Adaptively Secure, Universally Composable, Multiparty Computation in Constant Rounds

Dana Dachman-Soled[1], Jonathan Katz[1,⋆], and Vanishree Rao[2,⋆⋆]

[1] University of Maryland, USA
danadach@ece.umd.edu, jkatz@cs.umd.edu
[2] University of California at Los Angeles, USA
vanishri@cs.ucla.edu

Abstract. Cryptographic protocols with *adaptive security* ensure that security holds against an adversary who can dynamically determine which parties to corrupt as the protocol progresses—or even after the protocol is finished. In the setting where all parties may potentially be corrupted, and secure erasure is not assumed, it has been a long-standing open question to design secure-computation protocols with adaptive security running in *constant* rounds.

Here, we show a constant-round, universally composable protocol for computing any functionality, tolerating a malicious, adaptive adversary corrupting any number of parties. Interestingly, our protocol can compute *all* functionalities, not just adaptively well-formed ones. The protocol relies on indistinguishability obfuscation, and assumes a common reference string.

1 Introduction

When designing and analyzing protocols for secure computation, there are several different adversarial models one can consider. The original definitions of security assume a *static* adversary who decides which parties to corrupt before execution of the protocol begins. Subsequently [3,11], researchers began to consider the more challenging setting in which the adversary may *adaptively* decide which parties to corrupt as the protocol progresses—or even after the protocol ends. It is easy to come up with examples of protocols that are secure in a static-corruption model, but that are trivially insecure in the adaptive setting.

Even in a setting where adaptive corruptions are considered, there are different assumptions one can make. Initial work on adaptive security [3] made the assumption that honest parties can securely *erase* local data (e.g., randomness or other internal state) when no longer needed. Later work, led by Canetti et al. [11], sought to avoid this assumption, arguing that it is unwise to rely on other parties to erase data (since there is no way such erasure can be verified) and that it is generally difficult—even for an honest party who intends

⋆ Work supported in part by NSF awards #1111599 and #1223623.
⋆⋆ Work done while visiting the University of Maryland.

Y. Dodis and J.B. Nielsen (Eds.): TCC 2015, Part II, LNCS 9015, pp. 586–613, 2015.

to erase data—to ensure that all traces of data are gone. Whether or not erasure is assumed has a significant impact on the complexity of adaptively secure protocols; for example, adaptively secure public-key encryption is fairly simple and efficient [3] if erasure is assumed, but much more complicated (and much less efficient) [11,2,18,16] without this assumption. Similarly, adaptively secure two-party computation is much easier with the assumption of secure erasure [30] than without [14].

Designing protocols without the assumption of secure erasure is difficult, in part, due to the need to deal with *post-execution corruption* (PEC), whereby an adversary can corrupt parties (and hence obtain the randomness they used) even after execution of the protocol has concluded. Handling PEC is inherent to the setting of universal composability (UC) [9], and is important for ensuring sequential composition even in the stand-alone setting [8]. If secure erasure is assumed, the definition of adaptive security does not change whether or not PEC is allowed [10], but without erasure the requirement of dealing with PEC adds significant additional complications.

Prior Work. We are interested in adaptive security, with PEC, in a model where secure erasure is not assumed. Some prior protocols for secure computation in this setting (e.g., [11,2]) assume a majority of the parties remain uncorrupted. Other work [28,27,22,25], including concurrent work of [19], allows *all but one* of the parties to be corrupted. While it may seem strange to worry about corruption of *all* parties, consideration of this case is important when a protocol Π_{outer} invokes some protocol Π_{inner} (not involving all parties running Π_{outer}) as a subroutine. In this case, all parties running Π_{inner} may eventually be corrupted, and security of Π_{outer} should still be guaranteed.

To the best of our knowledge, all prior work giving adaptively secure protocols for general functionalities (without erasure), and tolerating an arbitrary number of corruptions, are based on the Goldreich-Micali-Wigderson [23] paradigm for semi-honest computation, and thus have round complexity linear in the depth of the circuit being computed. These include protocols in the common reference string model [14], the "sunspots" model [15], the key-registration model [1], and, more generally, based on adaptively secure UC puzzles [17]. In addition, all prior work in this setting handles only "adaptively well-formed functionalities" (see [14] for a definition).

1.1 Our Result

We show a constant-round, universally composable protocol for multiparty computation of arbitrary functionalities, with security against a malicious, adaptive adversary corrupting any number of parties. We highlight that our protocol can be used to securely compute *all* functionalities, not just adaptively well-formed ones. Our protocol relies on indistinguishability obfuscation, and assumes a common reference string.

Overview of Our Techniques. The main difficulty in our setting is to construct a constant-round protocol with security against a *semi-honest*, adaptive

adversary corrupting any number of parties. Given any such protocol, we can compile it as in [14] to obtain a universally composable protocol with security against a *malicious*, adaptive adversary, and still running in constant rounds. We may also assume secure channels, which can be implemented using adaptively secure encryption.

Our protocol in the semi-honest setting relies on a common reference string (CRS). While it would be more elegant to avoid this assumption, a CRS—or some other form of setup—is anyway needed [12] in order to obtain universally composable computation in the presence of *malicious* adversaries corrupting half or more of the parties, even in a static-corruption model. Thus, as far as our final result (i.e., our protocol with security in the malicious setting) is concerned, some form of setup is unavoidable. Moreover, results of Garg and Sahai [22] indicate that a CRS (or some other form of setup) is needed to obtain constant-round, universally composable, multiparty protocols with adaptive security even in the semi-honest case; see further discussion below.

At its core, our protocol relies on the ability to make arbitrary algorithms *explainable*, an idea we explain in more detail now. Fix some randomized algorithm Alg. Informally, an *explainable* version of Alg is an algorithm $\widetilde{\mathsf{Alg}}$ along with an associated *explain* algorithm $\mathsf{Explain}$ such that, for any input, (1) the distributions over the outputs of $\mathsf{Alg}(\mathsf{input})$ and $\widetilde{\mathsf{Alg}}(\mathsf{input})$ are statistically close, and (2) choosing random coins r, computing $\mathsf{output} := \widetilde{\mathsf{Alg}}(\mathsf{input}; r)$, and outputting (output, r) is computationally indistinguishable from choosing random coins r, computing $\mathsf{output} := \widetilde{\mathsf{Alg}}(\mathsf{input}; r)$, and then outputting $(\mathsf{output}, \mathsf{Explain}(\mathsf{input}, \mathsf{output}))$. That is, the $\mathsf{Explain}$ algorithm provides the ability to sample random coins for $\widetilde{\mathsf{Alg}}$ that "explain" any given input/output pair. (A related notion was considered by Ishai et al. [26], though without any construction being given.)

Sahai and Waters [31] introduced the notion of explainability for the specific case of public-key encryption schemes, in the context of constructing a deniable encryption scheme. We observe that their techniques can be suitably generalized to give an explainable version of *arbitrary* algorithms based on indistinguishability obfuscation for general circuits (and one-way functions). We refer the reader to Section 3 for a formal statement of this result.

Let C be a circuit taking n-bit inputs.[1] Consider the following functionality $\mathsf{NextMsg}$ that (essentially) computes the next-message function for a two-round secure-computation protocol for C based on garbled circuits: $\mathsf{NextMsg}$ takes as input a sequence of first-round messages $\mathsf{OT}_{1,1}, \ldots, \mathsf{OT}_{1,n}$ for a two-round, adaptively secure, oblivious-transfer (OT) protocol (e.g., the protocol of [14]); it then (1) computes a garbled circuit GC corresponding to C, along with input-wire labels $\{(y_{i,0}, y_{i,1})\}_{i=1}^{n}$, and (2) computes a sequence of OT responses $\mathsf{OT}_{2,1}, \ldots, \mathsf{OT}_{2,n}$. (These responses allow the party that generated $\mathsf{OT}_{1,i}$ using input bit b to recover $y_{i,b}$ while learning nothing about $y_{i,1-b}$.) The output of $\mathsf{NextMsg}$ is $(\mathsf{GC}, \mathsf{OT}_{2,1}, \ldots, \mathsf{OT}_{2,n})$. The CRS for our protocol will be $\widetilde{\mathsf{NextMsg}}$,

[1] We assume for simplicity here that C is deterministic. Randomized functionalities are handled in Section 4.

an explainable version of $\mathsf{NextMsg}$.[2] We note that, in contrast to [31], in the real-world execution no parties have access to the $\mathsf{Explain}$ algorithm corresponding to $\widetilde{\mathsf{NextMsg}}$.

Our multiparty protocol computing C can now be described quite simply. The protocol proceeds in four rounds. Say we have n parties $P_1, \ldots, P_n$ holding inputs $x_1, \ldots, x_n$, respectively. These parties generate first-round OT messages $\mathsf{OT}_{1,1}, \ldots, \mathsf{OT}_{1,n}$ (with the party who is supposed to provide the ith input generating $\mathsf{OT}_{1,i}$), and send these to P_n. Party P_n then runs $\widetilde{\mathsf{NextMsg}}(\mathsf{OT}_{1,1}, \ldots, \mathsf{OT}_{1,n})$ to obtain $\mathsf{GC}, \mathsf{OT}_{2,1}, \ldots, \mathsf{OT}_{2,n}$, and sends $\mathsf{OT}_{2,i}$ to the corresponding party (which might be itself). Each party P_i then locally recovers y_i, the label for the ith input wire of the garbled circuit, and sends y_i to P_n. Finally, P_n evaluates the garbled circuit GC using the provided input-wire labels to obtain the output z, and sends z to all the other parties.[3] Only the third- and fourth-round messages need to be sent via a secure channel.

We now describe the simulator informally. Our simulator begins by generating $\widetilde{\mathsf{NextMsg}}$ along with its associated $\mathsf{Explain}$ algorithm, and letting $\widetilde{\mathsf{NextMsg}}$ be the CRS. It simulates $\mathsf{OT}_{1,1}, \ldots, \mathsf{OT}_{1,n}$ and $\mathsf{OT}_{2,1}, \ldots, \mathsf{OT}_{2,n}$ using the simulator for the OT protocol (recall the OT protocol is adaptively secure), and uses these for the first two rounds of the protocol. Upon corruption of party P_i, the simulator corrupts that party in the ideal world and learns its input x_i and the output z. Then:

- If this is the first corruption, the simulator generates a simulated garbled circuit GC consistent with output z, along with n input-wire labels $y_1, \ldots, y_n$. It also uses the $\mathsf{Explain}$ algorithm to generate random coins r^* consistent with running $\widetilde{\mathsf{NextMsg}}$ on input $\mathsf{OT}_{1,1}, \ldots, \mathsf{OT}_{1,n}$ and obtaining output $\mathsf{GC}, \mathsf{OT}_{2,1}, \ldots, \mathsf{OT}_{2,n}$.
- The simulator uses the simulator for the OT protocol to generate internal state for P_i consistent with input x_i and output y_i, and returns this to the adversary. In addition, if $P = P_n$ then it returns r^* to the adversary.

Notably, our simulator is "corruption oblivious" [4]. Roughly, this means the behavior of the simulator upon corruption of a party is independent of the ideal state learned previously. By [4, Theorem 1.2], this means our protocol is also the first *leakage-tolerant* protocol with arbitrary leakage for general functionalities under semi-honest corruption. Moreover, by [5, Theorem 1], we also obtain the first construction of a *2-component OCL compiler* (see [5] for a definition).

Impossibility Results? We briefly mention two impossibility results regarding (constant-round) adaptively secure computation, and explain why they do not apply in our setting.

[2] As described, the CRS depends on the circuit C. However, by taking C to be a universal circuit, the CRS can be fixed independently of the "actual" function f the parties wish to compute (other than the size of a circuit for f).

[3] As described, all parties learn the output of the computation. Standard techniques can be used to handle the general case in which each party learns a possibly different function of the inputs.

First, our protocol can compute *arbitrary* randomized functionalities, not just *adaptively well-formed* ones. (We refer to [14] for a definition of this term.) This may seem somewhat surprising in light of an impossibility result of Ishai et al. [26] showing that adaptively secure computation of functionalities that are not adaptively well-formed is impossible. A closer examination of their result, however, reveals that it does *not* hold in the CRS model.

Second, Garg and Sahai [22] show that no constant-round, adaptively secure, multiparty protocol can be proven secure using black-box techniques; although they only claim this result for protocols with security against malicious adversaries, their proof appears to extend to the case of semi-honest adversaries as well. Again, however, their impossibility result only applies to the "plain" model with no setup, whereas we assume a CRS.

Concurrent Work. Independent of our work, two other groups of researchers have also studied the problem of constant-round adaptively secure computation. Canetti et al. [13] give a protocol that is similar in spirit to ours, but works only for the *two-party* case and requires sub-exponentially hard indistinguishability obfuscation. Garg and Polychroniadou [21], though also relying on indistinguishability obfuscation, follow a different approach. They give a *round-optimal*, adaptively secure protocol for the multiparty setting. We remark that both these other works only consider adaptively well-formed functionalities.

1.2 Organization of the Paper

We review some standard cryptographic background and primitives in Section 2. In Section 3, we introduce the notion of an *explainable* algorithm, and show how the Sahai-Waters compiler [31] can be used to make any algorithm explainable. Finally, in Section 4 we present a constant-round multiparty computation protocol tolerating a semi-honest, adaptive adversary corrupting any number of parties. Applying the compiler of Canetti et al. [14] yields a constant-round protocol tolerating a *malicious*, adaptive adversary corrupting any number of parties.

2 Preliminaries

We let λ denote the security parameter. We refer to previous work [8,10,30] for definitions of secure computation in the adaptive-corruption setting (with PEC).

