Two-round secure MPC from Indistinguishability Obfuscation

Sanjam Garg
IBM T. J. Watson
sanjamg@cs.ucla.edu

Craig Gentry
IBM T. J. Watson
craigbgentry@gmail.com

Shai Halevi
IBM T. J. Watson
shaih@alum.mit.edu

Mariana Raykova
IBM T. J. Watson
mariana@cs.columbia.edu

September 17, 2013

Abstract

One fundamental complexity measure of an MPC protocol is its *round complexity*. Asharov et al. recently constructed the first three-round protocol for general MPC in the CRS model. Here, we show how to achieve this result with only two rounds. We obtain UC security with abort against static malicious adversaries, and fairness if there is an honest majority. Additionally the communication in our protocol is only proportional to the input and output size of the function being evaluated and independent of its circuit size. Our main tool is indistinguishability obfuscation, for which a candidate construction was recently proposed by Garg et al.

The technical tools that we develop in this work also imply virtual black box obfuscation of a new primitive that we call a *dynamic point function*. This primitive may be of independent interest.

The second and third authors were supported by the Intelligence Advanced Research Projects Activity (IARPA) via Department of Interior National Business Center (DoI/NBC) contract number D11PC20202. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright annotation thereon. Disclaimer: The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of IARPA, DoI/NBC, or the U.S. Government.

1 Introduction

Secure multiparty computation (MPC) allows a group of mutually distrusting parties to jointly compute a function of their inputs without revealing their inputs to each other. This fundamental notion was introduced in the seminal works of [Yao82, GMW87], who showed that *any* function can be computed securely, even in the presence of malicious parties, provided the fraction of malicious parties is not too high. Since these fundamental feasibility results, much of the work related to MPC has been devoted to improving *efficiency.* There are various ways of measuring the efficiency of a MPC protocol, the most obvious being its computational complexity. In this paper, we focus on minimizing the *communication complexity* of MPC, primarily in terms of the number of *rounds* of interaction needed to complete the MPC protocol, but also in terms of the number of *bits* transmitted between the parties.

1.1 Our Main Result: Two-Round MPC from Indistinguishability Obfuscation

Our main result is a *compiler* that transforms *any* MPC protocol into a *2-round* protocol in the CRS model. Our compiler is conceptually very simple, and it uses as its main tool *indistinguishability obfuscation* ($i\mathcal{O}$) [BGI+12]. Roughly, in the first round the parties commit to their inputs and randomness, and in the second round each party provides an *obfuscation* of their "next-message" function in the underlying MPC protocol. The parties then separately evaluate the obfuscated next-message functions to obtain the output.

A bit more precisely, our main result is as follows:

Informal Theorem. *Assuming indistinguishability obfuscation, CCA-secure public-key encryption, and statistically-sound noninteractive zero-knowledge, any multiparty function can be computed securely in just two rounds of broadcast.*

We prove that our MPC protocol resists static malicious corruptions in the UC setting [Can01]. Moreover, the same protocol also achieves fairness if the set of corrupted players is a strict minority. Finally the communication in our protocol can be made to be only proportional to the input and output size of the function being evaluated and independent of its circuit size.

Minimizing round complexity is not just of theoretical interest. Low-interaction secure computation protocols are also applicable in the setting of computing on the web [HLP11], where a single server coordinates the computation, and parties "log in" at different times without coordination.

1.2 Indistinguishability Obfuscation

Obfuscation was first rigorously defined and studied by Barak et al. [BGI+12]. Most famously, they defined a notion of *virtual black box (VBB)* obfuscation, and proved that this notion is impossible to realize in general – i.e., some functions are VBB unobfuscatable.

Barak et al. also defined a weaker notion of *indistinguishability obfuscation ($i\mathcal{O}$)*, which avoids their impossibility results. $i\mathcal{O}$ provides the same functionality guarantees as VBB obfuscation, but a weaker security guarantee. Namely, that for any two circuits C_0, C_1 of similar size that *compute the same function*, it is hard to distinguish an obfuscation of C_0 from an obfuscation of C_1. Barak et al. showed that $i\mathcal{O}$ is *always* realizable, albeit inefficiently: the $i\mathcal{O}$ can simply *canonicalize* the input circuit C by outputting the lexicographically first circuit that computes the same function. More recently, Garg et al. [GGH+13b] proposed an efficient construction of $i\mathcal{O}$ for all circuits, basing security in part on assumptions related to multilinear maps [GGH13a].

It is clear that $i\mathcal{O}$ is a weaker primitive than VBB obfuscation. In fact, it is not hard to see that we cannot even hope to prove that $i\mathcal{O}$ implies one-way functions: Indeed, if $P = NP$ then one-way functions do not exist but $i\mathcal{O}$ does exist (since the canonicalizing $i\mathcal{O}$ from above can be implemented efficiently). Therefore we do not expect to build many "cryptographically interesting" tools just from $i\mathcal{O}$, but usually need to combine it with other assumptions. (One exception is witness encryption [GGSW13], which can be constructed from $i\mathcal{O}$ alone.)

It is known that $i\mathcal{O}$ can be combined with one-way functions (OWFs) to construct many powerful primitives such as public-key encryption, identity-based encryption, attribute-based encryption (via witness encryption), as well as NIZKs, CCA encryption, and deniable encryption [SW13]. However, there are still basic tools that are trivially constructible from VBB obfuscation that we do not know how to construct from $i\mathcal{O}$ and OWFs: for example, collision-resistant hash functions, or compact homomorphic encryption. (Compact homomorphic encryption implies collision-resistant hash functions [IKO05].) The main challenge in constructing primitives from $i\mathcal{O}$ is that the indistinguishability guarantee holds only in a limited setting: when the two circuits in question are perfectly functionally equivalent.

1.3 Our Techniques

To gain intuition and avoid technical complications, let us begin by considering how we would construct a 2-round protocol if we could use "perfect" VBB obfuscation. For starters, even with VBB obfuscation we still need at least two rounds of interaction, since a 1-round protocol would inherently allow the corrupted parties to repeatedly evaluate the "residual function" associated to the inputs of the honest parties on many different inputs of their choice (e.g., see [HLP11]).

It thus seems natural to split our 2-round protocol into a commitment round in which all players "fix their inputs," and then an evaluation round where the output is computed. Moreover, it seems natural to use CCA-secure encryption to commit to the inputs and randomness, as this would enable a simulator to extract these values from the corrupted players.

As mentioned above, our idea for the second round is a simple compiler: take any (possibly highly interactive) underlying MPC protocol, and have each party obfuscate their "next-message" function in that protocol, one obfuscation for each round, so that the parties can independently evaluate the obfuscations to obtain the output. Party i's next-message function for round j in the underlying MPC protocol depends on its input x_i and randomness r_i (which are hardcoded in the obfuscations), it takes as input the transcript through round $j - 1$, and it produces as output the next broadcast message.

However, there is a complication: unlike the initial interactive protocol, the obfuscations are susceptible to a "reset" attack – i.e., they can be evaluated on multiple inputs. To prevent such attacks, we ensure that the obfuscations can be used for evaluation only on a unique set of values – namely, values consistent with the inputs and randomness that the parties committed to in the first round, and the current transcript of the underlying MPC protocol. To ensure such consistency, naturally we use non-interactive zero-knowledge (NIZK) proofs. Since the NIZKs apply not only to the committed values of the first round, but also to the transcript as it develops in the second round, the obfuscations themselves must output these NIZKs "on the fly". In other words, the obfuscations are now augmented to perform not only the next-message function, but also to prove that their output is consistent. Also, obfuscations in round j of the underlying MPC protocol verify NIZKs associated to obfuscations in previous rounds before providing any output.

If we used VBB obfuscation, we could argue security intuitively as follows. Imagine an augmented version of the underlying MPC protocol, where we prepend a round of commitment to the inputs and randomness, after which the parties (interactively) follow the underlying MPC protocol, except that they provide NIZK proofs that their messages are consistent with their committed inputs and randomness and the developing transcript. It is fairly easy to see that the security of this augmented protocol (with some minor modifications to how the randomness is handled) reduces to the security of the underlying MPC protocol (and the security of the CCA encryption and NIZK proof system). Now, remove the interaction by providing VBB obfuscations of the parties in the second round. These VBB obfuscations "virtually emulate" the parties of the augmented protocol while providing no additional information – in particular, the obfuscations output $\perp$ unless the input conforms exactly to the transcript of the underlying MPC protocol on the committed inputs and randomness; the obfuscations might accept many valid proofs, but since the proofs are statistically sound this gives no more information than one obtains in the augmented protocol.

Instead, we use indistinguishability obfuscation, and while the our protocol is essentially as described above, the proof of security is more subtle. Here, we again make use of the fact that the transcript in the underlying MPC protocol is completely determined by the commitment round, but in a different way. Specifically, there is a step in the proof where we change the obfuscations, so that instead of actually computing the next-message function (with proofs), these values are extracted and simply hardcoded in the obfuscations as the output on any accepting input. We show that these two types of obfuscations are functionally equivalent, and invoke $i\mathcal{O}$ to prove that they are indistinguishable. Once these messages have been "hardcoded" and separated from the computation, we complete the security proof using standard tricks. The most interesting remaining step in the proof is where we replace hardcoded real values with hardcoded simulated values generated by the simulator of the underlying MPC protocol.

1.4 Additional Results

Two-Round MPC with Low Communication. In our basic 2-round MPC protocol, the communication complexity grows polynomially with the circuit size of the function being computed. In Section 3.2, we show how to combine our basic 2-round protocol with *multikey fully homomorphic encryption* [LATV12] to obtain an MPC that is still only two rounds, but whose communication is basically independent of the circuit size. Roughly speaking, this protocol has a first round where the players encrypt their inputs and evaluate the function under a shared FHE key (and commit to certain values as in our basic protocol), followed by a second round where the players apply the second round of our basic protocol to decrypt the final FHE ciphertext.

Dynamic Point Functions. As a side effect of our technical treatment, we observe that $i\mathcal{O}$ can be used to extend the reach of (some) known VBB obfuscators. For example, we can VBB obfuscate *dynamic point functions*. In this setting, the obfuscation process is partitioned between two parties, the "point owner" Penny and the "function owner" Frank. Penny has a secret string (point) $x \in \{0,1\}^*$, and she publishes a commitment to her point $c_x = \mathrm{com}(x)$. Frank has a function $f : \{0,1\}^* \to \{0,1\}^*$ and knows c_x but not x itself. Frank wants to allow anyone who happens to know x to compute $f(x)$. A dynamic point function obfuscator allows Frank to publish an obfuscated version of the point function

$$F_{f,x}(z) = \begin{cases} f(x) & \text{if } z = x \\ \perp & \text{otherwise.} \end{cases}$$

The security requirement here is that $F_{f,x}$ is obfuscated in the strong VBB sense (and that c_x hides x computationally). We believe that this notion of dynamic point functions is interesting on its own and that it may find future applications.

1.5 Other Related Work

The round complexity of MPC has been studied extensively: both lower and upper bounds, for both the two-party and multiparty cases, in both the semi-honest and malicious settings, in plain, CRS and PKI models. See [AJLA$^+$12, Section 1.3] for a thorough overview of this work.

Here, we specifically highlight the recent work of Asharov et al. [AJLA$^+$12], which achieves 3-round MPC in the CRS model (and 2-round MPC in the PKI model) against static malicious adversaries. They use fully homomorphic encryption (FHE) [RAD78, Gen09], but not as a black box. Rather, they construct threshold versions of *particular* FHE schemes – namely, schemes by Brakerski, Gentry and Vaikuntanathan [BV11, BGV12] based on the learning with errors (LWE) assumption. (We note that Myers, Sergi and shelat [MSS11] previously thresholdized a different FHE scheme based on the approximate gcd assumption [vDGHV10], but their protocol required more rounds.)

In more detail, Asharov et al. observe that these particular LWE-based FHE schemes have a key homomorphic property. Thus, in the first round of their protocol, each party can encrypt its message under its own FHE key, and then the parties can use the key homomorphism to obtain encryptions of the inputs under a shared FHE key. Also, in the last round of their protocol, decryption is a simple one-round process, where decryption of the final ciphertext under the individual keys reveals the decryption under the shared key. In between, the parties use FHE evaluation to compute the encrypted output under the shared key. Unfortunately, they need a third (middle) round for technical reasons: LWE-based FHE schemes typically also have an "evaluation key" – namely, an encryption of a function of the secret key under the public key. They need the extra round to obtain an evaluation key associated to their shared key.

Recently, Gentry, Sahai and Waters [GSW13] proposed an LWE-based FHE scheme without such an evaluation key. Unfortunately, eliminating the evaluation key in their scheme does not seem to give 2-round MPC based on threshold FHE, since their scheme lacks the key homomorphism property needed by Asharov et al.

We note that our basic two-round protocol does not rely on any *particular* constructions for $i\mathcal{O}$ (or CCA-secure PKE or NIZK proofs), but rather uses these components as black boxes.

