On Adaptively Secure Multiparty Computation with a Short CRS[*]

Ran Cohen[†] Chris Peikert[‡]

July 18, 2016

Abstract

In the setting of multiparty computation, a set of mutually distrusting parties wish to securely compute a joint function of their private inputs. A protocol is *adaptively secure* if honest parties might get corrupted *after* the protocol has started. Recently (TCC 2015) three constant-round adaptively secure protocols were presented [CGP15, DKR15, GP15]. All three constructions assume that the parties have access to a *common reference string* (CRS) whose size depends on the function to compute, even when facing semi-honest adversaries. It is unknown whether constant-round adaptively secure protocols exist, without assuming access to such a CRS.

In this work, we study adaptively secure protocols which only rely on a short CRS that is independent on the function to compute.

- First, we raise a subtle issue relating to the usage of *non-interactive non-committing encryption* within security proofs in the UC framework, and explain how to overcome it. We demonstrate the problem in the security proof of the adaptively secure oblivious-transfer protocol from [CLOS02] and provide a complete proof of this protocol.
- Next, we consider the two-party setting where one of the parties has a polynomial-size input domain, yet the other has no constraints on its input. We show that assuming the existence of adaptively secure oblivious transfer, every deterministic functionality can be computed with adaptive security in a constant number of rounds.
- Finally, we present a new primitive called *non-committing indistinguishability obfuscation*, and show that this primitive is *complete* for constructing adaptively secure protocols with round complexity independent of the function.

Keywords: secure multiparty computation, adaptive security, non-committing encryption, round complexity.

[*]An extended abstract of this work appeared at *SCN 2016*.

[†]Department of Computer Science, Bar-Ilan University. E-mail: `cohenrb@cs.biu.ac.il`. Work supported by the European Research Council under the ERC consolidators grant agreement n. 615172 (HIPS), by a grant from the Israel Ministry of Science, Technology and Space (grant 3-10883) and by the National Cyber Bureau of Israel.

[‡]Computer Science and Engineering, University of Michigan. E-mail: `cpeikert@umich.edu`. This material is based upon work supported by the National Science Foundation under CAREER Award CCF-1054495 and CNS-1606362, the Alfred P. Sloan Foundation, and by a Google Research Award. The views expressed are those of the authors and do not necessarily reflect the official policy or position of the National Science Foundation, the Sloan Foundation, or Google.

Contents

1 Introduction

1.1 Background

In the setting of *secure multiparty computation*, a set of mutually distrusting parties wish to jointly compute a function on their private inputs in a secure manner. Loosely speaking, the security requirements ensure that even if a subset of dishonest parties collude, nothing is learned from the protocol other than the output (*privacy*), and the output is distributed according to the prescribed functionality (*correctness*). This threat is normally modeled by a central adversarial entity, that might corrupt a subset of the parties and control them. A protocol is considered secure if whatever an adversary can achieve when attacking an execution of the protocol, can be emulated in an ideal world, where an incorruptible trusted party helps the parties to compute the function.

Initial constructions of secure protocols were designed under the assumption that the adversary is *static*, meaning that the set of corrupted parties is determined prior to the beginning of the protocol's execution [30, 20]. Starting from the work of Beaver and Haber [2] and of Canetti et al. [7], protocols that remain secure facing *adaptive* adversaries were considered. In this setting, the adversary can decide which parties to corrupt during the course of the protocol and based on its dynamic view. Adaptive security forms a greater challenge compare to static security, in particular because the adversary can corrupt honest parties *after* the protocol has completed. Furthermore, it can corrupt *all* the parties, thus learning all the randomness that was used in the protocol.[1]

The first adaptively secure protocol, which remains secure facing an arbitrary number of corrupted parties, was presented by Canetti, Lindell, Ostrovsky, and Sahai [8]. They showed that under some standard cryptographic assumptions, any *adaptively well-formed* functionality[2] can be securely computed facing adaptive malicious adversaries. This result follows the GMW paradigm [20], and consists of two stages: First, a protocol secure against adaptive semi-honest adversaries was constructed. This protocol is secure in the plain model, where no setup assumptions are needed; however, the number of communication rounds in this protocol depends on the circuit-depth of the underlying functionality. In the second stage, the protocol was compiled into a protocol secure against adaptive malicious adversaries; the semi-honest to malicious compiler, presented in [8], maintains the round complexity, and is secure assuming that all parties have access to a *common reference string* (CRS).[3]

Recently, three adaptively secure protocols that run in a constant number of rounds were independently presented by Canetti et al. [10], Dachman-Soled et al. [11] and Garg and Polychroniadou [15]. All three protocols are designed in the CRS model and share the idea of embedding inside the CRS an obfuscated program that receives the circuit to compute as one of its input variables. It follows that the *size* of the CRS depends of the *size* of the circuit, and moreover, the CRS is needed even when considering merely semi-honest adversaries. Dachman-Soled et al. [11] and Garg and Polychroniadou [15] raised the question of whether these requirements are necessary.

[1]In this work we do not assume the existence *secure erasures*, meaning that we do not rely on the ability of an honest party to erase specific parts of its memory.

[2]An adaptively well-formed functionality is a functionality that reveals its random input in case all parties are corrupted [8].

[3]Since the protocol of [8] is designed in the UC framework of Canetti [5], security against malicious adversaries requires some form of a trusted-setup assumption, see [6, 9, 26].

1.2 Our Contribution

In this work we consider *adaptive security with a short CRS*. By this we mean two security notions: adaptive security facing semi-honest adversaries in the plain model (i.e., without a CRS) and adaptive security facing malicious adversaries in the CRS model, where the CRS does not depend on the *size* of the circuit to compute.

Non-interactive non-committing encryption in the UC framework. A non-interactive non-committing encryption scheme is a public-key encryption scheme augmented with the ability to generate a fake public key and a fake ciphertext that can later be explained as an encryption of any message. This primitive serves as a building block for several cryptographic constructions, e.g., instantiating adaptively secure communication channels [7], adaptively secure oblivious transfer (OT) [8] and leakage-resilient protocols [3].

Although (interactive) non-committing encryption (NCE) was introduced well before the standard security models for adaptive security have been formalized, mainly the *modular-composition* framework of [4] and the *universal-composability (UC) framework* of [5], it has been a folklore belief that non-interactive NCE is secure in these frameworks. We revisit the security of non-interactive NCE and show that although it is straightforward to prove the security in the framework of modular composition, it is not as obvious in the UC framework. The reason lies in a subtle difference between the two frameworks: in the framework of [4], all the parties are initialized with their inputs *prior* to the beginning of the protocol, whereas in the UC framework, the environment can adaptively provide inputs to the parties *after* the protocol has started.

This may lead to the following attack. The environment first activates the receiver that generates a public key. This is simulated by generating the (fake) non-committing public key and ciphertext. Next, the adversary corrupts the receiver and learns its random coins (before the sender has been activated with input). At this point, the simulator must explain the key generation *before* the plaintext has been determined. Finally, the environment activates the sender with a random message. The problem is that once the random coins for the key generation have been fixed, the ciphertext becomes committing, and with a non-negligible probability will fail to decrypt to the random plaintext.

Not realizing these subtleties may lead to incomplete security proofs when using non-interactive NCE as a building block for protocols in the UC framework. We show that the simulator can in fact cater for such form of attacks, without any adjustments to the protocols, by carefully combining between non-committing ciphertexts and committing ciphertexts during the simulation. We thus prove that the definition of non-interactive NCE is valid in the UC framework. We further show that the proof of security of the adaptively secure OT in Canetti et al. [8] is incomplete and explain how to rectify it. We emphasize that the results in [8] are valid, and merely the proof is incomplete.

Functionalities with one-sided polynomial-size domain. We next consider deterministic two-party functionalities $f(x_1, x_2)$, where the input domain of P_1, denoted D_1, is of polynomial-size. We observe that in this situation, P_2 can locally compute f on its input x_2 and *every* possible input of P_1 and obtain all possible outputs. All that P_1 needs to do now is to select the output corresponding to its input x_1. Therefore, the computation of such functionalities boils down to the ability to compute 1-out-of-$|D_1|$ adaptively secure oblivious transfer. Using the adaptively secure OT from [8], we conclude that for every such functionality there exists a three-message protocol that is secure in the presence of adaptive semi-honest adversaries. Security against malicious adversaries follows using the CLOS compiler.

This result can be interpreted in two ways. On the one hand, it shows that restricting the domain of one of the parties yields a constant-round adaptively secure protocol. On the other hand, it shows that in order to try and prove a lower bound for constant-round adaptively secure protocols in general, one must consider either functionalities with super-polynomial input domains, or probabilistic functionalities.

Non-committing indistinguishability obfuscation. An indistinguishability obfuscator $i\mathcal{O}$ [1] is a machine that given a circuit, creates an "unintelligible" version of it, while maintaining its functionality. "Unintelligible" means, in this case, that given two circuits of the same length that compute exactly the same function, it is infeasible to distinguish between an obfuscation of the first circuit from an obfuscation of the second. This primitive has been shown to be useful for a vast amount of applications, and recently led to a construction of constant-round adaptively secure protocols in the CRS model [10, 11, 15].

All three protocols [10, 11, 15] share a clever idea of embedding an obfuscated program inside the CRS, such that a certain amount of the randomness that is used in the execution of the protocol is kept hidden, even if all parties are eventually corrupted. In this section we explore a different approach to this problem, inspired by the concept of NCE. We present an adaptive analogue for $i\mathcal{O}$ called *non-committing indistinguishability obfuscator*, which essentially allows the simulator to produce an obfuscated circuit for some circuit class, and later, given any circuit in the class, produce appropriate random coins explaining the obfuscation process. We then show that assuming the existence of non-committing $i\mathcal{O}$, every adaptively well-formed functionality can be computed with adaptive security and round complexity that is independent of the functionality.

We emphasize that currently we do not know how to construct non-committing $i\mathcal{O}$, or even if such a construction is possible. Rather, this result serves as a reduction from the problem of constructing adaptively secure protocols with round complexity independent of the function to the problem of constructing non-committing $i\mathcal{O}$.

Informally, by a non-committing indistinguishability obfuscator for some class of equivalent circuits (i.e., circuits that compute the same function), we mean an $i\mathcal{O}$ scheme for this class, augmented with a simulation algorithm that generates an obfuscated circuit $\tilde{C}$, such that later, given any circuit C from the class, it is possible to generate random coins that explain the obfuscated circuit $\tilde{C}$ as an obfuscation of the circuit C. It is not hard to see that if non-committing $i\mathcal{O}$ schemes exist in general, then the polynomial hierarchy collapses (see Section 5). In order to overcome this barrier, we consider a limited set of circuit classes, which turns out to be sufficient for our needs. In particular, we consider classes of equivalent "constant circuits", i.e., all circuits in the class are of the same size, receive no input and output the same value.

We next explain how to use non-committing $i\mathcal{O}$ in order to construct a protocol for any two-party functionality f, where the round complexity depends on the obfuscator rather than on f (this idea extends in a straightforward way to the multiparty setting). First, the parties use any adaptively secure protocol, e.g., the protocol from [8], to compute an intermediate functionality that given the parties' inputs and a circuit computing f, hard-wires the input values to the input wires of the circuit. This way the intermediate functionality generates a "constant circuit" computing the desired output. Next, the intermediate functionality obfuscates this "constant circuit" using random coins provided by the parties and outputs to each party an obfuscated constant circuit.

Finally, each party locally computes the output of the obfuscated constant circuit.

The underlying idea is that upon the first corruption request, the ideal-process adversary learns both the input and the output of the corrupted party, and so can prepare a simulated obfuscated constant circuit that outputs the correct value. Upon the second corruption request, the ideal-process adversary learns the input of the second party and can prepare the constant circuit as generated by the intermediate functionality. Using the non-committing properties of the obfuscation, the random coins explaining the obfuscated circuit can be computed at this point, and so the ideal-process adversary can correctly adjust the random coins that are used for the obfuscation.

1.3 Additional Related Work

Constant-round protocols that are secure facing adaptive adversaries corrupting an arbitrary number of parties that rely on a short CRS are not known to exist in general. Nonetheless, positive and negative results have been established in weaker models.