2.1 Garbled Circuits

We rely on the standard notion of garbled circuits [32]. However, we use slightly non-standard notation that we introduce here. Let C be a randomized circuit taking n-bit inputs and using λ bits of randomness. We abstract the construction/evaluation of a garbled circuit for C via algorithms $\mathsf{GenGC}, \mathsf{EvalGC}$ with the following properties. GenGC is a randomized algorithm that takes as input 1^λ and C, and outputs a garbled circuit GC along with $2n$ input-wire labels

$y_{1,0}, y_{1,1}, \ldots, y_{n,0}, y_{n,1} \in \{0,1\}^\lambda$ and 2λ random-wire labels $w_{1,0}, w_{1,1}, \ldots, w_{\lambda,0}$, $w_{\lambda,1} \in \{0,1\}^\lambda$. Deterministic algorithm EvalGC takes as input GC and $n+\lambda$ labels $y_1, \ldots, y_n, w_1, \ldots, w_\lambda$, and outputs a value z.

Correctness requires that for any GC, $\left(\{y_{i,0}, y_{i,1}\}_{i=1}^n, \{w_{i,0}, w_{i,1}\}_{i=1}^\lambda\right)$ output by $\mathsf{GenGC}(1^\lambda, C)$, any $x \in \{0,1\}^n$ and any $r \in \{0,1\}^\lambda$, we have

$$\mathsf{EvalGC}\left(\mathsf{GC}, \{y_{i,x_i}\}_{i=1}^n, \{w_{i,r_i}\}_{i=1}^\lambda\right) = C(x;r).$$

Security requires an efficient simulator SimGC such that for all x, r, the distribution

$$\left\{\left(\mathsf{GC}, \{(y_{i,0}, y_{i,1})\}_{i=1}^n, \{(w_{i,0}, w_{i,1})\}_{i=1}^\lambda\right) \leftarrow \mathsf{GenGC}(1^\lambda, C) : \left(\mathsf{GC}, \{y_{i,x_i}\}_{i=1}^n, \{w_{i,r_i}\}_{i=1}^\lambda\right)\right\}$$

is computationally indistinguishable from the output of $\mathsf{SimGC}(1^\lambda, C, C(x;r))$.

2.2 Adaptively Secure Oblivious Transfer

Our protocol uses a two-round, semi-honest, adaptively secure OT protocol as a building block. A suitable construction can be found in [14].

A two-round OT protocol Π_{OT} comprises three algorithms: a receiver algorithm R_{OT}, a sender algorithm S_{OT}, and an evaluation algorithm E_{OT}. Algorithm R_{OT} takes as input a bit b and random coins r_R, and outputs initial message OT_1. Algorithm S_{OT} takes as input an initial message OT_1, a pair of λ-bit strings (y_0, y_1), and randomness r_S, and outputs message OT_2. The evaluation algorithm E_{OT} takes as input b, r_R, and OT_2 and outputs the λ-bit string y_b.

For our purposes we require the following property that is implied by semi-honest, adaptive security of Π_{OT}. There is exist an efficient simulator $\mathsf{SimOT} = (\mathsf{SimOT}_1, \mathsf{SimOT}_2)$, where SimOT_2 is deterministic, such that (1) SimOT_1 outputs a transcript $(\mathsf{OT}_1, \mathsf{OT}_2)$ along with state st and (2) SimOT_2, given as input b, y, and st, outputs coins r_R for the receiver consistent with $(\mathsf{OT}_1, \mathsf{OT}_2)$ and the receiver holding input b and obtaining output y; for any b, y_0, y_1, the distribution

$$\left\{r_R, r_S \leftarrow \{0,1\}^*; \mathsf{OT}_1 := R_{\mathsf{OT}}(b; r_R) : \left(r_R,\ \mathsf{OT}_1,\ S_{\mathsf{OT}}(\mathsf{OT}_1, y_0, y_1; r_S)\right)\right\}$$

is computationally indistinguishable from

$$\left\{\begin{array}{c}(\mathsf{OT}_1, \mathsf{OT}_2, \mathsf{st}) \leftarrow \mathsf{SimOT}_1(1^\lambda); \\ r_R := \mathsf{SimOT}_2(1^\lambda, b, y_b, \mathsf{st})\end{array} : \ (r_R, \mathsf{OT}_1, \mathsf{OT}_2)\right\}.$$

That is, we only require "one-sided security" [25] for adaptive corruption of the receiver.

If we define algorithm $\mathsf{SimOT}'_1(1^\lambda)$ to run $\mathsf{SimOT}_1(1^\lambda)$ and output only $(\mathsf{OT}_1, \mathsf{st})$, and define the algorithm $\mathsf{SimOT}'_2(1^\lambda, b, \mathsf{st})$ to simply run $\mathsf{SimOT}_2(1^\lambda, b, 0^\lambda, \mathsf{st})$, then for any b the distribution $\left\{r_R \leftarrow \{0,1\}^* : \left(r_R,\ R_{\mathsf{OT}}(b; r_R)\right)\right\}$ is computationally indistinguishable from

$$\left\{\begin{array}{c}(\mathsf{OT}_1, \mathsf{st}) \leftarrow \mathsf{SimOT}'_1(1^\lambda); \\ r_R := \mathsf{SimOT}'_2(1^\lambda, b, \mathsf{st})\end{array} : \ (r_R, \mathsf{OT}_1)\right\}.$$

2.3 Indistinguishability Obfuscation

We use an indistinguishability obfuscator as a building block. A PPT machine $\mathsf{i}\mathcal{O}$ is an *indistinguishability obfuscator* for a circuit class $\{\mathcal{C}_\lambda\}$ if the following conditions are satisfied:

Correctness. For all λ, and all $C \in \mathcal{C}_\lambda$, it holds that C and $\mathsf{i}\mathcal{O}(1^\lambda, C)$ compute the same function.

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

Indistinguishability. For any sequence $\{(C_{\lambda,0}, C_{\lambda,1}, \mathsf{aux}_\lambda)\}_\lambda$ where $C_{\lambda,0}, C_{\lambda,1} \in \mathcal{C}_\lambda$, $C_{\lambda,0} \equiv C_{\lambda,1}$, and $|C_{\lambda,0}| = |C_{\lambda,1}|$, and any PPT distinguisher D, there is a negligible function negl such that:

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

When clear from the context, we will often omit the security parameter 1^λ as an input to $\mathsf{i}\mathcal{O}$ and as a subscript for C.

$\mathsf{i}\mathcal{O}$ is an *indistinguishability obfuscator* for $\mathsf{P/poly}$ if there is a polynomial p such that $\mathsf{i}\mathcal{O}$ is an indistinguishability obfuscator for $\{\mathcal{C}_\lambda\}$, where $\mathcal{C}_\lambda$ contains all circuits of size at most $p(\lambda)$. Garg et al. [20] have shown the first candidate construction of indistinguishability obfuscators for $\mathsf{P/poly}$.

3 Explainability Compilers

Sahai and Waters [31] define a notion of *explainability* for public-key encryption, and show a compiler that transforms any public-key encryption scheme into an explainable version. Here, we generalize the notion of explainability for an *arbitrary* algorithm Alg, and show that the Sahai-Waters compiler can be used to transform any algorithm Alg into an explainable version $\widetilde{\mathsf{Alg}}$.

At a high level, an explainability compiler takes as input (a description of) a randomized algorithm Alg, and outputs two algorithms $\widetilde{\mathsf{Alg}}, \mathsf{Explain}$. The first of these is a randomized algorithm computing the same functionality as Alg. The second algorithm, roughly speaking, takes an input/output pair $\mathsf{input}, \mathsf{output}$ and produces random coins r consistent with running $\widetilde{\mathsf{Alg}}(\mathsf{input})$ and obtaining the result output. That is, the algorithm "explains" the input/output pair $\mathsf{input}, \mathsf{output}$. We now give a formal definition.

Definition 1. *A* PPT *algorithm* Comp *is an* explainability compiler *if for every efficient, randomized circuit* Alg*, the following hold:*

Polynomial slowdown. *There is a polynomial* $p(\cdot)$ *such that, for any* $(\widetilde{\mathsf{Alg}}, \mathsf{Explain})$ *output by* $\mathsf{Comp}(1^\lambda, \mathsf{Alg})$ *it holds that* $|\widetilde{\mathsf{Alg}}| \leq p(\lambda) \cdot |\mathsf{Alg}|$.

Statistical functional equivalence. *With overwhelming probability over choice of* $(\widetilde{\mathsf{Alg}}, \star)$ *as output by* $\mathsf{Comp}(1^\lambda, \mathsf{Alg})$*, the distribution of* $\widetilde{\mathsf{Alg}}(\mathsf{input})$ *is statistically close to the distribution of* $\mathsf{Alg}(\mathsf{input})$ *for all* input.

$\overline{\mathsf{Alg}}$

Hardwired constants: Keys K_1, K_2, and K_3.
Input: Input input and randomness $u = (u[1], u[2])$.

1. Let $\mathsf{input}', \mathsf{output}', r' := F_3(K_3, u[1]) \oplus u[2]$. If it is the case that $\mathsf{input} = \mathsf{input}'$ and $u[1] = F_2(K_2, (\mathsf{input}', \mathsf{output}', r'))$, then output $\mathsf{output} := \mathsf{output}'$ and end.
2. Else let $x := F_1(K_1, (\mathsf{input}, u))$ and output $\mathsf{output} := \mathsf{Alg}(\mathsf{input}; x)$.

Fig. 1. Program $\overline{\mathsf{Alg}}$

Explainability. *The success probability of every non-uniform, polynomial-time adversary $\mathcal{A}$ in the following experiment is negligibly close to $1/2$:*

1. $\mathcal{A}(1^\lambda)$ *outputs* input^* *of its choice.*
2. $\mathsf{Comp}(1^\lambda, \mathsf{Alg})$ *is run to obtain* $(\widetilde{\mathsf{Alg}}, \mathsf{Explain})$.
3. *Choose uniform coins* $r_0 \in \{0,1\}^*$ *and compute* $\mathsf{output}^* := \widetilde{\mathsf{Alg}}(\mathsf{input}^*; r_0)$.
4. *Compute* $r_1 \leftarrow \mathsf{Explain}(\mathsf{input}^*, \mathsf{output}^*)$.
5. *Choose a uniform bit b and give* $\widetilde{\mathsf{Alg}}, \mathsf{output}^*, r_b$ *to* $\mathcal{A}$.
6. $\mathcal{A}$ *outputs a bit b', and* $\mathsf{succeeds}$ *if* $b' = b$.

We highlight one key difference between our definition and the corresponding one from [31]: in our case input^* is an arbitrary length value (depending on the domain of Alg) chosen by the adversary, whereas in [31] the input to the explainable algorithm is a *single bit* chosen uniformly (and given to the adversary). Because of this, and due to the way the explainability compiler is constructed, we require the adversary to choose input^* "non-adaptively," i.e., before being given $\widetilde{\mathsf{Alg}}$. This definition of explainability suffices for our eventual protocol.

3.1 Constructing an Explainability Compiler

Following [31], we now show how to construct an explainability compiler. As in [31], we rely on an indistinguishability obfuscator, $\mathsf{i}\mathcal{O}$, for $\mathsf{P/poly}$ and three different pseudorandom function (PRF) variants (cf. Appendix A):

- A *puncturable, extracting* PRF $F_1(K_1, \cdot)$ that accepts inputs of length $\ell_1 + \ell_2 + \ell_{\mathsf{in}}$, and outputs strings of length ℓ_r. It is extracting when the input min-entropy is greater than $\ell_r + 2\lambda + 4$, with statistical closeness less than $2^{-(\lambda+1)}$. Observe that $\ell_{\mathsf{in}} + \ell_1 + \ell_2 \geq \ell_r + 2\lambda + 4$, and thus if one-way functions exist then such a PRF exists by Theorem 4.
- A *puncturable, statistically injective* PRF $F_2(K_2, \cdot)$ that accepts inputs of length $2\lambda + \ell_{\mathsf{in}} + \ell_{\mathsf{out}}$, and outputs strings of length ℓ_1. Observe that $\ell_1 \geq 2 \cdot (2\lambda + \ell_{\mathsf{in}} + \ell_{\mathsf{out}}) + \lambda$, and thus if one-way functions exist then such a PRF exists by Theorem 3.
- A *puncturable* PRF $F_3(K_3, \cdot)$ that accepts inputs of length ℓ_1 and outputs strings of length ℓ_2. If one-way functions exist, then such a PRF exists by Theorem 2.

We define $\mathsf{Comp}(1^\lambda, \mathsf{Alg})$ as follows. Let $\mathsf{Alg} : \{0,1\}^{\ell_{\mathsf{in}}} \times \{0,1\}^{\ell_r} \to \{0,1\}^{\ell_{\mathsf{out}}}$ be an algorithm with domain $\{0,1\}^{\ell_{\mathsf{in}}}$, range $\{0,1\}^{\ell_{\mathsf{out}}}$, and randomness length ℓ_r. Our compiled program $\widetilde{\mathsf{Alg}}$ will take input $\mathsf{input} \in \{0,1\}^{\ell_{\mathsf{in}}}$ and randomness $u = (u[1], u[2])$ of length $\ell_1 + \ell_2$, where $|u[1]| = \ell_1 = 5\lambda + 2(\ell_{\mathsf{in}} + \ell_{\mathsf{out}}) + \ell_r$ and $|u[2]| = \ell_2 = 2\lambda + \ell_{\mathsf{in}} + \ell_{\mathsf{out}}$. Our compiler first samples keys K_1, K_2, and K_3 for PRFs F_1, F_2, and F_3, respectively. It then defines algorithms $\overline{\mathsf{Alg}}$ and $\overline{\mathsf{Explain}}$ as in Figures 1 and 2, respectively. Finally, it computes $\widetilde{\mathsf{Alg}} \leftarrow i\mathcal{O}(\overline{\mathsf{Alg}})$ and $\mathsf{Explain} \leftarrow i\mathcal{O}(\overline{\mathsf{Explain}})$, and outputs $(\widetilde{\mathsf{Alg}}, \mathsf{Explain})$.

