determined by the public key. For notational convenience, we assume here that the group is the "additive" group Z_q; however, an analogous construction works for the "multiplicative" group over Z_q as well.

2. Given *any* public-key pk generated by the key generation algorithm G and *any* two ciphertexts $c_1 = E_{pk}(m_1)$ and $c_2 = E_{pk}(m_2)$ under that key, it is possible to efficiently compute a *random* encryption $E_{pk}(m_1)+_E E_{pk}(m_2) = E_{pk}(m_1+m_2)$. Consequently, it is also possible to efficiently compute $E_{pk}(m_1 \cdot \alpha)$ for any known integer α. (This follows because repeated squaring – or addition – can be used.)

We also assume that (G, E, D) has *no decryption errors*. Such encryption schemes can be constructed under the quadratic-residuosity, N-residuosity, decisional Diffie-Hellman (DDH) and other assumptions; see [1, 17] for some references.

5.1 The Basic Protocol

Protocol 5.1 (oblivious transfer from errorless homomorphic encryption):

- **Inputs:** *The sender S has a pair of strings (x_0, x_1) for input; the receiver R has a bit σ. Both parties have the security parameter 1^n as auxiliary input. (In order to satisfy the constraints that all inputs are of the same length, it is possible to define $|x_0| = |x_1| = k$ and give the receiver $(\sigma, 1^{2k-1})$.)*

- **Assumption:** *We assume that the group determined by the homomorphic encryption scheme with security parameter n is large enough to contain all strings of length k. Thus, if the homomorphic encryption scheme only works for single bits, we will only consider $k = 1$ (i.e., bit oblivious transfer).*

- **The protocol:**

  1. *The receiver R chooses two sets of two pairs of keys:*
     (a) $(pk_1^0, sk_1^0), (pk_2^0, sk_2^0) \leftarrow G(1^n)$ using random coins r_G^0, and
     (b) $(pk_1^1, sk_1^1), (pk_2^1, sk_2^1) \leftarrow G(1^n)$ using random coins r_G^1
     R sends (pk_1^0, pk_2^0) and (pk_1^1, pk_2^1) to the sender S.
  2. Key-generation challenge:
     (a) S chooses a random coin $b \in_R \{0, 1\}$ and sends b to R.
     (b) R sends S the random-coins r_G^b that it used to generate (pk_1^b, pk_2^b).
     (c) S checks that the public keys output by the key-generation algorithm G when given input 1^n and the appropriate portions of the random-tape r_G^b equal pk_1^b and pk_2^b. If this does not hold, or if R did not send any message here, S outputs $\mathsf{corrupted}_R$ and halts. Otherwise, it proceeds.
     Denote $pk_1 = pk_1^{1-b}$ and $pk_2 = pk_2^{1-b}$.

3. *Let H be the message-space of the encryption scheme. Then, R chooses two random values $\sigma_0, \sigma_1 \in_R H$ with the constraint that $\sigma_0 + \sigma_1 = \sigma$.*

   (a) *R computes*

   $$c_0^1 = E_{pk_1}(\sigma_0) \qquad c_1^1 = E_{pk_1}(\sigma_1)$$
   $$c_0^2 = E_{pk_2}(\sigma_0) \qquad c_1^2 = E_{pk_2}(\sigma_1)$$

   using random coins r_0^1, r_1^1, r_0^2 and r_1^2, respectively. (Note that c_0^1 and c_0^2 are encryptions of the same value, and likewise c_1^1 and c_1^2. However, the encryptions use independent randomness and different keys.)

   (b) *R sends c_0^1, c_1^1 and c_0^2, c_1^2 to S.*

4. Encryption-generation challenge:

   (a) *S chooses a random bit $b' \in_R \{0,1\}$ and sends b' to R.*

   (b) *R sends $\sigma_{b'}$ to S, together with $r_{b'}^1$ and $r_{b'}^2$ (i.e., R sends an opening to the ciphertexts $c_{b'}^1$ and $c_{b'}^2$).*

   (c) *S checks that $c_{b'}^1$ and $c_{b'}^2$ are both encryptions of the same value. If not (including the case that no message is sent by R), S outputs $\mathsf{corrupted}_R$ and halts. Otherwise, it continues to the next step.*

5. *S uses the homomorphic property and $c_0^1, c_1^1, c_0^2, c_1^2$ to compute $E_{pk_1}(\sigma) = c_0^1 +_E c_1^1 = E_{pk_1}(\sigma_0) +_E E_{pk_1}(\sigma_1)$ and $E_{pk_2}(\sigma) = c_0^2 + c_1^2 = E_{pk_2}(\sigma_0) +_E E_{pk_2}(\sigma_1)$. The sender S chooses two random values $s_0, s_1 \in_R G$ and uses the homomorphic property to compute*

   $$\tilde{c}_0 = E_{pk_1}(x_0 + \sigma \cdot s_0) \quad \text{and} \quad \tilde{c}_1 = E_{pk_2}(x_1 + (1-\sigma) \cdot s_1)$$

   S sends $\tilde{c}_0$ and $\tilde{c}_1$ to R. (Notice that $\tilde{c}_0$ is encrypted with key pk_1 and $\tilde{c}_1$ is encrypted with key pk_2.)

6. *If $\sigma = 0$, the receiver R outputs $x_0 = D_{sk_1}(\tilde{c}_0)$. Otherwise, R outputs $x_1 = D_{sk_2}(\tilde{c}_1)$.*

7. *If at any stage during the protocol, S does not receive the next message that it expects to receive from R or the message it receives is invalid and cannot be processed, it outputs abort_R (unless it was already instructed to output $\mathsf{corrupted}_R$). Likewise, if R does not receive the next message that it expects to receive from S or it receives an invalid message, it outputs abort_S.*

The intuition as to why Protocol 5.1 is secure is as follows. First note that if the receiver follows the instructions of the protocol, it learns only a single value x_0 or x_1. This is because both $c_0^1 +_E c_1^1$ and $c_0^2 +_E c_1^2$ are encryptions of the same bit σ. Thus, if $\sigma = 0$, then $\tilde{c}_1$ is an encryption of a uniformly distributed value due to the blinding by s_1, and vice versa if $\sigma = 1$. (If $\sigma \notin \{0,1\}$ then both $\tilde{c}_0$ and $\tilde{c}_1$ are encryptions of uniformly distributed values.) The first problem that arises is that the receiver may not generate the encryptions $c_0^1, c_1^1, c_0^2, c_1^2$ properly (and so it may be that $c_0^1 +_E c_1^1$ is an encryption of 0 and $c_0^2 +_E c_1^2$ is an encryption of 1). This is prevented by the encryption-generation challenge. That is, the receiver may try to cheat in this way. However, then it is guaranteed to be caught with probability 1/2. The above intuition relates to *privacy*. However, we need to prove security via simulation. In order to do this, we have to show how to *extract* the receiver's implicit input and how to *simulate* its view. Extraction works by first providing the corrupted receiver with the encryption-challenge bit $b' = 0$ and then rewinding it and providing it with the challenge

$b' = 1$. If the corrupted receiver replies to both challenges, then the simulator can construct σ from σ_0 and σ_1. A crucial point here is that if the receiver does not reply to both challenges then an honest sender would output $\mathsf{corrupted}_R$ with probability 1/2, and so this corresponds to a cheat_R input in the ideal world. The task of simulating the receiver's view in this case is straightforward once its input is extracted; see the proof below.

In the case that the sender is corrupted, it is easy to see that it cannot learn anything about the receiver's input due to the semantic security of the encryption scheme. However, as above, we need to show how extraction and simulation works. Extraction here works by providing inconsistent encryptions in $c_0^1, c_1^1, c_0^2, c_1^2$. That is, these encryptions are generated so that $c_0^1 +_E c_1^1$ is an encryption of 0 and $c_0^2 +_E c_1^2$ is an encryption of 1. This ensures that $\tilde{c}_0$ is an encryption of x_0 and $\tilde{c}_1$ is an encryption of $\tilde{c}_1$ (i.e., both the s_0 and s_1 masks are neutralized). An important point here is that unlike a real receiver, the simulator can do this without being "caught". Specifically, the simulator generates the ciphertexts so that for some β it holds that c_β^1 and c_β^2 are to the same value σ_β, whereas $c_{1-\beta}^1$ and $c_{1-\beta}^2$ are to different values $\sigma_{1-\beta}$ and $\sigma'_{1-\beta}$. (Importantly, $\sigma_\beta + \sigma_{1-\beta} = 0$ whereas $\sigma_\beta + \sigma'_{1-\beta} = 1$.) Then, the simulator "hopes" that the corrupted sender asks it to open the ciphertexts c_β^1 and c_β^2 which look as they should. In such a case, the simulator proceeds and succeeds in extracting both x_0 and x_1. However, if the corrupted sender asks the simulator to open the other ciphertexts (that are clearly invalid), the simulator just rewinds the corrupted sender and tries again. Thus, extraction can be achieved. Regarding the simulation of the sender's view, this follows from the fact that the only differences between the above and a real execution are the values encrypted in the ciphertexts $c_0^1, c_1^1, c_0^2, c_1^2$. These distributions are therefore indistinguishable by the semantic security of the encryption scheme.

We now formally prove that Protocol 5.1 meets Definition 3.5 with $\epsilon = \frac{1}{2}$ (of course, this immediately implies security under Definitions 3.2 and 3.4 as well).

Theorem 5.2 *Assuming that (G, E, D) constitutes a semantically secure homomorphic encryption scheme (with errorless decryption), Protocol 5.1 securely computes the oblivious transfer functionality $((x_0, x_1), \sigma) \mapsto (\lambda, x_\sigma)$ in the presence of covert adversaries with ϵ-deterrent for $\epsilon = \frac{1}{2}$, under Definition 3.5.*

Proof: We will separately consider the case that no parties are corrupted, the case that the receiver is corrupted and the case that the sender is corrupted (the case that both parties are corrupted is trivial). We note that although we construct three different simulators (one for each corruption case), a single simulator as required by the definition can be constructed by simply combining the three simulators into one machine, and working appropriately given the corruption set I.