In a model where the CRS can depend on the function, Canetti et al. [10], Dachman-Soled et al. [11] and Garg and Polychroniadou [15] have independently presented constant-round protocols that are adaptively secure facing an arbitrary number of corrupted parties. Garg and Sahai [16] showed that in the plain model (without assuming a CRS) constant-round protocols that are adaptively secure facing malicious adversaries, cannot be proven secure using a black-box simulator. The authors further showed that using non-black-box techniques, there exists a constant-round adaptively secure multiparty protocol, resilient to corruptions of all but one of the parties.

In case the adaptive adversary cannot corrupt all the parties, i.e., at least one party remains honest, there exist several constant-round protocols. Katz and Ostrovsky [23] showed that any statically secure constant-round two-party protocol can be transformed into an adaptive protocol with a single corruption by wrapping the communication with non-committing encryption. Hazay and Patra [21] achieved better efficiency using one-sided secure primitives. In the multiparty case, Damgård and Ishai [12] constructed a constant-round adaptively secure protocol assuming an honest majority. Compiling this protocol with the IPS compiler from Ishai et al. [22] yields a constant-round adaptively secure protocol that tolerates corruptions of all but one of the parties. Damgård et al. [14] used equivocal FHE to get better concrete constants for the round complexity.

Assuming the existence of secure erasures, Lindell [27] constructed a constant-round protocol that UC-realizes any two-party functionality facing adaptive semi-honest adversaries.

Organization of the Paper

In Section 2 we discuss the subtleties relating to non-interactive NCE and in Section 3, the implications to the security proof of the adaptive OT protocol from [8]. In Section 4 we construct a constant-round two-party protocol for one-sided polynomial-size domain. In Section 5 we define the notion of non-committing $i\mathcal{O}$ and show that this is a complete primitive for adaptively secure protocols with round complexity independent of the function. The security model is defined in Appendix A.

2 Universally Composable Non-Interactive NCE

Non-committing encryption (NCE) is a cryptographic tool, used mainly for constructing adaptively secure multiparty protocols. This notion was first introduced by Canetti et al. [7] as an analogue in the adaptive setting to the instantiation of statically secure communication channels (using "standard" public-key encryption schemes). Since the introduction of NCE, further applications have been based on this primitive, for example, adaptively secure oblivious transfer [8] and leakage-resilient protocols [3].

When constructing adaptively secure protocols, two security models are normally considered: the framework of [4] which provides *modular composition* and the *universal-composability (UC) framework* of [5]. The definition of NCE has evolved over the years, starting from a multiparty protocol instantiating the *secure message transmission* functionality and stabilizing on a non-interactive definition, which is an extension of standard public-key encryption schemes. Although it is fairly easy to verify that the various definitions are equivalent in the framework of [4], certain subtleties arise when considering non-interactive NCE in the UC framework. Not realizing these subtleties may lead to incomplete security proofs when using non-interactive NCE as a building block for UC-secure protocols. In this section, we prove that the definition of non-interactive NCE is valid in the UC framework.

2.1 Non-Committing Encryption

Canetti et al. [7] introduced the notion of NCE as an analogue to the way that public-key encryption is used to instantiate secure channels in the static setting. That is, NCE is defined as a multiparty protocol realizing the n-party functionality $f_{\text{SMT}}(\mu, \lambda, \ldots, \lambda) = (\lambda, \mu, \lambda, \ldots, \lambda)$,[4] for $\mu \in \{0,1\}^*$, that is secure in the presence of adaptive semi-honest adversaries that can corrupt a subset of the parties. The authors constructed an n-party protocol that is an $(n-1)$-resilient NCE scheme assuming the existence of a common-domain trapdoor system, and observed that basing the protocol on specific number-theoretic assumptions, such as RSA or CDH, yields two-party protocols of two rounds.

The definition above encounters several weaknesses. It considers a multiparty protocol in order to compute essentially a functionality involving two parties. In addition, the definition allows a subset of the parties to remain uncorrupted, which is undesirable in order to achieve composition of protocols in the adaptive setting. Furthermore, the adversary is limited to be semi-honest, and finally, the security model of [7] is somewhat weak as it does not even allow for sequential composition. Following these observations, Damgård and Nielsen [13] introduced a stronger definition of NCE as a two-party protocol for the two-party secure message transmission functionality f_{SMT}, in the presence of adaptive malicious adversaries, in the framework of Canetti [4].

Definition 2.1 (strong NCE)**.** *A strong non-committing encryption is a two-party protocol that securely computes the two-party functionality $f_{\text{SMT}}(\mu, \lambda) = (\lambda, \mu)$, for $\mu \in \{0,1\}^*$, in the presence of adaptive malicious adversaries that can corrupt an arbitrary number of parties.*

The definition above does not require non-interactiveness, and indeed the authors proposed an interactive strong NCE protocol, assuming the existence of simulatable public-key encryption schemes.

Non-interactive NCE can be defined by extending Definition 2.1 and requiring that the protocol will consist of 2 rounds. However, proving that a protocol is adaptively secure is quite a tedious task. A simpler definition that captures the non-interactive property of non-committing encryption is given by Canetti et al. [8]. According to this definition, an NCE scheme is a public-key encryption scheme in which public keys and ciphertexts can be simulated and later be explained for any message.

[4]The input of P_1 is μ, the output of P_2 is μ, and all other parties have no input nor output.

Definition 2.2 (non-interactive NCE)**.** *A non-interactive non-committing (bit) encryption scheme consists of four algorithms* (Gen, Enc, Dec, Sim) *such that the following properties hold:*

- *The triplet* (Gen, Enc, Dec) *forms a public-key encryption scheme.*
- Sim *is a simulation algorithm that on input* 1^κ*, outputs* $(pk, c, \rho_G^0, \rho_E^0, \rho_G^1, \rho_E^1)$*, such that for any* $\mu \in \{0,1\}$ *the following distributions are computationally indistinguishable:*
  - *the joint view of an honest sender and an honest receiver in a normal encryption of* μ
  $$\{(pk, c, r_G, r_E) \mid (sk, pk) = \mathrm{Gen}(1^\kappa; r_G), c = \mathrm{Enc}(pk, \mu; r_E)\},$$
  - *the simulated view of an encryption of* μ
  $$\left\{(pk, c, \rho_G^\mu, \rho_E^\mu) \mid (pk, c, \rho_G^0, \rho_E^0, \rho_G^1, \rho_E^1) \leftarrow \mathrm{Sim}(1^\kappa)\right\}.$$

It is easy to verify that in the framework of [4], non-interactive NCE as in Definition 2.2 implies strong NCE. This follows since upon a corruption of either party, the simulator learns the message μ and can provide the appropriate randomness.

2.2 Non-Interactive NCE in the UC Framework

The definition of strong NCE can be easily adjusted to the UC framework, by considering protocols that UC-realize the secure message transmission ideal functionality $\mathcal{F}_{\text{SMT}}^l$ (see Appendix A.4.1). It is also not hard to see that the (interactive) protocol presented in [13] is UC-secure. However, when trying to use non-interactive NCE in the UC framework, things are not as immediate. Consider the standard protocol π_{SMT} for realizing $\mathcal{F}_{\text{SMT}}^l$ using non-interactive NCE, as presented in Figure 1.

Protocol π_{SMT}

Let (Gen, Enc, Dec, Sim) be a non-interactive NCE scheme.

- Upon the first activation with sid, the receiver P_2 computes $(sk, pk) \leftarrow \mathrm{Gen}(1^\kappa)$ and sends (sid, pk) to P_1.
- Upon receiving $(\mathsf{send}, \mathsf{sid}, \mu)$ from $\mathcal{Z}$ and having received (sid, pk) from P_2, party P_1 encrypts the message $c \leftarrow \mathrm{Enc}_{pk}(\mu)$ and sends (sid, c) to P_2.
- Having received (sid, c) from P_1, party P_2 decrypts $\mu' = \mathrm{Dec}_{sk}(c)$ and outputs $(\mathsf{sent}, \mathsf{sid}, \mu')$.

Figure 1: The adaptive, malicious secure message transmission protocol

The difficulty arises from a subtle difference between the framework of [4] and the UC framework. In the former, the parties are set with their inputs *before* the protocol begins, whereas in the later, the environment can adaptively set the inputs of the parties, meaning that parties may be set with inputs *after* the protocol has started. This may lead into a potential attack on π_{SMT} in the UC framework. The environment first activates the receiver P_2 (without input). The adversary waits for the public key pk to be sent from P_2 to P_1, and corrupts P_2 after it is sent. At this point, the internal state of P_2, which consists of the random coins r_G, used to generate (sk, pk), is revealed to the adversary which can pass it to the environment. Next, the environment activates the sender

P_1 with a uniformly chosen bit $\mu \in_R \{0,1\}$, and the protocol resumes: P_1 encrypts $c \leftarrow \mathrm{Enc}_{pk}(\mu)$ and sends the ciphertext c to P_2. Once the adversary receives c, it sends it to the environment. The environment now has possession of r_G and c, and can verify that c decrypts to μ.

The ideal-process adversary cannot use committing public key and ciphertext, generated by Gen and Enc, during the simulation, since he must be able to explain the transcript upon a late corruption of the parties (after the bit μ has been provided by the environment). However, if the ideal-process adversary simulates this scenario using non-committing public key and ciphertext, generated as $(pk, c, \rho_G^0, \rho_E^0, \rho_G^1, \rho_E^1) \leftarrow \mathrm{Sim}(1^\kappa)$, it needs to guess whether to reveal ρ_G^0 or ρ_G^1 as the random coins of P_1 upon the first corruption, and the ciphertext c will fail to decrypt to μ with probability $1/2$.

Fortunately, there is a solution to this issue. The key observation is that although for any simulated public key pk there exists a ciphertext c such that the pair (pk, c) is equivocal, the public key pk can still be used to encrypt other messages, albeit in a committing way. Therefore, if the ideal-process adversary $\mathcal{S}$ generates $(pk, c, \rho_G^0, \rho_E^0, \rho_G^1, \rho_E^1) \leftarrow \mathrm{Sim}(1^\kappa)$ and receives a corruption request of the receiver P_2 after the first message pk has been simulated and before the sender P_1 has been activated with an input, $\mathcal{S}$ can choose the random coins for P_2 arbitrarily between ρ_G^0 and ρ_G^1. Say $\mathcal{S}$ sets ρ_G^0 as the random coins, this means that c is now a committing encryption of 0, however, it will no longer be used. Next, once the environment activates P_1 with some bit μ, $\mathcal{S}$ receives μ from $\mathcal{F}_{\text{SMT}}^l$ and can use the public key pk with fresh random coins r_E in order to encrypt μ as $c' = \mathrm{Enc}_{pk}(\mu; r_E)$. The second message is now simulated using c' rather than c. Upon a late corruption of P_1, $\mathcal{S}$ sets the random coins to be r_E. Indistinguishability from the view of $\mathcal{A}$ in the real execution follows since otherwise the simulated public key generated using Sim can be distinguished from a public key generated using Gen.

Theorem 2.3. *If* $(\mathrm{Gen}, \mathrm{Enc}, \mathrm{Dec}, \mathrm{Sim})$ *is a non-interactive non-committing encryption scheme then Protocol* π_{SMT} *UC-realizes* $\mathcal{F}_{\text{SMT}}^l$*, in the presence of adaptive malicious adversaries.*

Proof. Let $\mathcal{A}$ be the dummy adversary and let $\mathcal{Z}$ be an environment.[5] We construct an ideal-process adversary $\mathcal{S}$, interacting with the environment $\mathcal{Z}$, the ideal functionality $\mathcal{F}_{\text{SMT}}^l$ and with ideal (dummy) parties $\tilde{P}_1$ and $\tilde{P}_2$. $\mathcal{S}$ constructs virtual real-model parties P_1 and P_2, and simulates the interaction of the dummy adversary $\mathcal{A}$ with π_{SMT}.

Simulating the first message: Before the beginning of the simulation, the simulator $\mathcal{S}$ computes $(\tilde{pk}, \tilde{c}, \rho_G^0, \rho_E^0, \rho_G^1, \rho_E^1) \leftarrow \mathrm{Sim}(1^\kappa)$. If the first message needs to be simulated (i.e., if $\tilde{P}_2$ is honest and has been activated by $\mathcal{Z}$), $\mathcal{S}$ simulates it as $(\mathsf{sid}, \tilde{pk})$.

Explain corruptions before the simulation of the second message: (Note that before the simulation of the second message, either $\tilde{P}_1$ has not been activated with input, or $\tilde{P}_1$ has been activated with input but $\tilde{P}_2$ was not activated yet; in the later case the first message was not simulated yet.)