The proofs of security for our compiler, given for completeness in Appendix B, follow closely along the lines of the analogous proofs in [31]. Specifically, the proof of statistical functional equivalence closely follows the proof used by Sahai and Waters to establish IND-CPA security of their deniable encryption scheme, and the proof of explainability follows the Sahai-Waters proof establishing explainability of their deniable encryption scheme. We highlight, however, that in our proof of explainability a difference arises because in our case the input input^* is an arbitrary length value (depending on the domain of Alg), whereas in [31] the input is just a single bit. We are able to adapt the proof to this case because we do not allow input^* to depend on $\widetilde{\mathsf{Alg}}$.

4 A Semi-Honest, Adaptively Secure Protocol

We describe here a protocol for secure computation of a randomized circuit C by a set of parties $P_1, \ldots, P_n$. We assume for simplicity that all parties learn the output of C; using standard techniques, we can handle the general case in which each party learns a possibly different function of the inputs. For ease of notation, we assume that the domain of C is $\{0,1\}^n$ with party P_i providing the ith input $x_i \in \{0,1\}$. (One can easily verify that our protocol and proof generalize to the case of arbitrary-length inputs.) We also assume without loss of generality that C uses λ random bits.

The CRS of our protocol is an "explainable" version $\widetilde{\mathsf{NextMsg}}$ of the algorithm $\mathsf{NextMsg}$ defined in Figure 3. That is, the CRS is generated by computing $(\widetilde{\mathsf{NextMsg}}, \mathsf{Explain}) \leftarrow \mathsf{Comp}(1^\lambda, \mathsf{NextMsg})$ and letting the CRS be $\widetilde{\mathsf{NextMsg}}$. As described, the CRS depends on C (since $\mathsf{NextMsg}$ does); however, by letting C

$\overline{\mathsf{Explain}}$

Hardwired constants: Keys K_2 and K_3.
Input: input, output, and randomness $r \in \{0,1\}^\lambda$.

1. Set $\alpha := F_2(K_2, (\mathsf{input}, \mathsf{output}, \mathsf{PRG}(r)))$ and let $\beta := F_3(K_3, \alpha) \oplus (\mathsf{input}, \mathsf{output}, \mathsf{PRG}(r))$.
   Output (α, β).

Fig. 2. Program $\overline{\mathsf{Explain}}$

NextMsg

Inputs: $\mathsf{OT}_{1,1}, \ldots, \mathsf{OT}_{1,n}$; randomness $r_1, \ldots, r_\lambda \in \{0,1\}$ and $r_{\mathsf{GC}}, r_{S,1}, \ldots, r_{S,n} \in \{0,1\}^*$.

1. Run $\mathsf{GenGC}(1^\lambda, C; r_{\mathsf{GC}})$ to produce the garbled circuit GC along with n pairs of input-wire labels $\{(y_{i,0}, y_{i,1})\}_{i=1}^n$ and λ pairs of random-wire labels $\{(w_{i,0}, w_{i,1})\}_{i=1}^\lambda$.
2. For $i \in [n]$, run S_{OT} on input $\mathsf{OT}_{1,i}$ and $(y_{i,0}, y_{i,1})$ using randomness $r_{S,i}$, to obtain $\mathsf{OT}_{2,i}$.
3. Output GC, OT messages $\{\mathsf{OT}_{2,i}\}_{i=1}^n$, and random-wire labels $w_{1,r_1}, \ldots, w_{\lambda,r_\lambda}$.

Fig. 3. Algorithm NextMsg. The security parameter 1^λ and circuit C are hardwired.

be a universal circuit the CRS can be fixed independently of the "actual" function the parties wish to compute. We note that we allow the environment $\mathcal{Z}$ to choose the parties' inputs depending on the CRS.

Let $\Pi_{\mathsf{OT}} = (R_{\mathsf{OT}}, S_{\mathsf{OT}}, E_{\mathsf{OT}})$ be a two-round, semi-honest, adaptively secure OT protocol (cf. Section 2.2). Our secure-computation protocol Π is defined in Figure 4. We describe the protocol assuming the existence of secure channels; these can be instantiated using any adaptively secure public-key encryption scheme.

Theorem 1. *Assume* Comp *is an explainability compiler, and* GenGC *and* Π_{OT} *satisfy the definitions from Sections 2.1 and 2.2, respectively. Then protocol* Π *in Figure 4 UC-realizes functionality* C *in the presence of a semi-honest, adaptive adversary corrupting any number of parties.*

Proof. Let SimGC, SimOT denote appropriate simulators as defined in Section 2. Fix an environment $\mathcal{Z}$ and a dummy adversary $\mathcal{A}$ attacking protocol Π. Recall that we allow the environment $\mathcal{Z}$ to adaptively choose the inputs of all parties *after* seeing the common reference string. Without loss of generality, we assume $\mathcal{Z}$ first observes the entire protocol transcript (which, since we use secure channels in rounds 3 and 4, consists only of the messages sent in the first two rounds) before corrupting any parties. Our simulator Sim for this adversary proceeds as follows:

1. Compute $(\widetilde{\mathsf{NextMsg}}, \mathsf{Explain}) \leftarrow \mathsf{Comp}(1^\lambda, \mathsf{NextMsg})$, and give $\widetilde{\mathsf{NextMsg}}$ to $\mathcal{Z}$ as the CRS.
2. Run $\mathsf{SimOT}_1(1^\lambda)$ a total of n times to obtain $\{(\mathsf{OT}_{1,i}, \mathsf{OT}_{2,i}, \mathsf{st}_i)\}_{i=1}^n$. Give $\mathsf{OT}_{1,1}, \ldots, \mathsf{OT}_{1,n-1}$ to $\mathcal{Z}$ as the first-round message, and $\mathsf{OT}_{2,1}, \ldots, \mathsf{OT}_{2,n-1}$ to $\mathcal{Z}$ as the second-round message.
3. When $\mathcal{Z}$ requests to corrupt party P_i, corrupt P_i in the ideal world to learn its input x_i and the output z. Then:
   - If this is the first party to be corrupted, compute $(\mathsf{GC}, \{y_i\}_{i=1}^n, \{w_i\}_{i=1}^\lambda) \leftarrow \mathsf{SimGC}(1^\lambda, C, z)$ and $r_n^* \leftarrow \mathsf{Explain}((\mathsf{OT}_{1,1}, \ldots, \mathsf{OT}_{1,n}), (\mathsf{GC}, \mathsf{OT}_{2,1}, \ldots,$

Semi-Honest, Adaptively Secure Multiparty Computation

Common input:

- $CRS = \widetilde{\mathsf{NextMsg}}$.
- Description of a randomized circuit C.

Private inputs: Every party P_i has private input $x_i \in \{0,1\}$.

Each P_i: *Compute first-round OT messages*:

- Sample random coins $r_{R,i} \leftarrow \{0,1\}^*$ of appropriate length.
- Compute $\mathsf{OT}_{1,i} := R_{\mathsf{OT}}(x_i; r_{R,i})$ and, for $i \in [n-1]$, send $\mathsf{OT}_{1,i}$ to P_n.

P_n: *Compute garbled circuit and second-round OT messages*:

- Sample random coins $r_n \leftarrow \{0,1\}^*$ of appropriate length.
- Compute

$$(\mathsf{GC}, \mathsf{OT}_{2,1}, \ldots, \mathsf{OT}_{2,n}, w_1, \ldots, w_\lambda) := \widetilde{\mathsf{NextMsg}}(\mathsf{OT}_{1,1}, \ldots, \mathsf{OT}_{1,n}; r_n).$$

- For $i \in [n-1]$, send $\mathsf{OT}_{2,i}$ to P_i.

Each P_i: *Recover OT output*:

- Compute $y_i := E_{\mathsf{OT}}(x_i, r_{R,i}, \mathsf{OT}_{2,i})$ and, for $i \in [n-1]$, send y_i to P_n over a secure channel.

P_n: *Evaluate garbled circuit and broadcast output*:

- Compute $z := \mathsf{EvalGC}(\mathsf{GC}, \{y_i\}_{i=1}^n, \{w_i\}_{i=1}^\lambda)$.
- For $i \in [n-1]$, send z to P_i over a secure channel.

Output: Each party P_i outputs z.

Fig. 4. Protocol Π for computing randomized circuit C.

$\mathsf{OT}_{2,n}, w_1, \ldots, w_n)$). Store these values to be used, as needed, in the rest of the simulation.
- In any case, compute $r_{R,i} := \mathsf{SimOT}_2(1^\lambda, x_i, y_i, \mathsf{st}_i)$ and give x_i, z, y_i, and $r_{R,i}$ to $\mathcal{Z}$. In addition, if $i = n$ give $\{y_i\}_{i=1}^{n-1}$ and r_n^* to $\mathcal{Z}$.

4. Output whatever $\mathcal{Z}$ outputs.

We prove that the output of $\mathcal{Z}$ when interacting with $\mathcal{A}$ and parties in a real-world execution of protocol Π is indistinguishable from the output of $\mathcal{Z}$ when interacting with Sim and the functionality C in an ideal-world execution of the protocol. We do so by considering a sequence of hybrid experiments, beginning with the real-world execution and ending with the ideal-world execution, and showing that each experiment is computationally indistinguishable from the preceding one.

Hybrid 0. This corresponds to the real-world execution of the protocol. We write the experiment in a format convenient for the proof. This experiment proceeds via the following steps:

1. Compute $(\widetilde{\mathsf{NextMsg}}, \mathsf{Explain}) \leftarrow \mathsf{Comp}(1^\lambda, \mathsf{NextMsg})$, and give $\widetilde{\mathsf{NextMsg}}$ to $\mathcal{Z}$ as the CRS. $\mathcal{Z}$ chooses inputs $x_1, \ldots, x_n$.
2. For $i \in [n]$, sample coins $r_{R,i}$ and compute $\mathsf{OT}_{1,i} := R_{\mathsf{OT}}(x_i; r_{R,i})$. Give the sequence of values $\mathsf{OT}_{1,1}, \ldots, \mathsf{OT}_{1,n-1}$ to $\mathcal{Z}$ as the first-round message.
3. Sample coins r_n and compute

$$(\mathsf{GC}, \mathsf{OT}_{2,1}, \ldots, \mathsf{OT}_{2,n}, w_1, \ldots, w_\lambda) := \widetilde{\mathsf{NextMsg}}(\mathsf{OT}_{1,1}, \ldots, \mathsf{OT}_{1,n}; r_n).$$

   Give $\mathsf{OT}_{2,1}, \ldots, \mathsf{OT}_{2,n-1}$ to $\mathcal{Z}$ as the second-round message.
4. When $\mathcal{Z}$ requests to corrupt party P_i, compute $y_i := E_{\mathsf{OT}}(x_i, r_{R,i}, \mathsf{OT}_{2,i})$ and give x_i, z, y_i, and $r_{R,i}$ to $\mathcal{Z}$. In addition, if $i = n$ then compute $y_i := E_{\mathsf{OT}}(x_i, r_{R,i}, \mathsf{OT}_{2,i})$ for $i \in [n-1]$ and give $\{y_i\}_{i=1}^{n-1}$ and r_n to $\mathcal{Z}$.

Hybrid 1. This experiment is similar to the previous one, except that the OT_1 messages and the random coins $\{r_{R,i}\}$ are generated by the simulator for the OT protocol (cf. Section 2.2). That is, the experiment proceeds via the following steps:

1. Compute $(\widetilde{\mathsf{NextMsg}}, \mathsf{Explain}) \leftarrow \mathsf{Comp}(1^\lambda, \mathsf{NextMsg})$, and give $\widetilde{\mathsf{NextMsg}}$ to $\mathcal{Z}$ as the CRS. $\mathcal{Z}$ chooses inputs $x_1, \ldots, x_n$.
2. Run $\mathsf{SimOT}'_1(1^\lambda)$ a total of n times to obtain $\{(\mathsf{OT}_{1,i}, \mathsf{st}_i)\}_{i=1}^n$. Give the sequence of values $\mathsf{OT}_{1,1}, \ldots, \mathsf{OT}_{1,n-1}$ to $\mathcal{Z}$ as the first-round message.
3. Sample coins r_n and compute

$$(\mathsf{GC}, \mathsf{OT}_{2,1}, \ldots, \mathsf{OT}_{2,n}, w_1, \ldots, w_\lambda) := \widetilde{\mathsf{NextMsg}}(\mathsf{OT}_{1,1}, \ldots, \mathsf{OT}_{1,n}; r_n).$$

   Give $\mathsf{OT}_{2,1}, \ldots, \mathsf{OT}_{2,n-1}$ to $\mathcal{Z}$ as the second-round message.
4. When $\mathcal{Z}$ corrupts party P_i, compute $r_{R,i} := \mathsf{SimOT}'_2(1^\lambda, x_i, \mathsf{st}_i)$ and $y_i := E_{\mathsf{OT}}(x_i, r_{R,i}, \mathsf{OT}_{2,i})$, and give x_i, z, y_i, and $r_{R,i}$ to $\mathcal{Z}$. In addition, if $i = n$ then for $i \in [n-1]$ compute $r_{R,i} := \mathsf{SimOT}'_2(1^\lambda, x_i, \mathsf{st}_i)$ and $y_i := E_{\mathsf{OT}}(x_i, r_{R,i}, \mathsf{OT}_{2,i})$, and give $\{y_i\}_{i=1}^{n-1}$ and r_n to $\mathcal{Z}$.