No corruptions. We first consider the case that no parties are corrupted (i.e., $I = \phi$). In this case, the real adversary $\mathcal{A}$'s view can be generated by a simulator Sim that simply runs S and R honestly, with inputs $x_0 = x_1 = 0^k$ and $\sigma = 0$ (recall that in this case we assume that the adversary's auxiliary input contains the input length k). The fact that this simulation is indistinguishable from a real execution (with the honest parties' real inputs) follows from the indistinguishability property of encryption scheme. The proof is straightforward and is therefore omitted. We remark that in order to show that the REAL and IDEAL outputs are indistinguishable, we also have to show that the honest parties' outputs in a real execution are correct (because this is the case in the ideal world). The sender's output is defined as λ and so this clearly holds. Regarding the receiver, recall that $\tilde{c}_0 = E_{pk_1}(x_0 + \sigma \cdot s_0)$ and $\tilde{c}_1 = E_{pk_2}(x_1 + (1 - \sigma) \cdot s_1)$. Thus, if $\sigma = 0$ it holds that $\tilde{c}_0 = E_{pk_1}(x_0)$ and if $\sigma = 1$ it holds that $\tilde{c}_1 = E_{pk_2}(x_1)$. This implies that the receiver correctly obtains x_σ as required.

Corrupted receiver: Let $\mathcal{A}$ be a real adversary that controls the receiver R. We construct a simulator Sim that works as follows:

1. Sim receives $(\sigma, 1^{2k-1})$ and z as input and invokes $\mathcal{A}$ on this input.
2. Sim plays the honest sender with $\mathcal{A}$ as receiver.
3. When Sim reaches the key-generation challenge step, it first sends $b = 0$ and receives back $\mathcal{A}$'s response. Then, Sim rewinds $\mathcal{A}$, sends $b = 1$ and receives back $\mathcal{A}$'s response.
   (a) If neither of the responses from $\mathcal{A}$ are valid (where by validity we mean a response that would not cause S to output $\mathsf{corrupted}_R$ in a real execution), Sim sends $\mathsf{corrupted}_R$ to the trusted party, simulates the honest S aborting due to detected cheating, and outputs whatever $\mathcal{A}$ outputs.
   (b) If $\mathcal{A}$ sends back exactly one valid response, then Sim sends cheat_R to the trusted party.
      i. If the trusted party replies with $\mathsf{corrupted}_R$, then Sim rewinds $\mathcal{A}$ and hands it the query for which $\mathcal{A}$'s response was not valid. Sim then simulates the honest S aborting due to detected cheating, and outputs whatever $\mathcal{A}$ outputs.
      ii. If the trusted party replies with $\mathsf{undetected}$ and the honest S's input pair (x_0, x_1), then Sim plays the honest sender with input (x_0, x_1) in a full execution with $\mathcal{A}$ as the receiver. At the conclusion, Sim outputs whatever $\mathcal{A}$ outputs.
   (c) If $\mathcal{A}$ sends back two valid responses, then Sim rewinds $\mathcal{A}$, gives it a random b' and proceeds as below.
4. Sim receives ciphertexts $c_0^1, c_1^1, c_0^2, c_1^2$ from $\mathcal{A}$.
5. Next, in the encryption-generation challenge step, Sim first sends $b' = 0$ and receives back $\mathcal{A}$'s response. Then, Sim rewinds $\mathcal{A}$, sends $b' = 1$ and receives back $\mathcal{A}$'s response.
   (a) If neither of the responses from $\mathcal{A}$ are valid (where by validity we mean a response that would not cause S to output $\mathsf{corrupted}_R$ in a real execution), Sim sends $\mathsf{corrupted}_R$ to the trusted party, simulates the honest S aborting due to detected cheating, and outputs whatever $\mathcal{A}$ outputs.
   (b) If $\mathcal{A}$ sends back exactly one valid response, then Sim sends cheat_R to the trusted party.
      i. If the trusted party replies with $\mathsf{corrupted}_R$, then Sim rewinds $\mathcal{A}$ and hands it the query for which $\mathcal{A}$'s response was not valid. Sim then simulates the honest S aborting due to detected cheating, and outputs whatever $\mathcal{A}$ outputs.
      ii. If the trusted party replies with $\mathsf{undetected}$ and the honest S's input pair (x_0, x_1), then Sim plays the honest sender with input (x_0, x_1) and completes the execution with $\mathcal{A}$ as the receiver. (Note that the sender has not yet used its input at this stage of the protocol. Thus, Sim has no problem completing the execution like an honest sender.) At the conclusion, Sim outputs whatever $\mathcal{A}$ outputs.
   (c) If $\mathcal{A}$ sends back two valid responses, then Sim computes $\sigma = \sigma_0 + \sigma_1$. If $\sigma \in \{0, 1\}$, then Sim sends σ to the trusted party and receives back $x = x_\sigma$. Simulator Sim then completes the execution playing the honest sender and using $x_0 = x_1 = x$. If $\sigma \notin \{0, 1\}$, then Sim completes the execution playing the honest sender and using $x_0 = x_1 = 0^k$, where $|x| = k$.

6. If at any point $\mathcal{A}$ sends a message that would cause the honest sender to halt and output abort_R, simulator Sim immediately sends abort_R to the trusted party, halts the simulation and proceeds to the final "output" step.

7. *Output:* At the conclusion, Sim outputs whatever $\mathcal{A}$ outputs.

This completes the description of Sim. Denoting π as Protocol 5.1 and noting that I here equals $\{R\}$ (i.e., the receiver is corrupted), we need to prove that for $\epsilon = \frac{1}{2}$,

$$\left\{\text{IDEALSC}^{\epsilon}_{\text{OT},\mathcal{S}(z),I}(((x_0,x_1),\sigma),n)\right\} \stackrel{\text{c}}{\equiv} \left\{\text{REAL}_{\pi,\mathcal{A}(z),I}(((x_0,x_1),\sigma),n)\right\}$$

It is clear that the simulation is perfect if Sim sends $\mathsf{corrupted}_R$ or cheat_R at any stage. This is due to the fact that the probability that an honest S outputs $\mathsf{corrupted}_R$ in the simulation is identical to the probability in a real execution (probability 1 in the case that $\mathcal{A}$ responds incorrectly to both challenges and probability 1/2 otherwise). Furthermore, in the case that Sim sends cheat_R and receives back $\mathsf{undetected}$ it concludes the execution using the true input of the sender. The simulation until the last step is perfect (it involves merely sending random challenges); therefore the completion using the true sender's input yields a perfect simulation. The above is clearly true of abort_R as well (because this can only occur before the last step where the sender's input is used).

It remains to analyze the case that Sim does not send $\mathsf{corrupted}_R$, cheat_R or abort_R to the trusted party. Notice that in this case, $\mathcal{A}$ responded correctly to both the key-generation challenges and the encryption-generation challenges. In particular, this implies that the keys pk_1 and pk_2 are correctly generated, and that Sim computes σ based on the encrypted values sent by $\mathcal{A}$.

Now, if $\sigma = 0$, then Sim hands $\mathcal{A}$ the ciphertexts $\tilde{c}_0 = E_{pk_1}(x_0)$ and $\tilde{c}_1 = E_{pk_2}(s)$ for some random value s, and if $\sigma = 1$, it hands $\mathcal{A}$ the ciphertexts $\tilde{c}_0 = E_{pk_1}(s')$ for some random value s' and $\tilde{c}_1 = E_{pk_2}(x_1)$. This follows from the instructions of Sim and the honest party (Sim plays the honest party with $x_0 = x_1 = x_\sigma$ and so $\tilde{c}_{1-\sigma}$ is an encryption of $s = x_\sigma + s_{1-\sigma}$). The important point to notice is that these messages are distributed identically to the honest sender's messages in a real protocol; the fact that Sim does not know $x_{1-\sigma}$ makes no difference because for a random s it holds that $x_\sigma + s$ is distributed identically to $x_{1-\sigma} + s$. The same analysis holds in the case that $\sigma \notin \{0,1\}$. Specifically, in this case, $\tilde{c}_0 = E_{pk_1}(s)$ and $\tilde{c}_1 = E_{pk_2}(s')$ for some random values s and s'. (In order to see this, set $s = \sigma \cdot s_0$ and $s' = (1-\sigma)\cdot s_1$. It follows that these strings are uniformly distributed because σ is fixed and s_0, s_1 are chosen uniformly.) We note that this assumes that the homomorphic property of the encryption scheme holds, but this is given by the fact that pk_1 and pk_2 are correctly formed. Regarding the rest of the messages sent by Sim, these are generated independently of the sender-input and so exactly like an honest sender.

We conclude that the view of $\mathcal{A}$ as generated by the simulator Sim is *identical* to the distribution generated in a real execution. Thus, its output is identically distributed in both cases. (Since the sender receives no output, we do not need to consider the output distribution of the honest sender in the real and ideal executions.) We conclude that

$$\left\{\text{IDEALSC}^{\epsilon}_{\text{OT},\mathcal{S}(z),I}(((x_0,x_1),\sigma),n)\right\} \equiv \left\{\text{REAL}_{\pi,\mathcal{A}(z),I}(((x_0,x_1),\sigma),n)\right\}$$

completing this corruption case.