Our low-communication two-round protocol uses multikey FHE, but only as a black box. This protocol can be seen as a realization of what Asharov et al. were trying to achieve: a first round where the players encrypt their inputs and evaluate the function under a shared FHE key, followed by a second round where the players decrypt the final FHE ciphertext.

2 Preliminaries

In this section we will start by briefly recalling the definition of different notions essential for our study. We refer the reader to Appendix A for additional background. The natural security parameter is λ, and all other quantities are implicitly assumed to be functions of λ. We use standard big-O notation to classify the growth of functions. We let $\mathsf{poly}(\lambda)$ denote an unspecified function $f(\lambda) = O(\lambda^c)$ for some constant c. A *negligible* function, denoted generically by $\mathsf{negl}(\lambda)$, is an $f(\lambda)$ such that $f(\lambda) = o(\lambda^{-c})$ for every fixed constant c. We say that a function is *overwhelming* if it is $1 - \mathsf{negl}(\lambda)$.

2.1 Indistinguishability Obfuscators

We will start by recalling the notion of indistinguishability obfuscation ($i\mathcal{O}$) recently realized in [GGH$^+$13b] using candidate multilinear maps[GGH13a].

Definition 1 (Indistinguishability Obfuscator ($i\mathcal{O}$))**.** A uniform PPT machine $i\mathcal{O}$ is called an *indistinguishability obfuscator* for a circuit class $\{\mathcal{C}_\lambda\}$ if the following conditions are satisfied:

- For all security parameters $\lambda \in \mathbb{N}$, for all $C \in \mathcal{C}_\lambda$, for all inputs x, we have that

$$\Pr[C'(x) = C(x) : C' \leftarrow i\mathcal{O}(\lambda, C)] = 1$$

- For any (not necessarily uniform) PPT distinguisher D, there exists a negligible function α such that the following holds: For all security parameters $\lambda \in \mathbb{N}$, for all pairs of circuits $C_0, C_1 \in \mathcal{C}_\lambda$, we have that if $C_0(x) = C_1(x)$ for all inputs x, then

$$\Big| \Pr\big[D(i\mathcal{O}(\lambda, C_0)) = 1\big] - \Pr\big[D(i\mathcal{O}(\lambda, C_1)) = 1\big] \Big| \leq \alpha(\lambda)$$

Definition 2 (Indistinguishability Obfuscator for NC^1)**.** A uniform PPT machine $i\mathcal{O}$ is called an *indistinguishability obfuscator* for NC^1 if for all constants $c \in \mathbb{N}$, the following holds: Let $\mathcal{C}_\lambda$ be the class of circuits of depth at most $c \log \lambda$ and size at most λ. Then $i\mathcal{O}(c, \cdot, \cdot)$ is an indistinguishability obfuscator for the class $\{\mathcal{C}_\lambda\}$.

Definition 3 (Indistinguishability Obfuscator for $P/poly$)**.** A uniform PPT machine $i\mathcal{O}$ is called an *indistinguishability obfuscator* for $P/poly$ if the following holds: Let $\mathcal{C}_\lambda$ be the class of circuits of size at most λ. Then $i\mathcal{O}$ is an indistinguishability obfuscator for the class $\{\mathcal{C}_\lambda\}$.

2.2 Semi-Honest MPC

We will also use a semi-honest n-party computation protocol π for any functionality f in the stand-alone setting. The existence of such a protocol follows from the existence of semi-honest 1-out-of-2 oblivious transfer [Yao82, GMW87] protocols. Now we build some notation that we will use in our construction.

Let $\mathcal{P} = \{P_1, P_2, \ldots P_n\}$ be the set of parties participating in a t round protocol π. Without loss of generality, in order to simplify notation, we will assume that in each round of π, each party broadcasts a single message that depends on its input and randomness and on the messages that it received from all parties in all previous rounds. (We note that we can assume this form without loss of generality, since in our setting where we have broadcast channels and CCA-secure encryption, and we only consider security against static corruptions.) We let $m_{i,j}$ denote the message sent by the i^{th} party in the j^{th} round. We define the function π_i such that $m_{i,j} = \pi_i(x_i, r_i, M_{j-1})$ where

$m_{i,j}$ is the j^{th} message generated by party P_i in protocol π with input x_i, randomness r_i and the series of previous messages M_{j-1}

$$M_{j-1} = \begin{pmatrix} m_{1,1} & m_{2,1} & \dots & m_{n,1} \\ m_{1,2} & m_{2,2} & \dots & m_{n,2} \\ \vdots & \ddots & & \\ m_{1,j-1} & m_{2,j-1} & \dots & m_{n,j-1} \end{pmatrix}$$

sent by all parties in π.

3 Our Protocol

In this section, we provide our construction of a two-round MPC protocol.

Protocol Π. We start by giving an intuitive description of the protocol. A formal description appears in Figure 1. The basic idea of our protocol is to start with an arbitrary round semi-honest protocol π and "squish" it into a two round protocol using indistinguishability obfuscation. The first round of our protocol helps set the stage for the "virtual" execution of π via obfuscations that all the parties provide in the second round.

The common reference string in our construction consists of a CRS σ for a NIZK Proof system and a public key pk corresponding to a CCA-secure public key encryption scheme. Next, the protocol proceeds in two rounds as follows:

Round 1: In the first round, the parties "commit" to their inputs and randomness, where the commitments are generated using the CCA-secure encryption scheme. The committed randomness will be used for coin-flipping and thereby obtaining unbiased random coins for all parties. Specifically, every party P_i, proceeds by encrypting its input x_i under the public key pk. Let c_i be the ciphertext. P_i also encrypts randomness $r_{i,j}$ for every $j \in [n]$. Let the ciphertext encrypting $r_{i,j}$ be denoted by $d_{i,j}$. Looking ahead the random coins P_i uses in the execution of π will be $s_i = \oplus_j r_{j,i}$. P_i broadcasts $\{c_i, \{d_{i,j}\}_j\}$ to everyone.

Round 2: In the second round parties will broadcast obfuscations corresponding to the next message function of π allowing for a "virtual emulation" of the interactive protocol π. Every party P_i proceeds as follows:

- P_i reveals the random values $\{r_{i,j}\}_{j \neq i \in [n]}$ and generates proofs $\{\gamma_{i,j}\}_{j \neq i \in [n]}$ that these are indeed the values that are encrypted in the ciphertexts $\{d_{i,j}\}_{j \neq i \in [n]}$.
- Recall that the underlying protocol π is a t round protocol where each party broadcasts one message per round. Each player P_i generates t obfuscations of its next-round function, $(i\mathcal{O}_{i,1}, \dots, i\mathcal{O}_{i,t})$.

  In more detail, each $i\mathcal{O}_{i,k}$ is an obfuscation of a function $F_{i,k}$ that takes as input the $r_{i,j}$ values sent by all the parties along with the proofs that they are well-formed, and also all the π-messages that were broadcast upto round $k-1$, along with the proof of correct generation of these messages. (These proofs are all with respect to the ciphertexts

Protocol Π

Protocol Π uses an Indistinguishability Obfuscator $i\mathcal{O}$, a NIZK proof system (K, P, V), a CCA-secure PKE scheme $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ with perfect correctness and an n-party semi-honest MPC protocol π.
Private Inputs: Party P_i for $i \in [n]$, receives its input x_i.
Common Reference String: Let $\sigma \leftarrow K(1^\lambda)$ and $(pk, \cdot) \leftarrow \mathsf{Gen}(1^\lambda)$ and then output (σ, pk) as the common reference string.

Round 1: Each party P_i proceeds as:

- $c_i = \mathsf{Enc}(i||x_i)$ and,
- $\forall j \in [n]$, sample randomness $r_{i,j} \in \{0,1\}^\ell$ and generate $d_{i,j} = \mathsf{Enc}(i||r_{i,j})$. (Here ℓ is the length of the maximum number of random coins needed by any party in π.)

It then sends $Z_i = \{c_i, \{d_{i,j}\}_{j\in[n]}\}$ to every other party.

Round 2: P_i generates:

- For every $j \in [n]$, $j \neq i$ generate $\gamma_{i,j}$ as the NIZK proof under σ for the NP-statement:

$$\left\{\exists\ \rho_{r_{i,j}}\ \middle|\ d_{i,j} = \mathsf{Enc}(i||r_{i,j}; \rho_{r_{i,j}})\right\}. \tag{1}$$

- A sequence of obfuscations $(i\mathcal{O}_{i,1}, \ldots i\mathcal{O}_{i,t})$ where $i\mathcal{O}_{i,j}$ is the obfuscation of the program $\mathsf{Prog}_{i,j}^{0,x_i,\rho_{x_i},r_{i,i},\rho_{r_{i,i}},\{Z_i\},0^{\ell_{i,j}}}$. (Where $\ell_{i,j}$ is output length of the program $\mathsf{Prog}_{i,j}$.)
- It sends $(\{r_{i,j}, \gamma_{i,j}\}_{j\in[n],j\neq i}, \{i\mathcal{O}_{i,j}\}_{j\in[t]})$ to every other party.

Evaluation (MPC in the Head): For each $j \in [t]$ proceed as follows:

- For each $i \in [n]$, evaluate the obfuscation $i\mathcal{O}_{i,j}$ of program $\mathsf{Prog}_{i,j}$ on input $(R, \Gamma, M_{j-1}, \Phi_{j-1})$ where

$$R = \begin{pmatrix} \cdot & r_{2,1} & \cdots & r_{n,1} \\ r_{1,2} & \cdot & \cdots & r_{n,2} \\ \vdots & \ddots & & \\ r_{1,n} & r_{2,n} & \cdots & \cdot \end{pmatrix}, \quad \Gamma = \begin{pmatrix} \cdot & \gamma_{2,1} & \cdots & \gamma_{n,1} \\ \gamma_{1,2} & \cdot & \cdots & \gamma_{n,2} \\ \vdots & \ddots & & \\ \gamma_{1,n} & \gamma_{2,n} & \cdots & \cdot \end{pmatrix}$$

$$M_{j-1} = \begin{pmatrix} m_{1,1} & m_{2,1} & \cdots & m_{n,1} \\ m_{1,2} & m_{2,2} & \cdots & m_{n,2} \\ \vdots & \ddots & & \\ m_{1,j-1} & m_{2,j-1} & \cdots & m_{n,j-1} \end{pmatrix}, \quad \Phi = \begin{pmatrix} \phi_{1,1} & \phi_{2,1} & \cdots & \phi_{n,1} \\ \phi_{1,2} & \phi_{2,2} & \cdots & \phi_{n,2} \\ \vdots & \ddots & & \\ \phi_{1,j-1} & \phi_{2,j-1} & \cdots & \phi_{n,j-1} \end{pmatrix}$$

- And obtain, $m_{1,j}, \ldots, m_{n,j}$ and $\phi_{1,j}, \ldots, \phi_{n,j}$.

Finally each party P_i outputs $m_{i,t}$.

Figure 1: Two Round MPC Protocol

$$\mathsf{Prog}_{i,j}^{\mathsf{flag},x_i,\rho_{x_i},r_{i,i},\rho_{r_{i,i}},\{Z_i\},\mathsf{fixedOutput}}$$

Program $\mathsf{Prog}_{i,j}^{\mathsf{flag},x_i,\rho_{x_i},r_{i,i},\rho_{r_{i,i}},\{Z_i\},\mathsf{fixedOutput}}$ takes as input $(R, \Gamma, M_{j-1}, \Phi)$ as defined above and outputs $m_{i,j}$ and $\phi_{i,j}$. Specifically, it proceeds as follows:

- $\forall p, q \in [n]$ such that $p \neq q$ check that $\gamma_{p,q}$ is an accepting proof under σ for the NP-statement:

$$\left\{\exists\ \rho_{r_{p,q}} \;\middle|\; d_{p,q} = \mathsf{Enc}(p||r_{p,q}; \rho_{r_{p,q}})\right\}.$$

- $\forall p \in [n], q \in [j-1]$ check that $\phi_{p,q}$ is an accepting proof for the NP-statement

$$\left\{\begin{array}{l} \exists\ (x_p, r_{p,p}, \rho_{x_p}, \rho_{r_{p,p}}) \mid \\ \left(c_p = \mathsf{Enc}(p||x_p; \rho_{x_p})\ \bigwedge\ d_{p,p} = \mathsf{Enc}(p||r_{p,p}, \rho_{r_{p,p}})\ \bigwedge\ m_{p,q} = \pi_p(x_p, \oplus_{k\in[n]} r_{k,p}, M_{q-1})\right). \end{array}\right\}$$

- If the checks above fail, output $\perp$. Otherwise, if $\mathsf{flag} = 0$ then output $(\pi_i(x_i, \oplus_{j\in[n]} r_{j,i}, M_{j-1}), \phi_{i,j})$ where $\phi_{i,j}$ is the proof for the NP-statement: (under some fixed randomness)

$$\left\{\begin{array}{l} \exists\ (x_i, r_{i,i}, \rho_{x_i}, \rho_{r_{i,i}}) \mid \\ \left(c_i = \mathsf{Enc}(i||x_i; \rho_{x_i})\ \bigwedge\ d_{i,i} = \mathsf{Enc}(i||r_{i,i}, \rho_{r_{i,i}})\ \bigwedge\ m_{i,j} = \pi_i(x_i, \oplus_{j\in[n]} r_{j,i}, M_{j-1})\right). \end{array}\right\}$$

  Otherwise, output $\mathsf{fixedOutput}$.

