How to Generate and use Universal Parameters

Dakshita Khurana
UCLA
dakshita@cs.ucla.edu

Amit Sahai [*]
UCLA
sahai@cs.ucla.edu

Brent Waters [†]
University of Texas at Austin
bwaters@cs.utexas.edu

June 27, 2014

Abstract

We introduce the notion of *universal parameters* as a method for generating the trusted parameters for many schemes from just a single trusted setup. In such a scheme a trusted setup process will produce universal parameters U. These parameters can then be combined with the description, $d(\cdot)$ of any particular cryptographic setup algorithm to produce parameters p_d that can be used by the cryptographic system associated with d. We give a solution in the random oracle model based on indistinguishability obfuscation.

[*]Research of first two authors supported in part from a DARPA/ONR PROCEED award, NSF grants 1228984, 1136174, 1118096, and 1065276, a Xerox Faculty Research Award, a Google Faculty Research Award, an equipment grant from Intel, and an Okawa Foundation Research Grant. This material is based upon work supported by the Defense Advanced Research Projects Agency through the U.S. Office of Naval Research under Contract N00014-11-1-0389. The views expressed are those of the author and do not reflect the official policy or position of the Department of Defense, the National Science Foundation, or the U.S. Government.

[†]Supported by NSF CNS-0915361 and CNS-0952692, CNS-1228599 DARPA through the U.S. Office of Naval Research under Contract N00014-11-1-0382, Google Faculty Research award, the Alfred P. Sloan Fellowship, Microsoft Faculty Fellowship, and Packard Foundation Fellowship.

Contents

1 Introduction

Many cryptographic systems rely on the trusted generation of common parameters to be used by participants. There are several types of reasons for using such parameters. For example, many cutting edge cryptographic protocols rely on the generation of a common reference string.[1] Constructions for other primitives such as aggregate signatures [BGLS03] or batch verifiable signatures [CHP12] require all users to choose their public keys using the same algebraic group structure. Finally, common parameters are sometimes for convenience and efficiency — such as when generating an EC-DSA public signing key, one can choose the elliptic curve parameters from a standard set and avoid the cost of completely fresh selection.

In most of these systems it is extremely important to make sure that the parameters were indeed generated in a trustworthy manner and failure to do so often results in total loss of security. In cryptographic protocols that explicitly create a common reference string it is obvious how and why a corrupt setup results in loss in security. In other cases security breaks are more subtle. A prominent recent example is the case of the elliptic curves parameters standardized in NIST Special Publications 800-90 [NIS12] for the Dual Elliptic Curve Deterministic Random Bit Generator (Dual EC) used in RSA's BSAFE product. Based on news articles [LPS13, BBG13, PLS13] that reported the Snowden leaks, it is speculated that these parameters may have been chosen with a trapdoor that allows subversion of the system. Recent research has shown [CFN$^+$14] that such trapdoors can lead to practical exploits.

Given these threats it is important to establish a trusted setup process that engenders the confidence of all users, even though users will often have competing interests and different trust assumptions. Realizing such trust is challenging and requires a significant amount of investment. For example, we might try to find a single trusted authority to execute the process. Alternatively, we might try to gather different parties that represent different interests and have them jointly execute a trusted setup algorithm using secure multiparty computation. For example, one could imagine gathering disparate parties ranging from the Electronic Frontier Foundation, to large corporations, to national governments.

Pulling together such a trusted process requires a considerable investment. While we typically measure the costs of cryptographic processes in terms of computational and storage costs, the organizational overhead of executing a trusted setup may often be the most significant barrier to adoption of a new cryptographic system. Given the large number of current and future cryposystems, it is difficult to imagine that a carefully executed trusted setup can be managed for each one of these. In this work we attempt to address this problem by asking an ambitious question:

Can a single trusted setup output a set of universal parameters,
which can (securely) serve all cryptographic protocols?