It follows immediately by security of the OT protocol (and a straightforward hybrid argument) that this experiment is computationally indistinguishable from the previous one.

Hybrid 2. This experiment is similar to the previous one, except that we now use the $\mathsf{Explain}$ algorithm to generate the random coins r_n. That is, the experiment proceeds as follow:

1. Compute $(\widetilde{\mathsf{NextMsg}}, \mathsf{Explain}) \leftarrow \mathsf{Comp}(1^\lambda, \mathsf{NextMsg})$, and give $\widetilde{\mathsf{NextMsg}}$ to $\mathcal{Z}$ as the CRS. $\mathcal{Z}$ chooses inputs $x_1, \ldots, x_n$.
2. Run $\mathsf{SimOT}'_1(1^\lambda)$ a total of n times to obtain $\{(\mathsf{OT}_{1,i}, \mathsf{st}_i)\}_{i=1}^n$. Give the sequence of values $\mathsf{OT}_{1,1}, \ldots, \mathsf{OT}_{1,n-1}$ to $\mathcal{Z}$ as the first-round message.
3. Sample coins r_n and compute

$$(\mathsf{GC}, \mathsf{OT}_{2,1}, \ldots, \mathsf{OT}_{2,n}, w_1, \ldots, w_\lambda) := \widetilde{\mathsf{NextMsg}}(\mathsf{OT}_{1,1}, \ldots, \mathsf{OT}_{1,n}; r_n).$$

   In addition, let $\mathsf{input}^* = (\mathsf{OT}_{1,1}, \ldots, \mathsf{OT}_{1,n})$ and $\mathsf{output}^* = (\mathsf{GC}, \mathsf{OT}_{2,1}, \ldots, \mathsf{OT}_{2,n}, w_1, \ldots, w_\lambda)$, and compute $r_n^* \leftarrow \mathsf{Explain}(\mathsf{input}^*, \mathsf{output}^*)$.
   Give $\mathsf{OT}_{2,1}, \ldots, \mathsf{OT}_{2,n-1}$ to $\mathcal{Z}$ as the second-round message.

4. When $\mathcal{Z}$ corrupts party P_i, compute $r_{R,i} := \mathsf{SimOT}'_2(1^\lambda, x_i, \mathsf{st}_i)$ and $y_i := E_{\mathsf{OT}}(x_i, r_{R,i}, \mathsf{OT}_{2,i})$, and give x_i, z, y_i, and $r_{R,i}$ to $\mathcal{Z}$. In addition, if $i = n$ then for $i \in [n-1]$ compute $r_{R,i} := \mathsf{SimOT}'_2(1^\lambda, x_i, \mathsf{st}_i)$ and $y_i := E_{\mathsf{OT}}(x_i, r_{R,i}, \mathsf{OT}_{2,i})$, and give $\{y_i\}_{i=1}^{n-1}$ and r_n^* to $\mathcal{Z}$.

Computationally indistinguishability of this experiment from the previous one follows from the definition of explainability (cf. Definition 1), and the fact that Comp is an explainability compiler. To see this, say there is an efficient adversary $\mathcal{Z}$ and a non-uniform, polynomial-time distinguisher D that distinguishes the outcome of Hybrid 1 from that of Hybrid 2. We show how to use this to construct an attacker $\mathcal{A}'$ violating explainability. $\mathcal{A}'$ works as follows: it runs $\mathsf{SimOT}'_1(1^\lambda)$ a total of n times to obtain $\{(\mathsf{OT}_{1,i}, \mathsf{st}_i)\}_{i=1}^n$, and outputs the value $\mathsf{input}^* = (\mathsf{OT}_{1,1}, \ldots, \mathsf{OT}_{1,n})$. Given $\widetilde{\mathsf{NextMsg}}, \mathsf{output}^*, r$ in response, where $\mathsf{output}^* = (\mathsf{GC}, \mathsf{OT}_{2,1}, \ldots, \mathsf{OT}_{2,n}, w_1, \ldots, w_\lambda)$, it then does:

1. Give $\widetilde{\mathsf{NextMsg}}$ to $\mathcal{Z}$ as the CRS. $\mathcal{Z}$ chooses inputs $x_1, \ldots, x_n$.
2. Give $\mathsf{OT}_{1,1}, \ldots, \mathsf{OT}_{1,n-1}$ to $\mathcal{Z}$ as the first-round message, and $\mathsf{OT}_{2,1}, \ldots, \mathsf{OT}_{2,n-1}$ to $\mathcal{Z}$ as the second-round message.
3. When $\mathcal{Z}$ corrupts party P_i, compute $r_{R,i} := \mathsf{SimOT}'_2(1^\lambda, x_i, \mathsf{st}_i)$ and $y_i := E_{\mathsf{OT}}(x_i, r_{R,i}, \mathsf{OT}_{2,i})$, and give x_i, z, y_i, and $r_{R,i}$ to $\mathcal{Z}$. In addition, if $i = n$ then for $i \in [n-1]$ compute $r_{R,i} := \mathsf{SimOT}'_2(1^\lambda, x_i, \mathsf{st}_i)$ and $y_i := E_{\mathsf{OT}}(x_i, r_{R,i}, \mathsf{OT}_{2,i})$, and give $\{y_i\}_{i=1}^{n-1}$ and r to $\mathcal{Z}$.

Finally, run D on the output of $\mathcal{Z}$ and output the result. It is easy to see that if the coins r are those used to run $\widetilde{\mathsf{NextMsg}}$, then the view of $\mathcal{Z}$ when run as a subroutine by $\mathcal{A}'$ corresponds to Hybrid 1, whereas if the coins r are those output by $\mathsf{Explain}$, then the view of $\mathcal{Z}$ when run as a subroutine by $\mathcal{A}'$ corresponds to Hybrid 2. Indistinguishability of the two experiments follows.

Hybrid 3. This is similar to the previous experiment, except that $\mathsf{NextMsg}$ and $\mathsf{Explain}$ are used in place of $\widetilde{\mathsf{NextMsg}}$. That is, the experiment proceeds as follows:

1. Compute $(\widetilde{\mathsf{NextMsg}}, \mathsf{Explain}) \leftarrow \mathsf{Comp}(1^\lambda, \mathsf{NextMsg})$, and give $\widetilde{\mathsf{NextMsg}}$ to $\mathcal{Z}$ as the CRS. $\mathcal{Z}$ chooses inputs $x_1, \ldots, x_n$.
2. Run $\mathsf{SimOT}'_1(1^\lambda)$ a total of n times to obtain $\{(\mathsf{OT}_{1,i}, \mathsf{st}_i)\}_{i=1}^n$. Give the sequence of values $\mathsf{OT}_{1,1}, \ldots, \mathsf{OT}_{1,n-1}$ to $\mathcal{Z}$ as the first-round message.
3. Compute

$$(\mathsf{GC}, \mathsf{OT}_{2,1}, \ldots, \mathsf{OT}_{2,n}, w_1, \ldots, w_\lambda) \leftarrow \mathsf{NextMsg}(\mathsf{OT}_{1,1}, \ldots, \mathsf{OT}_{1,n}).$$

   In addition, let $\mathsf{input}^* = (\mathsf{OT}_{1,1}, \ldots, \mathsf{OT}_{1,n})$ and $\mathsf{output}^* = (\mathsf{GC}, \mathsf{OT}_{2,1}, \ldots, \mathsf{OT}_{2,n}, w_1, \ldots, w_\lambda)$, and compute $r_n^* \leftarrow \mathsf{Explain}(\mathsf{input}^*, \mathsf{output}^*)$.
   Give $\mathsf{OT}_{2,1}, \ldots, \mathsf{OT}_{2,n-1}$ to $\mathcal{Z}$ as the second-round message.
4. When $\mathcal{Z}$ corrupts party P_i, compute $r_{R,i} := \mathsf{SimOT}'_2(1^\lambda, x_i, \mathsf{st}_i)$ and $y_i := E_{\mathsf{OT}}(x_i, r_{R,i}, \mathsf{OT}_{2,i})$, and give x_i, z, y_i, and $r_{R,i}$ to $\mathcal{Z}$. In addition, if $i = n$ then for $i \in [n-1]$ compute $r_{R,i} := \mathsf{SimOT}'_2(1^\lambda, x_i, \mathsf{st}_i)$ and $y_i := E_{\mathsf{OT}}(x_i, r_{R,i}, \mathsf{OT}_{2,i})$, and give $\{y_i\}_{i=1}^{n-1}$ and r_n^* to $\mathcal{Z}$.

Indistinguishability of this experiment from the previous one follows by statistical equivalence of $\mathsf{NextMsg}$ and $\widetilde{\mathsf{NextMsg}}$.

Hybrid 4. In this experiment, we first make explicit the steps of $\mathsf{NextMsg}$. (This is just a syntactic rewriting, and does not affect the experiment.) In addition, we now set $y_i = y_{i,x_i}$ instead of computing y_i using the OT-evaluation algorithm E_{OT}. This experiment proceeds as follows:

1. Compute $(\widetilde{\mathsf{NextMsg}}, \mathsf{Explain}) \leftarrow \mathsf{Comp}(1^\lambda, \mathsf{NextMsg})$, and give $\widetilde{\mathsf{NextMsg}}$ to $\mathcal{Z}$ as the CRS. $\mathcal{Z}$ chooses inputs $x_1, \ldots, x_n$.
2. Run $\mathsf{SimOT}'_1(1^\lambda)$ a total of n times to obtain $\{(\mathsf{OT}_{1,i}, \mathsf{st}_i)\}_{i=1}^n$. Give the sequence of values $\mathsf{OT}_{1,1}, \ldots, \mathsf{OT}_{1,n-1}$ to $\mathcal{Z}$ as the first-round message.
3. Compute $(\mathsf{GC}, \{(y_{i,0}, y_{i,1})\}_{i=1}^n, \{(w_{i,0}, w_{i,1})\}_{i=1}^\lambda) \leftarrow \mathsf{GenGC}(1^\lambda, C)$ and set $y_i = y_{i,x_i}$ for all i. For $i \in [n]$, run $\mathsf{OT}_{2,i} \leftarrow S_{\mathsf{OT}}(\mathsf{OT}_1, y_{i,0}, y_{i,1})$. Choose uniform $r_1, \ldots, r_\lambda \in \{0,1\}$, and let $\mathsf{input}^* = (\mathsf{OT}_{1,1}, \ldots, \mathsf{OT}_{1,n})$ and $\mathsf{output}^* = (\mathsf{GC}, \mathsf{OT}_{2,1}, \ldots, \mathsf{OT}_{2,n}, w_{r_1}, \ldots, w_{r_\lambda})$. Compute $r_n^* \leftarrow \mathsf{Explain}(\mathsf{input}^*, \mathsf{output}^*)$. Give $\mathsf{OT}_{2,1}, \ldots, \mathsf{OT}_{2,n-1}$ to $\mathcal{Z}$ as the second-round message.
4. When $\mathcal{Z}$ corrupts party P_i, compute $r_{R,i} := \mathsf{SimOT}'_2(1^\lambda, x_i, \mathsf{st}_i)$. Give x_i, z, y_i, and $r_{R,i}$ to $\mathcal{Z}$. In addition, if $i = n$ then give $\{y_i\}_{i=1}^{n-1}$ and r_n^* to $\mathcal{Z}$.

Computational indistinguishability of this experiment from the previous one follows from security of the OT protocol.

Hybrid 5. In the previous experiment the OT_2 messages were generated honestly as part of $\mathsf{NextMsg}$. Here, we have the OT simulator output them instead. That is, we now do:

1. Compute $(\widetilde{\mathsf{NextMsg}}, \mathsf{Explain}) \leftarrow \mathsf{Comp}(1^\lambda, \mathsf{NextMsg})$, and give $\widetilde{\mathsf{NextMsg}}$ to $\mathcal{Z}$ as the CRS. $\mathcal{Z}$ chooses inputs $x_1, \ldots, x_n$.
2. Run $\mathsf{SimOT}_1(1^\lambda)$ a total of n times to obtain $\{(\mathsf{OT}_{1,i}, \mathsf{OT}_{2,i}, \mathsf{st}_i)\}_{i=1}^n$. Give the sequence of values $\mathsf{OT}_{1,1}, \ldots, \mathsf{OT}_{1,n-1}$ to $\mathcal{Z}$ as the first-round message, and give $\mathsf{OT}_{2,1}, \ldots, \mathsf{OT}_{2,n-1}$ to $\mathcal{Z}$ as the second-round message.
3. Compute $(\mathsf{GC}, \{(y_{i,0}, y_{i,1})\}_{i=1}^n, \{(w_{i,0}, w_{i,1})\}_{i=1}^\lambda) \leftarrow \mathsf{GenGC}(1^\lambda, C)$ and set $y_i = y_{i,x_i}$ for all i. Choose uniform values $r_1, \ldots, r_\lambda \in \{0,1\}$, and let $\mathsf{input}^* = (\mathsf{OT}_{1,1}, \ldots, \mathsf{OT}_{1,n})$ and $\mathsf{output}^* = (\mathsf{GC}, \mathsf{OT}_{2,1}, \ldots, \mathsf{OT}_{2,n}, w_{r_1}, \ldots, w_{r_\lambda})$. Compute $r_n^* \leftarrow \mathsf{Explain}(\mathsf{input}^*, \mathsf{output}^*)$.
4. When $\mathcal{Z}$ corrupts party P_i, compute $r_{R,i} := \mathsf{SimOT}_2(1^\lambda, x_i, y_i, \mathsf{st}_i)$. Give x_i, z, y_i, and $r_{R,i}$ to $\mathcal{Z}$. In addition, if $i = n$ then give $\{y_i\}_{i=1}^{n-1}$ and r_n^* to $\mathcal{Z}$.