- **Explain a corruption of** P_1**:** Upon a corruption of P_1, $\mathcal{S}$ corrupts $\tilde{P}_1$ and proceeds as follows. $\mathcal{S}$ sets the contents of the random tape to be a uniformly distributed string r_1. If $\tilde{P}_1$ has already been activated with input $(\mathsf{send}, \mathsf{sid}, \mu)$, set the contents of P_1's input tape to

[5]Recall that following [5, Claim 10], security with respect to the dummy adversary implies security against arbitrary adversaries.

$(\mathsf{send}, \mathsf{sid}, \mu)$. Otherwise, upon a future activation of $\tilde{P}_1$, set the contents of P_1's input tape as above.

- **Explain a corruption of P_2:** Upon a corruption of P_2, $\mathcal{S}$ corrupts $\tilde{P}_2$ and sets the contents of P_2's random tape to be $r_2 = \rho_G^0$.

Simulating the second message (sid, c): The simulation of the second message is done based on the status of $\tilde{P}_1$ and $\tilde{P}_2$ at that point in the simulation. (Note that the second message needs to be simulated *only after* $\mathcal{Z}$ activates $\tilde{P}_1$ with input.)

- If both $\tilde{P}_1$ and $\tilde{P}_2$ are honest, $\mathcal{S}$ sets the message to be $(\mathsf{sid}, \tilde{c})$.
- If $\tilde{P}_1$ is honest and $\tilde{P}_2$ is corrupted, then $\mathcal{S}$ learns the output $(\mathsf{sent}, \mathsf{sid}, \mu)$ that $\tilde{P}_2$ received from $\mathcal{F}_{\text{SMT}}^l$. Next:
  - If $\tilde{P}_2$ was honest when the first message was simulated as $(\mathsf{sid}, \tilde{pk})$, $\mathcal{S}$ samples a uniformly random string r_1, computes $c = \text{Enc}_{\tilde{pk}}(\mu; r_1)$ and sets the second message to be (sid, c).
  - If $\tilde{P}_2$ was corrupted before the first message, denote by (sid, pk) the message sent by the environment. Next, $\mathcal{S}$ computes $c = \text{Enc}_{pk}(\mu; r_1)$, using a uniformly random string r_1, and sets the second message to be (sid, c).
- In case $\tilde{P}_1$ is corrupted, there is no need to simulate the second message. In this case, if $\tilde{P}_2$ is honest, denote by (sid, c) the message sent by the environment, decrypt c using ρ_G^0 to a bit μ and send $(\mathsf{send}, \tilde{P}_1, \tilde{P}_2, \mathsf{sid}, \mu)$ to $\mathcal{F}_{\text{SMT}}^l$.

Explain corruptions after the simulation of the second message: (Note that the second message is simulated after $\tilde{P}_1$ has been activated with input; in this case $\tilde{P}_2$ has received the output message $(\mathsf{sent}, \mathsf{sid}, \mu)$ from $\mathcal{F}_{\text{SMT}}^l$.)

- **Explain a corruption of P_1:** Upon a corruption of P_1, $\mathcal{S}$ corrupts $\tilde{P}_1$, learns its input $(\mathsf{send}, \mathsf{sid}, \mu)$, and proceeds by setting the contents of P_1's input tape to $(\mathsf{send}, \mathsf{sid}, \mu)$. Next:
  - **If $\tilde{P}_2$ is honest:** $\mathcal{S}$ sets the contents of the random tape to be $r_1 = \rho_E^\mu$.
  - **If $\tilde{P}_2$ is corrupted:** In case the second message was simulated using $\tilde{c}$, $\mathcal{S}$ sets and the contents of P_1's random tape to be $r_1 = \rho_E^\mu$; otherwise the contents of the random tape is set to be r_1 that was determined during the simulation of the second message.
- **Explain a corruption of P_2:** Upon a corruption of P_2, $\mathcal{S}$ corrupts $\tilde{P}_2$, learns its output tape $(\mathsf{sent}, \mathsf{sid}, \mu)$ and sets the content of P_2's output tape to be $(\mathsf{sent}, \mathsf{sid}, \mu)$. Next:
  - In case the second message was simulated using $\tilde{c}$, $\mathcal{S}$ sets the contents of the random tape to be $r_2 = \rho_G^\mu$.
  - In case the second message was sent by the environment (i.e., if $\tilde{P}_1$ was corrupted), $\mathcal{S}$ sets the contents of the random tape to be $r_2 = \rho_G^0$.

We now prove computational indistinguishability between the view of $\mathcal{Z}$ when interacting with the dummy adversary and parties P_1 and P_2 running π_{SMT} and the view of $\mathcal{Z}$ when interacting with $\mathcal{S}$ in the ideal computation of $\mathcal{F}_{\text{SMT}}^l$. Let $\mathcal{Z}$ be an environment that distinguishes with probability

$1/2+\epsilon$ between the real computation of π_{SMT} with the dummy adversary and the ideal computation of $\mathcal{F}^{l}_{\text{SMT}}$ with simulator $\mathcal{S}$. We construct a distinguisher $\mathcal{D}$ for the non-interactive NCE scheme. Initially, $\mathcal{D}$ receives a tuple $(pk^*, c^*, r^*_G, r^*_E)$ and invokes the environment $\mathcal{Z}$ on the value written on its advice tape. The environment $\mathcal{Z}$ can activate the parties and send corruption requests.

The distinguisher follows the operations of the simulator $\mathcal{S}$ with the following exceptions.

- The first message is simulated as (sid, pk^*) (recall that this is needed only if P_2 is honest).
- Upon a corruption of P_2 (before or after the simulation of the second message) $\mathcal{D}$ sets the random coins of P_2 to be $r_2 = r^*_G$.
- When simulating the second message:
  - If the environment did not issue yet any corruption requests, i.e., both parties are honest, $\mathcal{D}$ simulates the second message as (sid, c^*).
  - If P_2 is corrupted but was honest during the simulation of the first message, $\mathcal{D}$ simulates the second message as (sid, c), where $c = \text{Enc}_{pk^*}(\mu; r_1)$.
  - If P_2 was corrupted before the simulation of the first message, $\mathcal{D}$ simulates the second message as (sid, c), where $c = \text{Enc}_{pk}(\mu; r_1)$ and pk was sent by $\mathcal{Z}$.
- If after the simulation of the second message P_2 is honest, $\mathcal{D}$ sets the contents of its output tape by decrypting the ciphertext from the second message (sid, c) (either simulated or sent by $\mathcal{Z}$) using r^*_G and sets the output to be the decrypted bit.
- Upon a corruption of P_1 after the simulation of the second message, then if the second message was simulated using c^*, $\mathcal{D}$ sets the random coins of P_1 to be $r_1 = r^*_E$ (otherwise to the string r_1 that was set during the simulation of the second message).

Whenever $\mathcal{Z}$ outputs a bit b, $\mathcal{D}$ outputs b and halts, with the following exceptions, in which case $\mathcal{D}$ outputs a uniformly distributed bit b'.:

1. If $\mathcal{D}$ simulated both messages as (sid, pk^*) and (sid, c^*), and at least one of the parties has been corrupted afterwards, and the decryption of c^* using r^*_G is different from the bit μ that was sent by $\mathcal{Z}$ to P_1 as $(\mathsf{send}, \mathsf{sid}, \mu)$, before the simulation of the second message.
2. If $\mathcal{Z}$ sent the second message as (sid, c), while P_2 is honest, and the decryption of c under r^*_G is different than the decryption of c^*) under r^*_G.

Denote by BAD the event that one of the above cases happened. It can be seen by inspection that if the event BAD did not happen, then:

- If the tuple $(pk^*, c^*, r^*_G, r^*_E)$ is generated as $(pk^*, c^*, \rho^0_G, \rho^0_E, \rho^1_G, \rho^1_E) \leftarrow \text{Sim}(1^\kappa)$, where $(r^*_G, r^*_E) = (\rho^\mu_G, \rho^\mu_E)$ for some $\mu \in \{0,1\}$, then the view of $\mathcal{Z}$ is identically distributed as its view in the ideal computation of $\mathcal{F}^{l}_{\text{SMT}}$ with simulator $\mathcal{S}$.
- If the tuple $(pk^*, c^*, r^*_G, r^*_E)$ is generated as $(sk^*, pk^*) = \text{Gen}(1^\kappa; r^*_G)$ and $c^* = \text{Enc}(pk^*, \mu; r^*_E)$, for some $\mu \in \{0,1\}$, then the view of $\mathcal{Z}$ is identically distributed as its view in the real computation of π_{SMT} with the dummy adversary.

Therefore, we conclude that:

$$\begin{aligned}\Pr[\mathcal{D}\text{ succeeds}] &\le \Pr[\mathcal{D}\text{ succeeds} \mid \mathsf{BAD}]\cdot\Pr[\mathsf{BAD}] + \Pr\left[\mathcal{D}\text{ succeeds} \mid \overline{\mathsf{BAD}}\right]\cdot\Pr\left[\overline{\mathsf{BAD}}\right]\\ &= \frac{1}{2}\cdot\Pr[\mathsf{BAD}] + \left(\frac{1}{2}+\epsilon\right)\cdot(1-\Pr[\mathsf{BAD}])\\ &= \frac{1}{2}+\epsilon\,(1-\Pr[\mathsf{BAD}])\,.\end{aligned}$$

From the security of the non-interactive NCE scheme it follows that $\epsilon\cdot(1-\Pr[\mathsf{BAD}])$ is negligible, and therefore so is ϵ. □

3 Proof of the Adaptively Secure Oblivious Transfer from CLOS

In this section we show that the proof of security of the adaptively secure OT in [8] (see also Lindell [26]) is incomplete and explain how to rectify it. We emphasize that the results in [8] are valid, and merely the proof is incomplete.

Canetti et al. [8] used an augmented version of non-interactive NCE in order to construct a protocol instantiating the adaptively secure 1-out-of-ℓ oblivious-transfer functionality $\mathcal{F}_{\mathrm{OT}}^{\ell}$ (see Appendix A.4.2). They considered a non-interactive NCE scheme with the additional algorithm OGen which allows to obliviously sample public keys without knowing their secret keys.

Definition 3.1 (augmented non-interactive NCE)**.** *An augmented non-interactive non-committing encryption scheme is a non-interactive NCE scheme* (Gen, Enc, Dec, Sim) *augmented with an oblivious-sampling algorithm for public keys* $pk \leftarrow \mathrm{OGen}(1^\kappa)$*. We require that the distribution of a public key generated by* Gen *is computationally indistinguishable from a public key generated by* OGen*, i.e.,*

$$\{pk \mid (sk,pk)\leftarrow \mathrm{Gen}(1^\kappa)\} \overset{\mathrm{c}}{\equiv} \{pk \mid pk \leftarrow \mathrm{OGen}(1^\kappa)\}\,.$$

Furthermore, the algorithm OGen *has invertible sampling, meaning that there exists an algorithm* I_{OGen} *such that the following distributions are computationally indistinguishable*

$$\{(1^\kappa, pk, r) \mid pk = \mathrm{OGen}(1^\kappa; r)\} \overset{\mathrm{c}}{\equiv} \{(1^\kappa, pk, I_{\mathrm{OGen}}(1^\kappa, pk)) \mid (sk,pk)\leftarrow \mathrm{Gen}(1^\kappa)\}\,.$$

Protocol π_{OT} in Figure 2 describes the adaptive OT protocol from [8]. The idea behind this construction is for the receiver to generate ℓ public keys such that it knows the secret key only to the ith one. The sender encrypts every message using the corresponding public key and sends all the ciphertexts to the receiver. The receiver can decrypt only the ith ciphertext and thus obtain only x_i.