Ideally, we want a universal setup process, that when executed will produce what we refer to as universal parameters. Suppose a group of users wishes to use a certain protocol, where d is the

[1] Several cryptographic primitives (e.g. NIZKs) are realizable using only a common *random* string and thus only need access to a trusted random source for setup. However, many cutting edge constructions need to use a common *reference* string that is setup by some private computation. For example, the recent two-round MPC protocol of Garg et. al. [GGHR14] uses a trusted setup phase that generates public parameters drawn from a nontrivial distribution, where the randomness underlying the specific parameter choice needs to be kept secret.

description of the setup algorithm. Any member of the group can use the universal parameters and d to (deterministically) compute the shared parameters for the scheme in question. We refer to these as the induced parameters. If the universal parameters were securely generated, then the induced parameters should be "as good as" if there were directly derived from a trusted setup.[2]

Our Approach. We now describe our approach. We begin with a high level overview of the definition we wish to satisfy; details of the definition are in Section 3. In our system there is a universal parameter generation algorithm, `UniversalGen`, which is called with a security parameter 1^n and randomness r. The output of this algorithm will be a set of universal parameters U. In addition, there is a second algorithm `InduceGen` which takes as input universal parameters U and the (circuit) description of a setup algorithm, d, and outputs the induced parameters p_d.

We model security as an ideal/real game. In the real game an attacker will receive a set of universal parameters U produced from the universal parameter generation algorithm. Next, it will query an oracle on multiple setup algorithm descriptions $d_1, \ldots, d_q$ and iteratively gets back $p_i = \texttt{InduceGen}(U, d_i)$ for $i = 1, \ldots, q$. In the ideal world the attacker will first get a set of universal parameters U as before. However, when he queries on d_i the challenger will reply back with $p_i = d_i(r_i)$ for fresh randomness r_i. (Since the universal parameter generation algorithm is deterministic we only let a particular d value be queried once without loss of generality.) A scheme is secure if no poly-time attacker can distinguish between the real and ideal game with non-negligible advantage after observing their transcripts. [3]

To make progress toward our eventual solution we begin by considering a relaxed security notion. We relax the definition in two ways: (1) we consider where the attacker makes only a single query to the oracle and (2) he commits to the query statically (a.k.a. selectively) before seeing the universal parameters U. While this security notion is too weak to satisfy our long term goals, developing a solution will serve as step towards our final solution and provide insights.

We achieve this selective and bounded notion of security by using indistinguishability obfuscation by applying punctured programming [SW14] techniques. In our solution we consider setup programs to all come from a polynominal circuit family of size $\ell(\lambda)$, where each setup circuit d takes in $m(\lambda)$ bits and outputs parameters of $k(\lambda)$ bits. The polynomials of ℓ, m, k are fixed for a class of systems; we often will drop the dependence on λ when it is clear from context.

The construction and proof is a fairly straightforward use of the Sahai-Waters [SW14] punctured programming methodology. The `UniversalGen` algorithm will first choose a puncturable pseudo random function (PRF) key K for function F where $F(K, \cdot)$ takes as input a circuit description d and outputs parameters $p \in \{0,1\}^k$. The universal parameters are created as an obfuscation of a program that on input d computes and output $d(F(K, d))$. To prove security we perform a hybrid argument between the real and ideal games in the 1-bounded and selective model. First, we puncture out d^*, the single program that the attacker queried on, from K to get the punctured key $K(d^*)$. We change the parameters to be an obfuscation of the program which uses $K(d^*)$ to

[2]There are some cryptosystems such as Identity-Based Encryption [Sha85, BF03] where an authority will run a setup algorithm that generates public parameters as well as some secret parameters that it uses later on. (E.g., to generate IBE private keys.) In this work we are only interested in a trusted setup that produces common parameters and then requires no further intervention.

[3]In our actual formalization in Section 3 we state the games differs slightly, however, the above description will suffice for this discussion.

compute the program for any $d \neq d^*$. And for $d = d^*$ we simply hardwire in the output z where $z = d(1^n, F(K, d))$. This computation is functionally equivalent to the original program — thus indistinguishability of this step from the previous follows from indistinguishability obfuscation. In this next step, we change the hardwired value from to $d(r)$ for freshly chosen randomness $r \in \{0,1\}^m$. This completes the transition to the ideal game.

Achieving Adaptive Security. We now turn our attention to achieving our original goal of universal parameter generation for adaptive security. While selective security might be sufficient in some limited situations, the adaptive security notion covers many plausible real world attacks. For instance, suppose a group of people perform a security analysis and agree to use a certain cryptographic protocol and its corresponding setup algorithm. However, for any one algorithm there will be a huge number of functionally equivalent implementations. In a real life setting an attacker could choose one of these implementations based on the universal parameters and might convince the group to use this one. A selectively secure system is not necessarily secure against such an attack, while this is captured by the adaptive model.