Corrupted sender: Let $\mathcal{A}$ be a real adversary that controls the sender S. We construct a simulator Sim that works as follows:

1. Sim receives (x_0, x_1) and z and invokes $\mathcal{A}$ on this input.

2. Sim interacts with $\mathcal{A}$ and plays the honest receiver until Step 3 of the protocol.

3. In Step 3 of the protocol, Sim works as follows:

   (a) Sim chooses a random bit $\beta \in_R \{0,1\}$

   (b) Sim chooses a random value $\sigma_\beta \in_R G$

   (c) Sim computes two values: $\sigma_{1-\beta}$ and $\sigma'_{1-\beta}$ such that $\sigma_\beta + \sigma_{1-\beta} = 0$ and $\sigma_\beta + \sigma'_{1-\beta} = 1$

   (d) Sim computes $c^1_\beta = E_{pk_1}(\sigma_\beta)$ and $c^2_\beta = E_{pk_2}(\sigma_\beta)$, as the honest receiver would. However, Sim computes the other ciphertexts differently. Specifically, it computes

   $$c^1_{1-\beta} = E_{pk_1}(\sigma_{1-\beta}) \quad \text{and} \quad c^2_{1-\beta} = E_{pk_2}(\sigma'_{1-\beta})$$

   Notice that

   $$c^1_0 +_E c^1_1 = E_{pk_1}(\sigma_\beta + \sigma_{1-\beta}) = E_{pk_1}(0)$$

   and

   $$c^2_0 +_E c^2_1 = E_{pk_2}(\sigma_\beta + \sigma'_{1-\beta}) = E_{pk_2}(1)$$

   (e) Sim sends $c^1_0, c^2_0, c^1_1, c^2_1$ to $\mathcal{A}$.

4. In the next step (Step 4 of the protocol), $\mathcal{A}$ sends a bit b'. If $b' = \beta$, then Sim opens the ciphertexts c^1_β and c^2_β as the honest receiver would (note that the ciphertexts are both to the same value σ_β). Otherwise, Sim returns to Step 3 of the simulation above (i.e., it returns to the beginning of Step 3 of the protocol) and tries again with fresh randomness.[5]

5. The simulator Sim receives from $\mathcal{A}$ the ciphertexts $\tilde{c}_0$ and $\tilde{c}_1$. Sim computes $x_0 = D_{sk_1}(\tilde{c}_0)$ and $x_1 = D_{sk_2}(\tilde{c}_1)$, and sends the pair (x_0, x_1) to the trusted party as S's input.

6. If at any stage in the simulation $\mathcal{A}$ does not respond, or responds with an invalid message that cannot be processed, then Sim sends abort_S to the trusted party for the sender's inputs. (Such behavior from $\mathcal{A}$ can only occur before the last step and so before any input (x_0, x_1) has already been sent to the trusted party.)

7. Sim outputs whatever $\mathcal{A}$ outputs.

Notice that Sim never sends cheat_S to the trusted party. Thus we actually prove standard security in this corruption case. That is, we prove that:

$$\left\{\text{IDEAL}_{\text{OT},\mathsf{Sim}(z),I}((x_0, x_1, \sigma), n)\right\} \stackrel{\text{c}}{\equiv} \left\{\text{REAL}_{\pi,\mathcal{A}(z),I}((x_0, x_1, \sigma), n)\right\} \tag{5}$$

By Proposition 3.8, this implies security for covert adversaries as well. In order to prove Eq. (5), observe that the only difference between the view of the adversary $\mathcal{A}$ in a real execution and in the simulation by Sim is due to the fact that Sim does not generate $c^1_0, c^2_0, c^1_1, c^2_1$ correctly. Thus, intuitively, Eq. (5) follows from the security of the encryption scheme. That is, we begin by showing that if the view of $\mathcal{A}$ in the real and ideal executions can be distinguished, then it is possible to break the security of the encryption scheme. We begin by showing that the view of $\mathcal{A}$ when interacting with an honest sender with input $\sigma = 0$ is indistinguishable from the view of $\mathcal{A}$ when interacting in a simulation with Sim.

[5] This yields an expected polynomial-time simulation because these steps are repeated until $b' = \beta$. A strict polynomial-time simulation can be achieved by just halting after n attempts. The probability that $b' \neq \beta$ in all of these attempts can be shown to be negligible, based on the hiding property of the encryption scheme.

Let $\mathcal{A}'$ be an adversary that attempts to distinguish encryptions under a key pk.[6] Adversary $\mathcal{A}'$ receives a key pk, chooses a random bit $\gamma \in_R \{0,1\}$ and sets $pk_2^{1-\gamma} = pk$. It then chooses the keys $pk_1^{1-\gamma}$, pk_1^{γ} and pk_2^{γ} by itself and sends $\mathcal{A}$ the keys (pk_1^0, pk_2^0) and (pk_1^1, pk_2^1). When $\mathcal{A}$ replies with a bit b, adversary $\mathcal{A}'$ acts as follows. If $b = \gamma$, then $\mathcal{A}'$ opens the randomness used in generating (pk_1^b, pk_2^b) as the honest receiver would ($\mathcal{A}'$ can do this because it chose $(pk_1^{\gamma}, pk_2^{\gamma})$ by itself and $\gamma = b$). If $b \neq \gamma$, then $\mathcal{A}'$ cannot open the randomness as an honest receiver would. Therefore, $\mathcal{A}'$ just halts. If $\mathcal{A}$ continues, then it has set $pk_1 = pk_1^{1-\gamma}$ and $pk_2 = pk_2^{1-\gamma}$ (and so pk_2 is the public-key pk that $\mathcal{A}'$ is "attacking"). Now, $\mathcal{A}'$ computes the ciphertexts $c_0^1, c_0^2, c_1^1, c_1^2$ in the following way. $\mathcal{A}'$ chooses σ_β at random, as the honest receiver would, and computes $\sigma_{1-\beta}$ and $\sigma'_{1-\beta}$ as the simulator would. (Recall that this means that $\sigma_\beta + \sigma_{1-\beta} = 0$ and $\sigma_\beta + \sigma'_{1-\beta} = 1$.) It computes $c_\beta^1 = E_{pk_1}(\sigma_\beta)$, $c_\beta^2 = E_{pk_2}(\sigma_\beta)$ and $c_{1-\beta}^1 = E_{pk_1}(\sigma_{1-\beta})$. It then outputs the pair of plaintexts $(m_0 = \sigma_{1-\beta}, m_1 = \sigma'_{1-\beta})$ and receives back $c = E_{pk}(m_b) = E_{pk_2}(m_b)$ (for $b \in \{0,1\}$). Adversary $\mathcal{A}'$ sets $c_{1-\beta}^2 = c$ (i.e., to equal the challenge ciphertext) and continues playing the honest receiver until the end. The key point here is that if $\mathcal{A}'$ does not halt and $b = 0$, then the simulation by $\mathcal{A}'$ is identical to a real execution between $\mathcal{A}$ and an honest receiver R who has input $\sigma = 0$ (because both $c_{1-\beta}^1$ and $c_{1-\beta}^2$ are encryptions of $\sigma_{1-\beta}$, and $\sigma_\beta + \sigma_{1-\beta} = 0$). In contrast, if $\mathcal{A}'$ does not halt and $b = 1$, then the simulation by $\mathcal{A}'$ is identical to the simulation carried out by Sim. Finally, note that $\mathcal{A}'$ halts with probability exactly $1/2$ in both cases (this is due to the fact that the distribution of the keys is identical for both choices of γ). Combining the above together, we have that if it is possible to distinguish the view of $\mathcal{A}$ in the simulation by Sim from a real execution with a receiver who has input 0, then it is possible to distinguish encryptions. Specifically, $\mathcal{A}'$ can just run the distinguisher that exists for these views and output whatever the distinguisher outputs.

The above shows that the view of $\mathcal{A}$ in the simulation is indistinguishable from its view in a real execution with an honest receiver with input $\sigma = 0$. However, we actually have to show that when the honest receiver has input $\sigma = 0$, the *joint distribution* of $\mathcal{A}$ and the honest receiver's outputs in a real execution is indistinguishable from the joint distribution of Sim and the honest receiver's outputs in the ideal model. The point to notice here is that the output of the honest receiver in both the real and ideal models is the value obtained by decrypting $\tilde{c}_0$ using key pk_1. (In the real model this is what the protocol instructs the honest party to output and in the ideal model this is the value that Sim sends to the trusted party as the sender's input x_0.) However, in this reduction $\mathcal{A}'$ knows the associated secret-key to pk_1, because it chose pk_1 itself. Thus, $\mathcal{A}'$ can append the decryption of $\tilde{c}_0$ to the view of $\mathcal{A}$, thereby generating a joint distribution. It follows that if $\mathcal{A}'$ received an encryption of m_0 then it generates the joint distribution of the outputs in the real execution, and if it received an encryption of m_1 then it generates the joint distribution of the outputs in the ideal execution. This completes the proof of Eq. (5) for the case that $\sigma = 0$.

The case for $\sigma = 1$ follows from an almost identical argument as above. In this case, the only difference between the real and ideal executions is that in the ideal execution $c_{1-\beta}^1$ is an encryption of $\sigma_{1-\beta}$ instead of an encryption of $\sigma'_{1-\beta}$ (as it should be so that $\sigma_\beta + \sigma'_{1-\beta} = 1$). Thus, the same reduction by $\mathcal{A}'$ is carried out based on the key pk_1. Regarding the joint distribution, $\mathcal{A}'$ chooses the other key (here pk_2) by itself and so can decrypt $\tilde{c}_1$ in order to obtain the value that the honest receiver would output. This concludes the proof of this corruption case. ■

Discussion. The proof of Protocol 5.1 in the case that the receiver is corrupted relies heavily on

[6] The game that $\mathcal{A}'$ plays is that it receives a key pk, outputs a pair of plaintexts m_0, m_1, receives back a *challenge ciphertext* $E_{pk}(m_b)$ for some $b \in \{0,1\}$, and outputs a "guess" bit b'. An encryption scheme is indistinguishable if the probability that $\mathcal{A}'$ outputs $b' = 1$ when $b = 1$ is negligibly close to the probability that $\mathcal{A}'$ outputs $b' = 1$ when $b = 0$.

the fact that the simulator can send cheat and therefore does not need to complete a "standard" simulation. Take for example the case that $\mathcal{A}$ (controlling the receiver) only replies with one valid response to the encryption-generation challenge. In this case, the simulator never succeeds in extracting the value of σ (and indeed, the value may not be uniquely defined). However, with probability 1/2, a real execution will terminate without any cheating being detected. Thus, the standard simulation requirements (that require the simulator to also "succeed" with probability 1/2) cannot be met here. This demonstrates why it is possible to achieve higher efficiency for this definition of security.