Again, computational indistinguishability between this experiment and the previous one follows by security of the OT protocol.

Hybrid 6. Here we use the garbled-circuit simulator (cf. Section 2.1) instead of the garbled-circuit generation algorithm. Thus, the experiment now proceeds as follows:

1. Compute $(\widetilde{\mathsf{NextMsg}}, \mathsf{Explain}) \leftarrow \mathsf{Comp}(1^\lambda, \mathsf{NextMsg})$, and give $\widetilde{\mathsf{NextMsg}}$ to $\mathcal{Z}$ as the CRS. $\mathcal{Z}$ chooses inputs $x_1, \ldots, x_n$.

2. Run $\mathsf{SimOT}_1(1^\lambda)$ a total of n times to obtain $\{(\mathsf{OT}_{1,i}, \mathsf{OT}_{2,i}, \mathsf{st}_i)\}_{i=1}^n$. Give $\mathsf{OT}_{1,1}, \ldots, \mathsf{OT}_{1,n-1}$ to $\mathcal{Z}$ as the first-round message, and $\mathsf{OT}_{2,1}, \ldots, \mathsf{OT}_{2,n-1}$ to $\mathcal{Z}$ as the second-round message.
3. Compute $(\mathsf{GC}, \{y_i\}_{i=1}^n, \{w_i\}_{i=1}^\lambda) \leftarrow \mathsf{SimGC}(1^\lambda, C, z)$. Let $\mathsf{input}^* = (\mathsf{OT}_{1,1}, \ldots, \mathsf{OT}_{1,n})$ and $\mathsf{output}^* = (\mathsf{GC}, \mathsf{OT}_{2,1}, \ldots, \mathsf{OT}_{2,n}, w_{r_1}, \ldots, w_{r_\lambda})$. Compute $r_n^* \leftarrow \mathsf{Explain}(\mathsf{input}^*, \mathsf{output}^*)$.
4. When $\mathcal{Z}$ corrupts party P_i, compute $r_{R,i} := \mathsf{SimOT}_2(1^\lambda, x_i, y_i, \mathsf{st}_i)$. Give x_i, z, y_i, and $r_{R,i}$ to $\mathcal{Z}$. In addition, if $i = n$ then for $i \in [n-1]$ give $\{y_i\}_{i=1}^{n-1}$ and r_n^* to $\mathcal{Z}$.

Computational indistinguishability between this experiment and the previous one follows from security of garbled circuits.

We conclude the proof by noting that Hybrid 6 is simply a syntactic rewriting of the ideal-world execution involving the simulator originally defined.

5 Conclusions and Open Questions

In this work we have shown the first constant-round, universally composable protocol tolerating a malicious, adaptive adversary that can corrupt any number of parties, in a setting where secure erasure is not assumed. In addition, we have shown the first adaptively secure protocol, regardless of round complexity, that can compute arbitrary functionalities (and not only adaptively well-formed ones) in the presence of any number of corruptions and without erasures.

Several interesting open questions remain. Although a CRS (or some other form of setup) is necessary if we wish to obtain a universally composable protocol with security against malicious adversaries corrupting an arbitrary number of parties, it is still possible that the CRS can be avoided in the semi-honest case, or in the stand-alone setting. Moreover, our protocol assumes that the CRS depends on the circuit C being computed or, if we let C be a universal circuit (cf. footnote 2), an a priori bound on the size of the circuit being computed. It would be interesting to see if this can be avoided.

References

1. Barak, B., Canetti, R., Nielsen, J.B., Pass, R.: Universally composable protocols with relaxed set-up assumptions. In: 45th Annual Symposium on Foundations of Computer Science (FOCS), pp. 186–195. IEEE (2004)
2. Beaver, D.: Plug and play encryption. In: Kaliski Jr., B.S. (ed.) CRYPTO 1997. LNCS, vol. 1294, pp. 75–89. Springer, Heidelberg (1997)
3. Beaver, D., Haber, S.: Cryptographic protocols provably secure against dynamic adversaries. In: Rueppel, R.A. (ed.) EUROCRYPT 1992. LNCS, vol. 658, pp. 307–323. Springer, Heidelberg (1993)
4. Bitansky, N., Canetti, R., Halevi, S.: Leakage-tolerant interactive protocols. In: Cramer, R. (ed.) TCC 2012. LNCS, vol. 7194, pp. 266–284. Springer, Heidelberg (2012)

5. Bitansky, N., Dachman-Soled, D., Lin, H.: Leakage-tolerant computation with input-independent preprocessing. In: Garay, J.A., Gennaro, R. (eds.) CRYPTO 2014, Part II. LNCS, vol. 8617, pp. 146–163. Springer, Heidelberg (2014)
6. Boneh, D., Waters, B.: Constrained pseudorandom functions and their applications. In: Sako, K., Sarkar, P. (eds.) ASIACRYPT 2013, Part II. LNCS, vol. 8270, pp. 280–300. Springer, Heidelberg (2013)
7. Boyle, E., Goldwasser, S., Ivan, I.: Functional signatures and pseudorandom functions. In: Krawczyk, H. (ed.) PKC 2014. LNCS, vol. 8383, pp. 501–519. Springer, Heidelberg (2014)
8. Canetti, R.: Security and composition of multiparty cryptographic protocols. Journal of Cryptology 13(1), 143–202 (2000)
9. Canetti, R.: Universally composable security: A new paradigm for cryptographic protocols. In: 42nd Annual Symposium on Foundations of Computer Science (FOCS), pp. 136–145. IEEE (2001), Full version at `http://eprint.iacr.org/2000/067/`
10. Canetti, R., Damgård, I., Dziembowski, S., Ishai, Y., Malkin, T.: Adaptive versus non-adaptive security of multi-party protocols. J. Crypto. 17(3), 153–207 (2004)
11. Canetti, R., Feige, U., Goldreich, O., Naor, M.: Adaptively secure multiparty computation. In: 28th Annual ACM Symposium on Theory of Computing (STOC), pp. 639–648. ACM Press (1996)
12. Canetti, R., Fischlin, M.: Universally composable commitments. In: Kilian, J. (ed.) CRYPTO 2001. LNCS, vol. 2139, pp. 19–40. Springer, Heidelberg (2001)
13. Canetti, R., Goldwasser, S., Poburinnaya, O.: Adaptively secure two-party computation from indistinguishability obfuscation. Cryptology ePrint Archive, Report 2014/845 (2014)
14. Canetti, R., Lindell, Y., Ostrovsky, R., Sahai, A.: Universally composable two-party and multi-party secure computation. In: 34th Annual ACM Symposium on Theory of Computing (STOC), pp. 494–503. ACM Press (2002), Full version available at `http://eprint.iacr.org/2002/140`
15. Canetti, R., Pass, R., Shelat, A.: Cryptography from sunspots: How to use an imperfect reference string. In: 48th Annual Symposium on Foundations of Computer Science (FOCS), pp. 249–259. IEEE (2007)
16. Choi, S.G., Dachman-Soled, D., Malkin, T., Wee, H.: Improved non-committing encryption with applications to adaptively secure protocols. In: Matsui, M. (ed.) ASIACRYPT 2009. LNCS, vol. 5912, pp. 287–302. Springer, Heidelberg (2009)
17. Dachman-Soled, D., Malkin, T., Raykova, M., Venkitasubramaniam, M.: Adaptive and concurrent secure computation from new adaptive, non-malleable commitments. In: Sako, K., Sarkar, P. (eds.) ASIACRYPT 2013, Part I. LNCS, vol. 8269, pp. 316–336. Springer, Heidelberg (2013)
18. Damgård, I.B., Nielsen, J.B.: Improved non-committing encryption schemes based on a general complexity assumption. In: Bellare, M. (ed.) CRYPTO 2000. LNCS, vol. 1880, pp. 432–450. Springer, Heidelberg (2000)
19. Damgård, I., Polychroniadou, A., Rao, V.: Secure UC constant round multi-party computation. Cryptology ePrint Archive, Report 2014/830 (2014)
20. Garg, S., Gentry, C., Halevi, S., Raykova, M., Sahai, A., Waters, B.: Candidate indistinguishability obfuscation and functional encryption for all circuits. In: 54th Annual Symposium on Foundations of Computer Science (FOCS), pp. 40–49. IEEE (2013)
21. Garg, S., Polychroniadou, A.: Two-round adaptively secure mpc from indistinguishability obfuscation. Cryptology ePrint Archive, Report 2014/844 (2014)

22. Garg, S., Sahai, A.: Adaptively secure multi-party computation with dishonest majority. In: Safavi-Naini, R., Canetti, R. (eds.) CRYPTO 2012. LNCS, vol. 7417, pp. 105–123. Springer, Heidelberg (2012)
23. Goldreich, O., Micali, S., Wigderson, A.: How to play any mental game, or a completeness theorem for protocols with honest majority. In: 19th Annual ACM Symposium on Theory of Computing (STOC), pp. 218–229. ACM Press (1987)
24. Goldreich, O., Goldwasser, S., Micali, S.: On the cryptographic applications of random functions. In: Blakely, G.R., Chaum, D. (eds.) CRYPTO 1984. LNCS, vol. 196, pp. 276–288. Springer, Heidelberg (1985)
25. Hazay, C., Patra, A.: One-sided adaptively secure two-party computation. In: Lindell, Y. (ed.) TCC 2014. LNCS, vol. 8349, pp. 368–393. Springer, Heidelberg (2014)
26. Ishai, Y., Kumarasubramanian, A., Orlandi, C., Sahai, A.: On invertible sampling and adaptive security. In: Abe, M. (ed.) ASIACRYPT 2010. LNCS, vol. 6477, pp. 466–482. Springer, Heidelberg (2010)
27. Ishai, Y., Prabhakaran, M., Sahai, A.: Founding cryptography on oblivious transfer – efficiently. In: Wagner, D. (ed.) CRYPTO 2008. LNCS, vol. 5157, pp. 572–591. Springer, Heidelberg (2008)
28. Katz, J., Ostrovsky, R.: Round-optimal secure two-party computation. In: Franklin, M. (ed.) CRYPTO 2004. LNCS, vol. 3152, pp. 335–354. Springer, Heidelberg (2004)
29. Kiayias, A., Papadopoulos, S., Triandopoulos, N., Zacharias, T.: Delegatable pseudorandom functions and applications. In: 20th ACM Conf. on Computer and Communications Security (CCS), pp. 669–684. ACM Press (2013)
30. Lindell, A.Y.: Adaptively secure two-party computation with erasures. In: Fischlin, M. (ed.) CT-RSA 2009. LNCS, vol. 5473, pp. 117–132. Springer, Heidelberg (2009)
31. Sahai, A., Waters, B.: How to use indistinguishability obfuscation: Deniable encryption, and more. In: 46th Annual ACM Symposium on Theory of Com- puting (STOC), pp. 475–484. ACM Press (2014)
32. Yao, A.C.-C.: How to generate and exchange secrets. In: 27th Annual Sympo- sium on Foundations of Computer Science (FOCS), pp. 162–167. IEEE (1986)

A Puncturable PRFs

Puncturable PRFs are a type of constrained PRF [6,7,29] whereby it is possible to generate a key that defines the function everywhere except on some polynomial-size set of inputs:

Definition 2. *A* puncturable family of PRFs *is defined by polynomials* $n(\cdot)$ *and* $m(\cdot)$ *and a triple of Turing machines* Key_F, $\mathsf{Puncture}_F$, *and* Eval_F *satisfying the following conditions:*

Functionality preserved under puncturing. *For all polynomial-size sets* $S \subseteq \{0,1\}^{n(\lambda)}$ *and all* $x \in \{0,1\}^{n(\lambda)} \setminus S$, *we have:*

$$\Pr\left[K \leftarrow \mathsf{Key}_F(1^\lambda); K_S = \mathsf{Puncture}_F(K,S) : \mathsf{Eval}_F(K,x) = \mathsf{Eval}_F(K_S,x)\right] = 1.$$

Pseudorandom at punctured points. *For every* PPT *adversary* (A_1, A_2) *such that* $A_1(1^\lambda)$ *outputs a set* $S \subseteq \{0,1\}^{n(\lambda)}$ *and state* σ, *consider an experiment*

where $K \leftarrow \mathsf{Key}_F(1^\lambda)$ *and* $K_S = \mathsf{Puncture}_F(K, S)$*. Then we have*

$$\left| \Pr\left[A_2(\sigma, K_S, S, \mathsf{Eval}_F(K, S)) = 1\right] - \Pr\left[A_2(\sigma, K_S, S, U_{m(\lambda)\cdot|S|}) = 1\right] \right| \leq \mathsf{negl}(\lambda)$$

where $\mathsf{Eval}_F(K, S)$ *is the concatenation of* $\mathsf{Eval}_F(K, x_1), \ldots, \mathsf{Eval}_F(K, x_k)$*, and* $S = \{x_1, \ldots, x_k\}$ *denoted the elements of* S *in lexicographic order.*

For ease of notation, we write $F(K, x)$ *to represent* $\mathsf{Eval}_F(K, x)$*. We also represent the punctured key* $\mathsf{Puncture}_F(K, S)$ *by* $K(S)$*.*

As observed by [6,7,29], the GGM construction [24] of PRFs from one-way functions yields puncturable PRFs. Thus:

Theorem 2. *[6,7,29] Assuming one-way functions exist, for all polynomials* $n(\lambda), m(\lambda)$ *there is a puncturable PRF family that maps* $n(\lambda)$ *bits to* $m(\lambda)$ *bits.*