Obtaining a solution for our original security definition will be significantly more difficult. We first make a trivial observation that no standard model proof can satisfy our definition. The reasons for such are akin to those for non-committing encryption [Nie02] — once universal parameters of a fixed polynomial size are fixed, it is not possible for a standard model proof to make an unbounded number of parameters consistent with the already-fixed universal parameters. For this reason we turn our attention to proofs in the random oracle model, where such limitations can be overcome, and meaningful security guarantees can be achieved.

A tempting idea is to simply replace the puncturable PRF call from our last construction with a call to a hash function modeled as a random oracle. However, this is problematic. When applying the random oracle heuristic we will model the hash function as a oracle, yet at the same time we will need its code for creating the obfuscated program — thus arriving at an impasse. Therefore we need to consider a solution where random oracle calls are only possible outside of obfuscated programs. This complicates matters considerably, since the obfuscated program then has no way of knowing whether a setup program d is connected to a particular hash output[4].

A new proof technique: delayed backdoor programming. To solve our problem we develop a novel way of allowing what we call "delayed backdoor programming" using a random oracle. In our construction users will be provided with universal parameters which consists of an obfuscated program U (produced from `UniversalGen`) as well as a hash function H modeled as a random oracle and use these to determine the induced parameters parameters for a particular protocol.

Let's begin by seeing how `UniversalGen` creates a program, P, that will be obfuscated to create U. The program will actually follow a similar structure to the Encryption algorithm from the Sahai-Waters deniable encryption solution. The program consists of two main stages. In the first stage, the program takes its input w and checks to see if it encodes a "hidden trigger" using secret key information. This step will either output a string $x \in \{0,1\}^n$ or it will output $\perp$ to indicate failure. For a uniformly randomly chosen string w, this step will fail to decode with very high probability

[4] Recently, Hofheinz [Hof14] also applied obfuscation in the random oracle model in the context of building constrained PRFs and needed to develop techniques for using an external hash output. However, his goals and methods diverge from ours.

since trigger values are encoded sparsely. Moreover, without the secret information it should be difficult to distinguish a hidden trigger value from a uniformly sampled string. At this point if the decoding succeeds the program will simply output x.

Otherwise it will move into its second stage. At this point the program will compute randomness $r = F(K, w)$ for a puncturable PRF F. Now instead of directly computing the induced parameters we add a level of indirection. The program will next run the Setup algorithm for a one-bounded universal parameter generation scheme — in particular it could call the scheme we just illustrated above. [5] Now the recently generated 1-bounded universal parameters U_w are output.

We now show how to generate a set of induced parameters by executing $\texttt{InduceGen}(\texttt{U}, \texttt{d})$. The algorithm will first call $H(d) = w$. Next, it will compute $U_w = U(w)$. Finally, it obtains the induced parameters by computing $p_d = U_w(d)$. The extra level of indirection is critical for obtaining our proof of security.

We now overview how our proof of security proceeds. At the highest level the goal of our proof is to construct a sequence of hybrids where parameter generation is "moved" from being directly computed by the second stage of U (as in the real game) to where the parameters for setup algorithm d are being programmed in by the first stage hidden trigger mechanism via the input $w = H(d)$. Any poly-time algorithm $\mathcal{A}$ will make at most a polynomial number $Q = Q(\lambda)$ (unique) queries $d_1, \ldots, d_Q$ to the random oracle with RO outputs $w_1, \ldots, w_Q$. We will perform a hybrid of Q outer steps where at outer step i we move from using U_{w_i} to compute the induced parameters for d_i, to having the induced parameter for d_i being encoded in w_i itself.

Let's zoom in on one of these transitions. We note that the reduction algorithm will first choose the output w_i of the i-th (unique) random oracle query call randomly, but decide it *before* the universal parameters U are published. The first step uses punctured programming techniques to replace the normal computation of the 1-time universal parameters U_{w_i} with a hardwired and randomly sampled value for U_{w_i}. However, this value must still be programmed ahead of time when the universal parameters U is set and cannot be specialized to d_i, which is known to the reduction only when a random oracle call is made later.

In our next step we prepare a "hand-off" operation where we move the source of the one time parameters U_{w_i} to the hidden trigger. In the prior step of the hybrid the string w_i and one time universal parameters $U' = U_{w_i}$ were chosen independently. In the next hybrid step[6] we first choose U' independently and then set w_i to be a hidden trigger encoding of U'. At this point on calling $U(w_i)$ the program will get $U_{w_i} = U'$ from the Stage 1 hidden trigger detection and never proceed to Stage 2. Since the second stage is no longer used, we can use obfuscation security to return to the normal point where U' is no longer hardwired in — thus freeing up the a-priori-bounded "hardwiring resources" for future outer hybrid steps.