Figure 2: Obfuscated Programs in the Protocol

generated in first round and the revealed $r_{i,j}$ values.) The output of the function $F_{i,j}$ is the next message of P_i in π, along with a NIZK proof that it was generated correctly.

P_i broadcasts all the values $\{r_{i,j}\}_{j\neq i\in[n]}$, $\{\gamma_{i,j}\}_{j\neq i\in[n]}$, and $\{i\mathcal{O}_{i,k}\}_{k\in[t]}$.

Evaluation: After completion of the second round each party can independently "virtually" evaluate the protocol π using the obfuscations provided by each of the parties and obtain the output.

Theorem 1. Let f be any deterministic poly-time function with n inputs and single output. Assume the existence of an Indistinguishability Obfuscator $i\mathcal{O}$, a NIZK proof system (K, P, V), a CCA secure PKE scheme $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ with perfect correctness and an n-party semi-honest MPC protocol π. Then the protocol Π presented in Figure 1 UC-securely realizes the ideal functionality $\mathcal{F}_f$ in the $\mathcal{F}_{CRS}$-hybrid model.

3.1 Correctness and Proof of Security

Correctness. The correctness of our protocol Π in Figure 1 follows from the correctness of the underlying semi-honest MPC protocol and the other primitives used. Next we will argue that all the messages sent in the protocol Π are of polynomial length and can be computed in polynomial time. It is easy to see that all the messages of round 1 are polynomially long. Again it is easy to see that the round 2 messages besides the obfuscations themselves are of polynomial length.

We will now argue that each obfuscation sent in round 2 is also polynomially long. Consider the obfuscation $i\mathcal{O}_{i,j}$, which obfuscates $\mathsf{Prog}_{i,j}$; we need to argue that this program for every i, j is only polynomially long. Observe that this program takes as input $(R, \Gamma, M_{i-1}, \Phi_{j-1})$, where Γ and

Φ_{j-1} consist of polynomially many NIZK proofs. This program roughly proceeds by first checking that all the proofs in Γ and Φ_{j-1} are accepting. If the proofs are accepting then Prog outputs $m_{i,j}$ and $\phi_{i,j}$.

Observe that Γ and Φ_{j-1} are proofs of NP-statements each of which is a fixed polynomial in the description of the next message function of the protocol π. Also observe that the time taken to evaluate $m_{i,j}$ and $\phi_{i,j}$ is bounded a fixed polynomial. This allows us to conclude that all the computation done by $\mathsf{Prog}_{i,j}$ can be bounded by a fixed polynomial.

Security. Let $\mathcal{A}$ be a malicious, static adversary that interacts with parties running the protocol Π from Figure 1 in the $\mathcal{F}_{CRS}$-hybrid model. We construct an ideal world adversary $\mathcal{S}$ with access to the ideal functionality $\mathcal{F}_f$, which simulates a real execution of Π with $\mathcal{A}$ such that no environment $\mathcal{Z}$ can distinguish the ideal world experiment with $\mathcal{S}$ and $\mathcal{F}_f$ from a real execution of Π with $\mathcal{A}$.

We now sketch the description of the simulator and the proof of security, restricting ourselves to the stand-alone setting. The fully detailed description of our simulator is provided in Appendix B and the proof of indistinguishability provided in Appendix C. Those more formal proofs are given for the general setting of UC-security.

Our simulator $\mathcal{S}$ roughly proceeds as follows:

- **Common reference string:** Recall that the common reference string in our construction consists of a CRS σ for a NIZK Proof system and a public key pk corresponding to a CCA secure public key encryption scheme. Our simulator uses the simulator of the NIZK proof system in order to generate the reference string σ. Note that the simulator for NIZK proof system also generates some trapdoor information that can be used to generate simulated NIZK proofs. Our simulator saves that for later use. $\mathcal{S}$ also generates the public key pk along with its secret key sk, which it will later use to decrypt ciphertexts generated by the adversary.

- **Round 1:** Recall that in round 1, honest parties generate ciphertexts corresponding to encryptions of their inputs and various random coins. Our simulator just generates encryptions of the zero-string on behalf of the honest parties. Also $\mathcal{S}$ uses the knowledge of the secret key sk to extract the input and randomness that the adversarial parties encrypt.

- **Round 2:** Recall that in the second round the honest parties are required to "open" some of the randomness values committed to in round 1 along with obfuscations necessary for execution of π.

  $\mathcal{S}$ proceeds by preparing a simulated transcript of the execution of π using the malicious party inputs previously extracted and the output obtained from the ideal functionality, which it needs to force onto the malicious parties. $\mathcal{S}$ opens the randomness on behalf of honest parties such that the randomness of malicious parties becomes consistent with the simulated transcript and generates simulated proofs for the same. The simulator generates the obfuscations on behalf of honest parties by hard-coding the messages as contained in the simulated transcript. The obfuscations also generate proofs proving that the output was generated correctly. Our simulator hard-codes these proofs in the obfuscations as well.

Very roughly, our proof proceeds by first changing all the obfuscations $\mathcal{S}$ generates on behalf of honest parties to output fixed values. The statistical soundness of the NIZK proof system allows us to base security on the weak notion of indistinguishability obfuscation. Once this change has been made, in a sequence of hybrids we change from honest execution of the underlying semi-honest MPC protocol to a the simulated execution. We refer the reader to Appendix C for a complete proof.

3.2 Extensions

Low Communication. Our protocol Π (as described in Figure 1) can be used to UC-securely realize any functionality $\mathcal{F}_f$. However the communication complexity of this protocol grows polynomially in the size of the circuit evaluating function f and the security parameter λ. We would like to remove this restriction and construct a protocol Π' whose communication complexity is independent of the the function being evaluated.

Protocol Π'

Let Π be the MPC Protocol from Figure 1.
Let $(\mathsf{Setup}_{MK}, \mathsf{Encrypt}_{MK}, \mathsf{Eval}_{MK}, \mathsf{Decrypt}_{MK})$ be a multikey FHE scheme.
Private Inputs: Party P_i for $i \in [n]$, receives its input x_i.
Common Reference String: Generate the CRS corresponding to Π.

Round 1: P_i proceeds as follows:

- $(pk_i, sk_i) \leftarrow \mathsf{Setup}_{MK}(1^\lambda; \rho_i)$ and generates encryption $c_i := \mathsf{Encrypt}_{MK}(pk_i, x_i; \varrho_i)$.
- Generates the first round message Z_i of Π playing as P_i with input (x_i, ρ_i, ϱ_i). (Recall that the first message of Π does not depend on the function Π is used to evaluate.)
- Sends3 (pk_i, c_i, Z_i) to all parties.

Round 2: Every party P_i computes $c^* := \mathsf{Eval}_{MK}(C, (c_1, pk_1), \ldots, (c_n, pk_n))$. P_i generates P_i's second round message of Π, where Π computes the following function:

- For every $i \in [n]$, check if $(pk_i, sk_i) \leftarrow \mathsf{Setup}_{MK}(1^\lambda; \rho_i)$ and $c_i := \mathsf{Encrypt}_{MK}(pk_i, x_i; \varrho_i)$.
- If all the checks pass then output $\mathsf{Decrypt}_{MK}(sk_1, \ldots, sk_n, c^*)$ and otherwise output $\perp$.

Evaluation: P_i outputs the output of P_i in Π.

Figure 3: Two Round MPC Protocol with Low Communication Complexity

A key ingredient of our construction is *multikey fully homomorphic encryption* [LATV12]. Intuitively, multikey FHE allows us to evaluate any circuit on ciphertexts that might be encrypted under different public keys. To guarantee semantic security, decryption requires all of the corresponding secret keys. We refer the reader to Appendix A.4 for more details.

Our protocol Π' works by invoking Π. Recall that Π proceeds in two rounds. Roughly speaking, in the first stage parties commit to their inputs, and in the second round the parties generate obfuscations that allow for "virtual" execution of sub-protocol π on the inputs committed in the first round. Our key observation here is that the function that the sub-protocol π evaluates does not have to be specified until the second round.

We will now give a sketch of our protocol Π'. Every party P_i generates a public key pk_i and a secret key sk_i using the setup algorithm of the multikey FHE scheme. It then encrypts its input x_i under the public key pk_i and obtains ciphertext c_i. It then sends (pk_i, c_i) to everyone along with the first message of Π with input the randomness used in generation of pk_i and c_i. This completes the first round. At this point, all parties can use the values $((pk_1, c_1), \ldots, (pk_n, c_n))$ to obtain an encryption of $f(x_1, \ldots x_n)$, where f is the function that we want to compute. The second round of protocol Π can be used to decrypt this value. A formal description of the protocol appears in Figure 3.

Theorem 2. Under the same assumptions as in Theorem 1 and assuming the semantic security of the multikey FHE scheme, the protocol Π' presented in Figure 3 UC-securely realizes the ideal functionality $\mathcal{F}_f$ in the $\mathcal{F}_{CRS}$-hybrid model. Furthermore the communication complexity of protocol Π' is polynomial in the input lengths of all parties and the security parameter. (It is independent of the size of f.)

Proof. The correctness of the our protocol Π' follows from the correctness of the protocol Π and the correctness of the multikey FHE scheme. Observe that the compactness of the multikey FHE (Appendix A.4) implies that the ciphertext c^* evaluated in Round 2 on the description of Protocol Π (Figure 3) is independent of the size of the function f being evaluated. Also note that no other messages in the protocol depend on the function f. This allows us to conclude that the communication complexity of protocol Π' is independent of the size of f.

We defer the formal description of our simulator and the proof of indistinguishability to Appendix D. ∎

General Functionality. Our basic MPC protocol as described in Figure 1 only considers deterministic functionalities (Appendix A.1.5) where all the parties receive the same output. We would like to generalize it to handle randomized functionalities and individual outputs (just as in [AJW11, Appendix D]). First, the standard transformation from a randomized functionality to a deterministic one (See [Gol04, Section 7.3]) works for this case as well. In this transformation, instead of computing some randomized function $g(x_1, \ldots x_n; r)$, the parties compute the deterministic function $f((r_1, x_1), \ldots, (r_n, x_n)) \overset{def}{=} g(x_1, \ldots, x_n; \oplus_{i=1}^{n} r_i)$. We note that this computation does not add any additional rounds.

Next, we move to individual outputs. Again, we use a standard transformation (See [LP09], for example). Given a function $g(x_1, \ldots, x_n) \to (y_1, \ldots, y_n)$, the parties can evaluate the following function which has a single output:

$$f((k_1, x_1), \ldots, (k_n; x_n)) = (g_1(x_1, \ldots, x_n) \oplus k_1 || \ldots || g_n(x_1, \ldots, x_n) \oplus k_n)$$

where $a||b$ denotes a concatenation of a with b, g_i indicates the i^{th} output of g, and k_i is randomly chosen by the i^{th} party. Then, the parties can evaluate f, which is a single output functionality, instead of g. Subsequently every party P_i uses its secret input k_i to recover its own output. The only difference is that f has one additional exclusive-or gate for every circuit-output wire. Again, this transformation does not add any additional rounds of interaction.

Corollary 1. Let f be any (possibly randomized) poly-time function with n inputs and n outputs. Assume the existence of an Indistinguishability Obfuscator $i\mathcal{O}$, a NIZK proof system (K, P, V), a CCA secure PKE scheme $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ with perfect correctness and an n-party semi-honest MPC protocol π. Then the protocol Π presented in Figure 1 UC-securely realizes the ideal functionality $\mathcal{F}_f$ in the $\mathcal{F}_{CRS}$-hybrid model.

Common Random String vs Common Reference String. Our basic MPC protocol as described in Figure 1 uses a common reference string. We can adapt the construction to work in the setting of common random string (Appendix A.1.4) by assuming the existence of a CCA secure public-key encryption scheme (Appendix A.3) with perfect correctness and pseudorandom public keys.

Fairness. We note that the same protocol Π can be used to securely and fairly UC-realize the generalized functionality in the setting of honest majority, by using a fair semi-honest MPC protocol for π.

4 Applications

In this section we will discuss additional applications of our results.

4.1 Secure Computation on the Web

In a recent work, Halevi, Lindell and Pinkas [HLP11] studied of secure computation in a client-server model where each client connects to the server once and interacts with it, without any other client necessarily being connected at the same time. They show that, in such a setting, only limited security is achievable. However, among other results, they also point out that if we can get each of the players to connect twice to the server (rather than once), then their protocols can be used for achieving the standard notion of privacy.

One key aspect of the two-pass protocols of Halevi et. al [HLP11] is that there is a preset order in which the clients must connect to the server. Our protocol Π from Section 3 directly improves on the results in this setting by achieving the same two-pass protocol, but without such a preset order. Also, we achieve this result in the common reference/random string model, while the original protocols of Halevi et. al [HLP11] required a public key setup.