Canetti et al. [8, Claim 4.2] proved that assuming (Gen, OGen, Enc, Dec, Sim) is an augmented non-interactive NCE scheme, then π_{OT} UC-realizes $\mathcal{F}_{\mathrm{OT}}^{\ell}$ in the presence of adaptive semi-honest adversaries. The idea behind the proof is for $\mathcal{S}$ to produce a non-committing transcript, i.e., for every $j\in[\ell]$, to generate $(pk_j, c_j, \rho^0_{G,j}, \rho^0_{E,j}, \rho^1_{G,j}, \rho^1_{E,j}) \leftarrow \mathrm{Sim}(1^\kappa)$. Next, the first message is simulated as $(\mathsf{sid}, pk_1,\ldots,pk_\ell)$ whereas the second message is simulated as $(\mathsf{sid}, c_1,\ldots,c_\ell)$. Upon a corruption of the ideal sender, $\mathcal{S}$ learns its input $(\mathsf{sender}, \mathsf{sid}, x_1,\ldots,x_\ell)$ and sets the virtual sender's random coins to be $(\rho^{x_1}_{E,1},\ldots,\rho^{x_\ell}_{E,\ell})$. Upon a corruption of the ideal receiver, $\mathcal{S}$ learns its input $(\mathsf{receiver}, \mathsf{sid}, i)$ and output (sid, x_i) and for every $j\in[\ell]\setminus\{i\}$ it computes the invertible sampling $\rho^j_G \leftarrow I_{\mathrm{OGen}}(1^\kappa, pk_j)$, denotes $\rho^i_G = \rho^{x_i}_G$ and finally sets the virtual receiver's random coins to be $(\rho^1_G,\ldots,\rho^\ell_G)$.

Protocol π_{OT}

Let $(\text{Gen}, \text{OGen}, \text{Enc}, \text{Dec}, \text{Sim})$ be a augmented non-interactive NCE scheme.

- Given input $(\mathsf{receiver}, \mathsf{sid}, i)$, the receiver R computes $(sk, pk_i) \leftarrow \text{Gen}(1^\kappa)$ and runs $\ell - 1$ times $pk_j \leftarrow \text{OGen}(1^\kappa)$ for $j \in [\ell] \setminus \{i\}$. Then R sends $(\mathsf{sid}, pk_1, \ldots, pk_\ell)$ to T.
- Given input $(\mathsf{sender}, \mathsf{sid}, x_1, \ldots, x_\ell)$, and having received $(\mathsf{sid}, pk_1, \ldots, pk_\ell)$ from R, sender T computes $c_j \leftarrow \text{Enc}_{pk_j}(x_j)$ for $j \in [\ell]$, and sends $(\mathsf{sid}, c_1, \ldots, c_\ell)$ to R.
- Having received $(\mathsf{sid}, c_1, \ldots, c_\ell)$ from T, receiver R computes $x_i = \text{Dec}_{sk}(c_i)$ and outputs (sid, x_i).

Figure 2: The adaptive, semi-honest oblivious-transfer protocol [8]

However, during the security proof of the protocol, it is assumed that upon a corruption of the ideal receiver, the ideal-process adversary knows its output x_i and so can denote $\rho_G^i = \rho_G^{x_i}$. As we discussed, although valid in the framework of [4], such an assumption cannot be made in the UC framework. Hence, the security proof should be adjusted to cater for the corruption strategy in which the environment activates the receiver and the adversary corrupts the receiver *immediately after* the first message is sent from R to T and *before* the sender is activated with its input.

Proposition 3.2. *If* $(\text{Gen}, \text{OGen}, \text{Enc}, \text{Dec}, \text{Sim})$ *is an augmented non-interactive non-committing encryption scheme then Protocol* π_{OT} *UC-realizes* $\mathcal{F}_{\text{OT}}^\ell$*, in the presence of adaptive semi-honest adversaries.*

Proof. The proof proceeds identically as the proof of [8, Claim 4.2] with the following modifications to the section *Dealing with "corrupt" commands*. This section explains the operations of $\mathcal{S}$ upon corruption requests *after* the first message has been sent. If the sender is corrupted before the receiver, the operations remain unchanged. Upon a corruption of the receiver (while the sender is honest), $\mathcal{S}$ must check whether the sender has been activated. If so, $\mathcal{S}$ resumes as in the original proof. Otherwise, if the sender has not yet been activated, $\mathcal{S}$ proceeds as follows: $\mathcal{S}$ corrupts the ideal receiver $\tilde{R}$, obtains its input $(\mathsf{receiver}, \mathsf{sid}, i)$ (but not its output – the output is not determined yet) and constructs a virtual real-model receiver R. The contents of the input tape of R is set to $(\mathsf{receiver}, \mathsf{sid}, i)$. In order to set the contents of R's random tape, for every $j \in [\ell] \setminus \{i\}$ $\mathcal{S}$ computes the invertible sampling $r_G^j \leftarrow I_{\text{OGen}}(1^\kappa, pk_j)$, denotes $r_G^i = \rho_G^0$ and finally sets the receiver's random coins to be $(r_G^1, \ldots, r_G^\ell)$.

The simulation can now proceed in several ways: either the second message needs to be simulated while the sender is honest or the sender is corrupted before the simulation of the second message.

- If a corruption request of the sender is *before* the simulation of the second message and *before* the ideal sender $\tilde{T}$ has been activated with input, $\mathcal{S}$ first corrupts $\tilde{T}$ and constructs a virtual real-model sender T. Since the sender has not yet been activated, the input tape need not be set, but only the random tape. For this, $\mathcal{S}$ samples uniformly distributed random coins $(r_E^1, \ldots, r_E^\ell)$ as the random tape of T. Later, upon an activation of the ideal sender $\tilde{T}$ with input $(\mathsf{sender}, \mathsf{sid}, x_1, \ldots, x_\ell)$, $\mathcal{S}$ sets the input tape of T with $(\mathsf{sender}, \mathsf{sid}, x_1, \ldots, x_\ell)$. The second message is then simulated by computing $c'_j = \text{Enc}_{pk_j}(x_j; r_E^j)$ for every $j \in [\ell]$ and setting it to be $(\mathsf{sid}, c'_1, \ldots, c'_\ell)$.
- If a corruption request of the sender is *before* the simulation of the second message and *after* the ideal sender $\tilde{T}$ has been activated with input, $\mathcal{S}$ first corrupts $\tilde{T}$, obtains its input

$(\mathsf{sender}, \mathsf{sid}, x_1, \ldots, x_\ell)$ and constructs a virtual real-model sender T as follows. The input tape of T is set to $(\mathsf{sender}, \mathsf{sid}, x_1, \ldots, x_\ell)$. In order to set the contents of T's random tape, for every $j \in [\ell]$, $\mathcal{S}$ samples random coins r_E^j and encrypts $c'_j = \mathrm{Enc}_{pk_j}(x_j; r_E^j)$. The random coins of T are set as $(r_E^1, \ldots, r_E^\ell)$ and finally the second message is set to be $(\mathsf{sid}, c'_1, \ldots, c'_\ell)$.

- If the second message needs to be simulated while the sender is honest, then the sender must have been activated with its input, hence $\mathcal{S}$ obtains the output of the receiver x_i from the ideal functionality. Next, $\mathcal{S}$ samples r_E^i, computes $c'_i = \mathrm{Enc}_{pk_i}(x_i; r_E^i)$ and sets the message to be $(\mathsf{sid}, c_1, \ldots, c_{i-1}, c'_i, c_{i+1}, \ldots, c_\ell)$. Upon a corruption request of the sender, $\mathcal{S}$ first corrupts the ideal sender $\tilde{T}$, obtains its input $(\mathsf{sender}, \mathsf{sid}, x_1, \ldots, x_\ell)$ and constructs a virtual real-model sender T as follows. The contents of the input tape of T is set to $(\mathsf{sender}, \mathsf{sid}, x_1, \ldots, x_\ell)$, whereas in order to set the contents of T's random tape, for every $j \in [\ell] \setminus \{i\}$, $\mathcal{S}$ denotes $r_E^j = \rho_{E,j}^{x_j}$. The random coins of T are set as $(r_E^1, \ldots, r_E^\ell)$.

The sections *Simulating the receiver* and *Simulating the sender* in the proof in [8] also require adjustments, in order to cater for the scenarios where the protocol starts when one of the parties is corrupted, but one (or both) of the parties has not been activated with input. These adjustments follow similarly to the changes above. □

We note that adding initialization messages, such that a party sends OK once it is activated and the protocol begins only after both parties have been initialized, does not solve the problem. Consider an environment that activates the receiver R with input; R then sends OK to the sender. Next, the adversary corrupts R (before the sender is activated with input). The random tape of R should contain now the key generation random coins that will be used to generate (sk, pk_i) using Gen and pk_j using OGen for $j \in [\ell] \setminus \{i\}$. This means that although the message $(\mathsf{sid}, pk_1, \ldots, pk_\ell)$ has not been transmitted, it is essentially determined because the random coins that will generate it have been fixed.

4 Functionalities with One-Sided Polynomial-Size Domain

In this section, we focus on two-party deterministic functionalities for which the size of the input domain of one of the parties is polynomial in the security parameter, there are no restrictions on the input domain of the other party. More specifically, we consider functionalities of the form

$$f \colon D_1 \times \{0,1\}^{l_2} \to \{0,1\}^{m_1} \times \{0,1\}^{m_2},$$

where $D_1 \subseteq \{0,1\}^{l_1}$ and $|D_1| = O(\mathrm{poly}(\kappa))$.[6] The reason we consider D_1 to be a subset of $\{0,1\}^{l_1}$, rather than requiring that $l_1 = O(\log(\mathrm{poly}(\kappa)))$, is that we do not limit the functionality to receive short inputs. The input of P_1 may consists of l_1 bits, however there are polynomially many inputs.

In this situation, since P_2 knows the input domain of P_1, it can locally compute all possible values $(y_x^1, y_x^2) = f(x, x_2)$, for every $x \in D_1$. P_1 should retrieve only $y_{x_1}^1$, i.e., the output corresponding to its input x_1. This is exactly the requirement of oblivious transfer, therefore using adaptively secure OT, P_1 obtains $y_{x_1}^1$ and nothing else whereas P_2 does not learn anything about x_1.

If P_2 also receives an output, then it may learn something about P_1's input, in particular, P_2 learns that the input of P_1 lies in the preimage of its output $y_{x_1}^2$ under the function $f_2(\cdot, x_2)$.

[6]The idea of using OT over domain which is of polynomial size first appeared in Poupard and Stern [29].

However, this is valid in the setting of secure function evaluation, because this information is leaked from the output of the functionality, and therefore can also be learned in the ideal process. In order for P_2 to get its output $y^2_{x_1}$ without revealing it to P_1, P_2 masks every output it computes with a random string u. Now, during the OT, P_1 receives $y^2_{x_1} \oplus u$ in addition to $y^1_{x_1}$ and returns it to P_2 that can remove the mask.

Protocol $\pi^{\text{POLY}}_{\text{SFE}}$

Common input: A description of a two-party function $f\colon D_1 \times \{0,1\}^{l_2} \to \{0,1\}^{m_1} \times \{0,1\}^{m_2}$ and of the domain D_1 of P_1.

- Upon receiving $(\mathsf{input}, \mathsf{sid}, x_1)$ from $\mathcal{Z}$, party P_1 sends $(\mathsf{receiver}, \mathsf{sid}, x_1)$ to $\mathcal{F}^{\ell}_{\text{OT}}$.
- Upon receiving $(\mathsf{input}, \mathsf{sid}, x_2)$ from $\mathcal{Z}$, party P_2 operates as follows:
  1. Sample a random string $u \in \{0,1\}^{m_2}$.
  2. For every $x \in D_1$, compute $(y^1_x, y^2_x) = f(x, x_2)$.
  3. Denote by Y the ordered tuple $(y^1_x, y^2_x \oplus u)$ for every $x \in D_1$.
  4. Send to $\mathcal{F}^{\ell}_{\text{OT}}$ the message $(\mathsf{sender}, \mathsf{sid}, Y)$.
- Upon receiving the message $(\mathsf{sid}, (w_1, w_2))$ from $\mathcal{F}^{\ell}_{\text{OT}}$, party P_1 sends (sid, w_2) to P_2 and outputs $(\mathsf{output}, \mathsf{sid}, w_1)$.
- Upon receiving (sid, w_2) from P_1, party P_2 outputs $(\mathsf{output}, \mathsf{sid}, w_2 \oplus u)$.