The proof of security for a corrupted sender. We stress that we have actually proven something stronger. Specifically, we have shown that Protocol 5.1 is secure in the presence of a covert receiver with 1/2-deterrent as stated. However, we have also shown that Protocol 5.1 is (fully) secure with abort in the presence of a *malicious sender*.

Efficiently recognizable public keys. We remark that in the case that it is possible to efficiently recognize that a public-key is in the range of the key-generator of the public-key encryption scheme, it is possible to skip the key-generation challenge step in the protocol (the sender can verify for itself if the key is valid).

5.2 Extensions

String oblivious transfer. In Protocol 5.1, x_0 and x_1 are elements in the group over which the homomorphic encryption scheme is defined. If this group is large, then we can carry out string oblivious transfer. This is important because later we will use Protocol 5.1 to exchange symmetric encryption keys. However, if the group contains only 0 and 1, then this does not suffice. In order to extend Protocol 5.1 to deal with string oblivious transfer, even when the group has only two elements, we only need to change the last two steps of the protocol. Specifically, instead of S computing a single encryption for x_0 and a single encryption for x_1, it computes an encryption for each bit. That is, denote the bits of x_0 by $x_0^1, \ldots, x_0^n$, and likewise for x_1. Then, S computes:

$$\tilde{c}_0^1 = E_{pk_1}(x_0^1 + \sigma \cdot s_0^1), \ldots, E_{pk_1}(x_0^n + \sigma \cdot s_0^n)$$

and

$$\tilde{c}_1^2 = E_{pk_2}(x_1^1 + (1-\sigma) \cdot s_1^1), \ldots, E_{pk_2}(x_1^n + (1-\sigma) \cdot s_1^n)$$

where $s_0^1, \ldots, s_0^n$ and $s_1^1, \ldots, s_1^n$ are independent random values. Note that the receiver can still only obtain one of the strings because if $\sigma = 0$ then $\tilde{c}_1^2$ contains encryptions to random independent values, and vice versa if $\sigma = 1$.

Parallel oblivious transfer. We will use Protocol 5.1 in Yao's protocol for secure two-party computation. This means that we will run one oblivious transfer for every bit of the input. In principle, these oblivious transfers can be run in parallel, as long as the protocol being used remains secure under parallel composition. Fortunately, Protocol 5.1 can be modified so that it can be run in parallel. We stress that we do not refer to parallel composition in the classic sense where each execution is considered separately by the honest parties (i.e., stateless composition). Rather, we show how to extend Protocol 5.1 so that it computes the functionality:

$$((x_1^0, x_1^1), \ldots, (x_n^0, x_1^n), (\sigma_1, \ldots, \sigma_n)) \mapsto (\lambda, (x_1^{\sigma_1}, \ldots, x_n^{\sigma_n}))$$

Thus, we essentially have n oblivious transfers where in the i^{th} such transfer, the sender has input (x_i^0, x_i^1) and the receiver has input σ_i.

The extension to Protocol 5.1 works as follows. First, the same public-key pair (pk_1, pk_2) can be used in all executions. Therefore, Steps 1 and 2 remain unchanged. Then, Step 3 is carried out *independently* for all n bits $\sigma_1, \ldots, \sigma_n$. That is, for every i, two values σ_i^0 and σ_i^1 are chosen randomly under the constraint that $\sigma_i^0 + \sigma_i^1 = \sigma_i$. Furthermore, for every $i \neq j$, the values are chosen independently of each other. The four encryptions generated are generated separately for every i, in exactly the same way as in Protocol 5.1. The important change comes in Step 4. Here, the *same* challenge bit b' is used for every i. The sender then replies as it should, opening the $c_{b'}^1$ and $c_{b'}^2$ ciphertexts for every i. The protocol then concludes by the sender computing the $\tilde{c}_0$ and $\tilde{c}_1$ ciphertexts for every i, and the receiver decrypting. (We stress that the $\tilde{c}_0$ and $\tilde{c}_1$ ciphertexts are all generating using independent randomness for encrypting and for choosing s_0 and s_1.)

The proof of the above extension is almost identical to the proof of Theorem 5.2. The main point is that since only a single challenge is used for both the key-generation challenge and encryption-generation challenge, the probability of achieving $b' = \beta$ (as needed for the simulation) and $b = \gamma$ (as needed for the reduction to the security of the encryption scheme) remains one half. Furthermore, the probability that a corrupted R will succeed in cheating remains the same because if there is any i for which the σ_i^0 and σ_i^1 are not correctly formed, then the receiver will be caught with probability one half.

Higher values of ϵ. Finally, we show how it is possible to obtain higher values of ϵ with only minor changes to Protocol 5.1. The basic idea is to increase the probability of catching a corrupted receiver in the case that it attempts to generate an invalid key-pair or send ciphertexts in Step 3 that do not encrypt the same value. Let $k = \text{poly}(n)$ be an integer. Then, first the receiver generates k pairs of public-keys $(pk_1^1, pk_2^1), \ldots, (pk_1^k, pk_2^k)$ instead of just two pairs. The sender then asks the receiver to reveal the randomness used in generating all the pairs except for one (the unrevealed key-pair is the one used in the continuation of the protocol). Note that if a corrupted receiver generated even one key-pair incorrectly, then it is caught with probability $1 - 1/k$. Likewise, in Step 3, the receiver chooses k random values $\sigma_1, \ldots, \sigma_k$ under the constraint that $\sum_{i=1}^k \sigma_i = \sigma$ and encrypts each value σ_i under both pk_1 and pk_2. Then, the sender asks the receiver to open all pairs of encryptions of σ_i except for one pair. Clearly, the sender still learns nothing about σ because one of the σ_j values remains hidden. Furthermore, if the receiver generates even one pair of ciphertexts so that pk_1 encrypts some value σ_i and pk_2 encrypts some different value σ_i', then it will be caught with probability $1 - 1/k$. The rest of the protocol remains the same (of course, we modify the sender so that it computes $E_{pk_1}(\sigma)$ by "adding" all k ciphertexts). We conclude that the resulting protocol is secure in the presence of covert adversaries with ϵ-deterrent where $\epsilon = 1 - 1/k$. Notice that this works as long as k is polynomial in the security parameter and thus ϵ can be made to be very close to 1, if desired. (Of course, this methodology *cannot* be used to make ϵ negligibly close to 1, because then k has to be super-polynomial.)

Summary. We conclude with the following theorem, derived by combining the extensions above:

Theorem 5.3 *Assume that there exist semantically secure homomorphic encryption schemes with errorless decryption. Then, for any $k = \text{poly}(n)$ there exists a protocol that securely computes 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 $\epsilon = 1 - \frac{1}{k}$.

6 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. Our 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 will assume familiarity with it. Nevertheless, in Appendix A, we briefly describe Yao's garbled circuit construction and present an important lemma regarding it.

6.1 The Protocol for Definition 3.5

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 5.3. 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 Lemma A.1. 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 decrypts the circuit to the specified one and another set of keys decrypts 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 3.2 and 3.4 (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 3.4 and 3.5. In summary, this means that the protocol achieves Definition 3.2 (with $\epsilon = 1/2$) but not Definition 3.4. 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 6.1 (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. (P_2 receives as 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 5.3. 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 $\hat{k}^0_{n+1}, \hat{k}^1_{n+1}, \ldots, \hat{k}^0_{n^2+n}, \hat{k}^1_{n^2+n}$, and in $G(C')_1$ they are $\tilde{k}^0_{n+1}, \tilde{k}^1_{n+1}, \ldots, \tilde{k}^0_{n^2+n}, \tilde{k}^1_{n^2+n}$). Let $\hat{k}^0_1, \hat{k}^1_1, \ldots, \hat{k}^0_n, \hat{k}^1_n$ be the keys associated with P_1's input in $G(C')_0$ and $\tilde{k}^0_1, \tilde{k}^1_1, \ldots, \tilde{k}^0_n, \tilde{k}^1_n$ the analogous keys in $G(C')_1$. Then, for every $i \in \{1, \ldots, n\}$ and $b \in \{0, 1\}$, party P_1 computes $\hat{c}^b_i = \mathsf{Com}(\hat{k}^b_i; \hat{r}^b_i)$ and $\tilde{c}^b_i = \mathsf{Com}(\tilde{k}^b_i; \tilde{r}^b_i)$, 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 $\{\hat{c}^0_i, \hat{c}^1_i\}$ in a random order, and in the second vector the i^{th} pair is $\{\tilde{c}^0_i, \tilde{c}^1_i\}$ 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 $(\hat{k}^0_1, 0), (\hat{k}^1_1, 1), \ldots, (\hat{k}^0_{n^2+n}, 0), (\hat{k}^1_{n^2+n}, 1)$ and $\hat{r}^0_1, \hat{r}^1_1, \ldots, \hat{r}^0_n, \hat{r}^1_n$ 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 $\hat{k} = \hat{k}^{z^i_2}_{n+i}$ if $G(C')_0$ was opened, and $\tilde{k} = \tilde{k}^{z^i_2}_{n+i}$ 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 $(\tilde{k}^{x^1_1}_1, \tilde{r}^{x^1_1}_1), \ldots, (\tilde{k}^{x^n_1}_n, \tilde{r}^{x^n_1}_n)$ to P_2. Otherwise, if $b = 1$, then P_2 sends the keys $(\hat{k}^{x^1_1}_1, \hat{r}^{x^1_1}_1), \ldots, (\hat{k}^{x^n_1}_n, \hat{r}^{x^n_1}_n)$.*

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 $C'(x_1, z_2) = C'(x_1, x^1_2, \ldots, x^n_2) = C(x_1, x_2)$, 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 $\mathsf{corrupted}_i$ instead (as in Step 8).

We now prove the security of the protocol.

Theorem 6.2 *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 3.5. Then, Protocol 6.1 securely computes* f *in the presence of covert adversaries with* $1/2$*-deterrent by Definition 3.5.*

Proof: Our analysis of the security of the protocol is in the hybrid model, where the parties are assumed to have access to a trusted party computing the oblivious transfer functionality; see Section 4. Thus the simulator that we describe will play the trusted party in the oblivious transfer, when simulating for the adversary. We separately consider the different corruption cases (when no parties are corrupted, and when either one of the parties is corrupted). In the case that no parties are corrupted, the security reduces to the semi-honest case which has already been proven in [18] (the additional steps in Protocol 6.1 don't make a difference here).