We follow [31] for the following definitions of puncturable PRFs with enhanced properties:

Definition 3. *A* statistically injective *(puncturable) PRF family with failure probability* $\epsilon(\cdot)$ *is a family of (puncturable) PRFs* F *such that with probability* $1-\epsilon(\lambda)$ *over the random choice of key* $K \leftarrow \mathsf{Key}_F(1^\lambda)$*, we have that* $F(K, \cdot)$ *is injective.*

Definition 4. *An* extracting *(puncturable) PRF family with error* $\epsilon(\cdot)$ *for min-entropy* $k(\cdot)$ *is a family of (puncturable) PRFs* F *mapping* $n(\lambda)$ *bits to* $m(\lambda)$ *bits such that for all* λ*, if* X *is any distribution over* $n(\lambda)$ *bits with min-entropy greater than* $k(\lambda)$*, then the statistical distance between* $(K \leftarrow \mathsf{Key}_F(1^\lambda), F(K, X))$ *and* $(K \leftarrow \mathsf{Key}_F(1^\lambda), U_{m(\lambda)})$ *is at most* $\epsilon(\lambda)$*.*

The following results were proved in [31]:

Theorem 3 ([31]). *If one-way functions exist, then for all efficiently computable functions* $n(\lambda)$*,* $m(\lambda)$*, and* $e(\lambda)$ *such that* $m(\lambda) \geq 2n(\lambda) + e(\lambda)$*, there exists a puncturable statistically injective PRF family with failure probability* $2^{-e(\lambda)}$ *that maps* $n(\lambda)$ *bits to* $m(\lambda)$ *bits.*

Theorem 4. *If one-way functions exist, then for all efficiently computable functions* $n(\lambda)$*,* $m(\lambda)$*,* $k(\lambda)$*, and* $e(\lambda)$ *such that* $n(\lambda) \geq k(\lambda) \geq m(\lambda) + 2e(\lambda) + 2$*, there exists an extracting puncturable PRF family that maps* $n(\lambda)$ *bits to* $m(\lambda)$ *bits with error* $2^{-e(\lambda)}$ *for min-entropy* $k(\lambda)$*.*

B Proof of Security for Our Explainability Compiler

In this section we prove security of our explainability compiler. We must show two properties: statistical functional equivalence and explainability. (Polynomial slowdown is obvious.) The proof of statistical functional equivalence is largely identical to the analogous proof in [31], and is omitted. Instead, we focus on explainability.

We first state the following lemma, whose proof is the same as in [31].

Lemma 1. *Except with negligible probability over the choice of key K_2, the following hold:*

1. *For any fixed $u[1] = \alpha$, there exists at most one pair (input, β) such that the input input with randomness $u = (\alpha, \beta)$ will cause the Step 1 check of $\overline{\mathsf{Alg}}$ to be satisfied.*
2. *There are at most $2^{2\lambda+\ell_{\mathsf{in}}+\ell_{\mathsf{out}}}$ values for the randomness u that can cause the Step 1 check of $\overline{\mathsf{Alg}}$ to be satisfied.*

Given the above, we prove:

Theorem 5. *If F_1, F_2, F_3 are PRFs that satisfy the properties specified in Section 3.1, and $i\mathcal{O}$ is an indistinguishability obfuscator for $\mathsf{P/poly}$, then our construction $\mathsf{Comp}(\cdot,\cdot)$ satisfies explainability.*

Proof. Recall the explainability experiment from Definition 1:

1. $\mathcal{A}(1^\lambda)$ outputs input^* of its choice.
2. $\mathsf{Comp}(1^\lambda, \mathsf{Alg})$ is run to obtain $(\widetilde{\mathsf{Alg}}, \mathsf{Explain})$.
3. Choose random coins $r_0 \leftarrow \{0,1\}^*$, and compute $\mathsf{output}^* \leftarrow \widetilde{\mathsf{Alg}}(\mathsf{input}^*; r_0)$.
4. Compute $r_1 \leftarrow \mathsf{Explain}(\mathsf{input}^*, \mathsf{output}^*)$.
5. Choose a uniform bit b and give $\widetilde{\mathsf{Alg}}, \mathsf{output}^*, r_b$ to $\mathcal{A}$.
6. $\mathcal{A}$ outputs a bit b', and $\mathsf{succeeds}$ if $b' = b$.

Let $\mathsf{Expl}_{\mathsf{Alg},\mathcal{A}}$ be a random variable set to 1 if $\mathcal{A}$ succeeds in outputting $b' = b$ in the above experiment. Security of $\mathsf{Comp}(1^\lambda, \mathsf{Alg})$ requires that for every PPT $\mathcal{A}$ and for every efficient algorithm Alg, we have $\Pr[\mathsf{Expl}_{\mathsf{Alg},\mathcal{A}} = 1] \leq 1/2 + \mathsf{negl}(\lambda)$.

Assume towards a contradiction that there is some PPT adversary $\mathcal{A}$ and some efficient algorithm Alg such that $\Pr[\mathsf{Expl}_{\mathsf{Alg},\mathcal{A}} = 1] \geq 1/2 + \varepsilon(\lambda)$, for non-negligible $\varepsilon(\cdot)$. We derive a contradiction via a sequence of hybrid experiments. The change between each experiment and the previous one will be denoted by underlined text.

Original Experiment. We consider the probability that $b' = b$ in the following experiment:

1. $b \leftarrow \{0,1\}$.
2. $\mathsf{input}^* \leftarrow \mathcal{A}(1^\lambda)$.
3. Choose K_1, K_2, K_3 at random.
4. Select u^* and r^* at random.
5. If $F_3(K_3, u[1]) \oplus u[2] = (\mathsf{input}', \mathsf{output}', r')$ for (proper length) strings output', r', input', and $\mathsf{input}' = \mathsf{input}^*$, and $u[1] = F_2(K_2, (\mathsf{input}', \mathsf{output}', r'))$, then let $\mathsf{output}^* = \mathsf{output}'$ and jump to the next step. Otherwise, let $x^* = F_1(K_1, (\mathsf{input}^*, u^*))$ and $\mathsf{output}^* = \mathsf{Alg}(\mathsf{input}^*; x^*)$.
6. Set $\alpha^* = F_2(K_2, (\mathsf{input}^*, \mathsf{output}^*, \mathsf{PRG}(r^*)))$. Let $\beta^* = F_3(K_3, \alpha^*) \oplus (\mathsf{input}^*, \mathsf{output}^*, \mathsf{PRG}(r^*))$, and set $e^* = (\alpha^*, \beta^*)$.
7. Let $\widetilde{\mathsf{Alg}} \leftarrow i\mathcal{O}(\overline{\mathsf{Alg}})$ for $\overline{\mathsf{Alg}}$ as in Figure 1. Let $\mathsf{Explain} \leftarrow i\mathcal{O}(\overline{\mathsf{Explain}})$ for $\overline{\mathsf{Explain}}$ as in Figure 2.

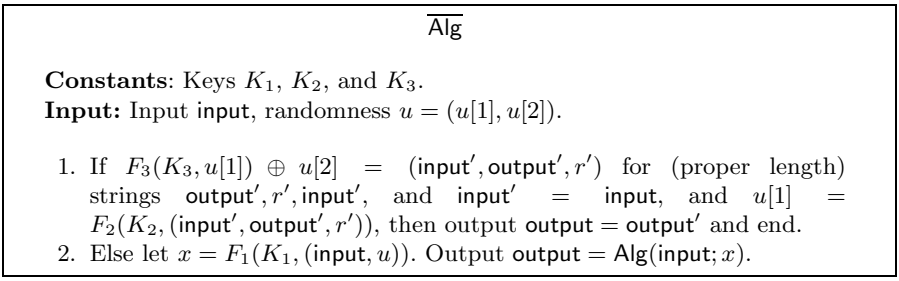

Fig. 5. Program $\overline{\mathsf{Alg}}$

8. If $b = 0$, set $b' \leftarrow \mathcal{A}(\widetilde{\mathsf{Alg}}, \mathsf{output}^*, u^*)$. If $b = 1$, set $b' \leftarrow \mathcal{A}(\widetilde{\mathsf{Alg}}, \mathsf{output}^*, e^*)$.

Hybrid 0. Next, we eliminate the check in Step 1 from the $\overline{\mathsf{Alg}}$ program when preparing the outputs of the fixed challenge input^*. Hybrid 0 is statistically close to the original experiment by Lemma 1. Consider the probability that $b' = b$ in the following experiment:

1. $b \leftarrow \{0,1\}$.
2. $\mathsf{input}^* \leftarrow \mathcal{A}(1^\lambda)$.
3. Choose K_1, K_2, K_3 at random.
4. Select u^* and r^* at random.
5. If $F_3(K_3, u[1]) \oplus u[2] = (\mathsf{input}', \mathsf{output}', r')$ for (proper length) strings output', r', input', and $\mathsf{input}' = \mathsf{input}^*$, and $u[1] = F_2(K_2, (\mathsf{input}', \mathsf{output}', r'))$, then let $\mathsf{output}^* = \mathsf{output}'$ and jump to the next step. Otherwise, let $x^* = F_1(K_1, (\mathsf{input}^*, u^*))$ and $\mathsf{output}^* = \mathsf{Alg}(\mathsf{input}^*; x^*)$.
6. Set $\alpha^* = F_2(K_2, (\mathsf{input}^*, \mathsf{output}^*, \mathsf{PRG}(r^*)))$. Let $\beta^* = F_3(K_3, \alpha^*) \oplus (\mathsf{input}^*, \mathsf{output}^*, \mathsf{PRG}(r^*))$, and set $e^* = (\alpha^*, \beta^*)$.
7. Let $\widetilde{\mathsf{Alg}} \leftarrow i\mathcal{O}(\overline{\mathsf{Alg}})$ for $\overline{\mathsf{Alg}}$ as in Figure 5. Let $\mathsf{Explain} \leftarrow i\mathcal{O}(\overline{\mathsf{Explain}})$ for $\overline{\mathsf{Explain}}$ as in Figure 6.
8. If $b = 0$, set $b' \leftarrow \mathcal{A}(\widetilde{\mathsf{Alg}}, \mathsf{output}^*, u^*)$. If $b = 1$, set $b' \leftarrow \mathcal{A}(\widetilde{\mathsf{Alg}}, \mathsf{output}^*, e^*)$.

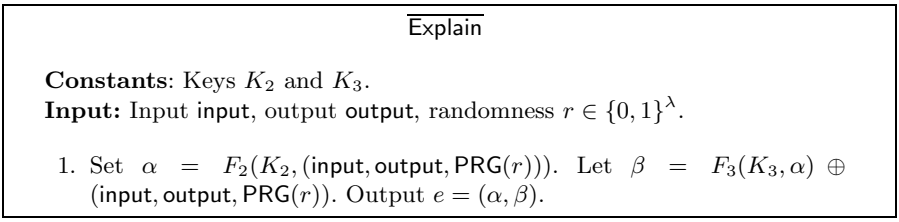

Fig. 6. Program $\overline{\mathsf{Explain}}$

Hybrid 1. Here, we modify the $\overline{\mathsf{Alg}}$ program as follows: First, we add constants $\mathsf{input}^*, \mathsf{output}^*, u^*, e^*$ to the program. Then, we add an "if" statement at the

$\overline{\mathsf{Alg}}$

Constants: $\underline{\mathsf{input}^*, \mathsf{output}^*, u^*, e^*}$ and PRF keys $K_1((\mathsf{input}^*, u^*), \underline{(\mathsf{input}^*, e^*)})$, K_2, and K_3.
Input: Input input, randomness $u = (u[1], u[2])$.

1. If $(\mathsf{input}, u) = (\mathsf{input}^*, u^*)$ or (input^*, e^*), output output^* and stop.
2. If $F_3(K_3, u[1]) \oplus u[2] = (\mathsf{input}', \mathsf{output}', r')$ for (proper length) strings $\mathsf{output}', r', \mathsf{input}'$, and $\mathsf{input}' = \mathsf{input}$, and $u[1] = F_2(K_2, (\mathsf{input}', \mathsf{output}', r'))$, then output $\mathsf{output} = \mathsf{output}'$ and end.
3. Else let $x = F_1(K_1, (\mathsf{input}, u))$. Output $\mathsf{output} = \mathsf{Alg}(\mathsf{input}; x)$.

Fig. 7. Program $\overline{\mathsf{Alg}}$

start that outputs output^* if the input is either (input^*, u^*) or (input^*, e^*), as this is exactly what the original $\overline{\mathsf{Alg}}$ program would do by our choice of u^*, e^*. Because this "if" statement is in place, we know that F_1 cannot be evaluated at either (input^*, u^*) or (input^*, e^*) within the program, and therefore we can safely puncture K_1 at those two positions.

By construction, the new $\overline{\mathsf{Alg}}$ program is functionally equivalent to the original $\overline{\mathsf{Alg}}$ program. Therefore, indistinguishability of Hybrid 0 and Hybrid 1 follows by the security of $\mathsf{i}\mathcal{O}$. Thus, the difference in the probabilities that $\mathcal{A}$ outputs $b' = b$ in Hybrid 0 and Hybrid 1 is negligible.