Figure 3: The adaptive, semi-honest two-party SFE protocol, in the $\mathcal{F}^{\ell}_{\text{OT}}$-hybrid model

Theorem 4.1. *Let f be a deterministic two-party functionality where the cardinality of the domain of P_1 is polynomial in the security parameter. Then Protocol $\pi^{\text{POLY}}_{\text{SFE}}$ UC-realizes $\mathcal{F}^{f}_{\text{SFE}}$ in the $\mathcal{F}^{\ell}_{\text{OT}}$-hybrid model in the presence of adaptive semi-honest adversaries.*

Proof. Let $\mathcal{A}$ be an adaptive, semi-honest adversary attacking $\pi^{\text{POLY}}_{\text{SFE}}$ in the $\mathcal{F}^{\ell}_{\text{OT}}$-hybrid model and let $\mathcal{Z}$ be an environment. We construct an ideal-process adversary $\mathcal{S}$, interacting with the environment $\mathcal{Z}$, the ideal functionality $\mathcal{F}^{f}_{\text{SFE}}$ and with ideal (dummy) parties $\tilde{P}_1$ and $\tilde{P}_2$. $\mathcal{S}$ constructs virtual real-model parties P_1 and P_2, and runs the adversary $\mathcal{A}$. $\mathcal{S}$ must simulate the view for $\mathcal{A}$, i.e., its communication with $\mathcal{Z}$, the messages sent by the uncorrupted parties and the internal states of the corrupted parties (including the interface to the ideal functionality $\mathcal{F}^{\ell}_{\text{OT}}$).

In order to simulate the communication with $\mathcal{Z}$, every input value that $\mathcal{S}$ receives from $\mathcal{Z}$ is written on $\mathcal{A}$'s input tape. Likewise, every output value written by $\mathcal{A}$ on its output tape is copied to $\mathcal{S}$'s own output tape.

We next explain how $\mathcal{S}$ simulates the message (sid, w_2) and how it behaves upon receiving corruption requests. Note that the message (sid, w_2) is sent in the protocol only *after* both P_1 and P_2 have been activated with input. In addition, each dummy party forwards its input immediately to $\mathcal{F}^{f}_{\text{SFE}}$ (both when it is honest and when it is corrupted) and $\mathcal{F}^{f}_{\text{SFE}}$ immediately computes the output once it receives both inputs. It follows that if a dummy party is corrupted after the simulation of the message, $\mathcal{S}$ learns both its input and its output.

Explain corruptions before simulating the message (sid, w_2):

- **Explain a corruption of P_1:** Upon a corruption of P_1, $\mathcal{S}$ corrupts $\tilde{P}_1$ and continues as follows. If $\tilde{P}_1$ has already been activated with input $(\mathsf{input}, \mathsf{sid}, x_1)$, $\mathcal{S}$ proceeds by setting the contents of P_1's input tape to $(\mathsf{input}, \mathsf{sid}, x_1)$ and the outgoing message to $\mathcal{F}_{\text{OT}}$ to $(\mathsf{receiver}, \mathsf{sid}, x_1)$. (Note that P_1 acts deterministically throughout the protocol, hence it has no random tape). In case $\tilde{P}_1$ has not been activated with input yet, $\mathcal{S}$ waits until it is activated and proceeds as above.

- **Explain a corruption of P_2:** Upon a corruption of P_2, $\mathcal{S}$ corrupts $\tilde{P}_2$ and continues as follows. If $\tilde{P}_2$ has already been activated with input $(\mathsf{input}, \mathsf{sid}, x_2)$, $\mathcal{S}$ proceeds by setting the contents of P_2's input tape to $(\mathsf{input}, \mathsf{sid}, x_2)$, samples a random string $u \in \{0,1\}^{m_2}$, sets the random tape of P_2 to u, computes Y as in the protocol and sets the outgoing message to $\mathcal{F}^{\ell}_{\text{OT}}$ to $(\mathsf{sender}, \mathsf{sid}, Y)$. In case $\tilde{P}_1$ has not been activated with input yet, $\mathcal{S}$ waits until it is activated and proceeds as above.

Simulating the message (sid, w_2): Note that the message is sent *after* P_1 has received the output from $\mathcal{F}^{\ell}_{\text{OT}}$, i.e., after both parties have been activated with input.

- **Case 1:** *P_2 is honest.* $\mathcal{S}$ samples a uniformly random string $w_2 \in \{0,1\}^{m_2}$ and sets the message to be (sid, w_2).

- **Case 2:** *P_2 is corrupted.* In this case $\tilde{P}_2$ must have received the output message $(\mathsf{output}, \mathsf{sid}, y_2)$ from $\mathcal{F}^{f}_{\text{SFE}}$ (and the value u has been already set). $\mathcal{S}$ computes $w_2 = y_2 \oplus u$ and sets the message to be (sid, w_2).

Explain corruptions after the simulation of (sid, w_2):

- **Explain a corruption of P_1:** Upon a corruption of P_1, $\mathcal{S}$ corrupts $\tilde{P}_1$, learns its input $(\mathsf{input}, \mathsf{sid}, x_1)$ and its output $(\mathsf{output}, \mathsf{sid}, y_1)$, and proceeds by setting the contents of P_1's input tape to $(\mathsf{input}, \mathsf{sid}, x_1)$, the output tape to $(\mathsf{output}, \mathsf{sid}, y_1)$, the outgoing message to $\mathcal{F}^{\ell}_{\text{OT}}$ to $(\mathsf{receiver}, \mathsf{sid}, x_1)$ and the incoming message from $\mathcal{F}^{\ell}_{\text{OT}}$ to $(\mathsf{sid}, (y_1, w_2))$.

- **Explain a corruption of P_2:** Upon a corruption of P_2, $\mathcal{S}$ corrupts $\tilde{P}_2$, learns its input $(\mathsf{input}, \mathsf{sid}, x_2)$ and its output $(\mathsf{output}, \mathsf{sid}, y_2)$, and proceeds by setting the contents of P_2's input tape to $(\mathsf{input}, \mathsf{sid}, x_2)$ and of the output tape to $(\mathsf{output}, \mathsf{sid}, y_2)$. Next, $\mathcal{S}$ sets the contents of P_2's random tape to be $u = y_2 \oplus w_2$, computes Y as in the protocol and sets the message to $\mathcal{F}^{\ell}_{\text{OT}}$ to be $(\mathsf{sender}, \mathsf{sid}, Y)$.

Proving (perfect) indistinguishability between the view of $\mathcal{A}$ when interacting with $\mathcal{Z}$, parties P_1, P_2 and with the ideal functionality $\mathcal{F}^{\ell}_{\text{OT}}$ and the view of $\mathcal{A}$ when interacting with $\mathcal{S}$ follows in a straight-forward way. □

Using the adaptively secure OT presented in [8] (see Section 3) and using the composition theorem from [5], we obtain the following corollary:

Corollary 4.2. *Assuming the existence of augmented non-interactive NCE schemes, every deterministic two-party functionality, for which the cardinality of the domain of P_1 is polynomial in the security parameter, can be securely UC-realized, in the presence of adaptive semi-honest adversaries using a three-message protocol.*

We note that this approach does not extend to probabilistic functionalities. The reason is that if P_2 locally computes f, then it must know the random coins used in the computation. However, this information is not available to the ideal-process adversary if only P_2 is corrupted. Alternatively, when using the standard transformation from a randomized functionality into a deterministic one, by computing $g((x_1, r_1), (x_2, r_2)) = f(x_1, x_2; r_1 \oplus r_2)$, the input domain of P_1 is no longer polynomial.

Another important corollary from Theorem 4.1 is that in order to prove impossibility of constant-round adaptively secure two-party protocols, one must consider either functionalities where both parties have super-polynomial domains, or probabilistic functionalities.

5 Non-Committing Indistinguishability Obfuscation

An *indistinguishability obfuscator* [1, 17] for a circuit class $\{\mathcal{C}_\kappa\}$ is a PPT machine $i\mathcal{O}$ satisfying the following conditions:

Correctness: For every κ and every $C \in \mathcal{C}_\kappa$, it holds that C and $i\mathcal{O}(C)$ compute the same function.

Polynomial slowdown: There is a polynomial p such that for all $C \in \mathcal{C}_\kappa$, $|i\mathcal{O}(1^\kappa, C)| \leq p(\kappa) \cdot |C|$.

Indistinguishability: For any sequence $\{C_{\kappa,0}, C_{\kappa,1}, \mathsf{aux}_\kappa\}_\kappa$, where $C_{\kappa,0}, C_{\kappa,1} \in \mathcal{C}_\kappa$, $|C_{\kappa,0}| = |C_{\kappa,1}|$ and $C_{\kappa,0}, C_{\kappa,1}$ compute the same function, and for any non-uniform PPT distinguisher $\mathcal{D}$, there exists a negligible function negl such that:

$$|\Pr\left[\mathcal{D}\left(i\mathcal{O}\left(1^\kappa, C_{\kappa,0}\right), \mathsf{aux}_\kappa\right) = 1\right] - \Pr\left[\mathcal{D}\left(i\mathcal{O}\left(1^\kappa, C_{\kappa,1}\right), \mathsf{aux}_\kappa\right) = 1\right]| \leq \mathrm{negl}(\kappa).$$

Indistinguishability obfuscation has recently led to a construction of a two-round statically secure protocol [18] and to constant-round adaptively secure protocols in the CRS model [10, 11, 15]. We consider an adaptive analogue for $i\mathcal{O}$ called *non-committing indistinguishability obfuscation* and show that this primitive is *complete* for constructing adaptively secure protocols with round complexity that is independent of the function to compute. We emphasize that currently we do not know how to construct non-committing indistinguishability obfuscation, and that this result serves as a reduction from the problem of constructing adaptively secure protocols with round complexity independent of the function to the problem of constructing non-committing $i\mathcal{O}$.

Given a circuit class consisting of circuits that compute the same function, we would like to have an indistinguishability obfuscator $i\mathcal{O}$ augmented with a simulation algorithm Sim_1 that outputs a "canonical" obfuscated circuit and some state s, such that later, given any circuit from the class and the state, a second algorithm Sim_2 can explain the randomness for the obfuscation algorithm to generate the canonical circuit as an obfuscation of this circuit. We note that such a notion of non-committing $i\mathcal{O}$ is unlikely to exists in general, since this will provide an efficient solution to the *circuit equivalence problem*, which is co-NP complete, and so will imply a collapse of the polynomial hierarchy. Given two circuits C_0, C_1, if and only if the circuits are equivalent, then there exists a non-committing $i\mathcal{O}$ for this family and it is possible to first compute $(\tilde{C}, s) \leftarrow \mathsf{Sim}_1(1^\kappa)$ and later explain $\tilde{C}$ both as $r_0 \leftarrow \mathsf{Sim}_2(s, C_0)$ and as $r_1 \leftarrow \mathsf{Sim}_2(s, C_1)$.

We overcome this difficulty by considering equivalent circuits that do not receive any input, i.e., a family of constant circuits that produce the same output. The circuit equivalence problem is easy in this scenario since one simply runs both circuits and compares the outputs. More specifically, consider a circuit C computing a function f and an input vector $\boldsymbol{x}$. We hard-wire to each input wire (i.e., to each *input terminal* in the terminology of Goldreich [19]) the corresponding input value. This yields a circuit that computes the constant function $f_{\boldsymbol{x}} = f(\boldsymbol{x})$. We say that a circuit is a *constant circuit* if all its input wires have hard-wired values (an so it computes a constant function).

Definition 5.1. *A* non-committing indistinguishability obfuscator scheme *for a circuit class* $\{\mathcal{C}_\kappa\}$*, consisting of constant circuits, is a triplet of PPT algorithms* $\Pi = (i\mathcal{O}, \mathsf{Sim}_1, \mathsf{Sim}_2)$ *such that:*

- $i\mathcal{O}$ *is an indistinguishability obfuscator for* $\{\mathcal{C}_\kappa\}$*.*
- *Upon receiving* 1^κ*, an integer* m *and a value* y*,* Sim_1 *outputs a constant circuit* $\tilde{C}$ *(of size* m *and output* y*) and a state* s*.*
- *Upon receiving a circuit* $C \in \mathcal{C}_\kappa$ *and a state* s*,* Sim_2 *outputs a string* r*.*
- *For any non-uniform* PPT $\mathcal{D}$ *and for large enough* $\kappa \in \mathbb{N}$*, it holds that:*

$$\left|\Pr\left[\textsc{Expt}^{\text{REAL}}_{\Pi,\mathcal{D}}(\kappa) = 1\right] - \Pr\left[\textsc{Expt}^{\text{IDEAL}}_{\Pi,\mathcal{D}}(\kappa) = 1\right]\right| \leq \text{negl}(\kappa),$$

where the experiments $\textsc{Expt}^{\text{REAL}}_{\Pi,\mathcal{D}}$ *and* $\textsc{Expt}^{\text{IDEAL}}_{\Pi,\mathcal{D}}$ *are defined below, and the probability is over the random coins of the experiments and of* $\mathcal{D}$*.*

Experiment $\textsc{Expt}^{\text{REAL}}_{\Pi,\mathcal{D}}(\kappa)$	Experiment $\textsc{Expt}^{\text{IDEAL}}_{\Pi,\mathcal{D}}(\kappa)$
	Send 1^κ to $\mathcal{D}$ and get back a circuit $C \in \mathcal{C}_\kappa$.
Send 1^κ to $\mathcal{D}$ and get back a circuit $C \in \mathcal{C}_\kappa$.	Run the circuit C and compute the output y.
Sample a uniformly distributed string r.	Compute $(\tilde{C}, s) \leftarrow \mathsf{Sim}_1(1^\kappa, \|C\|, y)$.
Compute $\tilde{C} = i\mathcal{O}(1^\kappa, C; r)$.	Compute $r \leftarrow \mathsf{Sim}_2(s, C)$.
Send $(\tilde{C}, r)$ to $\mathcal{D}$ and get back a bit b.	Send $(\tilde{C}, r)$ to $\mathcal{D}$ and get back a bit b.
Return b.	Return b.