Interestingly, all proof steps to this point were independent of the actual program d_i. We observe that this fact is essential to our proof since the reduction was able to choose ahead of time and program the "setup program neutral" one-time parameters U_{w_i} ahead of time into U which had to be published well before d_i was known. However, at this point U_{w_i} comes programmed in from the

[5] In our construction of Section 5 we actually directly put our one bounded scheme into the construction. However, we believe this can be be adapted to work for any one bounded scheme.

[6]This is actually performed by a sequence of smaller steps in our proof. We simplify to bigger steps in this overview.

random oracle output which is *after* the call to $H(d_i)$ was made. Therefore we will be able to use our previous techniques from the selective case to move $U_{w_i}(d_i)$ to the ideally generated parameters $d_i(r)$, but only after the oracle call $H(d_i)$ is made.

We believe our "delayed backdoor programming" technique may be useful in other situations where an unbounded number of backdoors are needed in a program of fixed size.

2 Preliminaries

2.1 Indistinguishability Obfuscation and PRFs

In this section, we define indistinguishability obfuscation, and variants of pseudo-random functions (PRFs) that we will make use of. All variants of PRFs that we consider will be constructed from one-way functions.

Indistinguishability Obfuscation. The definition below is adapted from [GGH$^+$13]; the main difference with previous definitions is that we uncouple the security parameter from the circuit size by directly defining indistinguishability obfuscation for all circuits:

Definition 1 (Indistinguishability Obfuscator (iO)). *A uniform PPT machine iO is called an* indistinguishability obfuscator *for circuits if the following conditions are satisfied:*

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

$$\Pr[C'(x) = C(x) : C' \leftarrow iO(\lambda, C)] = 1$$

- *For any (not necessarily uniform) PPT adversaries Samp, D, there exists a negligible function α such that the following holds: if $\Pr[|C_0| = |C_1|$ and $\forall x, C_0(x) = C_1(x) : (C_0, C_1, \sigma) \leftarrow Samp(1^\lambda)] > 1 - \alpha(\lambda)$, then we have:*

$$\Big| \Pr\big[D(\sigma, iO(\lambda, C_0)) = 1 : (C_0, C_1, \sigma) \leftarrow Samp(1^\lambda)\big]$$
$$-\Pr\big[D(\sigma, iO(\lambda, C_1)) = 1 : (C_0, C_1, \sigma) \leftarrow Samp(1^\lambda)\big] \Big| \leq \alpha(\lambda)$$

Such indistinguishability obfuscators for circuits were constructed under novel algebraic hardness assumptions in [GGH$^+$13].

PRF variants. We first consider some simple types of constrained PRFs [BW13, BGI13, KPTZ13], where a PRF is only defined on a subset of the usual input space. We focus on *puncturable* PRFs, which are PRFs that can be defined on all bit strings of a certain length, except for any polynomial-size set of inputs:

Definition 2. *A* puncturable *family of PRFs F is given by a triple of Turing Machines* Key_F, $\mathrm{Puncture}_F$*, and* Eval_F*, and a pair of computable functions $n(\cdot)$ and $m(\cdot)$, satisfying the following conditions:*

- ***[Functionality preserved under puncturing]*** *For every PPT adversary A such that $A(1^\lambda)$ outputs a set $S \subseteq \{0,1\}^{n(\lambda)}$, then for all $x \in \{0,1\}^{n(\lambda)}$ where $x \notin S$, we have that:*

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

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

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

where $\mathrm{Eval}_F(K,S)$ denotes the concatenation of $\mathrm{Eval}_F(K,x_1)), \ldots, \mathrm{Eval}_F(K,x_k))$ where $S = \{x_1,\ldots,x_k\}$ is the enumeration of the elements of S in lexicographic order, $negl(\cdot)$ is a negligible function, and U_ℓ denotes the uniform distribution over ℓ bits.

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

The GGM tree-based construction of PRFs [GGM84] from one-way functions are easily seen to yield puncturable PRFs, as recently observed by [BW13, BGI13, KPTZ13]. Thus we have:

Theorem 1. *[GGM84, BW13, BGI13, KPTZ13] If one-way functions exist, then for all efficiently computable functions $n(\lambda)$ and $m(\lambda)$, there exists a puncturable PRF family that maps $n(\lambda)$ bits to $m(\lambda)$ bits.*

3 Definitions

In this section, we describe our definitional framework for universal parameter schemes. The essential property of a universal parameter scheme is that given the universal parameters, and given any program d that generates parameters from randomness (subject to certain size constraints, see below), it should be possible for any party to use the universal parameters and the description of d to obtain induced parameters that look like the parameters that d would have generated given uniform and independent randomness.

