Adaptively Secure, Universally Composable, Multi-Party Computation in Constant Rounds

Dana Dachman-Soled* Jonathan Katz*† Vanishree Rao‡

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.

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, 9], 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. [9], 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 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) [9, 2, 15, 13] without this assumption. Similarly, adaptively secure two-party computation is much easier with the assumption of secure erasure [26] than without [11].

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

*University of Maryland. Email: danadach@ece.umd.edu, jkatz@cs.umd.edu.

†Work supported in part by NSF awards #1111599 and #1223623.

‡UCLA. Work done while visiting the University of Maryland. Email: vhvanshvansh@gmail.com.

(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) [7], and is important for ensuring sequential composition even in the stand-alone setting [6]. If secure erasure is assumed, the definition of adaptive security does not change whether or not PEC is allowed [8], 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., [9, 2]) assume a majority of the parties remain uncorrupted. Other work [24, 23, 18, 21], including concurrent work of [16], 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 [19] 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 [11], the "sunspots" model [12], the key-registration model [1], and, more generally, based on adaptively secure UC puzzles [14]. In addition, all prior work in this setting handles only "adaptively well-formed functionalities" (see [11] for a definition).

1.1 Our Result

We show a constant-round, universally composable protocol for multi-party computation of arbitrary functionalities, with security against a malicious, adaptive adversary corrupting any number of parties. Once again, we stress that we do not assume secure erasure.

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 [11] 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, note that a CRS—or some other form of setup—is anyway needed [10] 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. We remark further that results of Garg and Sahai [18] indicate that a CRS (or some other form of setup) is needed to obtain constant-round protocols with adaptive security even in the semi-honest case; see further discussion at the end of this section.

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.

Sahai and Waters [27] 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 [11]); 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 $\widetilde{\mathsf{NextMsg}}$, an explainable version of $\mathsf{NextMsg}$.[2] We note that, in contrast to [27], in the real-world execution no parties have access to the $\mathsf{Explain}$ algorithm corresponding to $\widetilde{\mathsf{NextMsg}}$.

Our multi-party 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.

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

[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 the parties wish to compute.

[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.

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

First, our protocol can compute *arbitrary* randomized functionalities, not just *adaptively well-formed* ones. (We refer to [11] for a definition of this term.) This may seem somewhat surprising in light of an impossibility result of Ishai et al. [22] showing that adaptively secure computation of *all* functionalities (and not just well-formed ones) is impossible. A closer examination of their result, however, reveals that it does *not* hold in the CRS model.[4]

Second, Garg and Sahai [18] show that no constant-round, adaptively secure, multi-party 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. Their impossibility result, though, explicitly only applies to the "plain" model where no setup is assumed, whereas in our work we assume a CRS.

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 [27] can be used to make any algorithm explainable. Finally, in Section 4 we present a constant-round multi-party computation protocol tolerating a semi-honest, adaptive adversary corrupting any number of parties. Applying the compiler of Canetti et al. [11] 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 [6, 8, 26] 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 [28]. 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 $\mathsf{SimGC}(1^\lambda, C, C(x; r))$.

[4] Although Ishai et al. claim that their result holds in the CRS model, they only provide a proof in the plain model and their proof seems to break down in the CRS model.

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 [11].

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

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

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" [21] 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 $\big\{r_R \leftarrow \{0,1\}^* : \big(r_R,\ R_{\mathsf{OT}}(b; r_R)\big)\big\}$ 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 $|\mathsf{i}\mathcal{O}(1^\lambda, C)| \le p(\lambda) \cdot |C|$ 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:

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

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. [17] have shown the first candidate construction of indistinguishability obfuscators for $\mathsf{P/poly}$.

3 Explainability Compilers

Sahai and Waters [27] 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.

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* succeeds *if* $b' = b$.

We highlight one key difference between our definition and the corresponding one from [27]: in our case input^* is an arbitrary length value (depending on the domain of Alg) chosen by the adversary, whereas in [27] 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 [27], we now show how to construct an explainability compiler. As in [27], we rely on an indistinguishability obfuscator, $\mathsf{i}\mathcal{O}$, for $\mathsf{P/poly}$ and three different pseudorandom function (PRF) variants (cf. Appendix A):

$\overline{\mathsf{Alg}}$

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 $\mathsf{output} := \mathsf{Alg}(\mathsf{input}; x)$ and end.