For our usage, we require that the depth of the circuit representing the obfuscator $i\mathcal{O}$ is independent of the depth of the (input variable) circuit C. This requirement is motivated by the construction of Garg et al. [17] for "standard" $i\mathcal{O}$, which satisfies this property.

We note that the technique of Katz et al. [25] does not seem to rule out non-committing $i\mathcal{O}$ for constant circuits, since the function that can be computed using the simulator is fixed in advance. Likewise, the technique of Nielsen [28] does not seem to work, since the number of constant circuits that can be explained is bounded in advance.

5.1 Adaptively Secure Protocol with Round Complexity Independent of f

We define the protocol in a hybrid model where the parties have access to an ideal obfuscate-circuit-with-input functionality $\mathcal{F}^C_{\text{OCWI}}$. For simplicity we present the protocol for public-output deterministic functionalities, and the extension to private-output randomized functionalities follows using standard techniques. $\mathcal{F}^C_{\text{OCWI}}$ is parametrized by a circuit C, each party sends its input to $\mathcal{F}^C_{\text{OCWI}}$, which hard-wires the inputs to the circuit, obfuscates it and returns the obfuscated circuit to the parties. Each party sends an additional random string that is used as a share of the random coins for the obfuscation. The obfuscate-circuit-with-input functionality is described in Figure 4.

Functionality $\mathcal{F}^{C}_{\text{OCWI}}$

The functionality $\mathcal{F}^{C}_{\text{OCWI}}$ proceeds as follows, interacting with parties $P_1, \ldots, P_n$ and an adversary $\mathcal{S}$, and parametrized by a circuit C and a non-committing $i\mathcal{O}$ scheme for constant circuits. For every party P_i initialize values $x_i \leftarrow \perp$ and $r_i \leftarrow 0$.

- Upon receiving $(\mathsf{ocwi\text{-}input}, \mathsf{sid}, (x, r))$ from P_i, set $x_i \leftarrow x$ and $r_i \leftarrow r$ and send a message $(\mathsf{ocwi\text{-}input}, \mathsf{sid}, P_i)$ to $\mathcal{S}$.
- Upon receiving $(\mathsf{ocwi\text{-}output}, \mathsf{sid})$ from P_i, do:
  1. If $x_i = \perp$ for some honest party P_i, return $\perp$.
  2. If the circuit $\tilde{C}$ has not been set yet:
     (a) Prepare the circuit C_1 by hard-wiring the values $x_1, \ldots, x_n$ to C.
     (b) Obfuscate the circuit C_1 as $\tilde{C} = i\mathcal{O}(1^\kappa, C_1; r_1 \oplus \ldots \oplus r_n)$.
  3. Send to P_i the obfuscated circuit $(\mathsf{ocwi\text{-}output}, \mathsf{sid}, \tilde{C})$.

Figure 4: The obfuscate-circuit-with-input Functionality

Based on the properties of non-committing $i\mathcal{O}$, the depth of a circuit computing $\mathcal{F}^{C}_{\text{OCWI}}$ depends only on the depth of the obfuscator $i\mathcal{O}$ and not the depth of C.

Protocol $\pi^{\text{NCIO}}_{\text{SFE}}$

Common input: an n-party functionality $f\colon (\{0,1\}^*)^n \to (\{0,1\}^*)^n$ and a circuit C computing f.

- Upon receiving $(\mathsf{input}, \mathsf{sid}, x_i)$ from $\mathcal{Z}$, party P_i samples a random string $r_i \in_R \{0,1\}^*$ and sends $(\mathsf{ocwi\text{-}input}, \mathsf{sid}, (x_i, r_i))$ to $\mathcal{F}^{C}_{\text{OCWI}}$.
- Upon receiving $(\mathsf{ocwi\text{-}output}, \mathsf{sid}, \tilde{C})$ from $\mathcal{F}^{C}_{\text{OCWI}}$, P_i runs the circuit $\tilde{C}$, receives an output y and outputs $(\mathsf{output}, \mathsf{sid}, y)$.

Figure 5: The adaptive, semi-honest protocol computing $\mathcal{F}^{f}_{\text{SFE}}$, in the $\mathcal{F}^{C}_{\text{OCWI}}$-hybrid model

Theorem 5.2. *Let f be an n-party functionality and let C be a circuit computing f. If $\Pi = (i\mathcal{O}, \mathsf{Sim}_1, \mathsf{Sim}_2)$ is a non-committing $i\mathcal{O}$ scheme for constant circuits, then Protocol $\pi^{\text{NCIO}}_{\text{SFE}}$ UC-realizes $\mathcal{F}^{f}_{\text{SFE}}$ in the $\mathcal{F}^{C}_{\text{OCWI}}$-hybrid model, in the presence of adaptive semi-honest adversaries.*

Proof. Let $\mathcal{A}$ be an adaptive, semi-honest adversary attacking $\pi^{\text{NCIO}}_{\text{SFE}}$ in the $\mathcal{F}^{C}_{\text{OCWI}}$-hybrid model and let $\mathcal{Z}$ be an environment. We construct an ideal-process adversary $\mathcal{S}$, interacting with the environment $\mathcal{Z}$, the ideal functionality $\mathcal{F}^{f}_{\text{SFE}}$ and with ideal (dummy) parties $\tilde{P}_1, \ldots, \tilde{P}_n$. $\mathcal{S}$ constructs virtual parties $P_1, \ldots, P_n$, and runs the adversary $\mathcal{A}$. $\mathcal{S}$ must simulate the view for $\mathcal{A}$, i.e., its communication with $\mathcal{Z}$, the internal states of the corrupted parties and the interface with $\mathcal{F}^{C}_{\text{OCWI}}$. Note that there is no communication between the parties during the protocol, therefore $\mathcal{S}$ need not simulate the messages sent by the uncorrupted parties,

In order to simulate the communication with $\mathcal{Z}$, every input value that $\mathcal{S}$ receives from $\mathcal{Z}$ is written on $\mathcal{A}$'s input tape. Likewise, every output value written by $\mathcal{A}$ on its output tape is copied to $\mathcal{S}$'s own output tape.

Simulating corruptions requests of party P_i: $\mathcal{S}$ corrupts $\tilde{P}_i$ and continues as follows:

- **If $\tilde{P}_i$ has not been activated with input yet:** $\mathcal{S}$ samples $r_i \in_R \{0,1\}^*$ and sets the contents of P_i's random tape to r_i.
- **If $\tilde{P}_i$ has been activated with input, but did not receive output yet:** $\mathcal{S}$ samples $r_i \in_R \{0,1\}^*$ and proceeds by setting the contents of P_i's input tape to $(\mathsf{input}, \mathsf{sid}, x_i)$, the contents of the random tape to r_i and the message P_i sends to $\mathcal{F}^C_{\text{OCWI}}$ to $(\mathsf{ocwi\text{-}input}, \mathsf{sid}, (x_i, r_i))$.
- **If $\tilde{P}_i$ has been activated with input, and has received output:** $\mathcal{S}$ sets the contents of P_i's input tape to $(\mathsf{input}, \mathsf{sid}, x_i)$. Next:
  - If $\tilde{C}$ has *not* been set yet (i.e., first time $\mathcal{S}$ learns the output y). $\mathcal{S}$ samples $r_i \in_R \{0,1\}^*$ and proceeds by setting the contents of P_i's random tape to r_i and the message P_i sends to $\mathcal{F}^C_{\text{OCWI}}$ to $(\mathsf{ocwi\text{-}input}, \mathsf{sid}, (x_i, r_i))$. Next, $\mathcal{S}$ computes $(\tilde{C}, s) \leftarrow \mathsf{Sim}_1(1^\kappa, |C|, y)$, and sets the messages from $\mathcal{F}^C_{\text{OCWI}}$ to be $(\mathsf{ocwi\text{-}output}, \mathsf{sid}, \tilde{C})$ for each corrupted party.
  - If $\tilde{C}$ has *already* been set:
    * If *not* all parties are corrupted, $\mathcal{S}$ samples $r_i \in_R \{0,1\}^*$ and proceeds by setting the contents of P_i's random tape to r_i, the message P_i sends to $\mathcal{F}^C_{\text{OCWI}}$ to $(\mathsf{ocwi\text{-}input}, \mathsf{sid}, (x_i, r_i))$ and the message it receives to $(\mathsf{ocwi\text{-}output}, \mathsf{sid}, \tilde{C})$.
    * Upon the n'th corruption, $\mathcal{S}$ computes the circuit C_1 by hard-wiring the input values $x_1, \ldots, x_n$ to C and computes $r \leftarrow \mathsf{Sim}_2(s, C_1)$. Next, $\mathcal{S}$ computes $r_i = r \oplus (\oplus_{j \neq i} r_j)$ and proceeds by setting the contents of P_i's random tape to r_i, the message P_i sends to $\mathcal{F}^C_{\text{OCWI}}$ to $(\mathsf{ocwi\text{-}input}, \mathsf{sid}, (x_i, r_i))$ and the message it receives to $(\mathsf{ocwi\text{-}output}, \mathsf{sid}, \tilde{C})$.

Upon an activation of a corrupted party: Once a corrupted dummy party $\tilde{P}_i$ is activated with input $(\mathsf{input}, \mathsf{sid}, x_i)$, $\mathcal{S}$ forward the input mesage to the ideal functionality $\mathcal{F}^f_{\text{SFE}}$ on behalf of party P_i and proceeds by setting the contents of P_i's input tape to $(\mathsf{input}, \mathsf{sid}, x_i)$ and the message P_i sends to $\mathcal{F}^C_{\text{OCWI}}$ to $(\mathsf{ocwi\text{-}input}, \mathsf{sid}, (x_i, r_i))$. (Recall that the random tape r_i has been set during the corruption.) In case *all* the parties have already been activated with input, $\mathcal{S}$ learns the output of the corrupted party $(\mathsf{output}, \mathsf{sid}, y)$ and operates as follows:

- If *all* parties are corrupted, $\mathcal{S}$ computes $\tilde{C}$ as in $\mathcal{F}^C_{\text{OCWI}}$ using the values (x_j, r_j) for every $j \in [n]$, and sets the messages from $\mathcal{F}^C_{\text{OCWI}}$ to be $(\mathsf{ocwi\text{-}output}, \mathsf{sid}, \tilde{C})$ for each party.
- If *not* all parties are corrupted, $\mathcal{S}$ computes $(\tilde{C}, s) \leftarrow \mathsf{Sim}_1(1^\kappa, |C|, y)$, and sets the messages from $\mathcal{F}^C_{\text{OCWI}}$ to be $(\mathsf{ocwi\text{-}output}, \mathsf{sid}, \tilde{C})$ for each corrupted party.

We now prove computational indistinguishability between the view of $\mathcal{Z}$ when interacting with $\mathcal{A}$, parties $P_1, \ldots, P_n$ running $\pi^{\text{NCIO}}_{\text{SFE}}$ and the view of $\mathcal{Z}$ when interacting with $\mathcal{S}$ in the ideal computation of $\mathcal{F}^f_{\text{SFE}}$. Let $\mathcal{Z}$ be an environment that distinguishes between the real computation of $\pi^{\text{NCIO}}_{\text{SFE}}$ with adversary $\mathcal{A}$ and the ideal computation of $\mathcal{F}^f_{\text{SFE}}$ with simulator $\mathcal{S}$. We construct a distinguisher $\mathcal{D}$ for the NCIO scheme. Initially, $\mathcal{D}$ receives 1^κ and invokes the environment $\mathcal{Z}$ on its advice and the adversary $\mathcal{A}$; the distinguisher follows the operations of the simulator $\mathcal{S}$, and in particular forwards all messages between $\mathcal{Z}$ and $\mathcal{A}$.

