

# Adaptively Secure Two-Party Computation with Erasures\*

Yehuda Lindell<sup>†</sup>

October 18, 2010

## Abstract

In the setting of multiparty computation a set of parties with private inputs wish to compute some joint function of their inputs, whilst preserving certain security properties (like privacy and correctness). An adaptively secure protocol is one in which the security properties are preserved even if an adversary can adaptively and dynamically corrupt parties during a computation. This provides a high level of security, that is arguably necessary in today's world of active computer break-ins. Until now, the work on adaptively secure multiparty computation has focused almost exclusively on the setting of an honest majority, and very few works have considered the honest minority and two-party cases. In addition, significant computational and communication costs are incurred by most protocols that achieve adaptive security.

In this work, we consider the two-party setting and assume that honest parties may *erase* data. We show that in this model it is possible to securely compute any two-party functionality in the presence of *adaptive semi-honest adversaries*. Furthermore, our protocol remains secure under concurrent general composition (meaning that it remains secure irrespective of the other protocols running together with it). Our protocol is based on Yao's garbled-circuit construction and, importantly, is as efficient as the analogous protocol for static corruptions. We argue that the model of adaptive corruptions with erasures has been unjustifiably neglected and that it deserves much more attention.

## 1 Introduction

In the setting of multiparty computation, a set of parties with private inputs wish to jointly compute some function of those inputs. Loosely speaking, the security requirements are that even if some of the participants are adversarial, nothing is learned from the protocol other than the output (*privacy*), and the output is distributed according to the prescribed functionality (*correctness*). The definition of security that has become standard today [17, 24, 1, 5] blends these two conditions (and adds more). This setting models essentially any cryptographic task of interest, including problems ranging from key exchange and authentication, to voting, elections and privacy-preserving data mining. Due to its generality, understanding what can and cannot be computed in this model, and at what complexity, has far reaching implications for the field of cryptography.

---

\*An extended abstract of this work appeared in *CT-RSA*, 2009. This research was supported by the ISRAEL SCIENCE FOUNDATION (grant No. 781/07).

<sup>†</sup>Bar-Ilan University, Israel. Email: lindell@cs.biu.ac.il.

One important issue regarding secure computation is the environment in which it takes place. In the basic setting, called the **stand-alone model**, the secure protocol is run only once and in isolation (or equivalently, the adversary attacks only this execution). A more advanced setting is that of composition, where the secure protocol may be run many times concurrently with itself and arbitrary other protocols. This setting is called **universal composability** [6], or equivalently **security under concurrent general composition** [20], and more accurately models the real-world security needs than the stand-alone model.

A central question that needs to be addressed in this setting is the power of the adversary. An adversary can be **semi-honest** (in which case it follows the protocol specification exactly but attempts to learn more information than it should by analyzing the messages it receives) or **malicious** (in which case it can take arbitrary actions). In addition, one can consider **static corruptions** (meaning that the set of corrupted parties that are under the control of the adversary is fixed before the protocol execution begins) or **adaptive corruptions** (in which case the adversary can choose to corrupt parties during the computation based on its view). There are two main models that have been considered for adaptive corruptions. In both models, upon corrupting an honest party the adversary receives the internal state of that party. The difference lies in the question of what is included in that state. In the **non-erasure model**, honest parties are not assumed to be able to reliably erase data. Therefore, the internal state of a party includes its input, randomness and *all* of the messages that it received in the past. In contrast, in the **model with erasures**, honest parties may erase data if so instructed by the protocol and so the state includes all data as above except for data that has been explicitly erased. (Of course, not all intermediate data can be erased because the party needs to be able to run the protocol and compute the output.) In this paper, we consider the problem of achieving security in the presence of *adaptive semi-honest adversaries with erasures*. We remark that adaptive corruptions model the setting where hackers actively break in to computers during secure computations. As such, it more accurately models real-world threats than the model of static corruptions.

**To erase or not to erase.** In the cryptographic literature, the non-erasure model of adaptive corruptions has received far more attention than the erasure model (see prior work below). The argument has typically been that it is generally hard to ensure that parties fully erase data. After all, this can depend on the operating system, and in real life passwords and other secret data can often be found on swap files way after they were supposedly erased. We counter this argument by commenting that non-swappable memory is provided by all modern operating systems today and it is possible to use this type of memory for the data which is to be erased (as specified by the protocol). Of course, it is more elegant to assume that there are no erasures. However, the price of this assumption has been very great. That is, the complexity and communication of protocols that are secure under adaptive corruptions without erasures are all much higher than the analogous protocols that are secure under static corruptions (for example, we don't even have a constant-round protocol for general two-party computation that is secure under adaptive corruptions). We argue that the result of this is that no protocol designer today would even consider adaptive corruptions if the aim is to construct an efficient protocol that could possibly be used in practice.

**Our results.** We begin by studying the stand-alone model and note that the current situation is actually very good, as long as erasures are considered. Specifically, by combining results from Beaver and Haber [3], Canetti [5], and Canetti et al. [7], we show the following:

**Theorem 1** *Let  $f$  be any two-party functionality and let  $\pi$  be a protocol that securely computes  $f$  in the presence of static malicious (resp., semi-honest) adversaries, in the stand-alone model. Then, assuming the existence of one-way functions, there exists a highly efficient transformation of  $\pi$  to  $\pi'$  such that  $\pi'$  securely computes  $f$  in the presence of adaptive malicious (resp., semi-honest) adversaries with erasures, in the stand-alone model.*

We have no technical contribution in deriving Theorem 1; rather our contribution here is to observe that a combination of known results yields the theorem. To the best of our knowledge, the fact that this theorem holds has previously gone unnoticed.

The only drawback of Theorem 1 is that it holds only for the stand-alone model (see Section 3 for an explanation as to why). As we have mentioned, this is a relatively unrealistic model in today's world where many different protocols are run concurrently. Our main technical contribution is therefore to show that it is possible to construct secure protocols for general two-party computation in the presence of semi-honest adaptive adversaries (and with erasures) that are secure under concurrent general composition [20] (equivalently, universally composable [6]). Importantly, the complexity of our protocol is analogous to the complexity of the most efficient protocol known for the case of semi-honest static adversaries (namely, Yao's protocol [26]). We prove the following theorem:

**Theorem 2** *Assume that there exist enhanced<sup>1</sup> trapdoor permutations. Then, for every two-party probabilistic polynomial-time functionality  $f$  there exists a constant-round protocol that securely computes  $f$  under concurrent general composition, in the presence of adaptive, semi-honest adversaries with erasures.*

The contributions of Theorem 2 are as follows:

1. *Round complexity:* Our protocol for adaptive two-party computation requires a constant number of rounds. The only other protocols for general adaptive two-party computation that are secure under concurrent composition follow the GMW paradigm [16] and the number of rounds is therefore equal to the depth of the circuit that computes the function  $f$ ; see [10]. We stress that [10] does not assume erasures, whereas we do.
2. *Hardness assumptions:* Our protocol requires the minimal assumption for secure computation in the static model of enhanced trapdoor permutations. In contrast, all known protocols for adaptive oblivious transfer (and thus adaptive secure computation) without erasures assume seemingly stronger assumptions, like a public-key cryptosystem with the ability to sample a public-key without knowing the corresponding secret key. In fact, in a recent paper, it was shown that adaptively secure computation *cannot* be achieved in a black-box way from enhanced trapdoor permutations alone [23]. Thus, without assuming erasures, it is not possible to construct secure protocols for the adaptive model under this minimal assumption (at least, not in a black-box way).

In addition to the above, our protocol has the same complexity as Yao's protocol for static adversaries in all respects. We view this as highly important and as a sign that it is well worth considering this model when constructing efficient secure protocols. In particular, if it is possible to achieve

---

<sup>1</sup>See [15, Appendix C.1].

security in the presence of adaptive corruptions with erasures “for free”, then this provides a significant advantage over protocols that are only secure for static corruptions (of course, as long as such erasures can really be carried out). In addition, the typical argument against considering security under composition is that the resulting protocols are far less efficient. Our results demonstrate that this is not the case for the setting of semi-honest adaptive adversaries with erasures. Indeed our protocol is no less efficient than the basic protocol for the semi-honest stand-alone setting.

We remark that our protocol is very similar to the protocol of Yao and we only need to slightly change the order of some operations and include some erase instructions (i.e., in some sense, the original protocol is “almost” adaptively secure). Nevertheless, our *proof of security* is significantly different and requires a completely different simulation strategy to that provided in [21].