4.2 Black-Box Obfuscation for More Functions

In this subsection, we generalize the class of circuits that can be obfuscated according to the strong (virtual black box (VBB) notion of obfuscation. This application does not build directly on our protocol for two-round MPC. Rather, the main ideas here are related to ideas (particularly within the security proof) that arose in our MPC construction.

Our Result. Let $\mathcal{C}$ be a class of circuits that we believe to be VBB obfuscatable, e.g., point functions or conjunctions. Roughly speaking, assuming indistinguishability obfuscation, we show that a circuit C can be VBB obfuscated if there exists a circuit C' such that $C' \in \mathcal{C}$ and $C(x) = C'(x)$ for every input x. The non-triviality of the result lies in the fact that it might not be possible to efficiently recover C' from C. We refer the reader to Appendix E for a formal statement and proof.

Dynamic Point Function Obfuscation. We will now highlight the relevance of the results presented above with an example related to point functions. We know how to VBB obfuscate point functions. Now, consider a setting of three players. Player 1 generates a (perfectly binding) commitment to a value x. Player 2 would like to generate an obfuscation of an arbitrary function

f that allows an arbitrary Player 3, if he knows x, to evaluate f on input x alone (and nothing other than x). Our construction above enables such obfuscation. We stress that the challenge here is that Player 2 is not aware of the value x, which is in fact computationally hidden from it.

References

[AJLA+12] Gilad Asharov, Abhishek Jain, Adriana López-Alt, Eran Tromer, Vinod Vaikuntanathan, and Daniel Wichs. Multiparty computation with low communication, computation and interaction via threshold FHE. In *Advances in Cryptology - EUROCRYPT 2012*, volume 7237 of *Lecture Notes in Computer Science*, pages 483–501, 2012.

[AJW11] Gilad Asharov, Abhishek Jain, and Daniel Wichs. Multiparty computation with low communication, computation and interaction via threshold fhe. Cryptology ePrint Archive, Report 2011/613, 2011. `http://eprint.iacr.org/`.

[BCL+05] Boaz Barak, Ran Canetti, Yehuda Lindell, Rafael Pass, and Tal Rabin. Secure computation without authentication. In Victor Shoup, editor, *Advances in Cryptology – CRYPTO 2005*, volume 3621 of *Lecture Notes in Computer Science*, pages 361–377, Santa Barbara, CA, USA, August 14–18, 2005. Springer, Berlin, Germany.

[BFM88] Manuel Blum, Paul Feldman, and Silvio Micali. Non-interactive zero-knowledge and its applications. In *STOC*, pages 103–112, 1988.

[BGI+01] Boaz Barak, Oded Goldreich, Russell Impagliazzo, Steven Rudich, Amit Sahai, Salil P. Vadhan, and Ke Yang. On the (im)possibility of obfuscating programs. In Joe Kilian, editor, *Advances in Cryptology – CRYPTO 2001*, volume 2139 of *Lecture Notes in Computer Science*, pages 1–18, Santa Barbara, CA, USA, August 19–23, 2001. Springer, Berlin, Germany.

[BGI+12] Boaz Barak, Oded Goldreich, Russell Impagliazzo, Steven Rudich, Amit Sahai, Salil P. Vadhan, and Ke Yang. On the (im)possibility of obfuscating programs. *J. ACM*, 59(2):6, 2012.

[BGV12] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (leveled) fully homomorphic encryption without bootstrapping. In *ITCS*, 2012.

[BV11] Zvika Brakerski and Vinod Vaikuntanathan. Efficient fully homomorphic encryption from (standard) lwe. In *FOCS*, pages 97–106, 2011.

[Can01] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *42nd Annual Symposium on Foundations of Computer Science*, pages 136–145, Las Vegas, Nevada, USA, October 14–17, 2001. IEEE Computer Society Press.

[Can04] Ran Canetti. Universally composable signature, certification, and authentication. In *CSFW*, pages 219–. IEEE Computer Society, 2004.

[CF01] Ran Canetti and Marc Fischlin. Universally composable commitments. In Joe Kilian, editor, *Advances in Cryptology – CRYPTO 2001*, volume 2139 of *Lecture Notes in Computer Science*, pages 19–40, Santa Barbara, CA, USA, August 19–23, 2001. Springer, Berlin, Germany.

[CLOS02] Ran Canetti, Yehuda Lindell, Rafail Ostrovsky, and Amit Sahai. Universally composable two-party and multi-party secure computation. In *34th Annual ACM Symposium on Theory of Computing*, pages 494–503, Montréal, Québec, Canada, May 19–21, 2002. ACM Press.

[CLP10] Ran Canetti, Huijia Lin, and Rafael Pass. Adaptive hardness and composable security in the plain model from standard assumptions. In *51st Annual Symposium on Foundations of Computer Science*, pages 541–550, Las Vegas, Nevada, USA, October 23–26, 2010. IEEE Computer Society Press.

[DDN91] Danny Dolev, Cynthia Dwork, and Moni Naor. Non-malleable cryptography. In *23rd Annual ACM Symposium on Theory of Computing*, pages 542–552, New Orleans, Louisiana, USA, May 6–8, 1991. ACM Press.

[DH76] Whitfield Diffie and Martin E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22(6):644–654, 1976.

[FLS99] Uriel Feige, Dror Lapidot, and Adi Shamir. Multiple non-interactive zero knowledge proofs under general assumptions. *SIAM Journal of Computing*, 29(1):1–28, 1999.

[Gen09] Craig Gentry. *A fully homomorphic encryption scheme*. PhD thesis, Stanford University, 2009. `crypto.stanford.edu/craig`.

[GGH13a] Sanjam Garg, Craig Gentry, and Shai Halevi. Candidate multilinear maps from ideal lattices. In *EUROCRYPT*, 2013.

[GGH$^+$13b] Sanjam Garg, Craig Gentry, Shai Halevi, Mariana Raykova, Amit Sahai, and Brent Waters. Candidate indistinguishability obfuscation and functional encryption for all circuits. In *FOCS'13, to appear*. IEEE, 2013. Available from `http://eprint.iacr.org/2013/451`.

[GGSW13] Sanjam Garg, Craig Gentry, Amit Sahai, and Brent Waters. Witness encryption and its applications. In *STOC*, 2013.

[GM84] Shafi Goldwasser and Silvio Micali. Probabilistic encryption. *Journal of Computer and System Sciences*, 28(2):270–299, 1984.

[GMR89] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof systems. *SIAM Journal on Computing*, 18(1):186–208, 1989.

[GMW87] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game, or a completeness theorem for protocols with honest majority. In Alfred Aho, editor, *19th Annual ACM Symposium on Theory of Computing*, pages 218–229, New York City,, New York, USA, May 25–27, 1987. ACM Press.

[Gol01] Oded Goldreich. *Foundations of Cryptography: Basic Tools*, volume 1. Cambridge University Press, Cambridge, UK, 2001.

[Gol04] Oded Goldreich. *Foundations of Cryptography: Basic Applications*, volume 2. Cambridge University Press, Cambridge, UK, 2004.

[GOS06] Jens Groth, Rafail Ostrovsky, and Amit Sahai. Perfect non-interactive zero knowledge for np. In *Proceedings of Eurocrypt 2006, volume 4004 of LNCS*, pages 339–358. Springer, 2006.

[GSW13] Craig Gentry, Amit Sahai, and Brent Waters. Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based. In Ran Canetti and Juan A. Garay, editors, *CRYPTO (1)*, volume 8042 of *Lecture Notes in Computer Science*, pages 75–92. Springer, 2013.

[HLP11] Shai Halevi, Yehuda Lindell, and Benny Pinkas. Secure computation on the web: Computing without simultaneous interaction. In Phillip Rogaway, editor, *CRYPTO*, volume 6841 of *Lecture Notes in Computer Science*, pages 132–150. Springer, 2011.

[IKO05] Yuval Ishai, Eyal Kushilevitz, and Rafail Ostrovsky. Sufficient conditions for collision-resistant hashing. In Joe Kilian, editor, *TCC*, volume 3378 of *Lecture Notes in Computer Science*, pages 445–456. Springer, 2005.

[LATV12] Adriana López-Alt, Eran Tromer, and Vinod Vaikuntanathan. On-the-fly multiparty computation on the cloud via multikey fully homomorphic encryption. In *STOC*, pages 1219–1234, 2012.

[LP09] Yehuda Lindell and Benny Pinkas. A proof of security of Yao's protocol for two-party computation. *Journal of Cryptology*, 22(2):161–188, April 2009.

[MSS11] Steven Myers, Mona Sergi, and Abhi Shelat. Threshold fully homomorphic encryption and secure computation. *IACR Cryptology ePrint Archive*, 2011:454, 2011.

[NY90] Moni Naor and Moti Yung. Public-key cryptosystems provably secure against chosen ciphertext attacks. In *22nd Annual ACM Symposium on Theory of Computing*, Baltimore, Maryland, USA, May 14–16, 1990. ACM Press.

[RAD78] Ron Rivest, Leonard Adleman, and Michael L. Dertouzos. On data banks and privacy homomorphisms. In *Foundations of Secure Computation*, pages 169–180, 1978.

[RS92] Charles Rackoff and Daniel R. Simon. Non-interactive zero-knowledge proof of knowledge and chosen ciphertext attack. In Joan Feigenbaum, editor, *Advances in Cryptology – CRYPTO'91*, volume 576 of *Lecture Notes in Computer Science*, pages 433–444, Santa Barbara, CA, USA, August 11–15, 1992. Springer, Berlin, Germany.

[RSA78] Ronald L. Rivest, Adi Shamir, and Leonard M. Adleman. A method for obtaining digital signature and public-key cryptosystems. *Communications of the Association for Computing Machinery*, 21(2):120–126, 1978.

[SW13] Amit Sahai and Brent Waters. How to use indistinguishability obfuscation: Deniable encryption, and more. *IACR Cryptology ePrint Archive*, 2013:454, 2013.

[vDGHV10] Marten van Dijk, Craig Gentry, Shai Halevi, and Vinod Vaikuntanathan. Fully homomorphic encryption over the integers. In *EUROCRYPT*, pages 24–43, 2010.

[Yao82] Andrew C. Yao. Protocols for secure computations. In *23rd Annual Symposium on Foundations of Computer Science*, pages 160–164, Chicago, Illinois, November 3–5, 1982. IEEE Computer Society Press.

A Additional Background

A.1 UC Security

In this section we briefly review UC security. For full details see [Can01]. A large part of this introduction has been taken verbatim from [CLP10].

A.1.1 The basic model of execution

Following [GMR89, Gol01], a protocol is represented as an interactive Turing machine (ITM), which represents the program to be run within each participant. Specifically, an ITM has three tapes that can be written to by other ITMs: the input and subroutine output tapes model the inputs from and the outputs to other programs running within the same "entity" (say, the same physical computer), and the incoming communication tapes and outgoing communication tapes model messages received from and to be sent to the network. It also has an identity tape that cannot be written to by the ITM itself. The identity tape contains the program of the ITM (in some standard encoding) plus additional identifying information specified below. Adversarial entities are also modeled as ITMs.

We distinguish between ITMs (which represent static objects, or programs) and *instances of ITMs*, or ITIs, that represent interacting processes in a running system. Specifically, an ITI is an ITM along with an identifer that distinguishes it from other ITIs in the same system. The identifier consists of two parts: A session-identifier (SID) which identifies which protocol instance the ITM belongs to, and a party identifier (PID) that distinguishes among the parties in a protocol instance. Typically the PID is also used to associate ITIs with "parties", or clusters, that represent some administrative domains or physical computers.

The model of computation consists of a number of ITIs that can write on each other's tapes in certain ways (specified in the model). The pair (SID,PID) is a unique identifier of the ITI in the system.

With one exception (discussed within) we assume that all ITMs are probabilistic polynomial time (PPT). An ITM is PPT if there exists a constant $c > 0$ such that, at any point during its run, the overall number of steps taken by M is at most n^c, where n is the overall number of bits written on the *input tape* of M in this run. (In fact, in order to guarantee that the overall protocol execution process is bounded by a polynomial, we define n as the total number of bits written to the input tape of M, *minus the overall number of bits written by M to input tapes of other ITMs.*; see [Can01].)

A.1.2 Security of protocols

Protocols that securely carry out a given task (or, protocol problem) are defined in three steps, as follows. First, the process of executing a protocol in an adversarial environment is formalized. Next, an "ideal process" for carrying out the task at hand is formalized. In the ideal process the parties do not communicate with each other. Instead they have access to an "ideal functionality," which is essentially an incorruptible "trusted party" that is programmed to capture the desired functionality of the task at hand. A protocol is said to securely realize an ideal functionality if the process of running the protocol amounts to "emulating" the ideal process for that ideal functionality. Below we overview the model of protocol execution (called the *real-life model*), the ideal process, and the notion of protocol emulation.