Initially, $\mathcal{Z}$ sends input messages to the parties and $\mathcal{A}$ may send corruption requests. When $\mathcal{D}$ receives from $\mathcal{Z}$ an input value x_i for party P_i, $\mathcal{D}$ stores the value.[7] When $\mathcal{D}$ receives a corruption request for some P_i, it follows the operations of $\mathcal{S}$, i.e., samples a random string r_i, returns r_i to $\mathcal{Z}$ as P_i's random coins and sets the value for $\mathcal{F}^{C}_{\text{OCWI}}$ to be (x_i, r_i). If $\mathcal{Z}$ outputs a bit b before distributing input values for all the parties, $\mathcal{D}$ sends a constant circuit of size $|C|$ with output 0 to the challenger, receives back $(\tilde{C}, \tilde{r})$, returns b and halts.

Otherwise, once $\mathcal{D}$ received input values for all the parties, it prepares the circuit C_1 by hard-wiring the input values to the circuit C, sets the output value for each (corrupted and uncorrupted) party to be the output value of C_1, sends C_1 to the challenger and gets back the response $(\tilde{C}, \tilde{r})$. We consider two cases:

- If the adversary $\mathcal{A}$ corrupts all the parties before all the input values were set, $\mathcal{D}$ ignores the response it received from the challenger.[8] Instead, $\mathcal{D}$ obfuscates the circuit $\tilde{C} = i\mathcal{O}(C_1; \oplus r_i)$ and sets the response from $\mathcal{F}^{C}_{\text{OCWI}}$ to be $\tilde{C}$ for all the parties.

- If the environment gave inputs to all the parties before the adversary corrupted all of them, $\mathcal{D}$ sets the output from $\mathcal{F}^{C}_{\text{OCWI}}$ to be $\tilde{C}$ for each corrupted party. $\mathcal{D}$ continues to answer a corruption request to party P_i as $\mathcal{S}$, by using uniformly distributed random coins r_i. Upon the n'th corruption request, for some party P_i, $\mathcal{D}$ computes the random coin as $r_i = \tilde{r} \oplus (\oplus_{j \neq i} r_j)$.

It can be seen by inspection that:

- In case $\mathcal{D}$ runs in the experiment $\textsc{Expt}^{\text{REAL}}_{\Pi,\mathcal{D}}(\kappa)$ the view of $\mathcal{Z}$ is identically distributed as its view in an execution of $\pi^{\text{NCIO}}_{\text{SFE}}$ in the $focwi$-hybrid model with adversary $\mathcal{A}$.

- In case $\mathcal{D}$ runs in the experiment $\textsc{Expt}^{\text{IDEAL}}_{\Pi,\mathcal{D}}(\kappa)$ the view of $\mathcal{Z}$ is identically distributed as its view in an ideal computation of $\mathcal{F}^{f}_{\text{SFE}}$ with $\mathcal{S}$.

It follows that $\mathcal{D}$ succeeds with the same probability as $\mathcal{Z}$. $\square$

When instantiating the ideal functionality $\mathcal{F}^{C}_{\text{OCWI}}$ using the protocol from Canetti et al. [8], the round complexity depends on the circuit representing $\mathcal{F}^{C}_{\text{OCWI}}$, which is independent from the depth of f. Hence, using the composition theorem from Canetti [5] we conclude with the following corollary.

Corollary 5.3. *Assume that enhanced trapdoor permutations, augmented non-committing encryption and non-committing $i\mathcal{O}$ scheme for constant circuits exist. Then for any adaptively well-formed multiparty functionality f, there exists a protocol that UC-realizes $\mathcal{F}^{f}_{\text{SFE}}$ in the presence of adaptive semi-honest adversaries, with round complexity that is independent of f.*

Acknowledgements

We would like to thank Yehuda Lindell for helpful discussions on the topic and to the anonymous referees for pointing to the work of [29] regarding OT over polynomial-size domain and for pointing out a problem in an earlier version of Definition 5.1.

[7]Note that since the adversary $\mathcal{A}$ is semi-honest, it does not change the input values for corrupted parties.

[8]Indeed, the simulation does not use the algorithms Sim_1 and Sim_2 in this case.

References

[1] B. Barak, O. Goldreich, R. Impagliazzo, S. Rudich, A. Sahai, S. Vadhan, and K. Yang. On the (Im)Possibility of Obfuscating Programs. In *Advances in Cryptology – CRYPTO 2001*, pages 1–18, 2001.

[2] D. Beaver and S. Haber. Cryptographic Protocols Provably Secure Against Dynamic Adversaries. In *Advances in Cryptology – EUROCRYPT '92*, pages 307–323, 1992.

[3] N. Bitansky, R. Canetti, and S. Halevi. Leakage-Tolerant Interactive Protocols. In *Proceedings of the 9th Theory of Cryptography Conference, TCC 2012*, pages 266–284, 2012.

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

[5] R. Canetti. Universally Composable Security: A New Paradigm for Cryptographic Protocols. In *Proceedings of the 42nd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 136–145, 2001.

[6] R. Canetti and M. Fischlin. Universally Composable Commitments. In *Advances in Cryptology – CRYPTO 2001*, pages 19–40, 2001.

[7] R. Canetti, U. Feige, O. Goldreich, and M. Naor. Adaptively Secure Multi-Party Computation. In *Proceedings of the 28th Annual ACM Symposium on Theory of Computing (STOC)*, pages 639–648, 1996.

[8] R. Canetti, Y. Lindell, R. Ostrovsky, and A. Sahai. Universally composable two-party and multi-party secure computation. In *Proceedings of the 34th Annual ACM Symposium on Theory of Computing (STOC)*, pages 494–503, 2002.

[9] R. Canetti, E. Kushilevitz, and Y. Lindell. On the Limitations of Universally Composable Two-Party Computation Without Set-Up Assumptions. *Journal of Cryptology*, 19(2):135–167, 2006.

[10] R. Canetti, S. Goldwasser, and O. Poburinnaya. Adaptively Secure Two-Party Computation from Indistinguishability Obfuscation. In *Proceedings of the 12th Theory of Cryptography Conference, TCC 2015, part II*, pages 557–585, 2015.

[11] D. Dachman-Soled, J. Katz, and V. Rao. Adaptively Secure, Universally Composable, Multi-party Computation in Constant Rounds. In *Proceedings of the 12th Theory of Cryptography Conference, TCC 2015, part II*, pages 586–613, 2015.

[12] I. Damgård and Y. Ishai. Constant-Round Multiparty Computation Using a Black-Box Pseudorandom Generator. In *Advances in Cryptology – CRYPTO 2005*, pages 378–394, 2005.

[13] I. Damgård and J. B. Nielsen. Improved Non-committing Encryption Schemes Based on a General Complexity Assumption. In *Advances in Cryptology – CRYPTO 2000*, pages 432–450, 2000.

[14] I. Damgård, A. Polychroniadou, and V. Rao. Adaptively Secure Multi-Party Computation from LWE (via Equivocal FHE). In *Proceedings of the 19th International Conference on the Theory and Practice of Public-Key Cryptography (PKC), part II*, pages 208–233, 2016.

[15] S. Garg and A. Polychroniadou. Two-Round Adaptively Secure MPC from Indistinguishability Obfuscation. In *Proceedings of the 12th Theory of Cryptography Conference, TCC 2015, part II*, pages 614–637, 2015.

[16] S. Garg and A. Sahai. Adaptively Secure Multi-Party Computation with Dishonest Majority. In *Advances in Cryptology – CRYPTO 2012*, pages 105–123, 2012.

[17] S. Garg, C. Gentry, S. Halevi, M. Raykova, A. Sahai, and B. Waters. Candidate indistinguishability obfuscation and functional encryption for all circuits. In *Proceedings of the 54th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 40–49, 2013.

[18] S. Garg, C. Gentry, S. Halevi, and M. Raykova. Two-Round Secure MPC from Indistinguishability Obfuscation. In *Proceedings of the 11th Theory of Cryptography Conference, TCC 2014*, pages 74–94, 2014.

[19] O. Goldreich. *Computational complexity - a conceptual perspective.* Cambridge University Press, 2008. ISBN 978-0-521-88473-0.

[20] O. Goldreich, S. Micali, and A. Wigderson. How to Play any Mental Game or A Completeness Theorem for Protocols with Honest Majority. In *Proceedings of the 19th Annual ACM Symposium on Theory of Computing (STOC)*, pages 218–229, 1987.

[21] C. Hazay and A. Patra. One-Sided Adaptively Secure Two-Party Computation. In *Proceedings of the 11th Theory of Cryptography Conference, TCC 2014*, pages 368–393, 2014.

[22] Y. Ishai, M. Prabhakaran, and A. Sahai. Founding Cryptography on Oblivious Transfer - Efficiently. In *Advances in Cryptology – CRYPTO 2008*, pages 572–591, 2008.

[23] J. Katz and R. Ostrovsky. Round-Optimal Secure Two-Party Computation. In *Advances in Cryptology – CRYPTO 2004*, pages 335–354, 2004.

[24] J. Katz, U. Maurer, B. Tackmann, and V. Zikas. Universally Composable Synchronous Computation. In *Proceedings of the 10th Theory of Cryptography Conference, TCC 2013*, pages 477–498, 2013.

[25] J. Katz, A. Thiruvengadam, and H.-S. Zhou. Feasibility and Infeasibility of Adaptively Secure Fully Homomorphic Encryption. In *Proceedings of the 16th International Conference on the Theory and Practice of Public-Key Cryptography (PKC)*, pages 14–31, 2013.

[26] Y. Lindell. *Composition of Secure Multi-Party Protocols, A Comprehensive Study*, volume 2815 of *Lecture Notes in Computer Science.* Springer, 2003.

[27] Y. Lindell. Adaptively Secure Two-Party Computation with Erasures. In *Proceedings of the Cryptographers' Track at the RSA Conference (CT-RSA)*, pages 117–132, 2009.

[28] J. B. Nielsen. Separating Random Oracle Proofs from Complexity Theoretic Proofs: The Non-committing Encryption Case. In *Advances in Cryptology – CRYPTO 2002*, pages 111–126, 2002.

[29] G. Poupard and J. Stern. Generation of Shared RSA Keys by Two Parties. In *Advances in Cryptology – ASIACRYPT '98*, pages 11–24, 1998.

[30] A. Yao. Protocols for Secure Computations (Extended Abstract). In *Proceedings of the 23rd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 160–164, 1982.

A The UC Framework

In this section we describe the universal-composability framework, for more details see [5].

A.1 The Real Model

An execution of a protocol π in the real model consists of n PPT *interactive Turing machines* (ITMs) $P_1, \ldots, P_n$ representing the parties, along with two additional ITMs, an *adversary* $\mathcal{A}$, describing the behavior of the corrupted parties and an *environment* $\mathcal{Z}$, representing the external network environment in which the protocol operates. The environment gives inputs to the honest parties, receives their outputs, and can communicate with the adversary at any point during the execution. The adversary controls the operations of the corrupted parties and the delivery of messages between the parties.

In more details, each ITM is initialized with the security parameter κ and random coins, where the environment may receive an additional auxiliary input. The protocol proceeds by a sequence of *activations*, where the environment is activated first and at each point a single ITM is active. When the environment is activated it can read the output tapes of all honest parties and of the adversary, and it can activate one of the parties or the adversary by writing on its input tape. Once a party is activated it can perform a local computation, write on its output tape or send messages to other parties by writing on its outgoing communication tapes. After the party completes its operations the control is returned to the environment. Once the adversary is activated it can send messages on behalf of the corrupted parties or send a message to the environment by writing on its output tape. In addition, $\mathcal{A}$ controls the communication between the parties, and so it can read the contents of the messages on outgoing tapes of honest parties and write messages on their incoming tapes. We assume that only messages that were sent in the past by some party can be delivered, and each message can be delivered at most once.[9] $\mathcal{A}$ can also corrupt an honest party, gain access to all its tapes and control all its actions. Whenever a party is corrupted the environment is notified. If $\mathcal{A}$ wrote on the incoming tape of an honest party, this party is activated next, otherwise the environment is activated. The protocol completes once $\mathcal{Z}$ stops activating other parties and outputs a single bit.