**Related work.** The vast majority of work on adaptive corruptions for secure computation has considered the setting of multiparty computation with an honest majority [3, 8, 2] and thus is not applicable to the two-party setting. To the best of our knowledge, the only two works that considered the basic question of adaptive corruptions for *general secure computation* in the setting of no honest majority are [10] and [7]. Canetti et al. [7] study the relation between adaptive and static security and present a series of results that greatly clarifies the definitions and their differences. However, this work only relates to the stand-alone setting. In the setting of composition, Canetti et al. [10] presented a protocol that is universally composable (equivalently, secure under concurrent general composition). The construction presented there considers a model with no erasures. As such, it is far less efficient (e.g., it is not constant-round), far more complicated, and relies on seemingly stronger cryptographic hardness assumptions. Regarding secure computation of functions of specific interest, there has also been little work on achieving adaptive security, with the notable exception of threshold cryptosystems [9, 13, 19] and oblivious transfer [14].

## 2 Definitions

### 2.1 Definitions of Security

We denote the security parameter by  $n$ . A function  $\mu(\cdot)$  is **negligible** in  $n$  (or just **negligible**) if for every polynomial  $p(\cdot)$  there exists a value  $N$  such that for all  $n > N$  it holds that  $\mu(n) < 1/p(n)$ . A machine is said to run in **polynomial-time** if its number of steps is polynomial in the *security parameter*, irrespective of the length of its input. Formally, each machine has a security-parameter tape upon which  $1^n$  is written. The machine is then polynomial in the contents of this tape.

Let  $X = \{X(n, a)\}_{n \in \mathbb{N}, a \in \{0,1\}^*}$  and  $Y = \{Y(n, a)\}_{n \in \mathbb{N}, a \in \{0,1\}^*}$  be distribution ensembles. Then, we say that  $X$  and  $Y$  are **computationally indistinguishable**, denoted  $X \stackrel{c}{\equiv} Y$ , if for every non-uniform polynomial-time distinguisher  $D$  there exists a function  $\mu(\cdot)$  that is negligible in  $n$ , such that for every  $a \in \{0, 1\}^*$ ,

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

Typically, the distributions  $X$  and  $Y$  will denote the output vectors of the parties in real and ideal executions, respectively. In this case,  $a$  denotes the parties’ inputs.

### 2.2 Stand-Alone versus Composition

The definitions for security in the stand-alone model are significantly simpler than those for security under concurrent general composition (or the equivalent definition of universal composable).

Kushilevitz et al. [18] showed that any protocol that has been proven secure in the stand-alone model, using a simulator that is *black-box* and *straight-line* and so doesn't rewind the adversary, is secure under concurrent-general composition. In actuality, an additional requirement for this to be true is something called *initial synchronization*. In the two-party setting, this just means that the parties send each other an init message before actually running the protocol.

Due to the above, we will present the definitions for the stand-alone model only and will derive security under concurrent general composition via the fact that all of our simulators are black-box and straight-line. (In fact, most known simulators for the semi-honest model are black-box and straight-line, with one notable exception being the simulators of [7]. See Section 3 for more discussion on this.)

### 2.3 Secure Two-Party Protocols for Semi-Honest Adversaries

The model that we consider here is that of two-party computation in the presence of *adaptive semi-honest* adversaries with *erasures*. An adaptive adversary can choose to corrupt parties at any time throughout the computation (including before it begins and after it terminates), where upon corruption the adversary receives the entire internal state of the party. (This is like breaking into the honest party's machine during the execution.) The fact that the adversary is semi-honest means that even the corrupted parties follow the protocol specification exactly. However, the adversary may try to learn more information than allowed by looking at the transcript of the messages received by the corrupted parties. We consider a model with erasures, meaning that the protocol may contain instructions to honest parties to erase data; it is assumed that if such an instruction is executed, the erased data is not available to the adversary if it corrupts the party at a later time. The definitions presented here are according to Canetti in [5], with the appropriate changes needed for considering erasures. Specifically, no post-execution phase is needed, and there is no need to include the external environment  $\mathcal{Z}$ ; see [5, Remark 5].

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

**Security – informal.** Intuitively, a protocol is secure if whatever can be computed by a party participating in the protocol can be computed based on its input and output only. This is formalized according to the ideal/real model paradigm. Loosely speaking, we require that the output of a real protocol execution be indistinguishable from the output of an *ideal* computation involving an incorruptible trusted third party. This trusted party receives the parties' inputs, computes the functionality on these inputs and returns to each their respective output. In addition, the adversary can issue "corrupt" commands upon which it receives the input and output of the newly corrupted party. Loosely speaking, a protocol is secure if any real-model adversary can be converted into an ideal-model adversary such that the output distributions of the real and ideal executions are computationally indistinguishable.

**Definition of security.** In order to define security, we first need to define the ideal and real models. We begin with an ideal execution between two parties  $P_1$  and  $P_2$ , an adaptive semi-honest adversary  $\mathcal{A}$  and a trusted third party:

**Inputs:** Each party ( $P_1$ ,  $P_2$  and  $\mathcal{A}$ ) has the security parameter written in unary form on its security parameter tape. Parties  $P_1$  and  $P_2$  obtain respective inputs  $x$  and  $y$ , and the adversary  $\mathcal{A}$  receives an auxiliary input  $z$ .

**First corruption phase:** The adversary  $\mathcal{A}$  can issue any `corrupt`  $P_i$  commands it wishes (for  $i \in \{1, 2\}$ ). Upon issuing such a command it receives the appropriate party's input. We stress that  $\mathcal{A}$  can issue any number of these commands in any order, and its decision can depend on the values it has already seen.

**Computation stage:** Each party  $P_i$  sends its input to the trusted party. The trusted party then computes  $(w_1, w_2) = f(x, y)$  and hands each  $P_i$  the value  $w_i$ . The adversary  $\mathcal{A}$  receives this output for any party that is already corrupted.

**Second corruption phase:** After receiving the outputs of the corrupted parties (if there are any), the adversary proceeds to another corruption phase which is the same as the first.

**Output:** Each uncorrupted party  $P_i$  outputs  $w_i$  as received from the trusted party and corrupted parties output  $\perp$ . The adversary  $\mathcal{A}$  outputs an arbitrary function of its view in the execution.

Let  $f : \{0, 1\}^* \times \{0, 1\}^* \rightarrow \{0, 1\}^* \times \{0, 1\}^*$  be a functionality, where  $f = (f_1, f_2)$ , and let  $\mathcal{A}$  be a non-uniform probabilistic polynomial-time machine (representing the adversary). The execution of  $f$  in the ideal model (on input  $(x, y)$  for the parties and auxiliary input  $z$  for the adversary), denoted  $\text{IDEAL}_{f, \mathcal{A}(z)}(n, x, y)$ , is defined as the output of the adversary  $\mathcal{A}$  and the parties  $P_1$  and  $P_2$  from the above ideal execution.

**Execution in the real model (with erasures).** We next consider the real model in which a real two-party protocol is executed (and there exists no trusted third party). Each party ( $P_1$ ,  $P_2$  and  $\mathcal{A}$ ) is invoked with the security parameter  $n$  written in unary on its security parameter tape and with its designated input ( $x$  for  $P_1$ ,  $y$  for  $P_2$ , and  $z$  for  $\mathcal{A}$ ). The parties then run the protocol according to its specification, including erase commands. The adversary may corrupt honest parties at any time before, during or after the computation. Upon corruption, the adversary receives the internal state of the party that includes the entire history of the execution, except for data that has been explicitly erased. Furthermore, the adversary continues to view all messages that the corrupted party receives. We stress that we consider the semi-honest adversary model here and so the corrupted parties continue to follow the protocol specification. At the end of the execution, the honest parties output whatever the protocol instructs them to output, the corrupted parties output  $\perp$ , and the adversary  $\mathcal{A}$  outputs an arbitrary function of its view in the execution.

Let  $f$  be as above and let  $\pi$  be a two-party protocol for computing  $f$ . The execution of  $\pi$  in the real model (on input  $(x, y)$  for the parties, and auxiliary input  $z$  for the adversary), denoted  $\text{REAL}_{\pi, \mathcal{A}(z)}(n, x, y)$ , is defined as the output of the adversary  $\mathcal{A}$  and the parties  $P_1$  and  $P_2$  from the above real execution.