Party P_2 is corrupted. Let $\mathcal{A}$ be an adversary controlling P_2. The simulator $\mathcal{S}$ works as follows:

1. $\mathcal{S}$ chooses two sets of $2n^2$ random keys as P_1 would.
2. $\mathcal{S}$ plays the trusted party for the oblivious transfers with $\mathcal{A}$ as the receiver. $\mathcal{S}$ receives the input that $\mathcal{A}$ sends to the trusted party (as its input as receiver to the oblivious transfers):
   (a) If the input is abort_2 or $\mathsf{corrupted}_2$, then $\mathcal{S}$ sends abort_2 or $\mathsf{corrupted}_2$ (respectively) to the trusted party computing f, simulates P_1 aborting and halts (outputting whatever $\mathcal{A}$ outputs).
   (b) If the input is cheat_2, then $\mathcal{S}$ sends cheat_2 to the trusted party. If it receives back $\mathsf{corrupted}_2$, then it hands $\mathcal{A}$ the message $\mathsf{corrupted}_2$ as if it received it from the trusted party, simulates P_1 aborting and halts (outputting whatever $\mathcal{A}$ outputs). If it receives back $\mathsf{undetected}$ then it uses the input x_1 of P_1 that it obtains in order to perfectly emulate P_1 in the rest of the execution. That is, it runs P_1's honest strategy with input x_1 while interacting with $\mathcal{A}$ playing P_2 for the rest of the execution. Let y_1 be the output for P_1 that it receives. $\mathcal{S}$ sends y_1 to the trusted party (for P_1's output) and outputs whatever $\mathcal{A}$ outputs. The simulation ends here in this case.
   (c) If the input is a series of bits $z_2^1, \ldots, z_2^{n^2}$, then $\mathcal{S}$ hands $\mathcal{A}$ the keys from above that are "chosen" by the z_2^i bits, and proceeds with the simulation below.
3. $\mathcal{S}$ defines $x_2 = \oplus_{i=0}^{n-1}(z_2^{i \cdot n+1}, \ldots, z_2^{i \cdot n+n})$ and sends x_2 to the trusted party computing f. $\mathcal{S}$ receives back some output y.
4. $\mathcal{S}$ chooses a random bit β and computes a garbled circuit $G(C')_\beta$ correctly (using the appropriate input keys from above as P_1 would). However, for the garbled circuit $G(C')_{1-\beta}$, the simulator $\mathcal{S}$ does not use the true circuit for computing f but rather a circuit that always evaluates to y (the value it received from the trusted party), using Lemma A.1. $\mathcal{S}$ uses the appropriate input keys from above also in generating $G(C')_{1-\beta}$. $\mathcal{S}$ also computes commitments to the keys associated with P_1's input in an honest way.
5. $\mathcal{S}$ sends $G(C')_0$, $G(C')_1$ and the commitments to $\mathcal{A}$ and receives back a bit b.
6. If $b \neq \beta$ then $\mathcal{S}$ rewinds $\mathcal{A}$ and returns to Step 4 above (using fresh randomness).
   Otherwise, if $b = \beta$, then $\mathcal{S}$ opens the garbled circuit $G(C')_b$ and the commitments as the honest P_1 would and proceeds to the next step.
7. For $i = 1, \ldots, n$, $\mathcal{S}$ sends $\mathcal{A}$ an arbitrary one of the two keys associated with the input wire w_i in $G(C')_{1-b}$ (one key per wire), together with its correct decommitments.

8. If at any stage, $\mathcal{S}$ does not receive a response from $\mathcal{A}$, it sends abort_2 to the trusted party (resulting in P_1 outputting abort_2). If the protocol proceeds successfully to the end, $\mathcal{S}$ sends $\mathsf{continue}$ to the trusted party and outputs whatever $\mathcal{A}$ outputs.

Denoting π as Protocol 6.1 and $I = \{2\}$ (i.e., party P_2 is corrupted), we prove that:

$$\left\{\text{IDEALSC}^{\epsilon}_{f,\mathcal{S}(z),I}((x_1,x_2),n)\right\} \stackrel{\text{c}}{\equiv} \left\{\text{HYBRID}^{\text{OT}}_{\pi,\mathcal{A}(z),I}((x_1,x_2),n)\right\} \tag{6}$$

In order to prove Eq. (6) we separately consider the cases of abort (including a "$\mathsf{corrupted}$" input), cheat or neither. If $\mathcal{A}$ sends abort_2 or $\mathsf{corrupted}_2$ as the oblivious transfer input, then $\mathcal{S}$ sends abort_2 or $\mathsf{corrupted}_2$ (respectively) to the trusted party computing f. In both cases the honest P_1 outputs the same (abort_2 or $\mathsf{corrupted}_2$) and the view of $\mathcal{A}$ is identical. Thus, the IDEAL and HYBRID output distributions are identical. The exact same argument is true if $\mathcal{A}$ sends cheat_2 and the reply to $\mathcal{S}$ from the trusted party is $\mathsf{corrupted}_2$. In contrast, if $\mathcal{A}$ sends cheat_2 and $\mathcal{S}$ receives back the reply $\mathsf{undetected}$, then the execution does not halt immediately. Rather, $\mathcal{S}$ plays the honest P_1 with its input x_1. Since $\mathcal{S}$ follows the exact same strategy as P_1, and the output received by P_1 from the execution is the same y_1 that $\mathcal{S}$ receives from the protocol execution, it is clear that once again the output distributions are identical (recall that in the ideal model, P_1 outputs the same y_1 obtained by $\mathcal{S}$). We remark that the probability of the trusted party answering $\mathsf{corrupted}_2$ or $\mathsf{undetected}$ is the same in the hybrid and ideal executions (i.e., 1/2), and therefore the output distributions in the cases of abort, $\mathsf{corrupted}$ or cheat are identical. We denote the event that $\mathcal{A}$ sends an abort, $\mathsf{corrupted}$ or cheat message in the oblivious transfers by bad_{OT}. Thus, we have shown that

$$\left\{\text{IDEALSC}^{\epsilon}_{f,\mathcal{S}(z),I}((x_1,x_2),n) \mid \mathsf{bad}_{\text{OT}}\right\} \equiv \left\{\text{HYBRID}^{\text{OT}}_{\pi,\mathcal{A}(z),I}((x_1,x_2),n) \mid \mathsf{bad}_{\text{OT}}\right\}$$

We now show that the IDEAL and HYBRID distributions are computationally indistinguishable in the case that $\mathcal{A}$ sends valid input in the oblivious transfer phase (i.e., in the event $\neg\mathsf{bad}_{\text{OT}}$). In order to show this, we consider a modified simulator $\mathcal{S}'$ who is also given the honest party P_1's real input x_1. Simulator $\mathcal{S}'$ works exactly as $\mathcal{S}$ does, except that it constructs $G(C')_{1-\beta}$ honestly, and not as $\tilde{G}(C)$ from Lemma A.1. Furthermore, in Step 7 it sends the keys associated with P_1's input x_1 and not arbitrary keys. It is straightforward to verify that the distribution generated by $\mathcal{S}'$ is identical to the distribution generated by $\mathcal{A}$ in an execution of the real protocol. This is due to the fact that both circuits received by $\mathcal{A}$ are honestly constructed and the keys that it receives from $\mathcal{S}'$ are associated with P_1's real input. The only difference is the rewinding. However, since β is chosen uniformly, this has no effect on the output distribution. Thus:

$$\left\{\text{IDEALSC}^{\epsilon}_{f,\mathcal{S}'(z,x_1),I}((x_1,x_2),n) \mid \neg\mathsf{bad}_{\text{OT}}\right\} \equiv \left\{\text{HYBRID}^{\text{OT}}_{\pi,\mathcal{A}(z),I}((x_1,x_2),n) \mid \neg\mathsf{bad}_{\text{OT}}\right\}$$

Next we prove that the distributions generated by $\mathcal{S}$ and $\mathcal{S}'$ are computationally indistinguishable. That is,

$$\left\{\text{IDEAL}_{f,\mathcal{S}(z),I}((x_1,x_2),n) \mid \neg\mathsf{bad}_{\text{OT}}\right\} \stackrel{\text{c}}{\equiv} \left\{\text{IDEAL}^{\epsilon}_{f,\mathcal{S}'(z,x_1),I}((x_1,x_2),n) \mid \neg\mathsf{bad}_{\text{OT}}\right\}$$

In order to see this, notice that the only difference between $\mathcal{S}$ and $\mathcal{S}'$ is in the construction of the garbled circuit $G(C')_{1-\beta}$. By Lemma A.1 it follows immediately that these distributions are computationally indistinguishable. (Note that we do not need to consider the joint distribution of $\mathcal{A}$'s view and P_1's output because P_1 has no output from Protocol 6.1.) This yields the above equation. In order to complete the proof of Eq. (6), note that the probability that the event bad_{OT} happens is *identical* in the IDEAL and HYBRID executions. This holds because the oblivious transfer

is the first step of the protocol and $\mathcal{A}$'s view in this step with $\mathcal{S}$ is identical to its view in a protocol execution with a trusted party computing the oblivious transfer functionality. Combining this fact with the above equations we derive Eq. (6).

We remark that the simulator $\mathcal{S}$ described above runs in expected polynomial-time. In order to see this, note that by Lemma A.1, a fake garbled circuit is indistinguishable from a real one. Therefore, the probability that $b = \beta$ is at most negligibly far from 1/2 (otherwise, this fact alone can be used to distinguish a fake garbled circuit from a real one). It follows that the expected number of attempts by $\mathcal{S}$ is close to two, and so its expected running-time is polynomial. By our definition, $\mathcal{S}$ needs to run in strict polynomial-time. However, this is easily achieved by having $\mathcal{S}$ halt if it fails after n rewinding attempts. Following the same argument as above, such a failure can occur with at most negligible probability.