Figure 1: Program $\overline{\mathsf{Alg}}$

- 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 [27]. 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 [27] 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 $\widetilde{\mathsf{Alg}}$.

$\overline{\mathsf{Explain}}$

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).

Figure 2: Program $\overline{\mathsf{Explain}}$

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 $w_{1,r_1}, \ldots, w_{\lambda,r_\lambda}$.

Figure 3: Algorithm $\mathsf{NextMsg}$. The security parameter 1^λ and circuit C are hardwired.

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 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:

Semi-Honest, Adaptively Secure Multi-Party 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.

Figure 4: Protocol Π for computing randomized circuit C.

- 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 $r_n \leftarrow \mathsf{Explain}\left((\mathsf{OT}_{1,1}, \ldots, \mathsf{OT}_{1,n}), (\mathsf{GC}, \mathsf{OT}_{2,1}, \ldots, \mathsf{OT}_{2,n}, w_1, \ldots, w_n)\right)$. 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 $r^* \leftarrow \mathsf{Explain}(\mathsf{input}^*, \mathsf{output}^*)$.

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 $r^* \leftarrow \mathsf{Explain}(\mathsf{input}^*, \mathsf{output}^*)$.

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 $r^* \leftarrow \mathsf{Explain}(\mathsf{input}^*, \mathsf{output}^*)$.

   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 $r^* \leftarrow \mathsf{Explain}(\mathsf{input}^*, \mathsf{output}^*)$.

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 $r^* \leftarrow \mathsf{Explain}(\mathsf{input}^*, \mathsf{output}^*)$.

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] Boaz Barak, Ran Canetti, Jesper Buus Nielsen, and Rafael Pass. Universally composable protocols with relaxed set-up assumptions. In *45th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 186–195. IEEE, 2004.

[2] Donald Beaver. Plug and play encryption. In *Advances in Cryptology—Crypto '97*, volume 1294 of *LNCS*, pages 75–89. Springer, 1997.

[3] Donald Beaver and Stuart Haber. Cryptographic protocols provably secure against dynamic adversaries. In *Advances in Cryptology—Eurocrypt '92*, volume 658 of *LNCS*, pages 307–323. Springer, 1992.

[4] Dan Boneh and Brent Waters. Constrained pseudorandom functions and their applications. In *Advances in Cryptology—Asiacrypt 2013, Part II*, volume 8270 of *LNCS*, pages 280–300. Springer, 2013.

[5] Elette Boyle, Shafi Goldwasser, and Ioana Ivan. Functional signatures and pseudorandom functions. In *17th Intl. Conference on Theory and Practice of Public Key Cryptography—PKC 2014*, volume 8383 of *LNCS*, pages 501–519. Springer, 2014.

[6] Ran Canetti. Security and composition of multiparty cryptographic protocols. *Journal of Cryptology*, 13(1):143–202, 2000.

[7] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *42nd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 136–145. IEEE, 2001. Full version at `http://eprint.iacr.org/2000/067/`.

[8] Ran Canetti, Ivan Damgård, Stefan Dziembowski, Yuval Ishai, and Tal Malkin. Adaptive versus non-adaptive security of multi-party protocols. *J. Crypto*, 17(3):153–207, 2004.

[9] Ran Canetti, Uriel Feige, Oded Goldreich, and Moni Naor. Adaptively secure multi-party computation. In *28th Annual ACM Symposium on Theory of Computing (STOC)*, pages 639–648. ACM Press, 1996.

[10] Ran Canetti and Marc Fischlin. Universally composable commitments. In *Advances in Cryptology—Crypto 2001*, volume 2139 of *LNCS*, pages 19–40. Springer, 2001.

[11] 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 (STOC)*, pages 494–503. ACM Press, 2002. Full version available at `http://eprint.iacr.org/2002/140`.