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

**Definition 3** Let  $f$  and  $\pi$  be as above. Protocol  $\pi$  is said to securely compute  $f$  in the adaptive semi-honest model with erasures 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:

$$\left\{ \text{IDEAL}_{f, \mathcal{S}(z)}(n, x, y) \right\}_{x, y, z \in \{0,1\}^*; n \in \mathbb{N}} \stackrel{c}{\equiv} \left\{ \text{REAL}_{\pi, \mathcal{A}(z)}(n, x, y) \right\}_{x, y, z \in \{0,1\}^*; n \in \mathbb{N}}$$

where  $|x| = |y|$ .

We note that the above definition assumes that the parties know the input lengths (this can be seen from the requirement that  $|x| = |y|$ ). Some restriction on the input lengths is unavoidable, see [15] for discussion.

## 2.4 Sequential Composition

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

We note that the definition of security that appears in [5] for the case of adaptive corruptions – and for which the composition theorem is proved – is significantly more complex than our definition above (specifically, it includes a post-execution corruption phase and a separate environmental entity). Nevertheless, as pointed out by [5], this additional complication is not needed in the case that honest parties can erase data. Rather, it suffices to adopt the convention that at the end of every execution of a secure protocol the parties erase all the data that was internal to the execution except for the output.

## 3 Stand-Alone Two-Party Computation for Malicious Adversaries

In this section, we observe that in the *stand-alone model*, any two-party protocol that is secure in the presence of static adversaries can be efficiently converted into a protocol that is secure in the presence of adaptive adversaries, as long as erasures are allowed. This powerful result is another good reason why it is worth considering erasures – indeed, adaptive security is obtained almost for free. In order to see why this is true, we combine three different results:

1. First, Beaver and Haber [3] proved that any protocol that is adaptively secure in a model with ideally secure communication channels can be efficiently converted into a protocol that is adaptively secure in a model with regular (authenticated) communication channels, assuming

erasures. The transformation of [3] requires public-key encryption and thus assumes the existence of trapdoor permutations. We stress that under specific assumptions, it can be implemented at very low cost.

2. Next, Canetti et al. [7] consider a modification of the standard definition of security for adaptive adversaries. The standard definition includes a *post-execution corruption phase* (known as PEC for short); this phase is necessary for obtaining sequential composition as described in [5].<sup>2</sup> Nevertheless, most of the results in [7] consider a modified definition where there is no PEC phase. Amongst many other results, it is proven in [7] that in a model with ideally secure channels and no PEC, any two-party protocol that is secure in the presence of static malicious adversaries is also secure in the presence of adaptive malicious adversaries. (The same holds also for semi-honest adversaries.)

A combination of the results of [3] and [7] yields the result that any two-party protocol that is secure in the presence of static malicious adversaries can be efficiently transformed into a protocol that is secure in the presence of adaptive malicious adversaries under a definition without PEC. (The requirement of [7] for ideally secure channels is removed by [3].) This result is still somewhat lacking because PEC is in general a necessary definitional requirement.

3. The post-execution corruption phase is not needed in the adaptive model where erasures are allowed [5, Remark 5]. In particular, modular sequential composition holds in this model even without this phase. (There is one requirement: the honest parties must erase the internal data they used at the end of the secure protocol execution, and can store only the input and output. There is no problem doing this because the input and output is all that they need.) Thus, in a model allowing erasures, the results of [7] guarantee adaptive security under a definition of security that is sufficient (and in particular implies sequential composition).

Combining the above three observations, we obtain the following theorem:

**Theorem 4** *Consider the stand-alone model of computation. Let  $f$  be a two-party functionality and let  $\pi$  be a protocol that securely computes  $f$  in the presence of malicious static adversaries. Then, assuming the existence of trapdoor permutations, there exists a protocol  $\pi'$  that securely computes  $f$  in the presence of malicious adaptive adversaries, with erasures.*

The theorem statement hides the fact that the transformation of  $\pi$  to  $\pi'$  is highly efficient and thus the boosting of the security guarantee from static to adaptive adversaries is obtained at almost no cost. Before concluding, we stress again that this result *only holds in a model assuming that honest parties can safely erase data*. This is due to the fact that in the non-erasure model the PEC requirement *is* needed, and so the combination of the results of [3] and [7] yields a protocol that is not useful. (In particular, it is not necessarily secure with PEC and so may not be secure under sequential composition.)

**Concurrent composition.** The above relates to the stand-alone model (and so, of course, also to sequential composition). What happens when considering concurrent composition? An analogous result cannot be achieved because the proof of [7] relies inherently on the fact that the simulator can *rewind* the adversary. Specifically, [7] prove the equivalence of adaptive and static security in the following way. The adaptive simulator begins by running the static simulator for the case that no

---

<sup>2</sup>We remark that there is no PEC requirement for the definition of security in the presence of static adversaries.

party is corrupted. Then, if the adversary corrupts a party (say party  $P_1$ ), the adaptive simulator rewinds the adversary and begins the simulation from scratch running the static simulator for the case that  $P_1$  is corrupted. The adaptive simulator runs the static simulator multiple times until the adversary asks to corrupt  $P_1$  in the same place as the first time. Since the static simulator assumes that  $P_1$  is corrupted, it can complete the simulation. This is the general idea of the simulation strategy; for more details and the actual proof, see [7]. In any case, since rewinding is an inherent part of the strategy, their proof cannot be used in the setting of concurrent composition (where rewinding simulation strategies do not work).

## 4 Two-Party Computation for Semi-Honest Adversaries

### 4.1 Adaptively-Secure Oblivious Transfer

We start by observing that in the setting of concurrent composition, the oblivious transfer protocol of [12] is *not* adaptively secure (at least, it is not simulatable without rewinding). In order to see this, recall that this protocol works by the sender  $P_1$  choosing an enhanced trapdoor permutation  $f$  with its trapdoor  $t$  and sending  $f$  to the receiver  $P_2$ . Upon input  $\sigma$ , party  $P_2$  then sends  $P_1$  values  $y_0, y_1$  so that it knows the preimage of  $y_\sigma$  but not of  $y_{1-\sigma}$ . Party  $P_1$  then masks its input bit  $z_0$  with the hard-core bit of  $f^{-1}(y_0)$  and masks its input bit  $z_1$  with the hard-core bit of  $f^{-1}(y_1)$ . The protocol concludes by  $P_2$  extracting  $z_\sigma$ ; it can do this because it knows the preimage of  $y_\sigma$  and so can compute the hard-core bit used to mask  $z_\sigma$ . Now, consider an adversarial strategy that waits until  $P_1$  sends its second message and then corrupts the receiver. Following this corruption, the adversary should be able to obtain  $P_2$ 's state and compute  $z_\sigma$  from  $P_1$ 's message (the adversary must be able to do this because  $P_2$  must be able to do this). However, when the messages from  $P_1$  are generated by the simulator and no party is corrupted, the simulator cannot know what values of  $z$  to place in the message. The simulation will therefore often fail. A similar (and in fact worse) problem appears in the known oblivious transfer protocols that rely on homomorphic encryption.

Our approach to solving this problem is novel and has great advantages. We show that *any* oblivious transfer protocol that is secure for static corruptions can be modified so that with a minor addition adaptive security (with erasures) is obtained. The idea is to run *any* secure oblivious transfer upon random inputs and then use the random values obtained to exchange the actual bit. This method was presented in [25] in order to show a reduction from standard OT to OT with random inputs. Here we use it to obtain adaptive security. We remark that our protocol is exactly that of [25]; our contribution is in observing and proving that it is adaptively secure.

**Protocol 1** (oblivious transfer):

- **Inputs:**  $P_1$  has two strings  $x_0, x_1 \in \{0, 1\}^n$ , and  $P_2$  has a bit  $\sigma \in \{0, 1\}$ .
- **The protocol:**
  1.  $P_1$  chooses random strings  $r_0, r_1 \in_R \{0, 1\}^n$  and  $P_2$  chooses a random bit  $b \in_R \{0, 1\}$ .  $P_1$  and  $P_2$  run an oblivious transfer protocol, using the chosen random inputs. (Note that  $P_2$ 's output is  $r_b$ .) At the conclusion of the protocol,  $P_1$  and  $P_2$  erase all of the randomness that they used, and remain only with their inputs and outputs from the subprotocol (i.e.,  $P_1$  remains with  $(r_0, r_1)$  and  $P_2$  remains with  $(b, r_b)$ ).
  2.  $P_2$  sends  $P_1$  the bit  $\beta = b \oplus \sigma$ .