We will consider two definitions – a simpler definition promising security for a single arbitrary but fixed protocol, and a more complex definition promising security in a strong adaptive sense against many protocols chosen after the universal parameters are fixed. All our security definitions follow a “Real World” vs. “Ideal World” paradigm.

Before we proceed to our definitions, we will first set up some notation and conventions that all our definitions will respect:

- We will consider programs d that are bounded in the following ways: Note that we will use d to refer to both the program, and the description of the program. Below, $\ell(\lambda), m(\lambda)$, and $k(\lambda)$ are all computable polynomials. The description of d is as an $\ell(\lambda)$-bit string describing a circuit[7] implementing d. The program d takes as input $m(\lambda)$ bits of randomness, and outputs parameters of length $k(\lambda)$ bits. Without loss of generality, we assume that $\ell(\lambda) \geq \lambda$ and $m(\lambda) \geq \lambda$. When context is clear, we omit the dependence on the security parameter λ. The quantities (ℓ, m, k) are bounds that are set during the setup of the universal parameter scheme.
- We enforce that every ℓ-bit description of d yields a circuit mapping m bits to k bits; this can be done by replacing any invalid description with a default circuit satisfying these properties.
- We will sometimes refer to the program d that generates parameters as a "protocol". This is to emphasize that d is designed to generate parameters for some protocol.

A universal parameter scheme consists of two algorithms:

(1) The first randomized algorithm `UniversalGen` takes as input a security parameter 1^λ and outputs universal parameters U.

(2) The second algorithm `InduceGen` takes as input universal parameters U and a circuit d of size at most ℓ, and outputs induced parameters p_d.

Intuition. Before giving formal definitions, we will now describe the intuition behind our definitions. We want to formulate security definitions that guarantee that induced parameters are indistinguishable from honestly generated parameters to an arbitrary interactive system of adversarial and honest parties.

We first consider an "ideal world," where a trusted party, on input a program description d, simply outputs $d(r_d)$ where r_d is independently chosen true randomness, chosen once and for all for each given d. In other words, if F is a truly random function, then the trusted party outputs $d(F(d))$. In this way, if any party asks for the parameters corresponding to a specific program d, they are all provided with the same honestly generated value. This corresponds precisely to the shared trusted public parameters model in which protocols are typically constructed.

In the real world, however, all parties would only have access to the trusted *universal* parameters. Parties would use the universal parameters to derive induced parameters for any specific program d. Following the ideal/real paradigm, we would like to argue that for any adversary that exists in the real world, there should exist an equivalently successful adversary in the ideal world. However, the general scenario of an interaction between multiple parties, some malicious and some honest, interacting in an arbitrary security game would be cumbersome to model in a definition. To avoid this, we note that the only way that honest parties ever use the universal parameters is to execute the parameter derivation algorithm using the universal parameters and some program descriptions d (corresponding to the protocols in which they participate) to obtain derived parameters, which these honest parties then use in their interactions with the adversary.

[7]Note that if we assume *iO* for Turing Machines, then we do not need to restrict the size of the description of d. Candidates for *iO* for Turing Machines were given by [ABG+13, BCP14].

Thus, instead of modeling these honest parties explicitly, we can "absorb" them into the adversary, as we now explain: We will require that for every real-world adversary $\mathcal{A}$, there exists a simulator $\mathcal{S}$ that can provide simulated universal parameters U to the adversary such that these simulated universal parameters U actually induce the completely honestly generated parameters $d(F(d))$ created by the trusted party: in other words, that $\texttt{InduceGen}(U, d) = d(F(d))$. Note that since honest parties are instructed to simply honestly compute induced parameters, this ensures that honest parties in the ideal world would obtain these completely honestly generated parameters $d(F(d))$. Thus, we do not need to model the honest parties explicitly – the adversary $\mathcal{A}$ can internally simulate any honest parties. By the condition we impose on the simulation, these honest parties would have the correct view in the ideal world.