The model for protocol execution. The model of computation consists of the parties running an instance of a protocol Π, an adversary $\mathcal{A}$ that controls the communication among the parties, and an *environment* $\mathcal{Z}$ that controls the inputs to the parties and sees their outputs. We assume that all parties have a security parameter $n \in \mathbb{N}$. (We remark that this is done merely for convenience and is not essential for the model to make sense). The execution consists of a sequence of *activations*, where in each activation a single participant (either $\mathcal{Z}$, $\mathcal{A}$, or some other ITM) is activated, and may write on a tape of at most *one* other participant, subject to the rules below. Once the activation of a participant is complete (i.e., once it enters a special waiting state), the participant whose tape was written on is activated next. (If no such party exists then the environment is activated next.)

The environment is given an external input z and is the first to be activated. In its first activation, the environment invokes the adversary $\mathcal{A}$, providing it with some arbitrary input. In the context of UC security, the environment can from now on invoke (namely, provide input to) only ITMs that consist of a single instance of protocol Π. That is, all the ITMs invoked by the environment must have the same SID and the code of Π.

Once the adversary is activated, it may read its own tapes and the outgoing communication tapes of all parties. It may either deliver a message to some party by writing this message on the party's incoming communication tape or report information to $\mathcal{Z}$ by writing this information on the subroutine output tape of $\mathcal{Z}$. For simplicity of exposition, in the rest of this paper we assume authenticated communication; that is, the adversary may deliver only messages that were actually sent. (This is however not essential as shown in [Can04, BCL+05].)

Once a protocol party (i.e., an ITI running Π) is activated, either due to an input given by the environment or due to a message delivered by the adversary, it follows its code and possibly writes a local output on the subroutine output tape of the environment, or an outgoing message on the adversary's incoming communication tape.

The protocol execution ends when the environment halts. The output of the protocol execution is the output of the environment. Without loss of generality we assume that this output consists of only a single bit.

Let $\mathrm{EXEC}_{\pi,\mathcal{A},\mathcal{Z}}(n, z, r)$ denote the output of the environment $\mathcal{Z}$ when interacting with parties running protocol Π on security parameter n, input z and random input $r = r_{\mathcal{Z}}, r_{\mathcal{A}}, r_1, r_2, \ldots$ as described above (z and $r_{\mathcal{Z}}$ for $\mathcal{Z}$; $r_{\mathcal{A}}$ for $\mathcal{A}$, r_i for party P_i). Let $\mathrm{EXEC}_{\pi,\mathcal{A},\mathcal{Z}}(n, z)$ random variable describing $\mathrm{EXEC}_{\pi,\mathcal{A},\mathcal{Z}}(n, z, r)$ where r is uniformly chosen. Let $\mathrm{EXEC}_{\pi,\mathcal{A},\mathcal{Z}}$ denote the ensemble $\{\mathrm{EXEC}_{\pi,\mathcal{A},\mathcal{Z}}(n, z)\}_{n\in\mathbb{N},z\in\{0,1\}^*}$.

Ideal functionalities and ideal protocols. Security of protocols is defined via comparing the protocol execution to an *ideal protocol* for carrying out the task at hand. A key ingredient in the ideal protocol is the *ideal functionality* that captures the desired functionality, or the specification, of that task. The ideal functionality is modeled as another ITM (representing a "trusted party") that interacts with the parties and the adversary. More specifically, in the ideal protocol for functionality $\mathcal{F}$ all parties simply hand their inputs to an ITI running $\mathcal{F}$. (We will simply call this

ITI $\mathcal{F}$. The SID of $\mathcal{F}$ is the same as the SID of the ITIs running the ideal protocol. (the PID of $\mathcal{F}$ is null.)) In addition, $\mathcal{F}$ can interact with the adversary according to its code. Whenever $\mathcal{F}$ outputs a value to a party, the party immediately copies this value to its own output tape. We call the parties in the ideal protocol dummy parties. Let $\Pi(\mathcal{F})$ denote the ideal protocol for functionality $\mathcal{F}$.

Securely realizing an ideal functionality. We say that a protocol Π *emulates* protocol ϕ if for any adversary $\mathcal{A}$ there exists an adversary $\mathcal{S}$ such that no environment $\mathcal{Z}$, on any input, can tell with non-negligible probability whether it is interacting with $\mathcal{A}$ and parties running Π, or it is interacting with S and parties running ϕ. This means that, from the point of view of the environment, running protocol Π is 'just as good' as interacting with ϕ. We say that Π *securely realizes* an ideal functionality $\mathcal{F}$ if it emulates the ideal protocol $\Pi(\mathcal{F})$. More precise definitions follow. A distribution ensemble is called *binary* if it consists of distributions over $\{0, 1\}$.

Definition 4. Let Π and ϕ be protocols. We say that Π UC-emulates ϕ if for any adversary $\mathcal{A}$ there exists an adversary $\mathcal{S}$ such that for any environment $\mathcal{Z}$ that obeys the rules of interaction for UC security we have $\mathrm{EXEC}_{\phi,\mathcal{S},\mathcal{Z}} \approx \mathrm{EXEC}_{\pi,\mathcal{A},\mathcal{Z}}$.

Definition 5. Let $\mathcal{F}$ be an ideal functionality and let Π be a protocol. We say that Π UC-realizes $\mathcal{F}$ if Π UC-emulates the ideal process $\Pi(\mathcal{F})$.

A.1.3 Hybrid protocols

Hybrid protocols are protocols where, in addition to communicating as usual as in the standard model of execution, the parties also have access to (multiple copies of) an ideal functionality. Hybrid protocols represent protocols that use idealizations of underlying primitives, or alternatively make *trust assumptions* on the underlying network. They are also instrumental in stating the universal composition theorem. Specifically, in an $\mathcal{F}$-hybrid protocol (i.e., in a hybrid protocol with access to an ideal functionality $\mathcal{F}$), the parties may give inputs to and receive outputs from an unbounded number of copies of $\mathcal{F}$.

The communication between the parties and each one of the copies of $\mathcal{F}$ mimics the ideal process. That is, giving input to a copy of $\mathcal{F}$ is done by writing the input value on the input tape of that copy. Similarly, each copy of $\mathcal{F}$ writes the output values to the subroutine output tape of the corresponding party. It is stressed that the adversary does not see the interaction between the copies of $\mathcal{F}$ and the honest parties.

The copies of $\mathcal{F}$ are differentiated using their SIDs. All inputs to each copy and all outputs from each copy carry the corresponding SID. The model does not specify how the SIDs are generated, nor does it specify how parties "agree" on the SID of a certain protocol copy that is to be run by them. These tasks are left to the protocol. This convention seems to simplify formulating ideal functionalities, and designing protocols that securely realize them, by freeing the functionality from the need to choose the SIDs and guarantee their uniqueness. In addition, it seems to reflect common practice of protocol design in existing networks.

The definition of a protocol securely realizing an ideal functionality is extended to hybrid protocols in the natural way.

The universal composition operation. We define the universal composition operation and state the universal composition theorem. Let ρ be an $\mathcal{F}$-hybrid protocol, and let Π be a protocol that securely realizes $\mathcal{F}$. The composed protocol ρ^{Π} is constructed by modifying the code of each ITM in ρ so that the first message sent to each copy of $\mathcal{F}$ is replaced with an invocation of a new copy of Π with fresh random input, with the same SID, and with the contents of that message as input. Each subsequent message to that copy of $\mathcal{F}$ is replaced with an activation of the corresponding copy of Π, with the contents of that message given to Π as new input. Each output value generated by a copy of Π is treated as a message received from the corresponding copy of $\mathcal{F}$. The copy of Π will start sending and receiving messages as specified in its code. Notice that if Π is a $\mathcal{G}$-hybrid protocol (i.e., ρ uses ideal evaluation calls to some functionality $\mathcal{G}$) then so is ρ^{Π}.

The universal composition theorem. Let $\mathcal{F}$ be an ideal functionality. In its general form, the composition theorem basically says that if Π is a protocol that UC-realizes $\mathcal{F}$ then, for any $\mathcal{F}$-hybrid protocol ρ, we have that an execution of the composed protocol ρ^{Π} "emulates" an execution of protocol ρ. That is, for any adversary $\mathcal{A}$ there exists a simulator $\mathcal{S}$ such that no environment machine $\mathcal{Z}$ can tell with non-negligible probability whether it is interacting with $\mathcal{A}$ and protocol ρ^{Π} or with $\mathcal{S}$ and protocol ρ, in a UC interaction. As a corollary, we get that if protocol ρ UC-realizes $\mathcal{F}$, then so does protocol ρ^{Π}. [1]

Theorem 3 (Universal Composition [Can01].)**.** Let $\mathcal{F}$ be an ideal functionality. Let ρ be a $\mathcal{F}$-hybrid protocol, and let Π be a protocol that UC-realizes $\mathcal{F}$. Then protocol ρ^{Π} UC-emulates ρ.

An immediate corollary of this theorem is that if the protocol ρ UC-realizes some functionality $\mathcal{G}$, then so does ρ^{Π}.

A.1.4 The Common Reference/Random String Model

In the common reference string (CRS) model [CF01, CLOS02], all parties in the system obtain from a trusted party a reference string, which is sampled according to a pre-specified distribution D. The reference string is referred to as the *CRS*. In the UC framework, this is modeled by an ideal functionality $\mathcal{F}_{CRS}^{D}$ that samples a string ρ from a pre-specified distribution D and sets ρ as the CRS. $\mathcal{F}_{CRS}^{D}$ is described in Figure 4.

Functionality $\mathcal{F}_{\mathbf{CRS}}^{\mathbf{D}}$

1. Upon activation with session id *sid* proceed as follows. Sample $\rho = D(r)$, where r denotes uniform random coins, and send $(\texttt{crs}, sid, \rho)$ to the adversary.
2. On receiving $(\texttt{crs}, sid)$ from some party send $(\texttt{crs}, sid, \rho)$ to that party.

Figure 4: The Common Reference String Functionality.

When the distribution D in $\mathcal{F}_{CRS}^{D}$ is sent to be the uniform distribution (on a string of appropriate length) then we obtain the common random string model.

[1] The universal composition theorem in [Can01] applies only to "subroutine respecting protocols", namely protocols that do not share subroutines with any other protocol in the system.

A.1.5 General Functionality

We consider the general-UC functionality $\mathcal{F}$, which securely evaluates any polynomial-time (possibly randomize) function $f : (\{0,1\}^{\ell_{in}})^n \rightarrow (\{0,1\}^{\ell_{out}})^n$. The functionality $\mathcal{F}_f$ is parameterized with a function f and is described in Figure 5. In this paper we will only be concerned with the *static* corruption model.

Functionality $\mathcal{F}_\mathbf{f}$

$\mathcal{F}_f$ parameterized by an (possibly randomized) n-ary function f, running with parties $\mathcal{P} = \{P_1, \ldots P_n\}$ (of which some may be corrupted) and an adversary $\mathcal{S}$, proceeds as follows:

1. Each party P_i (and $\mathcal{S}$ on behalf of P_i if P_i is corrupted) sends $(\mathsf{input}, \mathsf{sid}, \mathcal{P}, P_i, x_i)$ to the functionality.
2. Upon receiving the inputs from all parties, evaluate $(y_1, \ldots y_n) \leftarrow f(x_1, \ldots, x_n)$. For every P_i that is corrupted send adversary $\mathcal{S}$ the message $(\mathsf{output}, \mathsf{sid}, \mathcal{P}, P_i, y_i)$.
3. On receiving $(\mathsf{generateOutput}, \mathsf{sid}, \mathcal{P}, P_i)$ from $\mathcal{S}$ the ideal functionality outputs $(\mathsf{output}, \mathsf{sid}, \mathcal{P}, P_i, y_i)$ to P_i. (And ignores the message if inputs from all parties in $\mathcal{P}$ have not been received.)

Figure 5: General Functionality.

Our protocol in Figure 1 (also Theorem 1) is for UC-securely realizing general functionality $\mathcal{F}_f$ when the function f is restricted to be any deterministic poly-time function with n inputs and single output. This functionality has been formally defined in Figure 6.

As explained in Section 3 the same protocol can be used to obtain a protocol that UC-securely realizes the general functionality $\mathcal{F}_f$ for any function f.

Fairness. Our default notion of UC-security, as described above, is "security-with abort" meaning that the ideal adversary (simulator) can abort the computation and cause the functionality to not give output to honest parties. In addition, we say that a protocol π *securely and fairly realizes* a functionality $\mathcal{F}$ if $\mathcal{S}$ does not get the ability to abort. Meaning the functionality always sends the outputs to the honest parties.

A.2 Non-Interactive Zero-Knowledge Proofs

Let R be an efficiently computable binary relation. For pairs $(x, w) \in R$ we call x the statement and w the witness. Let L be the language consisting of statements in R.