[12] Ran Canetti, Rafael Pass, and Abhi Shelat. Cryptography from sunspots: How to use an imperfect reference string. In *48th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 249–259. IEEE, 2007.

[13] Seung Geol Choi, Dana Dachman-Soled, Tal Malkin, and Hoeteck Wee. Improved non-committing encryption with applications to adaptively secure protocols. In *Advances in Cryptology—Asiacrypt 2009*, volume 5912 of *LNCS*, pages 287–302. Springer, 2009.

[14] Dana Dachman-Soled, Tal Malkin, Mariana Raykova, and Muthuramakrishnan Venkitasubramaniam. Adaptive and concurrent secure computation from new adaptive, non-malleable commitments. In *Advances in Cryptology—Asiacrypt 2013, Part I*, volume 8269 of *LNCS*, pages 316–336. Springer, 2013.

[15] Ivan Damgård and Jesper Buus Nielsen. Improved non-committing encryption schemes based on a general complexity assumption. In *Advances in Cryptology—Crypto 2000*, volume 1880 of *LNCS*, pages 432–450. Springer, 2000.

[16] Ivan Damgård, Antigoni Polychroniadou, and Vanishree Rao. Secure UC constant round multi-party computation, 2014. Cryptology ePrint Archive, Report 2014/830.

[17] Sanjam Garg, Craig Gentry, Shai Halevi, Mariana Raykova, Amit Sahai, and Brent Waters. Candidate indistinguishability obfuscation and functional encryption for all circuits. In *54th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 40–49. IEEE, 2013.

[18] Sanjam Garg and Amit Sahai. Adaptively secure multi-party computation with dishonest majority. In *Advances in Cryptology—Crypto 2012*, volume 7417 of *LNCS*, pages 105–123. Springer, 2012.

[19] O. Goldreich, S. Micali, and A. Wigderson. 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)*, pages 218–229. ACM Press, 1987.

[20] Oded Goldreich, Shafi Goldwasser, and Silvio Micali. On the cryptographic applications of random functions. In *Advances in Cryptology—Crypto '84*, volume 196 of *LNCS*, pages 276–288. Springer, 1985.

[21] Carmit Hazay and Arpita Patra. One-sided adaptively secure two-party computation. In *9th Theory of Cryptography Conference—TCC 2014*, volume 8349 of *LNCS*, pages 368–393. Springer, 2014.

[22] Yuval Ishai, Abishek Kumarasubramanian, Claudio Orlandi, and Amit Sahai. On invertible sampling and adaptive security. In *Advances in Cryptology—Asiacrypt 2010*, volume 6477 of *LNCS*, pages 466–482. Springer, 2010.

[23] Yuval Ishai, Manoj Prabhakaran, and Amit Sahai. Founding cryptography on oblivious transfer—efficiently. In *Advances in Cryptology—Crypto 2008*, volume 5157 of *LNCS*, pages 572–591. Springer, 2008.

[24] Jonathan Katz and Rafail Ostrovsky. Round-optimal secure two-party computation. In *Advances in Cryptology—Crypto 2004*, volume 3152 of *LNCS*, pages 335–354. Springer, 2004.

[25] Aggelos Kiayias, Stavros Papadopoulos, Nikos Triandopoulos, and Thomas Zacharias. Delegatable pseudorandom functions and applications. In *20th ACM Conf. on Computer and Communications Security (CCS)*, pages 669–684. ACM Press, 2013.

[26] Andrew Lindell. Adaptively secure two-party computation with erasures. In *Cryptographers' Track—RSA 2009*, LNCS, pages 117–132. Springer, 2009.

[27] Amit Sahai and Brent Waters. How to use indistinguishability obfuscation: Deniable encryption, and more. In *46th Annual ACM Symposium on Theory of Computing (STOC)*, pages 475–484. ACM Press, 2014.

[28] Andrew C.-C. Yao. How to generate and exchange secrets. In *27th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 162–167. IEEE, 1986.