We conclude that $\mathcal{S}$ meets the requirements of Definition 3.5. (Note that $\mathcal{S}$ only sends cheat_2 due to the oblivious transfer. Thus, if a "fully secure" oblivious transfer protocol were to be used, the protocol would meet the standard definition of security for malicious adversaries for the case that P_2 is corrupted.)

Party P_1 is corrupted. Let $\mathcal{A}$ be an adversary controlling P_1. The simulator $\mathcal{S}$ works as follows:

1. $\mathcal{S}$ invokes $\mathcal{A}$ and plays the trusted party for the oblivious transfers with $\mathcal{A}$ as the sender. $\mathcal{S}$ receives the input that $\mathcal{A}$ sends to the trusted party (as its input to the oblivious transfers):

   (a) If the input is abort_1 or $\mathsf{corrupted}_1$, then $\mathcal{S}$ sends abort_1 or $\mathsf{corrupted}_1$ (respectively) to the trusted party computing f, simulates P_2 aborting and halts (outputting whatever $\mathcal{A}$ outputs).

   (b) If the input is cheat_1, then $\mathcal{S}$ sends cheat_1 to the trusted party. If it receives back $\mathsf{corrupted}_1$, then it hands $\mathcal{A}$ the message $\mathsf{corrupted}_1$ as if it received it from the trusted party, simulates P_2 aborting and halts (outputting whatever $\mathcal{A}$ outputs). If it receives back $\mathsf{undetected}$ then it uses the input x_2 of P_2 that it obtains in order to perfectly emulate P_2 in the rest of the execution. That is, it runs P_2's honest strategy with input x_2 while interacting with $\mathcal{A}$ playing P_1 for the rest of the execution. Let y_2 be the output for P_2 that it receives. $\mathcal{S}$ sends y_2 to the trusted party (for P_2's output) and outputs whatever $\mathcal{A}$ outputs. The simulation ends here in this case.

   (c) If the input is a series of pairs of keys $\left([\hat{k}^0_{n+i}, \tilde{k}^0_{n+i}], [\hat{k}^1_{n+i}, \tilde{k}^1_{n+i}]\right)$ for $i = 1, \ldots, n^2$, then $\mathcal{S}$ proceeds below.

2. $\mathcal{S}$ receives from $\mathcal{A}$ two garbled circuits $G(C')_0$ and $G(C')_1$ and a series of commitments.

3. $\mathcal{S}$ sends $\mathcal{A}$ the bit $b = 0$ and receives its reply. $\mathcal{S}$ then rewinds $\mathcal{A}$, sends it the bit $b = 1$ and receives its reply.

4. $\mathcal{S}$ continues the simulation differently, depending on the validity of the circuit openings. In order to describe the cases, we introduce some terminology. We call a circuit opening $\mathsf{detectably\ bad}$ if it is not opened at all (i.e., P_1 doesn't send a message at all), if the keys sent by P_1 in the opening for the input wires associated with its own input or P_2's input fail to decrypt the circuit (i.e., at least one of them is an invalid decryption key), if the circuit decrypts but does not equal the auxiliary input circuit C', or if *both keys of a given wire* that P_1 provides in the oblivious transfer (as received by $\mathcal{S}$) are different to the keys that P_1 provides to open the circuit. (We stress that if only one of the keys in the last condition does not match, then the opening is *not* detectably bad. We also stress that only one of

the conditions needs to be fulfilled in order to make the opening detectably bad.) $\mathcal{S}$ works according to the follows cases:

(a) *Case 1 – both circuit openings are detectably bad:* If both of the openings of the circuit (i.e., for $b = 0$ and $b = 1$) provided by P_1 are detectably bad, then $\mathcal{S}$ sends $w_1 = \mathsf{corrupted}_1$ to the trusted party (causing the honest P_2 to output $\mathsf{corrupted}_1$). $\mathcal{S}$ then sends $\mathcal{A}$ a random bit b, simulates P_2 aborting due to detected cheating, and outputs whatever $\mathcal{A}$ outputs.

(b) *Case 2 – exactly one of the circuit openings is detectably bad:* If one of the openings of the circuit is detectably bad, then $\mathcal{S}$ sends $w_1 = \mathsf{cheat}_1$ to the trusted party. For the sake of concreteness, we describe first the case that the opening of circuit $G(C')_1$ (corresponding to the query $b = 1$) is detectably bad while the other is not:

i. If $\mathcal{S}$ receives the message $\mathsf{corrupted}_1$ from the trusted party, then it rewinds $\mathcal{A}$ to after the point that it sends the garbled circuits and sends it $b = 1$. Then, $\mathcal{S}$ receives back $\mathcal{A}$'s detectably bad opening as above and simulates P_2 aborting due to detected cheating. $\mathcal{S}$ then outputs whatever $\mathcal{A}$ outputs and halts.

ii. If $\mathcal{S}$ receives the message $\mathsf{undetected}$ from the trusted party, then it rewinds $\mathcal{A}$ as above, sends it $b = 0$ and continues to the end of the execution playing the honest P_2 with the input x_2 that it received as well. (When computing the circuit, $\mathcal{S}$ takes the keys from the oblivious transfer that P_2 would have received when using input x_2 and when acting as the honest P_2 to define the values $x_2^1, \ldots, x_2^n$.) Let y_2 be the output that $\mathcal{S}$ received when playing P_2 in this execution. $\mathcal{S}$ sends y_2 to the trusted party (to be the output of P_2) and outputs whatever $\mathcal{A}$ outputs. Note that if the output of P_2 in this execution would have been $\mathsf{corrupted}_1$ then $\mathcal{S}$ sends $y_2 = \mathsf{corrupted}_1$ to the trusted party.[7]

If the opening of the circuit $G(C')_0$ is detectably bad and the other is not, $\mathcal{S}$ works as above except that it reverses the values of b depending on the case.

(c) *Case 3 – neither of the circuit openings are detectably bad:* We begin by defining the notion of $\mathsf{inconsistent\ keys}$, that relates to the question of whether the keys provided by P_1 in the oblivious transfers are the same as those provided in the circuit opening. We say that the keys are $\mathsf{consistent}$ in a circuit if all of the keys provided by P_1 as input in the oblivious transfer are the same as the keys that it provides in the circuit opening. Note that since in this case neither of the circuits are detectably bad, only one of the two keys on any wire can be inconsistent. In this case $\mathcal{S}$ first checks that there is no input bit of P_2 for which all n wires that are associated with it are inconsistent. (Recall that each bit of P_2's input is "split" over n wires. Here $\mathcal{S}$ checks that it does not hold that all of the wires of P_2 for some bit have inconsistent keys.) If there is such a bit, and this holds in both circuits, then $\mathcal{S}$ acts exactly as in the case that both circuits are detectably bad. If this holds in one circuit, then $\mathcal{S}$ acts exactly as in the case that one circuit is detectably bad.[8] Otherwise, it proceeds as follows.

For each wire for which there are inconsistent keys, $\mathcal{S}$ chooses a random key and checks whether it chose any inconsistent key. (We stress that if for some wire w_i, $\mathcal{S}$ chooses

[7] We remark that P_2 may output $\mathsf{corrupted}_1$ with probability that is higher than 1/2. This possibility is dealt with by having $\mathcal{S}$ play P_2 and force a $\mathsf{corrupted}_1$ output if this would have occurred in the execution.

[8] Notice that such a circuit is actually detectably bad because except with probability 2^{-n}, party P_2 will output $\mathsf{corrupted}_1$ if this circuit is opened. This holds because in the protocol, P_2 checks the consistency of the keys obtained from the oblivious transfer protocol and in the circuit opening, and outputs $\mathsf{corrupted}$ if they are not the same.

the key associated with zero, for example, then it chooses the key associated with zero in the wire w_i in *both* circuits.) There are three cases:

i. *Case 3a – $\mathcal{S}$ only chose consistent keys:* In this case, $\mathcal{S}$ proceeds to Step 4d below.

ii. *Case 3b – $\mathcal{S}$ chose inconsistent keys in both circuits:* In this case, $\mathcal{S}$ sends $w_1 = \mathsf{corrupted}_1$ to the trusted party. Next, it rewinds $\mathcal{A}$ to after the point that it sends the garbled circuits and sends it a random bit b. Then, $\mathcal{S}$ receives back the circuit opening to $G(C')_b$ as above and simulates P_2 aborting due to detected cheating. $\mathcal{S}$ then outputs whatever $\mathcal{A}$ outputs and halts.

iii. *Case 3c – $\mathcal{S}$ chose inconsistent keys in exactly one circuit:* In this case, $\mathcal{S}$ sends $w_1 = \mathsf{cheat}_1$ to the trusted party. Then:

   A. If $\mathcal{S}$ receives the message $\mathsf{undetected}$ from the trusted party, it rewinds $\mathcal{A}$ as above and sends it the bit b that opens the circuit for which no inconsistent keys were chosen. Then, P_2 continues to the end of the execution playing the honest P_2 with the input x_2 that it received as well, with one exception: $\mathcal{A}$ uses the same choice of keys that it made above with the wires with inconsistent keys. (Notice that since not all n wires are inconsistent for any bit, $\mathcal{S}$ can complete the keys for any real input bit of P_2 so that it is correct and consistent with the random choices that it made.) Let y_2 be the output that $\mathcal{S}$ received when playing P_2 in this execution (including a possible $\mathsf{corrupted}_1$ that can happen if the inconsistent key does not decrypt the circuit at all). $\mathcal{S}$ sends y_2 to the trusted party (to be the output of P_2) and outputs whatever $\mathcal{A}$ outputs.

   B. If $\mathcal{S}$ receives back the message $\mathsf{corrupted}_1$ from the trusted party, then it rewinds $\mathcal{A}$ to after the point that it sends the garbled circuits and sends it the bit b that opens the circuit for which inconsistent keys were chosen. Then, $\mathcal{S}$ receives back the circuit opening to $G(C')_b$ as above and simulates P_2 aborting due to detected cheating. $\mathcal{S}$ then outputs whatever $\mathcal{A}$ outputs and halts.