1. $b \leftarrow \{0, 1\}$.
2. $\mathsf{input}^* \leftarrow \mathcal{A}(1^\lambda)$.
3. Choose K_1, K_2, K_3 at random.
4. Select u^* and r^* at random.
5. Let $x^* = F_1(K_1, (\mathsf{input}^*, u^*))$ and let $\mathsf{output}^* = \mathsf{Alg}(\mathsf{input}^*; x^*)$.
6. Set $\alpha^* = F_2(K_2, (\mathsf{input}^*, \mathsf{output}^*, \mathsf{PRG}(r^*)))$. Let $\beta^* = F_3(K_3, \alpha^*) \oplus (\mathsf{input}^*, \mathsf{output}^*, \mathsf{PRG}(r^*))$, and set $e^* = (\alpha^*, \beta^*)$.
7. Let $\widetilde{\mathsf{Alg}} \leftarrow \mathsf{i}\mathcal{O}(\overline{\mathsf{Alg}})$ for $\overline{\mathsf{Alg}}$ as in Figure 7. Let $\mathsf{Explain} \leftarrow \mathsf{i}\mathcal{O}(\overline{\mathsf{Explain}})$ for $\overline{\mathsf{Explain}}$ as in Figure 8.
8. If $b = 0$, set $b' \leftarrow \mathcal{A}(\widetilde{\mathsf{Alg}}, \mathsf{output}^*, u^*)$. If $b = 1$, set $b' \leftarrow \mathcal{A}(\widetilde{\mathsf{Alg}}, \mathsf{output}^*, e^*)$.

$\overline{\mathsf{Explain}}$

Constants: PRF keys K_2, and K_3.
Input: Input input, output output, randomness r.

1. Set $\alpha = F_2(K_2, (\mathsf{input}, \mathsf{output}, \mathsf{PRG}(r)))$. Let $\beta = F_3(K_3, \alpha) \oplus (\mathsf{input}, \mathsf{output}, \mathsf{PRG}(r))$. Output $e = (\alpha, \beta)$.

Fig. 8. Program $\overline{\mathsf{Explain}}$

Hybrid 2. Here, the value x^* is chosen uniformly instead of as the output of $F_1(K_1, (\mathsf{input}^*, u^*))$. Pseudorandomness of F_1 thus implies that the difference in the probabilities that $\mathcal{A}$ outputs $b' = b$ in Hybrid 1 and Hybrid 2 is negligible.

1. $b \leftarrow \{0,1\}$.
2. $\mathsf{input}^* \leftarrow \mathcal{A}(1^\lambda)$.
3. Choose K_1, K_2, K_3 at random.
4. Select u^* and r^* at random.
5. Choose uniform x^* and let $\mathsf{output}^* = \mathsf{Alg}(\mathsf{input}^*; x^*)$.
6. Set $\alpha^* = F_2(K_2, (\mathsf{input}^*, \mathsf{output}^*, \mathsf{PRG}(r^*)))$. Let $\beta^* = F_3(K_3, \alpha^*) \oplus (\mathsf{input}^*, \mathsf{output}^*, \mathsf{PRG}(r^*))$, and set $e^* = (\alpha^*, \beta^*)$.
7. Let $\widetilde{\overline{\mathsf{Alg}}} \leftarrow i\mathcal{O}(\overline{\mathsf{Alg}})$ for $\overline{\mathsf{Alg}}$ as in Figure 9. Let $\mathsf{Explain} \leftarrow i\mathcal{O}(\overline{\mathsf{Explain}})$ for $\overline{\mathsf{Explain}}$ as in Figure 10.
8. If $b = 0$, set $b' \leftarrow \mathcal{A}(\widetilde{\overline{\mathsf{Alg}}}, \mathsf{output}^*, u^*)$. If $b = 1$, set $b' \leftarrow \mathcal{A}(\widetilde{\overline{\mathsf{Alg}}}, \mathsf{output}^*, e^*)$.

$\overline{\mathsf{Alg}}$

Constants: $\mathsf{input}^*, \mathsf{output}^*, u^*, e^*$ and PRF keys $K_1((\mathsf{input}^*, u^*), (\mathsf{input}^*, e^*))$, K_2, and K_3.
Input: Input input, randomness $u = (u[1], u[2])$.

1. If $(\mathsf{input}, u) = (\mathsf{input}^*, u^*)$ or (input^*, e^*), output output^* and stop.
2. If $F_3(K_3, u[1]) \oplus u[2] = (\mathsf{input}', \mathsf{output}', r')$ for (proper length) strings $\mathsf{output}', r', \mathsf{input}'$, and $\mathsf{input}' = \mathsf{input}$, and $u[1] = F_2(K_2, (\mathsf{input}', \mathsf{output}', r'))$, then output $\mathsf{output} = \mathsf{output}'$ and end.
3. Else let $x = F_1(K_1, (\mathsf{input}, u))$. Output $\mathsf{output} = \mathsf{Alg}(\mathsf{input}; x)$.

Fig. 9. Program $\overline{\mathsf{Alg}}$

$\overline{\mathsf{Explain}}$

Constants: PRF keys K_2, and K_3.
Input: Input input, output output, randomness r.

1. Set $\alpha = F_2(K_2, (\mathsf{input}, \mathsf{output}, \mathsf{PRG}(r)))$. Let $\beta = F_3(K_3, \alpha) \oplus (\mathsf{input}, \mathsf{output}, \mathsf{PRG}(r))$. Output $e = (\alpha, \beta)$.

Fig. 10. Program $\overline{\mathsf{Explain}}$

Hybrid 3. Here, instead of choosing uniform r^* and applying a PRG to it, a value $\tilde{r}$ is chosen uniformly from the range of the PRG. Security of the PRG implies that the difference in the probabilities that $\mathcal{A}$ outputs $b' = b$ in Hybrid 2 and Hybrid 3 is negligible.

1. $b \leftarrow \{0,1\}$.
2. $\mathsf{input}^* \leftarrow \mathcal{A}(1^\lambda)$.
3. Choose K_1, K_2, K_3 at random.
4. Select u^* and $\tilde{r}$ at random.
5. Choose uniform x^* and let $\mathsf{output}^* = \mathsf{Alg}(\mathsf{input}^*; x^*)$.
6. Set $\alpha^* = F_2(K_2, (\mathsf{input}^*, \mathsf{output}^*, \tilde{r}))$. Let $\beta^* = F_3(K_3, \alpha^*) \oplus (\mathsf{input}^*, \mathsf{output}^*, \tilde{r})$, and set $e^* = (\alpha^*, \beta^*)$.
7. Let $\widetilde{\mathsf{Alg}} \leftarrow \mathsf{i}\mathcal{O}(\overline{\mathsf{Alg}})$ for $\overline{\mathsf{Alg}}$ as in Figure 11. Let $\mathsf{Explain} \leftarrow \mathsf{i}\mathcal{O}(\overline{\mathsf{Explain}})$ for $\overline{\mathsf{Explain}}$ as in Figure 12.
8. If $b = 0$, set $b' \leftarrow \mathcal{A}(\widetilde{\mathsf{Alg}}, \mathsf{output}^*, u^*)$. If $b = 1$, set $b' \leftarrow \mathcal{A}(\widetilde{\mathsf{Alg}}, \mathsf{output}^*, e^*)$.

$\overline{\mathsf{Alg}}$

Constants: $\mathsf{input}^*, \mathsf{output}^*, u^*, e^*$ and PRF keys $K_1((\mathsf{input}^*, u^*), (\mathsf{input}^*, e^*))$, K_2, and K_3.
Input: Input input, randomness $u = (u[1], u[2])$.

1. If $(\mathsf{input}, u) = (\mathsf{input}^*, u^*)$ or (input^*, e^*), output output^* and stop.
2. If $F_3(K_3, u[1]) \oplus u[2] = (\mathsf{input}', \mathsf{output}', r')$ for (proper length) strings $\mathsf{output}', r', \mathsf{input}'$, and $\mathsf{input}' = \mathsf{input}$, and $u[1] = F_2(K_2, (\mathsf{input}', \mathsf{output}', r'))$, then output $\mathsf{output} = \mathsf{output}'$ and end.
3. Else let $x = F_1(K_1, (\mathsf{input}, u))$. Output $\mathsf{output} = \mathsf{Alg}(\mathsf{input}; x)$.

Fig. 11. Program $\overline{\mathsf{Alg}}$

Hybrid 4. Here, the $\overline{\mathsf{Alg}}$ and $\overline{\mathsf{Explain}}$ programs are modified as shown below. In Lemma 2, (proven below), we argue that except with negligible probability over choice of constants, these modifications do not alter the functionality of either program. Thus, the $\mathsf{i}\mathcal{O}$ security property implies that the difference in the probabilities that $\mathcal{A}$ outputs $b' = b$ in Hybrid 3 and Hybrid 4 is negligible.

1. $b \leftarrow \{0,1\}$.
2. $\mathsf{input}^* \leftarrow \mathcal{A}(1^\lambda)$.
3. Choose K_1, K_2, K_3 at random.
4. Select u^* and $\tilde{r}$ at random.
5. Choose uniform x^* and let $\mathsf{output}^* = \mathsf{Alg}(\mathsf{input}^*; x^*)$.
6. Set $\alpha^* = F_2(K_2, (\mathsf{input}^*, \mathsf{output}^*, \tilde{r}))$. Let $\beta^* = F_3(K_3, \alpha^*) \oplus (\mathsf{input}^*, \mathsf{output}^*, \tilde{r})$, and set $e^* = (\alpha^*, \beta^*)$.
7. Let $\widetilde{\mathsf{Alg}} \leftarrow \mathsf{i}\mathcal{O}(\overline{\mathsf{Alg}})$ for $\overline{\mathsf{Alg}}$ as in Figure 13. Let $\mathsf{Explain} \leftarrow \mathsf{i}\mathcal{O}(\overline{\mathsf{Explain}})$ for $\overline{\mathsf{Explain}}$ as in Figure 14.

$\overline{\mathsf{Explain}}$

Constants: PRF keys K_2, and K_3.
Input: Input input, output output, randomness r.

1. Set $\alpha = F_2(K_2, (\mathsf{input}, \mathsf{output}, \mathsf{PRG}(r)))$. Let $\beta = F_3(K_3, \alpha) \oplus (\mathsf{input}, \mathsf{output}, \mathsf{PRG}(r))$. Output $e = (\alpha, \beta)$.

Fig. 12. Program $\overline{\mathsf{Explain}}$

$\overline{\mathsf{Alg}}$

Constants: $\mathsf{input}^*, \mathsf{output}^*, u^*, e^*$ and PRF keys $K_1((\mathsf{input}^*, u^*), (\mathsf{input}^*, e^*))$, K_2, and $K_3(\underline{u^*[1], e^*[1]})$.
Input: Input input, randomness $u = (u[1], \underline{u[2]})$.

1. If $(\mathsf{input}, u) = (\mathsf{input}^*, u^*)$ or (input^*, e^*), output output^* and stop.
2. $\underline{\text{If } u[1] = e^*[1] \text{ or } u[1] = u^*[1], \text{ then skip this step.}}$ If $F_3(K_3, u[1]) \oplus u[2] = (\mathsf{input}', \mathsf{output}', r')$ for (proper length) strings $\mathsf{output}', r', \mathsf{input}'$, and $\mathsf{input}' = \mathsf{input}$, and $u[1] = F_2(K_2, (\mathsf{input}', \mathsf{output}', r'))$, then output $\mathsf{output} = \mathsf{output}'$ and end.
3. Else let $x = F_1(K_1, (\mathsf{input}, u))$. Output $\mathsf{output} = \mathsf{Alg}(\mathsf{input}; x)$.

Fig. 13. Program $\overline{\mathsf{Alg}}$

$\overline{\mathsf{Explain}}$

Constants: PRF keys K_2, and $K_3(\underline{u^*[1], e^*[1]})$.
Input: Input input, output output, randomness r.

1. Set $\alpha = F_2(K_2, (\mathsf{input}, \mathsf{output}, \mathsf{PRG}(r)))$. Let $\beta = F_3(K_3, \alpha) \oplus (\mathsf{input}, \mathsf{output}, \mathsf{PRG}(r))$. Output $e = (\alpha, \beta)$.

Fig. 14. Program $\overline{\mathsf{Explain}}$

8. If $b = 0$, set $b' \leftarrow \mathcal{A}(\widetilde{\mathsf{Alg}}, \mathsf{output}^*, u^*)$. If $b = 1$, set $b' \leftarrow \mathcal{A}(\widetilde{\mathsf{Alg}}, \mathsf{output}^*, e^*)$.

Hybrid 5. Here, the value $e^*[2]$, denoted β^*, is chosen at random instead of being chosen as $\beta^* = F_3(K_3, \alpha^*) \oplus (\mathsf{input}^*, \mathsf{output}^*, \tilde{r})$. Pseudorandomness of F_3 thus implies that the difference in the probabilities that $\mathcal{A}$ outputs $b' = b$ in Hybrid 4 and Hybrid 5 is negligible.

1. $b \leftarrow \{0, 1\}$.
2. $\mathsf{input}^* \leftarrow \mathcal{A}(1^\lambda)$.
3. Choose K_1, K_2, K_3 at random.
4. Select u^* and $\tilde{r}$ at random.
5. Choose uniform x^* and let $\mathsf{output}^* = \mathsf{Alg}(\mathsf{input}^*; x^*)$.
6. Set $\alpha^* = F_2(K_2, (\mathsf{input}^*, \mathsf{output}^*, \tilde{r}))$. $\underline{\text{Choose uniform } \beta^*}$, and set $e^* = (\alpha^*, \beta^*)$.

7. Let $\widetilde{\overline{\mathsf{Alg}}} \leftarrow \mathsf{i}\mathcal{O}(\overline{\mathsf{Alg}})$ for $\overline{\mathsf{Alg}}$ as in Figure 15. Let $\mathsf{Explain} \leftarrow \mathsf{i}\mathcal{O}(\overline{\mathsf{Explain}})$ for $\overline{\mathsf{Explain}}$ as in Figure 16.
8. If $b = 0$, set $b' \leftarrow \mathcal{A}(\widetilde{\overline{\mathsf{Alg}}}, \mathsf{output}^*, u^*)$. If $b = 1$, set $b' \leftarrow \mathcal{A}(\widetilde{\overline{\mathsf{Alg}}}, \mathsf{output}^*, e^*)$.