A Puncturable PRFs

Puncturable PRFs are a type of constrained PRF [4, 5, 25] 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*, and 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*

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

where $\mathsf{Eval}_F(K,S)$ denotes the concatenation of $\mathsf{Eval}_F(K,x_1), \ldots, \mathsf{Eval}_F(K,x_k)$, and $S = \{x_1, \ldots, x_k\}$ is an enumeration of 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 [4, 5, 25], the GGM construction [20] of PRFs from one-way functions yields puncturable PRFs. Thus:

Theorem 2. *[4, 5, 25] If one-way functions exist, then for all polynomials $n(\lambda)$ and $m(\lambda)$ there exists a puncturable PRF family that maps $n(\lambda)$ bits to $m(\lambda)$ bits.*

We follow [27] 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 [27]:

Theorem 3 ([27])**.** *If one-way functions exist, then for all efficiently computable functions $n(\lambda)$, $m(\lambda)$, and $e(\lambda)$ such that $m(\lambda) \ge 2n(\lambda) + e(\lambda)$, 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 $n(\lambda) \ge k(\lambda) \ge m(\lambda) + 2e(\lambda) + 2$, 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 [27], so is omitted. Instead, we focus on explainability.

We first state the following lemma, whose proof is the same as in [27].

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 $\mathsf{i}\mathcal{O}$ is an indistinguishability obfuscator for $\mathsf{P/poly}$, then our construction $\mathsf{Comp}(\cdot, \cdot)$ satisfies explainability.*

Proof: Recall the explainability game 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 game. 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 $\varepsilon(\cdot)$. Then, we shall arrive at a contradiction through several hybrids. To maintain ease of verification for the reader, we present a full description of each hybrid experiment, each one given on a separate page. The change between each hybrid and the previous hybrid will be denoted in underlined font. The hybrids are chosen so that the indistinguishability of each successive hybrid experiment follows in a relatively straightforward manner.

We unwrap the explainability game specifically with respect to our construction Comp. Recall that we consider an adversary whose objective is to output $b' = b$ in the following game.

Original Game. We consider the probability that $b' = b$ in the following game:

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^* at random. Select r^* at random.

5. - 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 let $\mathsf{output}^* = \mathsf{output}'$ and jump to Step 5. Otherwise, perform the following Step.
   - Let $x^* = F_1(K_1, (\mathsf{input}^*, u^*))$ and let $\mathsf{output}^* = \mathsf{Alg}(\mathsf{input}^*; x^*)$.
6. Do the following. Set $\alpha^* = F_2(K_2, (\mathsf{input}^*, \mathsf{output}^*, \mathsf{PRG}(r^*)))$. Let $\beta^* = F_3(K_3, \alpha^*) \oplus (\mathsf{input}^*, \mathsf{output}^*, \mathsf{PRG}(r^*))$.
   Set $e^* = (\alpha^*, \beta^*)$.
7. Let $\widetilde{\mathsf{Alg}} \leftarrow \mathsf{i}\mathcal{O}(\overline{\mathsf{Alg}})$ for $\overline{\mathsf{Alg}}$ as in Figure 1. Let $\mathsf{Explain} \leftarrow \mathsf{i}\mathcal{O}(\overline{\mathsf{Explain}})$ for $\overline{\mathsf{Explain}}$ as in Figure 2.
8. If $b = 0$, set $b' \leftarrow \mathcal{A}(\widetilde{\mathsf{Alg}}, \mathsf{output}^*, u^*)$.
   If $b = 1$, set $b' \leftarrow \mathcal{A}(\widetilde{\mathsf{Alg}}, \mathsf{output}^*, e^*)$.

Next, we jump to Hybrid 0, where we eliminate Step 1 check from the $\overline{\mathsf{Alg}}$ program when preparing the outputs of the fixed challenge input input^*. Hybrid 0 is statistically close to the original Explainability game by Lemma 1.