3.  $P_1$  sends  $P_2$  the pair  $y_0 = x_0 \oplus r_\beta$  and  $y_1 = x_1 \oplus r_{1-\beta}$ .
4.  $P_2$  outputs  $y_\sigma \oplus r_b$ .

Before proving security, we first show that the protocol is correct. If  $\sigma = 0$  then  $\beta = b$  and so  $y_0 = x_0 \oplus r_b$ , implying that  $P_2$  outputs  $y_0 \oplus r_b = x_0$  as required. Likewise, if  $\sigma = 1$  then  $\beta = b \oplus 1$  and so  $y_1 = x_1 \oplus r_b$ , implying that  $P_2$  outputs  $y_1 \oplus r_b = x_1$  as required. We have the following theorem.

**Theorem 5** *If the oblivious transfer used in Step 1 of Protocol 1 is secure in the presence of semi-honest static adversaries in the stand-alone model, then Protocol 1 is secure under concurrent general composition in the presence of semi-honest adaptive adversaries with erasures.*

**Proof:** Before beginning the proof, we remark that we cannot analyze the security of the protocol in a hybrid model where the oblivious transfer of Step 1 is run by a trusted party. This is because the oblivious transfer protocol of Step 1 is only secure in the presence of static adversaries, and we are working in the adaptive model. We now proceed with the proof. Intuitively, the protocol is adaptively secure because any corruptions that occur before Step 1 are easily dealt with due to the fact that even the honest parties use random inputs in this stage (and thus inputs that are independent of their real input). Furthermore, any corruptions that take place after Step 1 can be dealt with because the oblivious transfer protocol used in Step 1 is statically secure, and so hides the actual inputs used. Given that the parties erase their internal state after this step, the simulator is able to lie about what “random” inputs the parties actually used.

Let  $\mathcal{A}$  be a probabilistic polynomial-time real adversary. We construct a simulator  $\mathcal{S}$  for Protocol 1, separately describing its actions for every corruption case (of course,  $\mathcal{S}$  doesn’t know when corruptions occur so its actions are the same until corruptions happen). Upon auxiliary input  $z$ , simulator  $\mathcal{S}$  invokes  $\mathcal{A}$  upon input  $z$  and works as follows:

1. *No corruption, or corruption at the end:*  $\mathcal{S}$  begins by choosing random  $r_0, r_1$  and  $b$  and playing the honest parties in the oblivious transfer protocol of Step 1. Then,  $\mathcal{S}$  simulates  $P_2$  sending a random  $\beta$  to  $P_1$ , and  $P_1$  replying with two random strings  $(y_0, y_1)$ .

If  $\mathcal{A}$  carries out corruptions following the execution, then  $\mathcal{S}$  acts as follows, according to the case:

- (a) *Corruption of  $P_1$  first:*  $\mathcal{S}$  corrupts  $P_1$  and obtains its input pair  $(x_0, x_1)$ . Then,  $\mathcal{S}$  sets  $r_\beta = x_0 \oplus y_0$  and  $r_{1-\beta} = x_1 \oplus y_1$ , where  $\beta, y_0, y_1$  are as above (and the values  $r_0, r_1$  chosen in the simulation of the oblivious transfer subprotocol are ignored).  $\mathcal{S}$  generates the view of  $P_1$  based on this  $(r_0, r_1)$ .

If  $\mathcal{A}$  corrupts  $P_2$  following this, then  $\mathcal{S}$  corrupts  $P_2$  and obtains its input bit  $\sigma$ . Then,  $\mathcal{S}$  sets the value  $b$  (that  $P_2$  supposedly used in its input to the OT subprotocol) to be  $\beta \oplus \sigma$ .  $\mathcal{S}$  generates the view of  $P_2$  based on it using input  $b$  to the OT and receiving output  $r_b$ , where the value of  $r_b$  is as fixed after the corruption of  $P_1$ .

- (b) *Corruption of  $P_2$  first:*  $\mathcal{S}$  corrupts  $P_2$  and obtains its input bit  $\sigma$  together with its output string  $x_\sigma$ . Then,  $\mathcal{S}$  sets  $b = \sigma \oplus \beta$  and  $r_b = x_\sigma \oplus y_\sigma$ , where  $\beta$  is the value set above and likewise  $y_\sigma$  is from the pair  $(y_0, y_1)$  above.  $\mathcal{S}$  then generates the view of  $P_2$  based on its input to the OT being  $b$  and its output being  $r_b$ .

If  $\mathcal{A}$  corrupts  $P_1$  follows this, then  $\mathcal{S}$  corrupts  $P_1$  and obtains its input pair  $(x_0, x_1)$  (note that  $x_\sigma$  was already obtained). Then,  $\mathcal{S}$  sets  $r_{1-b} = x_{1-\sigma} \oplus y_{1-\sigma}$  and generates the view of  $P_1$  such that its input to the OT subprotocol was  $(r_0, r_1)$  as generated upon the corruption of  $P_2$  and the later corruption of  $P_1$ .

2. *Corruption of both  $P_1$  and  $P_2$  at any point until Step 1 concludes:*  $\mathcal{S}$  begins by emulating the OT subprotocol with random  $(r_0, r_1)$  and  $b$  as described above. Then, upon corruption of party  $P_i$ , simulator  $\mathcal{S}$  corrupts  $P_i$  and obtains its input. It can then just hand  $\mathcal{A}$  the input of  $P_i$  together with its view in the emulated subprotocol using the inputs  $(r_0, r_1)$  and  $b$ .
3. *Corruption of  $P_1$  up until Step 1 concludes and  $P_2$  after it concludes:* The corruption of  $P_1$  is dealt with exactly as in the previous case. We remark that once  $P_1$  is corrupted,  $\mathcal{S}$  continues to play  $P_2$  while interacting with  $\mathcal{A}$  controlling  $P_1$  as if in a real execution (and using the random input  $b$  that was chosen). After the OT subprotocol concludes,  $\mathcal{S}$  simulates  $P_2$  sending a random  $\beta$  to  $P_1$ , and obtains back a pair  $(y_0, y_1)$  from  $\mathcal{A}$  who controls  $P_1$ .<sup>3</sup>

If  $\mathcal{A}$  corrupts  $P_2$  at this point, then  $\mathcal{S}$  corrupts  $P_2$  and obtains  $\sigma$ .  $\mathcal{S}$  sets  $P_2$ 's input in the OT subprotocol to be  $b = \sigma \oplus \beta$  and generates the view accordingly.

4. *Corruption of  $P_2$  up until Step 1 concludes and  $P_1$  after it concludes:* The corruption of  $P_2$  is dealt with exactly as in the corruption case in item 2 above. As previously, once  $P_2$  is corrupted  $\mathcal{S}$  continues to play  $P_1$  while interacting with  $\mathcal{A}$  controlling  $P_2$  as if in a real execution (and using the random input  $(r_0, r_1)$  that was chosen). Let  $\sigma$  be  $P_1$ 's input and let  $x_\sigma$  be its output. After the OT subprotocol concludes,  $\mathcal{S}$  obtains a random bit  $\beta$  from  $\mathcal{A}$  controlling  $P_2$  and sets  $y_\sigma = x_\sigma \oplus r_b$  where  $b$  is the input used by  $P_2$  in the OT subprotocol (whether corrupted or not) and  $r_b$  is from above. Furthermore,  $\mathcal{S}$  chooses a random  $y_{1-\sigma} \in_R \{0, 1\}^n$ , and simulates  $P_1$  sending  $(y_0, y_1)$  to  $P_2$ .

If  $\mathcal{A}$  corrupts  $P_1$  at this point (or before  $(y_0, y_1)$  were sent – but it makes no difference), then  $\mathcal{S}$  corrupts  $P_1$  and obtains  $(x_0, x_1)$ . It then redefines the value of  $r_{1-b}$  to be  $y_{1-\sigma} \oplus x_{1-\sigma}$ , and generates the view based on these values.