$\overline{\mathsf{Alg}}$

Constants: $\mathsf{input}^*, \mathsf{output}^*, u^*, e^*$ and PRF keys $K_1((\mathsf{input}^*, u^*), (\mathsf{input}^*, e^*))$, K_2, and $K_3(u^*[1], e^*[1])$.
Input: Input input, randomness $u = (u[1], u[2])$.

1. If $(\mathsf{input}, u) = (\mathsf{input}^*, u^*)$ or (input^*, e^*), output output^* and stop.
2. If $u[1] = e^*[1]$ or $u[1] = u^*[1]$, then skip this step. If $F_3(K_3, u[1]) \oplus u[2] = (\mathsf{input}', \mathsf{output}', r')$ for (proper length) strings $\mathsf{output}', r', \mathsf{input}'$, and $\mathsf{input}' = \mathsf{input}$, and $u[1] = F_2(K_2, (\mathsf{input}', \mathsf{output}', r'))$, then output $\mathsf{output} = \mathsf{output}'$ and end.
3. Else let $x = F_1(K_1, (\mathsf{input}, u))$. Output $\mathsf{output} = \mathsf{Alg}(\mathsf{input}; x)$.

Fig. 15. Program $\overline{\mathsf{Alg}}$

Hybrid 6. First we modify the $\overline{\mathsf{Alg}}$ program to add a condition to the check in Step 2 to determine if $(\mathsf{input}', \mathsf{output}', r') = (\mathsf{input}^*, \mathsf{output}^*, \tilde{r})$ and, if so, to skip this check. This does not change the functionality of the program, because $e^*[1] = F_2(K_2, (\mathsf{input}^*, \mathsf{output}^*, \tilde{r}))$, and therefore the check cannot be satisfied if $(\mathsf{input}', \mathsf{output}', r') = (\mathsf{input}^*, \mathsf{output}^*, \tilde{r})$, since Step 2 is skipped entirely if $u[1] = e^*[1]$. Furthermore, both the $\overline{\mathsf{Alg}}$ and $\overline{\mathsf{Explain}}$ programs are modified to have K_2 punctured at the points $(\mathsf{input}^*, \mathsf{output}^*, \tilde{r})$. This puncturing does not change the functionality of the $\overline{\mathsf{Alg}}$ program because of the new "if" condition just added. With overwhelming probability, $\tilde{r}$ is not in the image of the PRG and therefore this puncturing also does not change the functionality of the $\overline{\mathsf{Explain}}$ program. Thus, the difference in the probabilities that $\mathcal{A}$ outputs $b' = b$ in Hybrid 5 and Hybrid 6 is negligible.

$\overline{\mathsf{Explain}}$

Constants: PRF keys K_2, and $K_3(u^*[1], e^*[1])$.
Input: Input input, output output, randomness r.

1. Set $\alpha = F_2(K_2, (\mathsf{input}, \mathsf{output}, \mathsf{PRG}(r)))$. Let $\beta = F_3(K_3, \alpha) \oplus (\mathsf{input}, \mathsf{output}, \mathsf{PRG}(r))$. Output $e = (\alpha, \beta)$.

Fig. 16. Program $\overline{\mathsf{Explain}}$

1. $b \leftarrow \{0,1\}$.
2. $\mathsf{input}^* \leftarrow \mathcal{A}(1^\lambda)$.
3. Choose K_1, K_2, K_3 at random.
4. Select u^* and $\tilde{r}$ at random.
5. Choose uniform x^* and let $\mathsf{output}^* = \mathsf{Alg}(\mathsf{input}^*; x^*)$.
6. Set $\alpha^* = F_2(K_2, (\mathsf{input}^*, \mathsf{output}^*, \tilde{r}))$. Choose uniform β^*, and set $e^* = (\alpha^*, \beta^*)$.
7. Let $\widetilde{\mathsf{Alg}} \leftarrow \mathsf{i}\mathcal{O}(\overline{\mathsf{Alg}})$ for $\overline{\mathsf{Alg}}$ as in Figure 17. Let $\mathsf{Explain} \leftarrow \mathsf{i}\mathcal{O}(\overline{\mathsf{Explain}})$ for $\overline{\mathsf{Explain}}$ as in Figure 18.
8. If $b = 0$, set $b' \leftarrow \mathcal{A}(\widetilde{\mathsf{Alg}}, \mathsf{output}^*, u^*)$. If $b = 1$, set $b' \leftarrow \mathcal{A}(\widetilde{\mathsf{Alg}}, \mathsf{output}^*, e^*)$.

$\overline{\mathsf{Alg}}$

Constants: $\mathsf{input}^*, \mathsf{output}^*, u^*, e^*, \tilde{r}$ and PRF keys $K_1((\mathsf{input}^*, u^*), (\mathsf{input}^*, e^*))$, $K_2(\underline{(\mathsf{input}^*, \mathsf{output}^*, \tilde{r})})$, and $K_3(u^*[1], e^*[1])$.
Input: Input input, randomness $u = (u[1], u[2])$.

1. If $(\mathsf{input}, u) = (\mathsf{input}^*, u^*)$ or (input^*, e^*), output output^* and stop.
2. If $u[1] = e^*[1]$ or $u[1] = u^*[1]$, then skip this step. If $F_3(K_3, u[1]) \oplus u[2] = (\mathsf{input}', \mathsf{output}', r')$ for (proper length) strings $\mathsf{output}', r', \mathsf{input}'$, and $\mathsf{input}' = \mathsf{input}$, and $\underline{(\mathsf{input}', \mathsf{output}', r') \neq (\mathsf{input}^*, \mathsf{output}^*, \tilde{r})}$, then also check if $u[1] = F_2(K_2, (\mathsf{input}', \mathsf{output}', r'))$, then output $\mathsf{output} = \mathsf{output}'$ and end.
3. Else let $x = F_1(K_1, (\mathsf{input}, u))$. Output $\mathsf{output} = \mathsf{Alg}(\mathsf{input}; x)$.

Fig. 17. Program $\overline{\mathsf{Alg}}$

$\overline{\mathsf{Explain}}$

Constants: PRF keys $K_2(\underline{(\mathsf{input}^*, \mathsf{output}^*, \tilde{r})})$, and $K_3(u^*[1], e^*[1])$.
Input: Input input, output output, randomness r.

1. Set $\alpha = F_2(K_2, (\mathsf{input}, \mathsf{output}, \mathsf{PRG}(r)))$. Let $\beta = F_3(K_3, \alpha) \oplus (\mathsf{input}, \mathsf{output}, \mathsf{PRG}(r))$. Output $e = (\alpha, \beta)$.

Fig. 18. Program $\overline{\mathsf{Explain}}$

Hybrid 7. Finally, we modify $e^*[1]$, denoted α^*, to be uniform, instead of being computed as $\alpha^* = F_2(K_2, (\mathsf{input}^*, \mathsf{output}^*, \tilde{r}))$. Pseudorandomness of F_2 implies that the difference in the probabilities that $\mathcal{A}$ outputs $b' = b$ in Hybrid 6 and Hybrid 7 is negligible.

1. $b \leftarrow \{0,1\}$.
2. $\mathsf{input}^* \leftarrow \mathcal{A}(1^\lambda)$.
3. Choose K_1, K_2, K_3 at random.
4. Select u^* and $\tilde{r}$ at random.
5. Choose uniform x^* and let $\mathsf{output}^* = \mathsf{Alg}(\mathsf{input}^*; x^*)$.
6. Choose uniform α^* and β^*, and set $e^* = (\alpha^*, \beta^*)$.
7. Let $\widetilde{\mathsf{Alg}} \leftarrow \mathsf{i}\mathcal{O}(\overline{\mathsf{Alg}})$ for $\overline{\mathsf{Alg}}$ as in Figure 19. Let $\mathsf{Explain} \leftarrow \mathsf{i}\mathcal{O}(\overline{\mathsf{Explain}})$ for $\overline{\mathsf{Explain}}$ as in Figure 20.
8. If $b = 0$, set $b' \leftarrow \mathcal{A}(\widetilde{\mathsf{Alg}}, \mathsf{output}^*, u^*)$. If $b = 1$, set $b' \leftarrow \mathcal{A}(\widetilde{\mathsf{Alg}}, \mathsf{output}^*, e^*)$.

$\overline{\mathsf{Alg}}$

Constants: $\mathsf{input}^*, \mathsf{output}^*, u^*, e^*, \tilde{r}$ and PRF keys $K_1((\mathsf{input}^*, u^*), (\mathsf{input}^*, e^*))$, $K_2((\mathsf{input}^*, \mathsf{output}^*, \tilde{r}))$, and $K_3(u^*[1], e^*[1])$.
Input: Input input, randomness $u = (u[1], u[2])$.

1. If $(\mathsf{input}, u) = (\mathsf{input}^*, u^*)$ or (input^*, e^*), output output^* and stop.
2. If $u[1] = e^*[1]$ or $u[1] = u^*[1]$, then skip this step. If $F_3(K_3, u[1]) \oplus u[2] = (\mathsf{input}', \mathsf{output}', r')$ for (proper length) strings $\mathsf{output}', r', \mathsf{input}'$, and $\mathsf{input}' = \mathsf{input}$, and, $(\mathsf{input}', \mathsf{output}', r') \neq (\mathsf{input}^*, \mathsf{output}^*, \tilde{r})$, then also check if $u[1] = F_2(K_2, (\mathsf{input}', \mathsf{output}', r'))$, then output $\mathsf{output} = \mathsf{output}'$ and end.
3. Else let $x = F_1(K_1, (\mathsf{input}, u))$. Output $\mathsf{output} = \mathsf{Alg}(\mathsf{input}; x)$.

Fig. 19. Program $\overline{\mathsf{Alg}}$

In Hybrid 7 we observe that the variables e^*, u^* are now uniform and independent. Thus, the inputs to $\mathcal{A}$ are distributed identically regardless of whether $b = 0$ or $b = 1$ are identical, and so $b = b'$ with probability exactly $1/2$. The lemma below thus concludes the proof.

$\overline{\mathsf{Explain}}$

Constants: PRF keys $K_2((\mathsf{input}^*, \mathsf{output}^*, \tilde{r}))$, and $K_3(u^*[1], e^*[1])$.
Input: Input input, output output, randomness r.

1. Set $\alpha = F_2(K_2, (\mathsf{input}, \mathsf{output}, \mathsf{PRG}(r)))$. Let $\beta = F_3(K_3, \alpha) \oplus (\mathsf{input}, \mathsf{output}, \mathsf{PRG}(r))$. Output $e = (\alpha, \beta)$.

Fig. 20. Program $\overline{\mathsf{Explain}}$

The proof above relies on the following lemma showing that the programs obfuscated in Hybrid 3 are equivalent to the corresponding programs in Hybrid 4.

Lemma 2. *Except with negligible probability over the choice of* $u^*[1]$ *and* $e^*[1]$*, the* $\overline{\mathsf{Alg}}$ *and* $\overline{\mathsf{Explain}}$ *programs in Hybrid 4 are equivalent to the* $\overline{\mathsf{Alg}}$ *and* $\overline{\mathsf{Explain}}$ *programs in Hybrid 3.*

Proof. We consider below each change to the programs.

First, an "if" statement is added to Step 2 of the $\overline{\mathsf{Alg}}$ program, to skip the check in Step 2 if either $u[1] = e^*[1]$ or $u[1] = u^*[1]$. To see why this change does not affect the functionality of the program, let us consider each case in turn. By Lemma 1, if $u[1] = e^*[1]$, then the only way the Step 2 check can be satisfied is if $\mathsf{input} = \mathsf{input}^*$ and $u[2] = e^*[2]$. But this case is already handled in Step 1; therefore, skipping Step 2 if $u[1] = e^*[1]$ does not affect the functionality of the program. On the other hand, recall that $u^*[1]$ is chosen at random, and therefore the probability that $u^*[1]$ is in the image of $F_2(K_2, \cdot)$ is negligible. Thus, with overwhelming probability over the choice of constants $u^*[1]$, the check in Step 2 cannot be satisfied if $u[1] = u^*[1]$. Therefore, the addition of this "if" statement does not alter the functionality of the $\overline{\mathsf{Alg}}$ program.

Also, the key K_3 is punctured at $u^*[1], e^*[1]$ in both the $\overline{\mathsf{Alg}}$ and $\overline{\mathsf{Explain}}$ programs. The new "if" statement above ensures that $F_3(K_3, \cdot)$ is never called at these values in the $\overline{\mathsf{Alg}}$ program. Recall that the $\overline{\mathsf{Explain}}$ program only calls $F_3(K_3, \cdot)$ on values computed as $F_2(K_2, (\mathsf{input}, \mathsf{output}, \mathsf{PRG}(r)))$ for some bit input and strings output and r. Furthermore, F_2 is statistically injective with a very sparse image set, by our choice of parameters. Since every $u^*[1]$ is randomly chosen, it is very unlikely to be in the image of $F_2(K_2, \cdot)$. Since every $e^*[1]$ is chosen based on a random $\tilde{r}$ value instead of a PRG output, it is very unlikely to correspond to $F_2(K_2, (\mathsf{input}, \mathsf{output}, \mathsf{PRG}(r)))$ for any $(\mathsf{input}, \mathsf{output}, r)$. Thus, these values are not called by the $\overline{\mathsf{Explain}}$ program, except with negligible probability over the choice of these constants $u^*[1]$ and $e^*[1]$.