Hybrid 0. We consider the probability that $b' = b$ in the following game:

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^* at random. Select r^* at random.
5. - 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 let $\mathsf{output}^* = \mathsf{output}'$ and jump to Step . Otherwise, perform the following Step.
   - Let $x^* = F_1(K_1, (\mathsf{input}^*, u^*))$ and let $\mathsf{output}^* = \mathsf{Alg}(\mathsf{input}^*; x^*)$.
6. Do the following. Set $\alpha^* = F_2(K_2, (\mathsf{input}^*, \mathsf{output}^*, \mathsf{PRG}(r^*)))$. Let $\beta^* = F_3(K_3, \alpha^*) \oplus (\mathsf{input}^*, \mathsf{output}^*, \mathsf{PRG}(r^*))$. Set $e^* = (\alpha^*, \beta^*)$.
7. Let $\widetilde{\mathsf{Alg}} \leftarrow \mathsf{i}\mathcal{O}(\overline{\mathsf{Alg}})$ for $\overline{\mathsf{Alg}}$ as in Figure 5. Let $\mathsf{Explain} \leftarrow \mathsf{i}\mathcal{O}(\overline{\mathsf{Explain}})$ 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 $b' \leftarrow \mathcal{A}(\widetilde{\mathsf{Alg}}, \mathsf{output}^*, e^*)$.

Hybrid 1. In this hybrid, 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 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 key K_1 at these two positions.

$\overline{\mathsf{Alg}}$

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

1. 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.
2. Else let $x = F_1(K_1, (\mathsf{input}, u))$. Output $\mathsf{output} = \mathsf{Alg}(\mathsf{input}; x)$.

Figure 5: Program $\overline{\mathsf{Alg}}$

$\overline{\mathsf{Explain}}$

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

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 $e = (\alpha, \beta)$.

Figure 6: Program $\overline{\mathsf{Explain}}$

By construction, the new $\overline{\mathsf{Alg}}$ program is functionally equivalent to the original $\overline{\mathsf{Alg}}$ program. Therefore the indistinguishability of Hybrid 0 and Hybrid 1 follows by the security of $i\mathcal{O}$. Thus, the probabilities that $\mathcal{A}$ outputs $b' = b$ in the two hybrids differ by a negligible amount.
Note: Implicitly, all "if" statements that are added to programs with multiple checks are written in lexicographic order; that is, if $u^* < e^*$ in lexicographic order, we write it as "If $(\mathsf{input}, u) = (\mathsf{input}^*, u^*)$ or (input^*, e^*)," otherwise we write it as "If $(\mathsf{input}, u) = (\mathsf{input}^*, e^*)$ or (input^*, u^*)."

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

Hybrid 2. Here, the value x^* is chosen uniformly instead of as the output of $F_1(K_1, (\mathsf{input}^*, u^*))$. The indistinguishability of Hybrid 2 from Hybrid 1 follows immediately from the pseudorandomness property of the punctured PRF F_1 (Definition 2). Thus, the difference in the probability $\mathcal{A}$ outputs $b' = b$ in the two hybrids is by a negligible amount.

$\overline{\mathsf{Alg}}$

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

1. $\underline{\text{If } (\mathsf{input}, u) = (\mathsf{input}^*, u^*) \text{ or } (\mathsf{input}^*, e^*)\text{, output } \mathsf{output}^* \text{ and stop.}}$
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 $\mathsf{output} = \mathsf{Alg}(\mathsf{input}; x)$.

Figure 7: 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 $e = (\alpha, \beta)$.

Figure 8: 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^* at random. Select r^* at random.
5. Let $\underline{x^* \text{ be chosen randomly}}$ and let $\mathsf{output}^* = \mathsf{Alg}(\mathsf{input}^*; x^*)$.
6. Do the following. Set $\alpha^* = F_2(K_2, (\mathsf{input}^*, \mathsf{output}^*, \mathsf{PRG}(r^*)))$. Let $\beta^* = F_3(K_3, \alpha^*) \oplus (\mathsf{input}^*, \mathsf{output}^*, \mathsf{PRG}(r^*))$. Set $e^* = (\alpha^*, \beta^*)$.
7. Let $\widetilde{\mathsf{Alg}} \leftarrow \mathsf{i}\mathcal{O}(\overline{\mathsf{Alg}})$ for $\overline{\mathsf{Alg}}$ as in Figure 9. Let $\mathsf{Explain} \leftarrow \mathsf{i}\mathcal{O}(\overline{\mathsf{Explain}})$ for $\overline{\mathsf{Explain}}$ as in Figure 10.
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 3. In this hybrid, instead of picking r^* at random and applying a PRG to it, a value $\tilde{r}$ is chosen at random from the co-domain of the PRG. The indistinguishability of Hybrid 2 and Hybrid 3 follows immediately from the security of the PRG. Thus, the difference in the probability $\mathcal{A}$ outputs $b' = b$ in the two hybrids differs by a negligible amount.