This covers all corruption cases. We now proceed to prove that

$$\left\{ \text{IDEAL}_{OT, \mathcal{S}(z)}(n, x_0, x_1, \sigma) \right\}_{x_0, x_1, \sigma, z; n \in \mathbb{N}} \stackrel{c}{\equiv} \left\{ \text{REAL}_{\pi, \mathcal{A}(z)}(n, x_0, x_1, \sigma) \right\}_{x_0, x_1, \sigma, z; n \in \mathbb{N}}$$

We present our analysis following the case-by-case description of the simulator:

1. *No corruption, or corruption at the end:* In order to prove this corruption case, we begin by showing that when no parties are corrupted, every oblivious transfer protocol (that is secure for the *static* corruption model) has the following property. Let  $\mathcal{A}$  be a probabilistic polynomial-time adversary that corrupts no parties, and only eavesdrops on the communication in the protocol. Then, for all strings  $r_0, r_1, r'_0, r'_1 \in \{0, 1\}^n$  and every probabilistic polynomial-time distinguisher  $D$ :

$$|\Pr[D(\text{REAL}_{\pi, \mathcal{A}(z)}(n, r_0, r_1, 0)) = 1] - \Pr[D(\text{REAL}_{\pi, \mathcal{A}(z)}(n, r'_0, r'_1, 1)) = 1]| \leq \text{negl}(n) \quad (1)$$

---

<sup>3</sup>Note that if  $\mathcal{A}$  was malicious and not semi-honest, then the simulation at this point would not work because  $\mathcal{S}$  cannot know which inputs  $\mathcal{A}$  used in the oblivious transfer (this is due to the fact that the corruption occurred in the middle of the oblivious transfer and the static simulator may not necessarily be able to deal with this). For this reason, we have only been able to prove our transformation for the semi-honest model.

for some negligible function  $\text{negl}$ . The above follows from the following three equations (all equations relate to the same quantification as above over all strings and all distinguishers):

$$|\Pr[D(\text{REAL}_{\pi, \mathcal{A}(z)}(n, r_0, r_1, 0)) = 1] - \Pr[D(\text{REAL}_{\pi, \mathcal{A}(z)}(n, r_0, r'_1, 0)) = 1]| \leq \text{negl}(n)$$

This holds because otherwise an adversary corrupting the receiver  $P_2$  could learn something about the second string of  $P_1$ 's input, even though it used input 0 and so received  $r_0$ . (This would contradict the security of the protocol in the ideal model; the formal statement of this is straightforward and so omitted.) Next,

$$|\Pr[D(\text{REAL}_{\pi, \mathcal{A}(z)}(n, r_0, r'_1, 0)) = 1] - \Pr[D(\text{REAL}_{\pi, \mathcal{A}(z)}(n, r_0, r'_1, 1)) = 1]| \leq \text{negl}(n)$$

This second equation holds because otherwise an adversary corrupting the sender  $P_1$  could distinguish the case that the receiver  $P_2$  has input 0 or 1. Finally,

$$|\Pr[D(\text{REAL}_{\pi, \mathcal{A}(z)}(n, r_0, r'_1, 1)) = 1] - \Pr[D(\text{REAL}_{\pi, \mathcal{A}(z)}(n, r'_0, r'_1, 1)) = 1]| \leq \text{negl}(n)$$

As with the first equation, this holds because otherwise an adversary corrupting  $P_2$  can learn something about the first string of  $P_1$ 's input even though it used input 1. Combining the above three together, Eq. (1) follows.

Now, let  $r'_0, r'_1, b$  be the values used by  $\mathcal{S}$  to simulate the oblivious transfer in Step 1 of the protocol, and let  $y_0, y_1, \beta$  be the random values sent by  $\mathcal{S}$  in the later steps. In addition, let  $x_0, x_1, \sigma$  be the real parties' inputs that are received by  $\mathcal{S}$  upon corruption of both  $P_1$  and  $P_2$ . As in the simulation description, we separately analyze the case that no parties are corrupted, the case that  $P_1$  was corrupted first and the case that  $P_2$  was corrupted first.

- (a) *No corruptions*: In the case of no corruptions, all the adversary sees is the oblivious transfer transcript, a random bit  $\beta$  and two random strings  $y_0, y_1$ . Let  $x_0, x_1, \sigma$  be the real inputs of the honest parties. Then, the values  $y_0, y_1, \beta$  seen by the adversary are "correct" if the inputs used in the oblivious transfer are  $r'_\beta = x_0 \oplus y_0$  and  $r'_{1-\beta} = x_1 \oplus y_1$  and  $b = \beta \oplus \sigma$ . However,  $\mathcal{S}$  used inputs  $r_0, r_1, b$  and not these  $r'_0, r'_1, b'$ . Nevertheless, Eq. (1) guarantees that the distribution over the transcript generated using  $r_0, r_1, b$  (as in the simulation) is computationally indistinguishable from the distribution over the transcript generated using  $r'_0, r'_1, b'$  (as in a real execution). Thus, indistinguishability holds for this case.
- (b) *Corruption of  $P_1$  first*: As described in the simulation,  $\mathcal{S}$  sets  $r_\beta = x_0 \oplus y_0$  and  $r_{1-\beta} = x_1 \oplus y_1$ . Then,  $\mathcal{S}$  sets  $b = \beta \oplus \sigma$ . If  $\mathcal{S}$  had used  $r_0, r_1, b$  as defined here in the simulation of Step 1 of the protocol, then the simulation would be perfect (because all of the values are constructed exactly as the honest parties would construct them upon inputs  $x_0, x_1, \sigma$ ). Thus, using Eq. (1), we have that the distributions are computationally indistinguishable. (Recall that Eq. (1) refers to the transcript generated when no parties are corrupted. However, this is exactly the case here because the corruptions occur *after* the subprotocol has concluded. Furthermore, because the parties erase their internal state, the only information about the subprotocol that  $\mathcal{A}$  receives is the transcript of messages sent, as required.)
- (c) *Corruption of  $P_2$  first*: The proof of this is almost identical to the case where  $P_1$  is corrupted first.

2. *Corruption of both  $P_1$  and  $P_2$  at any point until Step 1 concludes:* This case is trivial because in Step 1 both  $\mathcal{S}$  and the honest parties use random inputs that are independent of the inputs. Thus the distribution generated by  $\mathcal{S}$  is identical as in a real execution.
3. *Corruption of  $P_1$  until Step 1 concludes and  $P_2$  after it concludes:* The distribution of the view of  $P_1$  generated by  $\mathcal{S}$  is identical to a real execution, because as in the previous case, the inputs used until the end of Step 1 are random and independent of the party's actual input. Regarding  $P_2$ 's view, indistinguishability follows from the fact that for any oblivious transfer protocol (that is secure for static corruptions), it holds that when  $\mathcal{A}$  has corrupted  $P_1$  only, we have that for all  $r_0, r_1 \in \{0, 1\}^n$

$$\left\{ \text{REAL}_{\pi, \mathcal{A}(z)}(n, r_0, r_1, 0) \right\}_{n \in \mathbb{N}} \stackrel{c}{\equiv} \left\{ \text{REAL}_{\pi, \mathcal{A}(z)}(n, r_0, r_1, 1) \right\}_{n \in \mathbb{N}}$$

This follows similarly to Eq. (1) because in the ideal model, a corrupted  $P_1$  cannot know if  $P_2$  has input 0 or 1.

4. *Corruption of  $P_2$  until Step 1 concludes and  $P_1$  after it concludes:* The proof of this case is almost identical to the previous case.

This completes the proof of the theorem. ■

## 4.2 The Two-Party Protocol for Semi-Honest Adversaries

We present a protocol for securely computing any functionality  $f$  that maps two  $n$ -bit inputs into an  $n$ -bit output. It is possible to generalize the construction to functions for which the input and output lengths vary. However, the security of our protocol relies crucially on the fact that the length of the output of  $f$  equals the length of the second input  $y$ . (This is because our simulation works by generating a garbled circuit computing  $f(x, y) = y$  which must be indistinguishable from a garbled circuit computing  $C(x, y)$ . Such indistinguishability can only hold if  $|C(x, y)| = |y|$ .) This can be achieved w.l.o.g. by padding the length of  $P_2$ 's input  $y$  with zeroes. Our description assumes familiarity with Yao's garbled circuit construction; see Appendix A for a full description. Observe that we consider only a "same-output functionality", meaning that both  $P_1$  and  $P_2$  receive the same output value  $f(x, y)$ . In [21], this was shown to be without loss of generality: given any protocol as we describe here it is possible to construct a protocol where the parties have different outputs with very little additional overhead.