(d) $\mathcal{S}$ reaches this point of the simulation if no circuits are detectably bad and if either all keys are consistent or it is simulating the case that no inconsistent keys are discovered. Thus, intuitively, the circuit and keys received by $\mathcal{S}$ from $\mathcal{A}$ are the same as from an honest P_1. The simulator $\mathcal{S}$ begins by rewinding $\mathcal{A}$, handing it a random bit b, and receiving its opening as before. In addition, $\mathcal{S}$ receives from $\mathcal{A}$ the set of keys and decommitments (for wires $w_1, \ldots, w_n$) for the unopened circuit $G(C')_{1-b}$. If the decommitments are invalid, $\mathcal{S}$ sends abort_1 to the trusted party (causing P_2 to output abort_1). Otherwise, $\mathcal{S}$ uses the opening of the circuit $G(C')_{1-b}$ obtained above, together with the keys obtained in order to derive the input x'_1 used by $\mathcal{A}$. This input is derived by comparing the keys for the wires $w_1, \ldots, w_n$ received by $\mathcal{S}$ from $\mathcal{A}$ with the keys provided by $\mathcal{A}$ in the opening of the circuit. This opening provides the association of each keys to a bit – the input x'_1 is derived using this association.

$\mathcal{S}$ sends the trusted party x'_1 (and $\mathsf{continue}$) and outputs whatever $\mathcal{A}$ outputs.

This concludes the description of $\mathcal{S}$. Denote by bad_{OT} the event that $\mathcal{A}$ sends abort_1, $\mathsf{corrupted}_1$ or cheat_1 in the oblivious transfers. The analysis of the event bad_{OT} is identical to the case that P_2 is corrupted and so denoting π as Protocol 6.1 and $I = \{1\}$ (i.e., party P_1 is corrupted), we have that:

$$\left\{\text{IDEALSC}^{\epsilon}_{f,\mathcal{S}(z),I}((x_1,x_2),n) \mid \mathsf{bad}_{\text{OT}}\right\} \equiv \left\{\text{HYBRID}^{\text{OT}}_{\pi,\mathcal{A}(z),I}((x_1,x_2),n) \mid \mathsf{bad}_{\text{OT}}\right\}$$

It remains to analyze the case that $\neg \mathsf{bad}_{\mathrm{OT}}$ (i.e., the oblivious transfer is not aborted). We will prove the case following the same case analysis as in the description of the simulator. Before doing so, notice that $\mathcal{A}$'s view when interacting with $\mathcal{S}$ is identical to its view when interacting with P_2 (when excluding P_2's output from $\mathcal{A}$'s view). This holds because P_2 uses its input only in the oblivious transfer and otherwise just sends a random bit (we will justify that the distribution over the bit b sent by $\mathcal{S}$ is also uniform). Now, when analyzing Protocol 6.1 in a hybrid model with a trusted party computing the oblivious transfer functionality, we have that $\mathcal{A}$'s view is the same in such an execution and in the simulation with $\mathcal{S}$. We remark that P_2's output *is* influenced by its input and so the above argument does not hold when the distribution includes the output. The focus of the proof below is thus to show that the distribution over the bit b sent by $\mathcal{S}$ during the simulation is uniform, and the distribution over the output of P_2 in the simulation is statistically close to its output in a real execution (even when viewed jointly with $\mathcal{A}$'s view).

1. *Case 1 – both circuit openings are detectably bad:* When $\mathcal{A}$ follows such a strategy, P_2 outputs $\mathsf{corrupted}_1$ with probability 1 in a real execution. Likewise, $\mathcal{S}$ sends $\mathsf{corrupted}_1$ in the ideal execution, with the result that P_2's output is $\mathsf{corrupted}_1$, also with probability 1. (Note that by the specification of $\mathcal{S}$ it hands $\mathcal{A}$ a uniformly distributed b, as required.)

2. *Case 2 – exactly one of the circuit openings is detectably bad:* In this case, P_2 outputs $\mathsf{corrupted}_1$ in a real execution with probability at least 1/2. In particular, if it asks $\mathcal{A}$ to open the detectably bad circuit, it certainly outputs $\mathsf{corrupted}_1$. However, it may output $\mathsf{corrupted}_1$ even if the detectably bad circuit is computed (depending on the keys and so on). The point to notice here is that the same distribution on the output is achieved by $\mathcal{S}$. This can be seen as follows.

   $\mathcal{S}$ sends cheat_1 to the trusted party and with probability 1/2 receives back $\mathsf{corrupted}_1$. This event simulates the case that P_2 asked P_1 to open the detectably-bad circuit (and in both the real and ideal executions, P_2 outputs $\mathsf{corrupted}_1$). On the other hand, with probability 1/2, simulator $\mathcal{S}$ receives back $\mathsf{undetected}$. This event simulates the case that P_2 asked P_1 to open the circuit that is not detectably bad. In this case, $\mathcal{S}$ has P_2's true input x_2 and so follows the honest strategy of P_2 (under the constraint that the circuit that is not detectably bad is opened). Clearly, the output distributions in the $\mathsf{undetected}$ case are the same (because $\mathcal{S}$ and P_2 follow exactly the same instructions). Furthermore, since the bit b is chosen according to the $\mathsf{corrupted}_1/\mathsf{undetected}$ response of the trusted party, it equals 0 with probability 1/2 and 1 with probability 1/2, as required.

3. *Case 3 – neither of the circuit openings are detectably bad:* First note that if all n wires of a certain bit of P_2 have inconsistent keys, then P_2 outputs $\mathsf{corrupted}_1$ except with probability 2^{-n} (this may actually depend on P_2's input; however the difference is either $\mathsf{corrupted}_1$ with probability 1 or with probability $1 - 2^{-n}$ and this therefore does not matter). Otherwise, we have the following cases:

   (a) *Case 3a – $\mathcal{S}$ only chose consistent keys:* This case occurs with exactly the same probability as P_2 in a real execution. The analysis of this is covered below.

   (b) *Case 3b – $\mathcal{S}$ chose inconsistent keys in both circuits:* In this case, $\mathcal{S}$ sends $\mathsf{corrupted}_1$. Notice that this case occurs in the simulation also with exactly the same probability as with P_2 in a real execution. However, clearly P_2 outputs $\mathsf{corrupted}_1$ whenever this happens (because when this happens it certainly detects cheating).

(c) *Case 3c – $\mathcal{S}$ chose inconsistent keys in exactly one circuit:* As above, this occurs with the same probability as with P_2 in a real execution. Notice, however, that there are two possibilities: either the circuit for which the inconsistent keys were chosen is opened (and P_2 outputs $\mathsf{corrupted}_1$) or the circuit for which the inconsistent keys is computed (and $\mathcal{A}$ succeeds in cheating). This is analogous to the case of one detectably bad circuit and the analysis is the same.

It remains to analyze the case that $\mathcal{S}$ proceeds to Step 4d in the simulation. We have already seen that $\mathcal{S}$ reaches this step with the correct probability in each case. This step perfectly simulates the case that P_2 opens a circuit that is not detectably bad and did not retrieve any inconsistent keys (in the opened and unopened circuits). Furthermore, by the description of $\mathcal{S}$ in Step 4d we have that the bit b is uniformly distributed. It remains to show that in this case P_2's output in a real execution equals $f(x'_1, x_2)$ where x'_1 is the value that $\mathcal{S}$ extracts from the circuit. First recall that in this case the circuit is correctly constructed and the keys that are provided all decrypt it correctly. That is, the keys that are associated with P_2's input decrypt it correctly (they were obtained in the oblivious transfer stage, but in the opening for the simulator it was verified that they are correct). Furthermore, the keys that are associated with P_1's input also decrypt it correctly. Note that since the keys that are associated with P_1's inputs are committed, $\mathcal{A}$ must use the same keys when opening the circuit and when computing it. Thus, if $\mathcal{A}$ correctly decommits to the keys that are associated with P_1's input in $G(C')_{1-b}$, it follows that P_2 computes the circuit correctly. Furthermore, the input of P_1 is fully defined by the keys that it decommits to (and this is known by $\mathcal{S}$ because it has already seen the circuit opening). Thus, the input x'_1 sent by $\mathcal{S}$ to the trusted party is such that P_2 will either output abort_1 (in the case that $\mathcal{A}$ sends invalid decommitments) or $f(x'_1, x_2)$ (in the case that $\mathcal{A}$ sends valid decommitments). However, this is exactly the same behavior as in the simulation by $\mathcal{S}$. This completes the proof. ■

6.2 Protocols for the Other Security Definitions

We present more efficient protocols for the two other security formulations (versions 1 and 2). 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. Recall that in Protocol 6.1, the input of P_2 is split into n shares. This requires the parties to run n^2 oblivious transfers instead of n (note that there is always one oblivious transfer per input bit of P_2). Since the oblivious transfers are asymmetric operations they are expensive, and thus this is a significant overhead.