1. $b \leftarrow \{0, 1\}$.
2. $\mathsf{input}^* \leftarrow \mathcal{A}(1^\lambda)$.
3. Choose K_1, K_2, K_3 at random.

$\overline{\mathsf{Alg}}$

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 $\mathsf{output} = \mathsf{Alg}(\mathsf{input}; x)$.

Figure 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 $e = (\alpha, \beta)$.

Figure 10: Program $\overline{\mathsf{Explain}}$

4. Select u^* at random. Select $\underline{\tilde{r}}$ at random.
5. Let x^* be chosen randomly and let $\mathsf{output}^* = \mathsf{Alg}(\mathsf{input}^*; x^*)$.
6. Do the following. Set $\alpha^* = F_2(K_2, (\mathsf{input}^*, \mathsf{output}^*, \underline{\tilde{r}}))$. Let $\beta^* = F_3(K_3, \alpha^*) \oplus (\mathsf{input}^*, \mathsf{output}^*, \underline{\tilde{r}})$. Set $e^* = (\alpha^*, \beta^*)$.
7. Let $\widetilde{\mathsf{Alg}} \leftarrow i\mathcal{O}(\overline{\mathsf{Alg}})$ for $\overline{\mathsf{Alg}}$ as in Figure 11. Let $\mathsf{Explain} \leftarrow i\mathcal{O}(\overline{\mathsf{Explain}})$ 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 $\mathsf{output} = \mathsf{Alg}(\mathsf{input}; x)$.

Figure 11: 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 $e = (\alpha, \beta)$.

Figure 12: Program $\overline{\mathsf{Explain}}$

Hybrid 4. In this hybrid, the $\overline{\mathsf{Alg}}$ and $\overline{\mathsf{Explain}}$ programs are modified as shown below. In Lemma 2, (proven below after all hybrids are given), we argue that except with negligible probability over choice of constants, these modifications do not alter the functionality of either program.

Thus, the indistinguishability of Hybrid 3 and Hybrid 4 follows from the $\mathsf{i}\mathcal{O}$ security property. and so the difference in the probability $\mathcal{A}$ outputs $b' = b$ in the two hybrids differs by a negligible amount.

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^* at random. Select $\tilde{r}$ at random.
5. Let x^* be chosen randomly and let $\mathsf{output}^* = \mathsf{Alg}(\mathsf{input}^*; x^*)$.
6. Do the following. Set $\alpha^* = F_2(K_2, (\mathsf{input}^*, \mathsf{output}^*, \tilde{r}))$. Let $\beta^* = F_3(K_3, \alpha^*) \oplus (\mathsf{input}^*, \mathsf{output}^*, \tilde{r})$. 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.
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. In this hybrid, the value $e^*[2]$, denoted β^*, is chosen at random instead of being chosen as $\beta^* = F_3(K_3, \alpha^*) \oplus (\mathsf{input}^*, \mathsf{output}^*, \tilde{r})$. The indistinguishability of Hybrid 4 and Hybrid 5 follows immediately from the pseudorandomness property of the puncturable PRF F_3. Thus, the difference in the probability $\mathcal{A}$ outputs $b' = b$ in the two hybrids differs by a negligible amount.

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^* at random. Select $\tilde{r}$ at random.
5. Let x^* be chosen randomly and let $\mathsf{output}^* = \mathsf{Alg}(\mathsf{input}^*; x^*)$.

$\overline{\mathsf{Alg}}$

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 $u = (u[1], u[2])$.