A non-interactive proof system [BFM88, FLS99, GOS06] for a relation R consists of a common reference string generation algorithm K, a prover P and a verifier V. We require that they all be probabilistic polynomial time algorithms, *i.e.*, we are looking at *efficient prover* proofs. The common reference string generation algorithm produces a common reference string σ of length $\Omega(\lambda)$. The prover takes as input (σ, x, w) and produces a proof π. The verifier takes as input (σ, x, π) and

Functionality $\mathcal{F}_\mathbf{f}$

$\mathcal{F}_f$ parameterized by an n-ary deterministic single output function f, running with parties $\mathcal{P} = \{P_1, \dots P_n\}$ (of which some may be corrupted) and an adversary $\mathcal{S}$, proceeds as follows:

1. Each party P_i (and $\mathcal{S}$ on behalf of P_i if P_i is corrupted) sends $(\mathsf{input}, \mathsf{sid}, \mathcal{P}, P_i, x_i)$ to the functionality.
2. Upon receiving the inputs from all parties, evaluate $y \leftarrow f(x_1, \dots, x_n)$. Send adversary $\mathcal{S}$ the message $(\mathsf{output}, \mathsf{sid}, \mathcal{P}, y)$.
3. On receiving $(\mathsf{generateOutput}, \mathsf{sid}, \mathcal{P}, P_i)$ from $\mathcal{S}$ the ideal functionality outputs $(\mathsf{output}, \mathsf{sid}, \mathcal{P}, y)$ to P_i. (And ignores the message if inputs from all parties in $\mathcal{P}$ have not been received.)

Figure 6: General Functionality for Deterministic Single Output Functionalities.

outputs 1 if the proof is acceptable and 0 if rejecting the proof. We call (K, P, V) a non-interactive proof system for R if it has the completeness and statistical-soundness properties described below.

Perfect completeness. A proof system is complete if an honest prover with a valid witness can convince an honest verifier. Formally we have

$$\Pr\left[\sigma \leftarrow K(1^\lambda) : \exists(x, \pi) : x \notin L : V(\sigma, x, \pi) = 1\right] = 1.$$

Statistical soundness. A proof system is sound if it is infeasible to convince an honest verifier when the statement is false. For all (even unbounded) adversaries $\mathcal{A}$ we have

$$\Pr\left[\sigma \leftarrow K(1^\lambda); (x, \pi) \leftarrow \mathcal{A}(\sigma) : V(\sigma, x, \pi) = 1 : x \notin L\right] = \mathsf{negl}(\lambda).$$

Computational zero-knowledge [FLS99]. A proof system is computational zero-knowledge if the proofs do not reveal any information about the witnesses to a bounded adversary. We say a non-interactive proof (K, P, V) is computational zero-knowledge if there exists a polynomial time simulator $S = (S_1, S_2)$, where S_1 returns a simulated common reference string σ together with a simulation trapdoor τ that enables S_2 to simulate proofs without access to the witness. For all non-uniform polynomial time adversaries $\mathcal{A}$ we have for all $x \in L$

$$\Pr\left[\sigma \leftarrow K(1^\lambda); \pi \leftarrow P(\sigma, x, w) : \mathcal{A}(x, \sigma, \pi) = 1\right] \approx \Pr\left[(\sigma, \tau) \leftarrow S_1(1^\lambda); \pi \leftarrow S_2(\sigma, \tau, x) : \mathcal{A}(x, \sigma, \pi) = 1\right].$$

A.3 CCA secure encryption

A public-key encryption scheme [DH76, RSA78, GM84] is a triple $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$, where (i) Gen is the (randomized) *key generation algorithm*, outputting a pair (pk, sk) consisting of a public-key and a secret-key, respectively (ii) Enc is the (randomized) *encryption algorithm* outputting a ciphertext

$c = \mathsf{Enc}(pk, m)$ for any message m and a valid public key pk and (iii) Dec is the deterministic decryption algorithm such that $\mathsf{Dec}(sk, c)$ that outputs a message or $\{\perp\}$. All algorithms additionally take (implicitly) as input the security parameter λ. We sometimes need to make the randomness used by Enc explicit: In these cases, we write $\mathsf{Enc}(pk, m; r)$ to highlight the fact that random coins r are used to encrypt the message m.

Perfect Correctness. We say that the encryption scheme has *perfect correctness* if for overwhelming fraction of the randomness used by the key generation algorithm, all all messages we have $\Pr[\mathsf{Dec}(sk, \mathsf{Enc}(pk, m)) = m] = 1$.

CCA Security. The CCA security [RS92, NY90, DDN91] of the $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ is defined via the following security game between a *challenger* and an adversary $\mathcal{A}$:

1. The challenger generates $(pk, sk) \leftarrow \mathsf{Gen}(1^\lambda)$ and $b \leftarrow \{0, 1\}$, and gives pk to $\mathcal{A}$.
2. The adversary $\mathcal{A}$ ask decryption queries c, which are answered with the message $\mathsf{Dec}(sk, c)$.
3. The adversary $\mathcal{A}$ inputs (m_0, m_1) with $|m_0| = |m_1|$ to the challenger, and receives a challenge ciphertext $c^* \leftarrow \mathsf{Enc}(pk, m_b)$.
4. the adversary $\mathcal{A}$ asks further decryption queries $c \neq c^*$, which are answered with message $\mathsf{Dec}(sk, c)$.
5. The adversary $\mathcal{A}$ outputs a bit b', and *wins* the game if $b' = b$.

We say that a PKE scheme is CCA secure if for all (non-uniform) probabilistic polynomial time adversaries $\mathcal{A}$ we have that the probability $\left|\Pr[b' = b] - \frac{1}{2}\right|$ is negligible.

A.4 Multikey Fully Homomorphic Encryption

In this section, we define multikey fully homomorphic encryption (taken verbatim from [LATV12]). However here we will present a special case of their primitive. This simple primitive suffices for our purposes.

Intuitively, multikey FHE allows us to evaluate any circuit on ciphertexts that might be encrypted under different public keys. To guarantee semantic security, decryption requires all of the corresponding secret keys.

Let n be the number of distinct keys in the system. We let all algorithms depend polynomially on n. This is similar to the definition of "leveled" FHE from [BGV12]. However, we note that in this definition, the algorithms depend on n but are independent of the depth of circuits that the scheme can evaluate. Thus, we consider schemes that are "leveled" with respect to the number of keys n, but fully homomorphic ("non-leveled") with respect to the circuits that are evaluated. We now define multikey FHE as follows, for arbitrary circuits.

Definition 6 (Multikey Homomorphic Encryption). $(\mathsf{Setup}_{MK}, \mathsf{Encrypt}_{MK}, \mathsf{Eval}_{MK}, \mathsf{Decrypt}_{MK})$ is a multikey homomorphic encryption scheme if it has the following properties:

- $(pk, sk) \leftarrow \mathsf{Setup}_{MK}(1^\lambda)$, for a security parameter λ, outputs a public key pk, a secret key sk.

- $c \leftarrow \mathsf{Encrypt}_{MK}(pk, m)$, given a public key pk and message m, outputs a ciphertext c.
- $c^* := \mathsf{Eval}_{MK}(C, (c_1, pk_1), \ldots, (c_n, pk_n))$, given a (description of) a boolean circuit C along with n tuples (c_i, pk_i), each consisting of a ciphertext c_i and a public key pk_i, outputs a ciphertext c^*.
- $m' := \mathsf{Decrypt}_{MK}(sk_1, \ldots sk_n, c)$, given n secret keys, and a ciphertext c outputs a message m'.

We require absence of decryption failures and compactness of ciphertexts. Formally: for every circuit C, all sequences of n key tuples $\{(pk_j, sk_j)\}_{j \in [n]}$ each of which is in the support of $\mathsf{Setup}_{MK}(1^\lambda)$, and all plaintexts $(m_1, \ldots, m_t)$ and ciphertexts $(c_1, \ldots, c_t)$ such that c_i is in the support of $\mathsf{Encrypt}_{MK}(pk_i, m_i)$, Eval_{MK} satisfies the following properties:

Correctness: Let $c^* := \mathsf{Eval}_{MK}(C, (c_1, pk_1), \ldots, (c_n, pk_n))$. Then $\mathsf{Decrypt}_{MK}(sk_1, \ldots, sk_n, c^*) = C(m_1, \ldots, m_n)$.

Compactness: Let $c^* := \mathsf{Eval}_{MK}(C, (c_1, pk_1), \ldots, (c_n, pk_n))$. There exists a polynomial P such that $|c^*| \leq P(\lambda, n)$. In other words, the size of c is independent of $|C|$. Note, however, that we allow the evaluated ciphertext to depend on the number of keys, n.

Semantic security of a multikey FHE follows directly from the semantic security of the underlying encryption scheme.

B Description of our Simulator

Let $\mathcal{A}$ be a malicious, static adversary that interacts with parties running the protocol Π from Figure 1 in the $\mathcal{F}_{CRS}$-hybrid model. We construct an ideal world adversary $\mathcal{S}$ with access to the ideal functionality $\mathcal{F}_f$, which simulates a real execution of Π with $\mathcal{A}$ such that no environment $\mathcal{Z}$ can distinguish the ideal world experiment with $\mathcal{S}$ and $\mathcal{F}_f$ from a real execution of Π with $\mathcal{A}$.

Recall that $\mathcal{S}$ interacts with the ideal functionality $\mathcal{F}_f$ and with the environment $\mathcal{Z}$. The ideal adversary $\mathcal{S}$ starts by invoking a copy of $\mathcal{A}$ and running a simulated interaction of $\mathcal{A}$ with the environment $\mathcal{Z}$ and the parties running the protocol. Our simulator $\mathcal{S}$ proceeds as follows:

Simulated CRS: The common reference string is chosen by $\mathcal{S}$ in the following manner (recall that $\mathcal{S}$ chooses the CRS for the simulated $\mathcal{A}$ as we are in the $\mathcal{F}_{CRS}$-hybrid model):

1. $\mathcal{S}$ generates $(\sigma, \tau) \leftarrow S_1(1^\lambda)$, the simulated common reference string for the NIZK proof system (K, P, V) with simulator $S = (S_1, S_2)$.
2. $\mathcal{S}$ runs the setup algorithm $\mathsf{Gen}(1^\lambda)$ of the CCA secure encryption scheme and obtains a public key pk and a secret key sk.

$\mathcal{S}$ sets the common reference string to equal (σ, pk) and locally stores (τ, sk). (The secret key sk will be later used to extract inputs of the corrupted parties and the trapdoor τ for the simulated CRS σ will be used to generate simulated proofs.)

Simulating the communication with $\mathcal{Z}$: Every input value that $\mathcal{S}$ receives from $\mathcal{Z}$ is written on $\mathcal{A}$'s input tape. Similarly, every output value written by $\mathcal{A}$ on its own output tape is directly copied to the output tape of $\mathcal{S}$.

Simulating actual protocol messages in Π: Note that there might be multiple sessions executing concurrently. Let sid be the session identifier for one specific session. We will specify the simulation strategy corresponding to this specific session. The simulator strategy for all other sessions will be the same. Let $\mathcal{P} = \{P_1, \ldots, P_n\}$ be the set of parties participating in the execution of Π corresponding to the session identified by the session identifier sid. Also let $\mathcal{P}^{\mathcal{A}} \subseteq \mathcal{P}$ be the set of parties corrupted by the adversary $\mathcal{A}$. (Recall that we are in the setting with static corruption.)

In the subsequent exposition we will assume that at least one party is honest. If no party is honest then the simulator does not need to do anything else.

Round 1 Messages $\mathcal{S} \to \mathcal{A}$: In the first round $\mathcal{S}$ must generate messages on behalf of the honest parties, i.e. parties in the set $\mathcal{P} \backslash \mathcal{P}^{\mathcal{A}}$. For each party $P_i \in \mathcal{P} \backslash \mathcal{P}^{\mathcal{A}}$ our simulator proceeds as:

1. $c_i = \mathsf{Enc}(i||0^{\ell_{in}})$ and, (recall that ℓ_{in} is the length of inputs of all parties)
2. $\forall j \in [n]$, and generate $d_{i,j} = \mathsf{Enc}(i||0^{\ell})$. (Recall that ℓ is the length of the maximum number of random coins needed by any party in π.)

It then sends $Z_i = \{c_i, \{d_{i,j}\}_{j \in [n]}\}$ to $\mathcal{A}$ on behalf of party P_i.

Round 1 Messages $\mathcal{A} \to \mathcal{S}$: Also in the first round the adversary $\mathcal{A}$ generates the messages on behalf of corrupted parties in $\mathcal{P}^{\mathcal{A}}$. For each party $P_i \in \mathcal{P}^{\mathcal{A}}$ our simulator proceeds as:

1. Let $Z_i = \{c_i, \{d_{i,j}\}_{j \in [n]}\}$ be the message that $\mathcal{A}$ sends on behalf of P_i. Our simulator $\mathcal{S}$ decrypts the ciphertexts using the secret key sk. In particular $\mathcal{S}$ sets $x'_i = \mathsf{Dec}(sk, c_i)$ and $r'_{i,j} = \mathsf{Dec}(sk, d_{i,j})$. Obtain $x_i \in \{0,1\}^{\ell_{in}}$ such that $x'_i = i||x_i$. If x'_i is not of this form the set $x_i = \bot$. Similarly obtain $r_{i,j}$ from $r'_{i,j}$ for every j setting the value to $\bot$ in case it is not of the right format.
2. $\mathcal{S}$ sends $(\mathsf{input}, \mathsf{sid}, \mathcal{P}, P_i, x_i)$ to $\mathcal{F}_f$ on behalf of the corrupted party P_i. It saves the values $\{r_{i,j}\}_j$ for later use.