**Convention.** We require that the circuit  $C$  used to compute  $f$  is of a given structure. Technically, what we need is that the same structure of the circuit (i.e., the positions of the wires connecting the gates) can be used to compute the function  $f'(x, y) = y$  (of course, the actual gates would be different, but this is fine). This can be achieved without difficulty, and so we will not elaborate further.

### Protocol 2

- **Inputs:**  $P_1$  has  $x \in \{0, 1\}^n$  and  $P_2$  has  $y \in \{0, 1\}^n$

- **Auxiliary input:** A boolean circuit  $C$  such that for every  $x, y \in \{0, 1\}^n$  it holds that  $C(x, y) = f(x, y)$ , where  $f: \{0, 1\}^n \times \{0, 1\}^n \rightarrow \{0, 1\}^n$ .<sup>4</sup>
- **The protocol:**
  1.  $P_1$  constructs the garbled circuit  $G(C)$  as described in Appendix A, but does not yet send it to  $P_2$ .
  2. Let  $w_1, \dots, w_n$  be the circuit-input wires corresponding to  $x$ , and let  $w_{n+1}, \dots, w_{2n}$  be the circuit-input wires corresponding to  $y$ . Then,
    - (a)  $P_1$  sends  $P_2$  the strings  $k_1^{x_1}, \dots, k_n^{x_n}$ .
    - (b) For every  $i$ ,  $P_1$  and  $P_2$  execute a 1-out-of-2 oblivious transfer protocol that is adaptively secure for semi-honest adversaries, in which  $P_1$ 's input equals  $(k_{n+i}^0, k_{n+i}^1)$  and  $P_2$ 's input equals  $y_i$ .  
The above oblivious transfers can all be run in parallel.
  3. After the previous step is completed,  $P_1$  erases all of the randomness it used to construct the garbled circuit (and in particular, erases all of the secret keys). Following this erasure,  $P_1$  sends  $P_2$  the garbled circuit  $G(C)$ .
  4. Following the above,  $P_2$  has obtained the garbled circuit and  $2n$  keys corresponding to the  $2n$  input wires to  $C$ . Party  $P_2$  then computes the circuit, as described in Appendix A, obtaining  $f(x, y)$ .  $P_2$  then sends  $f(x, y)$  to  $P_1$  and they both output this value.

The only differences between Protocol 2 and Yao's protocol as it appears in [21] are:

1.  $P_1$  does not send  $G(C)$  to  $P_2$  until the oblivious transfers have concluded.
2.  $P_1$  erases all of its internal state before it actually sends  $G(C)$ .
3. The oblivious transfers must be secure in the presence of *adaptive* semi-honest adversaries.

The above differences make no difference whatsoever to the proof of security in the static case. However, the simulator provided in [21] does not work in the case of adaptive corruptions. In order to see this, recall that the simulator there works by constructing a special fake circuit that outputs a predetermined value. In the setting of static security this suffices because the simulator is given the output value before it begins the simulation; it can therefore set the predetermined output value of the fake circuit to the party's output. However, in the setting of adaptive corruptions, the simulator may need to generate a fake garbled circuit before it knows any of the party's outputs (in particular, this happens if corruptions occur only at the end of the execution). It therefore does not know the circuit's output when it generates it. One way to overcome this problem is to use an equivocal encryption scheme that can be opened to any value. However, this raises a whole other set of problems, and would also be far less efficient. In our proof below we show that, in fact, the construction need not be modified at all. Rather, the simulator can construct a fake garbled circuit that computes the function  $f(x, y) = y$  (instead of a fake circuit that outputs a predetermined value). This means that by choosing appropriate keys for the input wires associated

---

<sup>4</sup>As in [21], we require that  $C$  is such that if a circuit-output wire leaves some gate  $g$ , then gate  $g$  has no other wires leading from it into other gates (i.e., no circuit-output wire is also a gate-input wire). Likewise, a circuit-input wire that is also a circuit-output wire enters no gates.

with  $y$  (i.e., those associated with  $P_2$ 's input), it is possible for the simulator to cause the circuit to output any value that it wishes. More specifically, after constructing such a fake garbled circuit, when the simulator receives an output value  $z = f(x, y)$  it can simply choose the keys to the input wires associated with  $P_2$ 's input to be those that are associated with the *output value*  $z$ . This will then result in the circuit being opened to  $z$ , as required. Our proof shows that such a circuit is computationally indistinguishable from a real garbled circuit, and so the simulation works. We use this simulation strategy to formally prove the following theorem:

**Theorem 6** *Let  $f$  be a deterministic same-output functionality. Furthermore, assume that the oblivious transfer protocol is secure in the presence of adaptive semi-honest adversaries (with erasures), and that the encryption scheme has indistinguishable encryptions under chosen plaintext attacks, and has an elusive and efficiently verifiable range. Then, Protocol 2 securely computes  $f$  in the presence of adaptive semi-honest adversaries (with erasures).*

**Proof:** Our proof is presented in the OT-hybrid model. Recall that this means that the oblivious transfers in the protocol are performed by the parties through an incorruptible trusted third party; the parties send their inputs to the trusted party via ideally secure channels and receive back their appropriate outputs. Thus, when no parties are corrupted, the adversary sees absolutely nothing in the oblivious transfer (apart from the fact that it takes place). Furthermore, when one of the parties is corrupted, the adversary playing the corrupted party must send its input explicitly to the trusted party.

Let  $\mathcal{A}$  denote the real adversary. We now describe the simulator  $\mathcal{S}$ . We begin by describing its actions from the beginning to the end when no corruptions take place. We then explain how it behaves when corruptions occur.

1.  $\mathcal{S}$  chooses  $4n$  random keys  $k_1^0, k_1^1, \dots, k_{2n}^0, k_{2n}^1$  as  $P_1$  would when constructing  $G(C)$ .  $\mathcal{S}$  simulates  $P_1$  sending  $P_2$  the keys  $k_1^0, \dots, k_n^0$ .
2.  $\mathcal{S}$  simulates  $P_1$  and  $P_2$  running the oblivious transfers via the trusted party. (When neither party is corrupted, this merely involves notifying  $\mathcal{A}$  that the transfers are taking place.)
3.  $\mathcal{S}$  constructs a circuit  $\tilde{C}$  that has the exact same gate and wire structure as  $C$ , except that  $\tilde{C}$  computes the function  $f(x, y) = y$ . The simulator  $\mathcal{S}$  then constructs a (proper) garbled circuit of  $\tilde{C}$ , denoted  $G(\tilde{C})$ , and simulates  $P_1$  sending the circuit to  $P_2$ .

This concludes the simulation for the case of no corruptions. We proceed to describe  $\mathcal{S}$ 's actions upon corruptions by  $\mathcal{A}$ . Note that we must describe the simulation for all possible corruption points. There are three corruption points for each party: (1) at the onset, (2) after the keys have been sent and the oblivious transfers run, and (3) after the garbled circuit has been sent. Since there are two parties, this defines 9 corruption cases. (Actually there are more cases because it may be that only one party is corrupted. Our description will deal only with the case that eventually both parties are corrupted. This is due to the fact that if the simulation works when a party is corrupted at the end, then it certainly works if that party is not corrupted at all.) We describe these cases now:

1. If  $\mathcal{A}$  corrupts both  $P_1$  and  $P_2$  at the onset, the simulation is trivial (when both parties are corrupted at the onset, all inputs are known and there is nothing to simulate).

2. If  $\mathcal{A}$  corrupts party  $P_1$  at the onset, then  $\mathcal{S}$  corrupts  $P_1$  in the ideal model and receives its input  $x$ .  $\mathcal{S}$  sends  $x$  to the trusted party and receives back  $z = f(x, y)$ . Then,  $\mathcal{S}$  simply runs the honest  $P_2$  in the protocol (interacting with the corrupted  $P_1$  with input  $x$ ); note that since the oblivious transfers are ideal, this involves just receiving messages from  $\mathcal{A}$ . For this reason, if  $\mathcal{A}$  corrupts  $P_2$  at any time after the corruption of  $P_1$ , simulator  $\mathcal{S}$  can just corrupt  $P_2$  in the ideal model, obtain its input  $y$  and that is all. (In fact,  $P_2$  does not even have to be probabilistic when the oblivious transfer is ideal and so its view consists merely of its input and the messages it received.)

This covers two corruption cases ( $P_1$  at the onset;  $P_2$  after the keys have been sent or after the garbled circuit has been sent).