If the adversary is *semi-honest*, it always instructs the corrupted parties to follow the protocol. If the adversary is *malicious*, it may instruct the corrupted parties to deviate from the protocol arbitrarily.

[9]We assume that all the communication is authenticated yet visible to the adversary; formally, we work in the $\mathcal{F}_{\text{AUTH}}$-hybrid model.

Let $\textsc{real}_{\pi,\mathcal{A},\mathcal{Z}}(\kappa, z, \boldsymbol{r})$ denote $\mathcal{Z}$'s output on input z and security parameter κ, after interacting with adversary $\mathcal{A}$ and parties $P_1, \ldots, P_n$ running protocol π with random tapes $\boldsymbol{r} = (r_1, \ldots, r_n, r_{\mathcal{A}}, r_{\mathcal{Z}})$ as described above. Let $\textsc{real}_{\pi,\mathcal{A},\mathcal{Z}}(\kappa, z)$ denote the random variable $\textsc{real}_{\pi,\mathcal{A},\mathcal{Z}}(\kappa, z, \boldsymbol{r})$, when the vector $\boldsymbol{r}$ is uniformly chosen.

A.2 The Ideal Model

A computation in the ideal model consists of n *dummy* parties $P_1, \ldots, P_n$, an *ideal-process adversary* (simulator) $\mathcal{S}$, an *environment* $\mathcal{Z}$, and an *ideal functionality* $\mathcal{F}$. As in the real model, the environment gives inputs to the honest (dummy) parties, receives their outputs, and can communicate with the ideal-process adversary at any point during the execution. The dummy parties act as channels between the environment and the ideal functionality, meaning that they send the inputs received from $\mathcal{Z}$ to $\mathcal{F}$ and vice-versa. The ideal functionality $\mathcal{F}$ defines the desired behaviour of the computation. $\mathcal{F}$ receives the inputs from the dummy parties, executes the desired computation and sends the output to the parties. The ideal-process adversary does not see the communication between the parties and the ideal functionality, however, $\mathcal{S}$ can communicate with $\mathcal{F}$.

Hiding the communication between the ideal functionality and the parties from the adversary may be too restrictive; it is often desired to provide the adversary the power to determine *when* a party will receive the message. We say that the ideal functionality $\mathcal{F}$ sends a *delayed output* v to a party P if $\mathcal{F}$ first sends to the adversary a message that it is ready to generate an output to P. In case the output is public $\mathcal{F}$ sends v to the adversary. When the adversary replies to the message, $\mathcal{F}$ outputs the value v to P.[10]

Let $\textsc{ideal}_{\mathcal{F},\mathcal{S},\mathcal{Z}}(\kappa, z, \boldsymbol{r})$ denote $\mathcal{Z}$'s output on input z and security parameter κ, after interacting with ideal-process adversary $\mathcal{S}$ and dummy parties $P_1, \ldots, P_n$ that interact with ideal functionality $\mathcal{F}$ with random tapes $\boldsymbol{r} = (r_{\mathcal{S}}, r_{\mathcal{Z}})$ as described above. Let $\textsc{ideal}_{\mathcal{F},\mathcal{S},\mathcal{Z}}(\kappa, z)$ denote the random variable $\textsc{ideal}_{\mathcal{F},\mathcal{S},\mathcal{Z}}(\kappa, z, \boldsymbol{r})$, when the vector $\boldsymbol{r}$ is uniformly chosen.

Definition A.1. *We say that a protocol π UC-realizes an ideal functionality $\mathcal{F}$ in the presence of adaptive malicious (resp., semi-honest) adversaries, if for any* PPT *adaptive malicious (resp., semi-honest) adversary $\mathcal{A}$ and any* PPT *environment $\mathcal{Z}$, there exists a* PPT *ideal-process adversary $\mathcal{S}$ such that the following two distribution ensembles are computationally indistinguishable*

$$\left\{\textsc{real}_{\pi,\mathcal{A},\mathcal{Z}}\left(\kappa, z\right)\right\}_{\kappa\in\mathbb{N},z\in\{0,1\}^*} \overset{\mathrm{c}}{\equiv} \left\{\textsc{ideal}_{\mathcal{F},\mathcal{S},\mathcal{Z}}\left(\kappa, z\right)\right\}_{\kappa\in\mathbb{N},z\in\{0,1\}^*}.$$

A.3 The Hybrid Model

The *$\mathcal{F}$-hybrid model* is a combination of the real and ideal models, it extends the real model with an ideal functionality $\mathcal{F}$. The parties communicate with each other in exactly the same way as in the real model described above, however, they can interact with $\mathcal{F}$ as in the ideal model. An important property of the UC framework is that the ideal functionality $\mathcal{F}$ in a $\mathcal{F}$-hybrid model can be replaced with a protocol that UC-realizes $\mathcal{F}$.

[10]The ideal-process adversary may never release messages from the ideal functionality to the dummy parties and so termination of the computation is not guaranteed. In order to rule out trivial protocols that never produce output, we follow [8] and consider *non-trivial protocols* that have the following property: if the real-model adversary delivers all messages and does not corrupt any parties, then the ideal-process adversary also delivers all messages and does not corrupt any parties. We note that using techniques from [24] guaranteed termination can be enforced.

Let the global output $\text{HYBRID}^{\mathcal{F}}_{\pi,\mathcal{A},\mathcal{Z}}(\kappa, \boldsymbol{x}, z)$ denote $\mathcal{Z}$'s output on input z and security parameter κ, after interacting in a $\mathcal{F}$-hybrid model with adversary $\mathcal{A}$ and parties $P_1, \ldots, P_n$ with input $\boldsymbol{x}$ and uniformly distributed random tapes $\boldsymbol{r} = (r_1, \ldots, r_n, r_{\mathcal{A}}, r_{\mathcal{Z}})$ running protocol π.

Theorem A.2 (Canetti [5])**.** *Let $\mathcal{F}$ be an ideal functionality and let ρ be a protocol that UC-realizes $\mathcal{F}$ in the presence of adaptive semi-honest (resp., malicious) adversaries, and let π be a protocol that UC-realizes $\mathcal{G}$ in the $\mathcal{F}$-hybrid model in the presence of adaptive semi-honest (resp., malicious) adversaries. Then for any* PPT *adaptive semi-honest (resp., malicious) real-model adversary $\mathcal{A}$ and any* PPT *environment $\mathcal{Z}$, there exists a* PPT *adaptive semi-honest (resp., malicious) adversary $\mathcal{S}$ in the $\mathcal{F}$-hybrid model such that*

$$\left\{\text{REAL}_{\pi^{\rho},\mathcal{A},\mathcal{Z}}\left(\kappa, z\right)\right\}_{\kappa\in\mathbb{N},z\in\{0,1\}^*} \overset{\text{c}}{\equiv} \left\{\text{HYBRID}^{\mathcal{F}}_{\pi,\mathcal{S},\mathcal{Z}}\left(\kappa, z\right)\right\}_{\kappa\in\mathbb{N},z\in\{0,1\}^*}.$$

We emphasize an important point when analyzing protocols in the $\mathcal{F}$-hybrid model. In addition to the input, output, randomness and incoming messages, the internal state of a party contains its interface with the ideal functionality $\mathcal{F}$, i.e., the input the party sent to $\mathcal{F}$ and the output it received. Therefore, upon a corruption of a party *after* an ideal call to $\mathcal{F}$ has been made, the simulator must also provide to the adversary the party's interface with $\mathcal{F}$.

A.4 Some Ideal Functionalities

We next describe several ideal functionalities that are used throughout the paper.

A.4.1 Secure Message Transmission

The *secure message transmission (SMT)* functionality models a secure and private channel between two parties. The sender can send a message to the receiver such that the adversary learns only a specified leakage of the message, e.g., its length. If the sender is corrupted before the message was delivered to the receiver, the adversary is allowed to change the message. The secure message transmission functionality is described in Figure 6.

Functionality $\mathcal{F}^{l}_{\text{SMT}}$

$\mathcal{F}^{l}_{\text{SMT}}$ proceeds as follows, parametrized with leakage function l and running with a sender T, a receiver R and an adversary $\mathcal{S}$.

- Upon receiving a message $(\mathsf{send}, T, R, \mathsf{sid}, m)$ from T, send the message $(\mathsf{sent}, T, R, \mathsf{sid}, l(m))$ to $\mathcal{S}$, generate a private delayed output $(\mathsf{sent}, T, \mathsf{sid}, m)$ to R and halt.
- Upon receiving a message $(\mathsf{corrupt}, \mathsf{sid}, P)$ from $\mathcal{S}$, where $P \in \{T, R\}$, disclose m to $\mathcal{S}$. Next, if $\mathcal{S}$ provides a value m' and $P = T$, and no output has been written yet to R, then output $(\mathsf{sent}, T, \mathsf{sid}, m')$ to R and halt.

Figure 6: The secure message transmission functionality

A.4.2 Oblivious Transfer

A *1-out-of-ℓ oblivious transfer* is a two-party functionality involving a sender and a receiver. The sender has ℓ messages as its input $x_1, \ldots, x_\ell$, whereas the receiver has an index $i \in [\ell]$ as its input.

At the end of the computation, the receiver should obtain x_i and nothing else whereas the sender should not learn anything new. We describe the ideal functionality $\mathcal{F}_{\text{OT}}^{\ell}$ in Figure 7, for simplicity, for assume that every x_j is a bit.

Functionality $\mathcal{F}_{\text{OT}}^{\ell}$

$\mathcal{F}_{\text{OT}}^{\ell}$ proceeds as follows, parametrized with an integer ℓ and running with an oblivious transfer sender T, a receiver R and an adversary $\mathcal{S}$.

- Upon receiving a message $(\mathsf{sender}, \mathsf{sid}, x_1, \ldots, x_\ell)$ from T, where each $x_j \in \{0,1\}$, record the tuple $(x_1, \ldots, x_\ell)$.
- Upon receiving a message $(\mathsf{receiver}, \mathsf{sid}, i)$ from R, where $i \in [\ell]$, send (sid, x_i) to R and sid to $\mathcal{S}$, and halt. (If no $(\mathsf{sender}, \ldots)$ message was previously sent, then send nothing to R.)

Figure 7: The oblivious-transfer functionality

A.4.3 Secure Function Evaluation

Secure function evaluation (SFE) is a multiparty primitive where a set of n parties wish to compute a (possibly randomized) function $f\colon (\{0,1\}^*)^n \times \{0,1\}^* \to (\{0,1\}^*)^n$, where $f = (f_1, \ldots, f_n)$. That is, for a vector of inputs $\boldsymbol{x} = (x_1, \ldots, x_n) \in (\{0,1\}^*)^n$ and random coins $r \in_R \{0,1\}^*$, the output-vector is $(f_1(\boldsymbol{x}; r), \ldots, f_n(\boldsymbol{x}; r))$. The output for the i'th party (with input x_i) is defined to be $f_i(\boldsymbol{x}; r)$. The secure function evaluation functionality, $\mathcal{F}_{\text{SFE}}^{f}$, is presented in Figure 8.

Functionality $\mathcal{F}_{\text{SFE}}^{f}$

$\mathcal{F}_{\text{SFE}}^{f}$ proceeds as follows, running with parties $P_1, \ldots, P_n$ and an adversary $\mathcal{S}$, and parametrized by an n-party function $f\colon (\{0,1\}^*)^n \times \{0,1\}^* \to (\{0,1\}^*)^n$. For every P_i initialize an input value $x_i = \bot$ and an output value $y_i = \bot$.

- Upon receiving a message $(\mathsf{input}, \mathsf{sid}, v)$ from some party P_i, set $x_i = v$ and send a message $(\mathsf{input}, \mathsf{sid}, P_i)$ to $\mathcal{S}$.
- Upon receiving a message $(\mathsf{output}, \mathsf{sid})$ from some party P_i, do:
  1. If $x_j = \bot$ for some honest P_j, ignore the message.
  2. Otherwise, if $y_1, \ldots, y_n$ have not been set yet, then choose $r \in_R \{0,1\}^*$ and compute $(y_1, \ldots, y_n) = f(x_1, \ldots, x_n; r)$.
  3. Generate a delayed output $(\mathsf{output}, \mathsf{sid}, y_i)$ to P_i and send $(\mathsf{output}, \mathsf{sid}, P_i)$ to $\mathcal{S}$.

Figure 8: The secure function evaluation functionality