Achieving Definition 3.2: For the *failed-simulation formulation*, 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 and is thus far more efficient. The revised protocol provides security in the presence of covert adversaries with deterrence factor 1/2. In order to see why this holds, notice that the issue of P_2's inputs comes into the proof where we discuss *inconsistent keys*. Now, assume that an adversarial P_1 provides one inconsistent key for one of the wires of P_2's input; for the sake of clarity, assume that it is the key associated with 0. Then, if P_2's input bit associated with that wire is 0, then P_2 will output $\mathsf{corrupted}_1$ if the circuit is opened and will possibly output something incorrect if the circuit is computed (each event happens with probability 1/2). In contrast, if P_2's input bit associated with that wire is 1, then the computation will always conclude correctly. Thus, in both cases, the probability that P_2 outputs $\mathsf{corrupted}_1$ is at least 1/2 times the distinguishing gap. A similar analysis follows if it provides inconsistent keys in both circuits. We stress that P_1 learns the exact value of P_2's input bit for that wire (by just

observing if it outputs $\mathsf{corrupted}_1$ or not) and thus this is not secure under the standard definition of security for malicious adversaries. However, according to Definition 3.2, if P_2 outputs $\mathsf{corrupted}_1$ with probability at least 1/2 (as in the case that its input bit is 0), then nothing is required and the IDEAL and REAL distributions are allowed to be easily distinguished. We also remark that this protocol does *not* meet Definition 3.4 or 3.5 because the decision of whether to have P_2 output $\mathsf{corrupted}_1$ can depend on P_2's actual input. However, observe that in the IDEALC and IDEALSC ideal models, the adversary must decide to send cheat or $\mathsf{corrupted}$ before it learns anything about the honest parties' inputs. We remark that this protocol has a complexity that is between two and four times that of Yao's protocol for the semi-honest case.[9]

Achieving Definition 3.4: For the *explicit cheat formulation* (not strong), we split the input of P_2 into 2 shares, instead of n. This modification reduces the number of oblivious transfers from n^2 to $2n$ and so once again is far more efficient than Protocol 6.1. This version of the protocol provides security for covert adversaries with deterrence 1/4 under Definition 3.4. In order to see why this holds, notice once again that this has an influence only on the analysis of inconsistent keys for P_2's wires, and hence on the construction of the simulator for the case when P_1 is corrupted. The simulator for this Definition is similar to the one described above, but simpler. Recall that in Definition 3.4 the adversary always gets the honest player's whenever it chooses to cheat. Also note that the simulator described above always detects an attempt cheat by P_1. Thus, in any case of an attempted cheat, the simulator can simulate the interaction simply by obtaining P_2's input and simulating the interaction with P_1. It thus remains to show that any attempted cheat results in $\mathsf{corrupted}_1$ with probability at least 1/4. Indeed, for all cases except for inconsistent keys, we have shown that the probability is at least 1/2. For the case of inconsistent keys, there must be at least one inconsistent key, in at least one of the wires in at least one of the garbled circuits. The probability to choose this circuit is 1/2, and the probability of choosing the inconsistent key is again 1/2 - for a total of at least 1/4.

6.3 Higher Deterrence Values

For all three versions of our protocol, it is possible to boost the deterrence value to $1 - 1/\text{poly}(n)$, with an increased price in performance. We demonstrate this for Protocol 6.1 only (and with respect to Definition 3.5). Let $p(\cdot)$ be a polynomial. Then, Protocol 6.1 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 6.1 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. The proof of this protocol is very similar to that of Protocol 6.1. The only difference is with the cases considered; instead of considering the case of one circuit versus both circuits (regarding detectably bad and inconsistent keys), we consider the case of one circuit versus *more than one* circuit.

[9] On the one hand, n oblivious transfers are used in both cases, so this does not "cost any more" here than in the semi-honest case. However, the oblivious transfer protocol of Section 5 is about 4 times the cost of a semi-honest equivalent. The rest of the protocol amounts to about twice that of a plain semi-honest version of Yao's protocol.

6.4 Non-Halting Detection Accuracy

It is possible to modify Protocol 6.1 so that it achieves *non-halting detection accuracy*; see Definition 3.3. Before describing how we do this, notice that the reason that we need to recognize a halting-abort as cheating in Protocol 6.1 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. In order to describe this in more detail, we restate the circuit opening stage of Protocol 6.1 as follows:

1. Party P_1 sends two garbled circuits $G(C')_0$ and $G(C')_1$ to party P_2.
2. P_2 sends a random challenge bit b.
3. P_1 opens $G(C')_b$ by sending decommitments, keys and so on. In addition, it sends the keys associated with its own input in $G(C')_{1-b}$.
4. P_2 checks the circuit $G(C')_b$ and computes $G(C')_{1-b}$ (using the keys from P_1 in the previous step and the keys it obtained earlier in the oblivious transfers). P_2's output is defined to be the output of $G(C')_{1-b}$.

Notice that P_2 only outputs $\mathsf{corrupted}_1$ if the checks from the circuit that is opened do not pass. As we have mentioned, there is no logical reason why an adversarial P_1 would ever actually reply with an invalid opening; rather it would just abort. Consider now the following modification:

1. Party P_1 sends two garbled circuits $G(C')_0$ and $G(C')_1$ to party P_2.
2. P_1 and P_2 participate in a (fully secure) oblivious transfer with the following inputs:
   (a) P_1 defines its input (x_0, x_1) as follows. Input x_0 consists of the opening of circuit $G(C')_0$ together with the keys associated with its own input in $G(C')_1$. Input x_1 consists of the opening of circuit $G(C')_1$ together with the keys associated with its own input in $G(C')_0$.
   (b) P_2's input is a random bit b.
3. P_2 receives an opening of one circuit together with the keys needed to compute the other and proceeds as above.

Notice that this modified protocol is essentially equivalent to Protocol 6.1 and thus its proof of security is very similar. However, in this case, an adversarial P_1 who constructs one faulty circuit must decide *before* the oblivious transfer if it wishes to abort (in which case there is no successful cheating) or if it wishes to proceed (in which case, P_2 will receive an explicitly invalid opening). Note that due to the security of the oblivious transfer, P_1 cannot know what value b party P_2 inputs, and so cannot avoid being detected.

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. (We remark that one should not be concerned with the length of x_0 and x_1 in P_1's input to the oblivious transfer. This is because P_1 can send them encrypted ahead of time with independent symmetric keys k_0 and k_1. Then the oblivious transfer takes place only on the keys.)

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), pages 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), pages 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. *Cryptology ePrint Archive,* Report 2004/175, 2004. To appear in the *Journal of Cryptology.*

[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 Security Symposium*, pages 287–302, 2004.

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

[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.

A Yao's Protocol for Semi-Honest Adversaries

We now describe Yao's protocol for secure two-party computation (in the presence of semi-honest adversaries) which is proven secure in [18]. Yao's protocol is based on the following "garbled-circuit" construction.

The garbled circuit construction. Let C be a Boolean circuit that receives two inputs $x_1, x_2 \in \{0,1\}^n$ and outputs $C(x_1, x_2) \in \{0,1\}^n$ (for simplicity in this description, 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.[10] (Likewise, if a circuit-input wire is itself also a circuit-output, then it is not input into any gate.) The reduction uses a private key encryption scheme (G, E, D) that has indistinguishable encryptions for multiple messages, and also a special property called an elusive efficiently verifiable range; see [18].[11]

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,

[10] This requirement is due to our labelling of gates described below, that does not provide a unique label to each wire (see [18] for more discussion). We note that this assumption on C increases the number of gates by at most n.

[11] Loosely speaking, an encryption scheme has an elusive range if without knowing the key, it is hard to generate a ciphertext that falls in the range. An encryption scheme has a verifiable range if given the key and a ciphertext, it is easy to verify that the ciphertext is in the range. Such encryption schemes can be constructed using pseudorandom functions by encrypting the message together with n zeroes. It is easy to see that this provides both an elusive range and an efficiently verifiable one. We denote by $\perp$ the result of decrypting a value not in the range.

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

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

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. By the properties of the encryption scheme, it can be shown 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 $G(C)$, 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.

Let $x_1 = x_1^1 \cdots x_1^n$ and $x_2 = x_2^1 \cdots x_2^n$ be two n-bit inputs for C. Furthermore, let $w_1, \ldots, w_n$ be the input labels corresponding to x_1, and let $w_{n+1}, \ldots, w_{2n}$ be the input labels corresponding to x_2. It is shown in [18] that given the garbled circuit $G(C)$ and the strings $k_1^{x_1^1}, \ldots, k_n^{x_1^n}, k_{n+1}^{x_2^1}, \ldots, k_{2n}^{x_2^n}$, it is possible to compute $C(x_1, x_2)$, except with negligible probability.

Yao's protocol. Yao's protocol works by designating one party, say P_1, to be the circuit constructor. P_1 builds a garbled circuit to compute f and hands it to P_2. In addition, P_1 sends P_2 the keys $k_1^{x_1^1}, \ldots, k_n^{x_1^n}$ that are associated with its input x_1. Finally, P_2 obtains the keys $k_{n+1}^{x_2^1}, \ldots, k_{2n}^{x_2^n}$ associated with its input via (semi-honest) oblivious transfer. That is, for every $i = 1, \ldots, n$, parties P_1 and P_2 run an oblivious transfer protocol. In the i^{th} execution, P_1 plays the sender with inputs (k_{n+i}^0, k_{n+i}^1) and P_2 plays the receiver with input x_2^i. Following this, P_2 has the keys $k_1^{x_1^1}, \ldots, k_n^{x_1^n}, k_{n+1}^{x_2^1}, \ldots, k_{2n}^{x_2^n}$ and so, as stated above, it can compute the circuit to obtain $C(x_1, x_2)$. Furthermore, since it has only these keys, it cannot compute the circuit for any other input.

A Lemma. In our proof of security, we will use the following lemma:

Lemma A.1 *Given a circuit C with inputs wires $w_1, \ldots, w_{2n}$ and an output value y (of the same length as the output of C) it is possible to efficiently construct a garbled circuit $\tilde{G}(C)$ such that:*

1. *The output of $\tilde{G}(C)$ is always y, regardless of the garbled values that are provided for P_1 and P_2's input wires, and*
2. *If $y = f(x_1, x_2)$, then no non-uniform probabilistic polynomial-time adversary $\mathcal{A}$ can distinguish between the distribution ensemble consisting of $\tilde{G}(C)$ and a single arbitrary key for every input wire, and the distribution ensemble consisting of a real garbled version of C, together with the keys $k_1^{x_1^1}, \ldots, k_n^{x_1^n}, k_{n+1}^{x_2^1}, \ldots, k_{2n}^{x_2^n}$.*

Proof Sketch: The proof of this lemma is taken from [18] (it is not stated in this way there, but is proven). We sketch the construction of $\tilde{G}(C)$ here for the sake of completeness, and refer the reader to [18] for a full description and proof. The first step in the construction of the fake circuit $\tilde{G}(C)$ 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 y by $y_1 \cdots y_n$, 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 $y_i = 0$, and $[(0, k'_{m-n+i}), (1, k_{m-n+i})]$ if $y_i = 1$. 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 $y = y_1 \cdots y_n$ 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 [18]. ■