1. If $(\mathsf{input}, u) = (\mathsf{input}^*, u^*)$ or (input^*, e^*), output output^* and stop.
2. <u>If $u[1] = e^*[1]$ or $u[1] = u^*[1]$, then skip this step.</u> 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 $\mathsf{output} = \mathsf{Alg}(\mathsf{input}; x)$.

Figure 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 $e = (\alpha, \beta)$.

Figure 14: Program $\overline{\mathsf{Explain}}$

6. Do the following. Set $\alpha^* = F_2(K_2, (\mathsf{input}^*, \mathsf{output}^*, \tilde{r}))$. Let <u>$\beta^*$ be random</u>. Set $e^* = (\alpha^*, \beta^*)$.
7. Let $\widetilde{\mathsf{Alg}} \leftarrow i\mathcal{O}(\overline{\mathsf{Alg}})$ for $\overline{\mathsf{Alg}}$ as in Figure 15. Let $\mathsf{Explain} \leftarrow i\mathcal{O}(\overline{\mathsf{Explain}})$ for $\overline{\mathsf{Explain}}$ as in Figure 16.
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 $K_3(u^*[1], e^*[1])$.
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 $\mathsf{output} = \mathsf{Alg}(\mathsf{input}; x)$.

Figure 15: Program $\overline{\mathsf{Alg}}$

Hybrid 6. In this hybrid, first we modify the $\overline{\mathsf{Alg}}$ program to add a condition to Step 2 check to determine if the decrypted $(\mathsf{input}', \mathsf{output}', r') = (\mathsf{input}^*, \mathsf{output}^*, \tilde{r})$, and if so, to skip this check. This

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 $e = (\alpha, \beta$

$\overline{\mathsf{Alg}}$

Constants: $\mathsf{input}^*, \mathsf{output}^*, u^*, e^*, \underline{\tilde{r}}$ 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 $\mathsf{output} = \mathsf{Alg}(\mathsf{input}; x)$.

Figure 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 $e = (\alpha, \beta)$.

Figure 18: Program $\overline{\mathsf{Explain}}$

4. Select u^* at random. Select $\tilde{r}$ at random.
5. Let x^* be chosen randomly and let $\mathsf{output}^* = \mathsf{Alg}(\mathsf{input}^*; x^*)$.
6. Do the following. Let α^* be random. Let β^* be random. Set $e^* = (\alpha^*, \beta^*)$.
7. Let $\widetilde{\mathsf{Alg}} \leftarrow i\mathcal{O}(\overline{\mathsf{Alg}})$ for $\overline{\mathsf{Alg}}$ as in Figure 19. Let $\mathsf{Explain} \leftarrow i\mathcal{O}(\overline{\mathsf{Explain}})$ 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 $b' \leftarrow \mathcal{A}(\widetilde{\mathsf{Alg}}, \mathsf{output}^*, e^*)$.

In Hybrid 7 we observe that the variables e^*, u^* are now both independently uniformly random strings, and they are treated entirely symmetrically. (Recall that the "if" statements above have the conditions written in lexicographic order, so they do not reveal any asymmetry between e^* and u^*.) Thus, the distributions output by this Hybrid for $b = 0$ and $b = 1$ are identical, and therefore even an unbounded adversary outputs $b = b'$ with probability exactly 1/2. ■

The proof above made use of the following lemma for arguing that the programs obfuscated by the indistinguishability obfuscator 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.*

$\overline{\mathsf{Alg}}$

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 $\mathsf{output} = \mathsf{Alg}(\mathsf{input}; x)$.

Figure 19: Program $\overline{\mathsf{Alg}}$

$\overline{\mathsf{Explain}}$

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 $e = (\alpha, \beta)$.

Figure 20: Program $\overline{\mathsf{Explain}}$

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 Step 2 check 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. Observe that 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 functionality. On the other hand, recall that every $u^*[1]$ is chosen at random, and therefore the probability that $u^*[1]$ would be in the image of $F_2(K_2, \cdot)$ is negligible, therefore with high probability over the choice of constants $u^*[1]$, Step 2 check 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 $e^*[1]$. ∎