3. If  $\mathcal{A}$  corrupts  $P_1$  after the keys have been sent and the oblivious transfers have been run, but before the garbled circuit  $G(\tilde{C})$  was sent by  $\mathcal{S}$  in the simulation, then the simulation continues as in the case that the corruption of  $P_1$  was at the onset. The only difference is that  $P_1$  has supposedly already constructed  $G(C)$  and the keys  $k_1^0, \dots, k_n^0$  are supposedly associated with  $P_1$ 's input. However, given  $x$  after the corruption of  $P_1$ , the simulator  $\mathcal{S}$  can construct a correct  $G(C)$  and can reassign the keys so that for every  $i = 1, \dots, n$  it holds that  $k_i^{x_i} \leftarrow k_i^0$  and  $k_i^{1-x_i} \leftarrow k_i^1$ .<sup>5</sup> (This is identical to  $\mathcal{A}$ 's view upon corrupting the real  $P_1$  at this point of the protocol execution.) A corruption of  $P_2$  following  $P_1$  proceeds as in the previous case.

Once again, this covers two corruption cases ( $P_1$  after the keys;  $P_2$  at the same time or after the garbled circuit has been sent).

4. If  $\mathcal{A}$  corrupts  $P_1$  after the garbled circuit  $G(\tilde{C})$  has been sent by  $\mathcal{S}$  in the simulation, then  $P_1$  has already erased all of its internal randomness. Therefore,  $\mathcal{S}$  just corrupts  $P_1$  in the ideal model in order to obtain  $x$  and hands  $x$  to  $\mathcal{A}$ . If  $P_2$  is corrupted also at this point, then  $\mathcal{S}$  corrupts  $P_2$  in the ideal model and obtains  $y$ . It then computes  $z = f(x, y)$  and assigns the keys that  $P_2$  received in the oblivious transfers as follows. For every  $i = 1, \dots, n$  it assigns  $k_{n+i}^{y_i} \leftarrow k_{n+i}^{z_i}$  and  $k_{n+i}^{1-y_i} \leftarrow k_{n+i}^{1-z_i}$  (where  $k_{n+i}^{z_i}$  and  $k_{n+i}^{1-z_i}$  were the keys used in defining  $G(\tilde{C})$ ). Thus,  $\mathcal{A}$  obtains the view of  $P_2$  containing the keys  $k_1^0, \dots, k_n^0$  and  $k_{n+1}^{z_1}, \dots, k_{2n}^{z_n}$ , and the garbled circuit  $G(\tilde{C})$ . We remark that since  $\tilde{C}(x, y) = y$  it follows that the decryption of  $G(\tilde{C})$  with the above keys yields the output  $z = f(x, y)$ .

This covers the corruption case of  $P_1$  and  $P_2$  being corrupted after  $G(\tilde{C})$  has been sent.

5. If  $\mathcal{A}$  corrupts  $P_2$  at the onset of the execution, then  $\mathcal{S}$  corrupts  $P_2$  in the ideal model and obtains  $y$ .  $\mathcal{S}$  sends  $y$  to the trusted party and obtains back  $z = f(x, y)$ . Simulator  $\mathcal{S}$  then simulates  $P_1$  sending the keys  $k_1^0, \dots, k_n^0$  as above, and hands  $\mathcal{A}$  the keys  $k_{n+1}^{z_1}, \dots, k_{2n}^{z_n}$  as its output from the oblivious transfers.

If  $\mathcal{A}$  corrupts  $P_1$  at this point, then  $\mathcal{S}$  corrupts  $P_1$  and obtains  $x$ . It then “explains” the keys  $k_1^0, \dots, k_n^0$  as above in item 3 (i.e., for every  $1 \leq i \leq n$  it defines  $k_i^{x_i} \leftarrow k_i^0$  and  $k_i^{1-x_i} \leftarrow k_i^1$ ), and the keys  $k_{n+1}^{z_1}, \dots, k_{2n}^{z_n}$  as in the previous item (i.e.,  $k_{n+i}^{y_i} \leftarrow k_{n+i}^{z_i}$  and  $k_{n+i}^{1-y_i} \leftarrow k_{n+i}^{1-z_i}$ ). At this point, both parties are corrupted and nothing further needs to be simulated.

If  $\mathcal{A}$  does not corrupt  $P_1$  at this point, then  $\mathcal{S}$  constructs the garbled circuit  $G(\tilde{C})$  as above and sends it to  $\mathcal{A}$  (who controls  $P_2$ ). If  $\mathcal{A}$  corrupts  $P_1$  after  $G(\tilde{C})$  has been sent, then it has

---

<sup>5</sup>This may look confusing. However, what we mean is that the association of the key  $k = k_i^0$  to the zero value on the wire  $w_i$  is modified to the value  $x_i$  on the wire  $w_i$ .

already erased its internal state so nothing but  $x$  has to be given to  $\mathcal{A}$ .

This covers two corruption cases ( $P_2$  at the onset;  $P_1$  either after the keys have been sent or after the garbled circuit has been sent).

6. If  $\mathcal{A}$  corrupts  $P_2$  after the keys have been sent but before  $G(\tilde{C})$  has been sent in the simulation, then  $\mathcal{S}$  acts exactly as in the previous case. The only difference is that  $\mathcal{A}$  has not yet seen  $P_2$ 's output keys from the oblivious transfer. In this case, after obtaining  $y$  and  $z$ ,  $\mathcal{S}$  just defines the keys that  $P_2$  received in the oblivious transfer to be  $k_{n+1}^{z_1}, \dots, k_{2n}^{z_n}$  as above.

The case that  $\mathcal{A}$  corrupts  $P_1$  also after the keys have been sent is covered in item 3 above.

If  $\mathcal{A}$  does not corrupt  $P_1$  after the keys have been sent, then  $\mathcal{S}$  constructs  $G(\tilde{C})$  exactly as above. Again, if  $P_1$  is corrupted after  $G(\tilde{C})$  was sent, then only  $x$  needs to be given to  $\mathcal{A}$ .

This covers the corruption case that  $P_2$  was corrupted after the keys and  $P_1$  after the garbled circuit was sent.

This completes all the corruption cases. In order to prove that

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

we first construct an alternative simulator  $\mathcal{S}'$  that works in exactly the same way as  $\mathcal{S}$ , except that it is given  $x$  and  $y$  at the onset and always constructs the correct garbled circuit  $G(C)$ . Furthermore, the keys sent by  $P_1$  to  $P_2$  are  $k_1^{x_1}, \dots, k_1^{x_n}$  and the keys received by  $P_2$  in the oblivious transfers are  $k_{n+1}^{y_1}, \dots, k_{2n}^{y_n}$ . (Thus, in the corruption cases,  $\mathcal{S}'$  does not need to “explain” the keys in any way; it just reveals the keys used.) It is clear that

$$\left\{ \text{IDEAL}_{f, \mathcal{S}'(z)}(n, x, y) \right\}_{x, y, z; n \in \mathbb{N}} \equiv \left\{ \text{REAL}_{\pi, \mathcal{A}(z)}(n, x, y) \right\}_{x, y, z; n \in \mathbb{N}}$$

It thus remains to prove that

$$\left\{ \text{IDEAL}_{f, \mathcal{S}(z)}(n, x, y) \right\}_{x, y, z; n \in \mathbb{N}} \stackrel{c}{=} \left\{ \text{IDEAL}_{f, \mathcal{S}'(z)}(n, x, y) \right\}_{x, y, z; n \in \mathbb{N}} \quad (2)$$

Now, the only difference between the simulation with  $\mathcal{S}$  and with  $\mathcal{S}'$  is that  $\mathcal{S}$  uses a fake circuit  $\tilde{C}$  whereas  $\mathcal{S}'$  uses the real one  $C$ . Intuitively, this cannot be distinguished because the randomness used to generate the garbled circuit is erased before the circuit is sent. Furthermore, the garbled circuit itself reveals nothing but the output. More formally, the following claim was proven in [22] (based on the proof in [21]):

**Claim 7** *Given a circuit  $C$  and an output value  $z$  (of the same length as the output of  $C$ ) it is possible to construct a garbled circuit  $\tilde{G}_z(C)$  such that:*