Round 2 Messages $\mathcal{S} \to \mathcal{A}$: In the second round $\mathcal{S}$ must generate messages on behalf of the honest parties, i.e. parties in the set $\mathcal{P} \backslash \mathcal{P}^{\mathcal{A}}$. $\mathcal{S}$ proceeds as follows:

1. $\mathcal{S}$ obtains the output $(\mathsf{output}, \mathsf{sid}, \mathcal{P}, y)$ from the ideal functionality $\mathcal{F}_f$ and now it needs to force this output onto the adversary $\mathcal{A}$.
2. In order to force the output, the simulator $\mathcal{S}$ executes the simulator $\mathcal{S}_\pi$ and obtains a simulated transcript. The simulated transcript specifies the random coins of all the parties in $\mathcal{P}^{\mathcal{A}}$ and the protocol messages. Let s_i denote the random coins of party $P_i \in \mathcal{P}^{\mathcal{A}}$ and let $m_{i,j}$ for $i \in [n]$ and $j \in [t]$ denote the protocol messages. (Semi-honest security of protocol π implies the existence of such a simulator.)
3. For each $P_j \in \mathcal{P}^{\mathcal{A}}$ sample $r_{i,j}$ randomly in $\{0,1\}^{\ell}$ for each $P_i \in \mathcal{P} \backslash \mathcal{P}^{\mathcal{A}}$ subject to the constraint that $\oplus_{i=1}^{n} r_{i,j} = s_j$.

4. For each $P_i \in \mathcal{P} \backslash \mathcal{P}^{\mathcal{A}}$, $\mathcal{S}$ proceeds as follows:

   (a) For every $j \in [n]$, $j \neq i$ generate $\gamma_{i,j}$ as a simulated NIZK proof under σ for the NP-statement:
   $$\left\{\exists\ \rho_{r_{i,j}} \ \middle|\ d_{i,j} = \mathsf{Enc}(i||r_{i,j}; \rho_{r_{i,j}})\right\}.$$

   (b) A sequence of obfuscations $(i\mathcal{O}_{i,1}, \ldots i\mathcal{O}_{i,t})$ where $i\mathcal{O}_{i,j}$ is the obfuscation of the program $\mathsf{Prog}_{i,j}^{1,x_i,\rho_{x_i},r_{i,i},\rho_{r_{i,i}},\{Z_i\},\mathsf{fixedOutput}}$, where $\mathsf{fixedOutput}$ is the value $(m_{i,j}, \phi_{i,j})$ such that $\phi_{i,j}$ is the simulated proof that $m_{i,j}$ was generated correctly. (Recall that the flag has been set to 1 and this program on accepting inputs always outputs the value $\mathsf{fixedOutput}$.)

   (c) It sends $(\{r_{i,j}, \gamma_{i,j}\}_{j\in[n],j\neq i}, \{i\mathcal{O}_{i,j}\}_{j\in[t]})$ to $\mathcal{A}$ on behalf of P_i.

Round 2 Messages $\mathcal{A} \to \mathcal{S}$: Also in the second round the adversary $\mathcal{A}$ generates the messages on behalf of corrupted parties $\mathcal{P}^{\mathcal{A}}$. For each party $P_i \in \mathcal{P} \backslash \mathcal{P}^{\mathcal{A}}$ that has obtained "correctly formed" second round messages from all parties in $\mathcal{P}^{\mathcal{A}}$, our simulator sends $(\mathsf{generateOutput}, \mathsf{sid}, \mathcal{P}, P_i)$ to the ideal functionality.

This completes the description of the simulator.

C Proof of Security

In this section, via a sequence of hybrids, we will prove that no environment $\mathcal{Z}$ can distinguish the ideal world experiment with $\mathcal{S}$ and $\mathcal{F}_f$ (as defined above) from a real execution of Π with $\mathcal{A}$. We will start with the real world execution in which the adversary $\mathcal{A}$ interacts directly with the honest parties holding their inputs and step-by-step make changes till we finally reach the simulator as described in Appendix B. At each step will argue that the environment cannot distinguish the change except with negligible probability.

- H_1: This hybrid corresponds to the $\mathcal{Z}$ interacting with the real world adversary $\mathcal{A}$ and honest parties that hold their private inputs.

  We can restate the above experiment with the simulator as follows. We replace the real world adversary $\mathcal{A}$ with the ideal world adversary $\mathcal{S}$. The ideal adversary $\mathcal{S}$ starts by invoking a copy of $\mathcal{A}$ and running a simulated interaction of $\mathcal{A}$ with the environment $\mathcal{Z}$ and the honest parties. $\mathcal{S}$ forwards the messages that $\mathcal{A}$ generates for it environment directly to $\mathcal{Z}$ and vice versa (as explained in the description of the simulator $\mathcal{S}$). In this hybrid the simulator $\mathcal{S}$ holds the private inputs of the honest parties and generates messages on their behalf using the honest party strategies as specified by Π.

- H_2: In this hybrid we change how the simulator generates the CRS. In particular we will change how $\mathcal{S}$ generates the public key pk of the CCA secure encryption scheme. We will not change the way CRS for the NIZK is generated.

  $\mathcal{S}$ runs the setup algorithm $\mathsf{Gen}(1^\lambda)$ of the CCA secure encryption scheme and obtains a public key pk and a secret key sk. $\mathcal{S}$ will use this public key pk as part of the CRS and use the secret key sk to decrypt the ciphertexts generated by $\mathcal{A}$ on behalf of $\mathcal{P}^{\mathcal{A}}$. In particular for each party $P_i \in \mathcal{P}^{\mathcal{A}}$ our simulator proceeds as:

– Let $Z_i = \{c_i, \{d_{i,j}\}_{j\in[n]}\}$ be the message that $\mathcal{A}$ sends on behalf of P_i. Our simulator $\mathcal{S}$ decrypts the ciphertexts using the secret key sk. In particular $\mathcal{S}$ sets $x'_i = \mathsf{Dec}(sk, c_i)$ and $r'_{i,j} = \mathsf{Dec}(sk, d_{i,j})$. Obtain $x_i \in \{0,1\}^{\ell_{in}}$ such that $x'_i = i||x_i$. If x'_i is not of this form the set $x_i = \perp$. Similarly obtain $r_{i,j}$ from $r'_{i,j}$ for every j setting the value to $\perp$ in case it is not of the right format.

Note that in hybrids H_2 and H_1 is that $\mathcal{S}$ now additionally uses the secret key sk to extract the inputs of the adversarial parties. Furthermore if at any point in the execution any of the messages of the adversary are inconsistent with the input and randomness extracted but the adversary succeeds in providing an accepting NIZK proof then the simulator aborts `Extract Abort`.

The distribution of the CRS, and hence the view of the environment $\mathcal{Z}$, in the two cases is identical. Also note that it follows from the perfect correctness of the encryption scheme and the statistical soundness of the NIZK proof system that the NIZK proofs adversary generates will have to be consistent with the extracted values. In other words over the random choices of the CRS we have that the probability of `Extract Abort` is negligible.

- H_3: In this hybrid we will change how the simulator generates the obfuscations on behalf of honest parties. Roughly speaking we observe that the obfuscations can only be evaluated to output one unique value (consistent with inputs and randomness extracted using sk) and we can just hardcode this value into the obfuscated circuit. More formally in the second round $\mathcal{S}$ generates the messages on behalf of the honest parties, i.e. parties in the set $\mathcal{P}\backslash\mathcal{P}^{\mathcal{A}}$ as follows:

  1. For every P_j, $\mathcal{S}$ obtains $s_j = \oplus_{i=1}^{n} r_{i,j}$.
  2. $\mathcal{S}$ virtually executes the protocol π with inputs $x_1, \ldots, x_n$ and random coins $s_1, \ldots, s_n$ for the parties $P_1, \ldots P_n$ respectively, and obtains the messages $m_{i,j}$ for all $i \in [n]$ and $j \in [t]$.
  3. For each $P_i \in \mathcal{P}\backslash\mathcal{P}^{\mathcal{A}}$, $\mathcal{S}$ proceeds as follows:
     (a) For every $j \in [n]$, $j \neq i$ generate $\gamma_{i,j}$ as a NIZK proof under σ for the NP-statement:
$$\left\{\exists\ \rho_{r_{i,j}} \;\middle|\; d_{i,j} = \mathsf{Enc}(i||r_{i,j}; \rho_{r_{i,j}})\right\}.$$
     (b) A sequence of obfuscations $(i\mathcal{O}_{i,1}, \ldots i\mathcal{O}_{i,t})$ where $i\mathcal{O}_{i,j}$ is the obfuscation of the program $\mathsf{Prog}_{i,j}^{1,x_i,\rho_{x_i},r_{i,i},\rho_{r_{i,i}},\{Z_i\},\mathsf{fixedOutput}}$, where $\mathsf{fixedOutput}$ is the value $(m_{i,j}, \phi_{i,j})$ such that $\phi_{i,j}$ is the proof that $m_{i,j}$ was generated correctly. (Recall that the flag has been set to 1 and this program on all accepting inputs always outputs the value $\mathsf{fixedOutput}$.)
     (c) It sends $(\{r_{i,j}, \gamma_{i,j}\}_{j\in[n],j\neq i}, \{i\mathcal{O}_{i,j}\}_{j\in[t]})$ to $\mathcal{A}$ on behalf of P_i.

We will now argue that hybrids H_2 and H_3 and computationally indistinguishable. More formally we will consider a sequence of $t \cdot |\mathcal{P}\backslash\mathcal{P}^{\mathcal{A}}|$ hybrids $H_{3,0,0}, \ldots H_{3,|\mathcal{P}\backslash\mathcal{P}^{\mathcal{A}}|,t}$. In hybrid $H_{3,i,j}$ all the obfusctaions by the first $i-1$ honest parties and the first j obfuscations generated by the i^{th} honest party are generated in the modified way as described above. It is easy to see that hybrid $H_{3,0,0}$ is same as hybrid H_2 and hybrid $H_{3,|\mathcal{P}\backslash\mathcal{P}^{\mathcal{A}}|,t}$ is same as hybrid H_3 itself.

We will now argue that the hybrids $H_{3,i,j-1}$ and $H_{3,i,j}$ for $j \in [t]$ are computationally indistinguishable. This implies the above claim, but in order to argue the above claim we first prove the following lemma.

Lemma 1.

$$\Pr\left[\begin{array}{lll} \exists\, a, b \ : & & \mathsf{Prog}_{i,j}^{0,x_i,\rho_{x_i},r_{i,i},\rho_{r_{i,i}},\{Z_i\},0^{\ell_{i,j}}}(a) \neq \mathsf{Prog}_{i,j}^{0,x_i,\rho_{x_i},r_{i,i},\rho_{r_{i,i}},\{Z_i\},0^{\ell_{i,j}}}(b) \\ & \wedge & \mathsf{Prog}_{i,j}^{0,x_i,\rho_{x_i},r_{i,i},\rho_{r_{i,i}},\{Z_i\},0^{\ell_{i,j}}}(a) \neq \perp \\ & \wedge & \mathsf{Prog}_{i,j}^{0,x_i,\rho_{x_i},r_{i,i},\rho_{r_{i,i}},\{Z_i\},0^{\ell_{i,j}}}(b) \neq \perp \end{array}\right] = \mathsf{negl}(\lambda)$$

where the probability is taken over the random choices of the generation of the CRS.

Proof. Recall that program $\mathsf{Prog}_{i,j}^{0,x_i,\rho_{x_i},r_{i,i},\rho_{r_{i,i}},\{Z_i\},0^{\ell_{i,j}}}$ represents the j^{th} message function of the i^{th} party in protocol π. Recall that the input to the program consists of two $(R, \Gamma, M_{j-1}, \Phi_{j-1})$. We will refer to the (R, M_{j-1}) as the *main input part* and the Γ, Φ_{j-1} as the *proof part.*

Observe that since the proofs are always consistent with the extracted inputs and randomness, we have that there is a unique main input part for which adversary can provide valid (or accepting) proof parts. Further note that if the proof part is not accepting then $\mathsf{Prog}_{i,j}$ just outputs $\perp$. In other words if the proof is accepting then the program outputs a fixed value that depends just on the values that are fixed based on $\{Z_i\}$ values. We stress that the output actually does include a NIZK proof as well, however it is not difficult to see that this NIZK proof is also unique as a fixed randomness is used in generation of the proof. ■

Armed with Lemma 1, we can conclude that the programs $\mathsf{Prog}_{i,j}^{0,x_i,\rho_{x_i},r_{i,i},\rho_{r_{i,i}},\{Z_i\},0^{\ell_{i,j}}}$ and $\mathsf{Prog}_{i,j}^{1,x_i,\rho_{x_i},r_{i,i},\rho_{r_{i,i}},\{Z_i\},\mathsf{fixedOutput}}$ are functionally equivalent. Next based on the indistinguisbaility obfuscation property, it is easy to see that the hybrids $H_{3,i,j-1}$ and $H_{3,i,j}$ are computationally indistinguishable.

- H_4: In this hybrid we change how the simulator generates the NIZKs on behalf of honest parties. Formally $\mathcal{S}$ generates the σ using the simulator S_1 of the NIZK proof system and generates all the proofs using the simulator S_2. The argument can be made formal by considering a sequence of hybrids and changing each of the NIZK proofs one at a time.

  The indistinguishability between hybrids H_3 and H_4 can be based on the zero-knowledge property of the NIZK proof system.

- H_5: In this hybrid we change how the simulator $\mathcal{S}$ generates the first round messages on behalf of honest parties. In particular $\mathcal{S}$ instead of encrypting inputs and randomness of honest parties just encrypts zero strings of appropriate length.

  We could try to base the indistinguishabilty between hybrids H_4 and H_5 on the semantic security of the PKE scheme. However observe that $\mathcal{S}$ at the same time should continue to be able to decrypt the ciphertexts that $\mathcal{A}$ generates on behalf of corrupted parties. Therefore we need to rely on the CCA security of the PKE scheme.

- H_6: In this hybrid instead of generating all the messages $m_{i,j}$ on behalf of honest parties honestly $\mathcal{S}$ uses $\mathcal{S}_\pi$ (the simulator for the underlying MPC protocol) to generated simulated messages.

  The indistinguishability between hybrids H_5 and H_6 directly follows for the indistinguishability of honestly generated transcript in the execution of π from the transcript generated by $\mathcal{S}_\pi$.

- H_7: Observe that in hybrid H_6, $\mathcal{S}$ uses inputs of honest parties just in obtaining the output of the computation. It can obtain the same value by sending extracted inputs of the malicious parties to the ideal functionality $\mathcal{F}_f$.

  Note that the hybrids H_6 and H_7 are identical.

Observe that hybrid H_7 is identical to the simulation strategy described in Appendix B. This concludes the proof.

D Proof of Theorem 2

In this section we will give a proof for Theorem 2. Let $\mathcal{A}$ be a malicious, static adversary that interacts with parties running the protocol Π' from Figure 3 in the $\mathcal{F}_{CRS}$-hybrid model. We construct an ideal world adversary $\mathcal{S}$ with access to the ideal functionality $\mathcal{F}_f$, which simulates a real execution of Π' with $\mathcal{A}$ such that no environment $\mathcal{Z}$ can distinguish the ideal world experiment with $\mathcal{S}$ and $\mathcal{F}_f$ from a real execution of Π' with $\mathcal{A}$.

Recall that our protocol Π' executes protocol Π, and also that in Π' party P_i sends (pk_i, c_i) in the first round. Another difference is that in Π' the parties decide on the function to be evaluated after seeing the first message of all the parties, while in Π we need to provide the funtion before the protocol execution of Π begins. Also, recall that we already proved that Π is UC-secure in the $\mathcal{F}_{CRS}$-hybrid model. In other words, for every adversary $\mathcal{A}_\Pi$ there exist a simulator $\mathcal{S}_\Pi$ such that no environment $\mathcal{Z}$ can distinguish the ideal world experiment with $\mathcal{S}_\Pi$ and $\mathcal{F}_f$ from a real execution of Π with $\mathcal{A}_\Pi$. We note that the simulator $\mathcal{S}_\Pi$ has the special property that the first round message of $\mathcal{S}_\Pi$ does not rely on the function that is being evaluated. We will crucially use this property.

Our ideal adversary $\mathcal{S}$ starts by invoking a copy of $\mathcal{A}$ and running a simulated interaction of $\mathcal{A}$ with the environment $\mathcal{Z}$ and the parties running the protocol. Our simulator $\mathcal{S}$ proceeds as follows:

Simulated CRS: Our simulator lets $\mathcal{S}_\Pi$ generate the CRS.

Simulating the communication with $\mathcal{Z}$: Every input value that $\mathcal{S}$ receives from $\mathcal{Z}$ is written on $\mathcal{A}$'s input tape. Similarly, every output value written by $\mathcal{A}$ on its own output tape is directly copied to the output tape of $\mathcal{S}$.

Round 1 Messages $\mathcal{S} \to \mathcal{A}$: For each party $P_i \in \mathcal{P} \backslash \mathcal{P}^{\mathcal{A}}$, our simulator generates a public key pk_i and a secret key sk_i using the setup algorithm Setup_{MK} and generates the ciphertext c_i as an encryption of the zero string.

In addition to the public key and the ciphertext, $\mathcal{S}$ invokes $\mathcal{S}_\Pi$ in order to generate the first message on behalf of the honest parties. Note here that we are making use of the observation that the first message of the simulator $\mathcal{S}_\Pi$ is independent of the function being evaluated.

Round 1 Messages $\mathcal{A} \to \mathcal{S}$: Also in the first round the adversary $\mathcal{A}$ generates the messages on behalf of corrupted parties in $\mathcal{P}^{\mathcal{A}}$. For each party $P_i \in \mathcal{P}^{\mathcal{A}}$ our simulator obtains a public key pk_i along with a ciphertext c_i and the first round message of Π that $\mathcal{A}$ sends on behalf of P_i. The adversary passes the Π-messages directly to the simulator $\mathcal{S}_\Pi$. The simulator $\mathcal{S}_\Pi$ at this point extracts the inputs that adversary uses on behalf of malicious paries and provides the messages $(\mathsf{input}, \mathcal{P}, P_i, (x_i, \rho_i, \varrho_i))$ corresponding to all corrupted parties. Instead of passing it directly to the ideal functionality $\mathcal{S}$ proceeds as follows. Recall that the input of party P_i consists of (x_i, ρ_i, ϱ_i). Our simulator checks to see if $(pk_i, sk_i) = \mathsf{Setup}_{MK}(1^\lambda; \rho_i)$ and $c_i = \mathsf{Encrypt}_{MK}(pk_i, x_i; \varrho_i)$. If the checks pass then it passes on the input x_i to the ideal functionality, and otherwise it sends $\perp$.

Round 2 Messages: To generate messages for round two, our simulator $\mathcal{S}$ just invokes $\mathcal{S}_\Pi$ with the response that it obtains from its ideal functionality.

This completes the description of the simulator.

Indistinguishability Proof. In this section, via a sequence of hybrids, we will prove that no environment $\mathcal{Z}$ can distinguish the ideal world experiment with $\mathcal{S}$ and $\mathcal{F}_f$ (as defined above) from a real execution of Π with $\mathcal{A}$. We will start with the real world execution in which the adversary $\mathcal{A}$ interacts directly with the honest parties holding their inputs and step-by-step make changes till we finally reach the simulator as described above. At each step will argue that the environment cannot distinguish the change except with negligible probability.

- H_1: This hybrid corresponds to the $\mathcal{Z}$ interacting with the real world adversary $\mathcal{A}$ and honest parties that hold their private inputs.

  We can restate the above experiment with the simulator as follows. We replace the real world adversary $\mathcal{A}$ with the ideal world adversary $\mathcal{S}$. The ideal adversary $\mathcal{S}$ starts by invoking a copy of $\mathcal{A}$ and running a simulated interaction of $\mathcal{A}$ with the environment $\mathcal{Z}$ and the honest parties. $\mathcal{S}$ forwards the messages that $\mathcal{A}$ generates for it environment directly to $\mathcal{Z}$ and vice versa (as explained in the description of the simulator $\mathcal{S}$). In this hybrid the simulator $\mathcal{S}$ holds the private inputs of the honest parties and generates messages on their behalf using the honest party strategies as specified by Π.

- H_2: In this hybrid we change how the simulator generates messages for protocol Π. In particular the simulator $\mathcal{S}$ uses the simulator $\mathcal{S}_\Pi$ to generate Π messages.

  Indistinguishability follows from the security of the simulator $\mathcal{S}_\Pi$.

- H_3: In this hybrid we change how the simulator $\mathcal{S}$ generates the first round non-Π messages on behalf of honest parties. In particular, $\mathcal{S}$ still generates the public keys as before, but instead of encrypting the inputs of honest parties, it just encrypts zero strings of appropriate length.

  The indistinguishability follows from the semantic security of the multikey FHE scheme.

Observe that hybrid H_3 is identical to the simulation strategy described earlier. This concludes the proof.

E Black-Box Obfuscation for More Functions

Toward formalizing the result sketched in Section 4.2, we start by recalling the notion of virtual black box (VBB) obfuscation.

Definition 7 (VBB Obfuscation [BGI+01])**.** A uniform PPT machine $\mathcal{O}$ is called a *VBB obfuscator* for a circuit class $\{\mathcal{C}_\lambda\}$ if the following conditions are satisfied:

- For all security parameters $\lambda \in \mathbb{N}$, for all $C \in \mathcal{C}_\lambda$, for all inputs x, we have that

$$\Pr[C'(x) = C(x) : C' \leftarrow i\mathcal{O}(\lambda, C)] = 1$$

- For any polynomial size circuit adversary $\mathcal{A}$, there exists a polynomial size simulator circuit $\mathcal{S}$ such that for every input length λ and every $C \in \mathcal{C}_\lambda$:

$$|\Pr[\mathcal{A}(\mathcal{O}(C)) = 1] - \Pr[\mathcal{S}^C(1^\lambda) = 1]| \leq \mathsf{negl}(\lambda)$$

  where the probability is over the coins of the simulator and the obfuscator.

We say that class of circuits $\{\mathcal{C}_\lambda\}$ have an $\alpha(\lambda)$-expanding VBB obfuscator if there exists an obfuscation procedure that can VBB obfuscate any circuit $C \in \mathcal{C}_\lambda$ such that the size of the obfuscated circuit is at most $\alpha(\lambda)$.

Theorem 4. Let $\{\mathcal{C}_\lambda\}$ be a class of circuits that are conjectured to have $\alpha(\lambda)$-expanding VBB obfuscatable. Then assuming that indistinguishability obfuscator exists, we show that a VBB obfuscator for circuit class $\{\mathcal{D}_\lambda\}$ such that:

- $\forall D \in \mathcal{D}_\lambda \ \exists \ C \in \mathcal{C}_\lambda$, such that $\forall \ x \ C(x) = D(x)$.

Proof. We start by giving the description of our obfuscator. Our obfuscator on input a circuit $D \in \mathcal{D}_\lambda$ outputs $i\mathcal{O}(D')$ where D' is the circuit D padded to size $\alpha(\lambda)$.

To prove security, we need to show that for every adversary $\mathcal{A}_D$ there exists a simulator $\mathcal{S}_D$ such that for all circuits in $D \in \mathcal{D}_\lambda$ the distributions $\mathcal{S}_D^D(1^\lambda)$ and $\mathcal{A}_D(i\mathcal{O}(D'))$ are close, where D' is the circuit D padded to size $\alpha(\lambda)$.

We will first describe our simulator S_D. Note that we are assuming that the class $\mathcal{C}_\lambda$ is VBB obfuscatable. This implies an obfuscator $\mathcal{O}$ such that for every adversary $\mathcal{A}_C$ there exists a simulator $\mathcal{S}_C$ such that for all $C \in \mathcal{C}_\lambda$ the distributions $\mathcal{A}_C(\mathcal{O}(C))$ and $\mathcal{S}_C^C(1^\lambda)$ are close. Hence the task of constructing our simulator reduces to the task of constructing an adversary $\mathcal{A}_C$. Our adversary $\mathcal{A}_C$ on input x simply outputs $\mathcal{A}_D(i\mathcal{O}(x))$.

We prove the indistinguishability between real and simulated executions using a sequence of hybrids.

- H_0: This corresponds to the distribution $\mathcal{A}_D(i\mathcal{O}(D'))$.

- H_1: The distribution for this hybrid is $\mathcal{A}_D(i\mathcal{O}(\mathcal{O}(C)))$, where $C \in \mathcal{C}_\lambda$ is such that $\forall \ x, \ C(x) = D(x)$. Observe that this hybrid is exactly $\mathcal{A}_C(\mathcal{O}(C))$.

  The indistinguishability between H_0 and H_1 follows from the indistinguishability obfuscation property of $i\mathcal{O}$.

- H_2: This hybrid corresponds to the distribution $\mathcal{S}_C^D(1^\lambda)$, where $\mathcal{S}_C$ is the simulator corresponding to the adversary $\mathcal{A}_C$ constructed in the previous hybrid. (Note that oracle access to C or D is equivalent.)

  Indistinguishability between H_1 and H_2 from the VBB obfuscation property of $\mathcal{O}$. Observe that hybrid H_2 corresponds to the simulation strategy itself.

This concludes the proof. ■