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

Applying Claim 7 we have that the real garbled circuit  $G(C)$  used by  $\mathcal{S}'$  is indistinguishable from  $\tilde{G}_z(C)$ . Furthermore, applying the claim again, we have that the fake garbled circuit  $G(\tilde{C})$  used by  $\mathcal{S}$  is also indistinguishable from  $\tilde{G}_z(\tilde{C})$ . However, the circuit  $\tilde{G}_z(C)$  equals the circuit  $\tilde{G}_z(\tilde{C})$ ; this holds because  $C$  and  $\tilde{C}$  have exactly the same gate structure. We therefore conclude that  $G(C)$  is computationally indistinguishable from  $G(\tilde{C})$ . In order to complete the proof of Eq. (2), it suffices to note that the entire simulation (of  $\mathcal{S}$  or  $\mathcal{S}'$ ) can be carried out when the garbled circuit and one key for each input wire is received externally (and not constructed by the simulator). If the garbled circuit received is  $G(C)$  then the result is exactly the same as the simulation with  $\mathcal{S}'$  and if the garbled circuit received is  $G(\tilde{C})$ , then the result is exactly the same as in the simulation with  $\mathcal{S}$ . Eq. (2) therefore follows from the indistinguishability of  $G(C)$  and  $G(\tilde{C})$ . This completes the proof. ■

Since, as we have shown, adaptively secure oblivious transfer (in the erasure model) can be achieved assuming only the existence of enhanced trapdoor permutations, Theorem 6 implies Theorem 2 as stated in the introduction.

## References

- [1] D. Beaver. Foundations of Secure Interactive Computing. In *CRYPTO'91*, Springer-Verlag (LNCS 576), pages 377–391, 1991.
- [2] D. Beaver. Plug and play encryption. In *CRYPTO'97*, Springer-Verlag (LNCS 1294), pages 75–89, 1997.
- [3] D. Beaver and S. Haber. Cryptographic Protocols Provably Secure Against Dynamic Adversaries. In *EUROCRYPT'92*, Springer-Verlag (LNCS 658), pages 307–323, 1992.
- [4] M. Ben-Or, S. Goldwasser and A. Wigderson. Completeness Theorems for Non-Cryptographic Fault-Tolerant Distributed Computation. In *20th STOC*, pages 1–10, 1988.
- [5] R. Canetti. Security and Composition of Multiparty Cryptographic Protocols. *Journal of Cryptology*, 13(1):143–202, 2000.
- [6] R. Canetti. Universally Composable Security: A New Paradigm for Cryptographic Protocols. In *42nd FOCS*, pages 136–145, 2001. Full version available at <http://eprint.iacr.org/2000/067>.
- [7] R. Canetti, I. Damgård, S. Dziembowski, Y. Ishai and T. Malkin. Adaptive versus Non-Adaptive Security of Multi-Party Protocols. *Journal of Cryptology*, 17(3):153–207, 2004.
- [8] R. Canetti, U. Feige, O. Goldreich and M. Naor. Adaptively Secure Multi-Party Computation. In *28th STOC*, pages 639–648, 1996.
- [9] R. Canetti, R. Gennaro, S. Jarecki H. Krawczyk and T. Rabin. Adaptive Security for Threshold Cryptosystems. In *CRYPTO 1999*, Springer-Verlag (LNCS 1666), pages 98–115, 1999.
- [10] R. Canetti, Y. Lindell, R. Ostrovsky and A. Sahai. Universally Composable Two-Party and Multi-Party Computation. In *34th STOC*, pages 494–503, 2002. Full version available at <http://eprint.iacr.org/2002/140>.

- [11] D. Chaum, C. Crépeau and I. Damgård. Multi-party Unconditionally Secure Protocols. In *20th STOC*, pages 11–19, 1988.
- [12] S. Even, O. Goldreich and A. Lempel. A Randomized Protocol for Signing Contracts. In *Communications of the ACM*, 28(6):637–647, 1985.
- [13] Y. Frankel, P.D. MacKenzie and M. Yung. Adaptively-Secure Optimal-Resilience Proactive RSA. In *ASIACRYPT 1999*, Springer-Verlag (LNCS 1716), pages 180–194, 1999.
- [14] J.A. Garay, P.D. MacKenzie and K. Yang. Efficient and Universally Composable Committed Oblivious Transfer and Applications. In *TCC 2004*, Springer-Verlag (LNCS 2951), pages 297–316, 2004.
- [15] O. Goldreich. *Foundations of Cryptography: Volume 2 – Basic Applications*. Cambridge University Press, 2004.
- [16] O. Goldreich, S. Micali and A. Wigderson. How to Play any Mental Game – A Completeness Theorem for Protocols with Honest Majority. In *19th STOC*, pages 218–229, 1987. For details see [15].
- [17] 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.
- [18] E. Kushilevitz, Y. Lindell and T. Rabin. Information-Theoretically Secure Protocols and Security Under Composition. In the em *38th STOC*, pages 109–118, 2006.
- [19] S. Jarecki and A. Lysyanskaya. Adaptively Secure Threshold Cryptography: Introducing Concurrency, Removing Erasures. In *EUROCRYPT 2000*, Springer-Verlag (LNCS 1807), pages 221–242, 2000.
- [20] Y. Lindell. General Composition and Universal Composability in Secure Multi-Party Computation. In *44th FOCS*, pages 394–403, 2003.
- [21] Y. Lindell and B. Pinkas. A Proof of Security of Yao’s Protocol for Two-Party Computation. To appear in the *Journal of Cryptology*.
- [22] Y. Lindell and B. Pinkas. An Efficient Protocol for Secure Two-Party Computation in the Presence of Malicious Adversaries. In *EUROCRYPT 2007*, Springer-Verlag (LNCS 4515), pages 52–78, 2007.
- [23] Y. Lindell and H. Zarusim. Adaptive Zero-Knowledge Proofs and Adaptively Secure Oblivious Transfer. To appear in the *6th TCC*, 2009.
- [24] S. Micali and P. Rogaway. Secure Computation. Unpublished manuscript, 1992. Preliminary version in *CRYPTO’91*, Springer-Verlag (LNCS 576), pages 392–404, 1991.
- [25] S. Wolf and J. Wullschleger. Oblivious Transfer Is Symmetric. In *EUROCRYPT 2006*, Springer-Verlag (LNCS 4004), pages 222–232, 2006.
- [26] A. Yao. How to Generate and Exchange Secrets. In *27th FOCS*, pages 162–167, 1986.

## A Yao's Garbled Circuit Construction

In this section, we describe Yao's protocol for secure two-party computation in the presence of semi-honest adversaries [26]. The specific construction described here is from [21], where a full proof of security is also provided.

Let  $C$  be a Boolean circuit that receives two inputs  $x, y \in \{0, 1\}^n$  and outputs  $C(x, y) \in \{0, 1\}^n$  (for simplicity, we assume that the input length, output length and the security parameter are all of the same length  $n$ ). We also assume that  $C$  has the property that if a circuit-output wire comes from a gate  $g$ , then gate  $g$  has no wires that are input to other gates.<sup>6</sup> (Likewise, if a circuit-input wire is itself also a circuit-output, then it is not input into any gate.)

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

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

where  $E$  is from a private key encryption scheme  $(G, E, D)$  that has indistinguishable encryptions for multiple messages, and has an elusive efficiently verifiable range; see [21]. 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  $k_1^\alpha$  and  $k_2^\beta$ , 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  $k_3^\gamma$  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  $k_1^\alpha$  and  $k_2^\beta$ . Later we will show that except with negligible probability, only one non- $\perp$  value is indeed obtained.)

We are now ready to show how to construct the entire garbled circuit. Let  $m$  be the number of *wires* in the circuit  $C$ , and let  $w_1, \dots, w_m$  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

---

<sup>6</sup>This requirement is due to our labelling of gates described below, that does not provide a unique label to each wire (see [21] for more discussion). We note that this assumption on  $C$  increases the number of gates by at most  $n$ .

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  $x = x_1 \cdots x_n$  and  $y = y_1 \cdots y_n$  be two  $n$ -bit inputs for  $C$ . Furthermore, let  $w_1, \dots, w_n$  be the input labels corresponding to  $x$ , and let  $w_{n+1}, \dots, w_{2n}$  be the input labels corresponding to  $y$ . It is shown in [21] that given the garbled circuit  $G(C)$  and the strings  $k_1^{x_1}, \dots, k_n^{x_n}, k_{n+1}^{y_1}, \dots, k_{2n}^{y_n}$ , it is possible to compute  $C(x, y)$ , except with negligible probability.