Selective (and bounded) vs. Adaptive (and unbounded) Security. We explore two natural formulations of the simulation requirement. The simpler variant is the *selective* case, where we require that the adversary declare at the start a single program d^* on which it wants the ideal world simulator to enforce equality between the honestly generated parameters $d^*(F(d^*))$ and the induced parameters $\texttt{InduceGen}(U, d^*)$. This simpler variant has two advantages: First, it is achievable in the standard model. Second, it is achieved by natural and simple construction based on indistinguishability obfuscation.

However, ideally, we would like our security definition to capture a scenario where universal parameters U are set, and then an adversary can potentially adaptively choose a program d for generating parameters for some adaptively chosen application scenario. For example, there may be several plausible implementations of a program to generate parameters, and an adversary could influence which specific program description d is used for a particular protocol. Note, however, that such an adaptive scenario is trivially impossible to achieve in the standard model: there is no way that a simulator can publish universal parameters U of polynomial size, and then with no further interaction with the adversary, force $\texttt{InduceGen}(U, d^*) = d^*(F(d^*))$ for a d^* chosen after U has already been declared. This impossibility is very similar to the trivial impossibility for reusable non-interactive non-committing public-key encryption [Nie02] in the plain model. Such causality problems can be addressed, however, in the random-oracle model. As discussed in the introduction, the sound use of the random oracle model together with obfuscation requires care: we do not assume that the random oracle itself can be obfuscated, which presents an intriguing technical challenge.

Furthermore, we would like our universal parameters to be useful to obtain induced parameters for an unbounded number of other application scenarios. We formulate and achieve such an adaptive unbounded definition of security in the random oracle model.

3.1 Selective One-Time Universal Parameters

We now proceed to our first formal definition, of a selective one-time secure universal parameter scheme.

Definition 3 (Selectively-Secure One-Time Universal Parameters Scheme). *Let $\ell(\lambda), m(\lambda), k(\lambda)$ be efficiently computable polynomials. A pair of efficient algorithms* $(\texttt{UniversalGen}, \texttt{InduceGen})$ *is a selectively-secure one-time universal parameters scheme if there exists an efficient algorithm* $\texttt{SimUGen}$ *such that:*

- *There exists a negligible function negl such that for all circuits d of length ℓ, taking m bits of input, and outputting k bits, and for all strings $p_d \in \{0,1\}^k$, we have that:*

$$\Pr[\texttt{InduceGen}(\texttt{SimUGen}(1^\lambda, d, p_d), d) = p_d] = 1 - negl(\lambda)$$

- *For every efficient adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$, where $\mathcal{A}_2$ outputs one bit, there exists a negligible function negl such that the following holds. Consider the following two experiments:*

  The experiment $\texttt{Real}(1^\lambda)$ *is as follows:*

  1. $(d^*, \sigma) \leftarrow \mathcal{A}_1(1^\lambda)$
  2. *Output* $\mathcal{A}_2(\texttt{UniversalGen}(1^\lambda), \sigma)$.

  The experiment $\texttt{Ideal}(1^\lambda)$ *is as follows:*

  1. $(d^*, \sigma) \leftarrow \mathcal{A}_1(1^\lambda)$
  2. *Choose r uniformly from $\{0,1\}^m$.*
  3. *Let $p_d = d^*(r)$.*
  4. *Output* $\mathcal{A}_2(\texttt{SimUGen}(1^\lambda, d^*, p_d), \sigma)$.

  Then we have:

$$\left|\Pr[\texttt{Real}(1^\lambda) = 1] - \Pr[\texttt{Ideal}(1^\lambda) = 1]\right| = negl(\lambda)$$

3.2 Adaptively Secure Universal Parameters

We now proceed to define universal parameter schemes for the adaptive setting in the random oracle model. Our definition will also handle an unbounded number of induced parameters simultaneously. Recall that in the random oracle model, the indistinguishability obfuscation *cannot* obfuscate circuits that call the random oracle. Thus, the random oracle can only be used outside of obfuscated programs.

Definition 4 (Adaptively-Secure Universal Parameters Scheme)**.** *Let $\ell(\lambda), m(\lambda), k(\lambda)$ be efficiently computable polynomials. A pair of efficient oracle algorithms* ($\texttt{UniversalGen}, \texttt{InduceGen}$) *is an adaptively-secure universal parameters scheme if there exist efficient interactive Turing Machines* $\texttt{SimUGen}, \texttt{SimRO}$ *such that:*

- *We define an* admissible *adversary $\mathcal{A}$ to be an efficient interactive Turing Machine that outputs one bit, with the following input/output behavior:*
  - *$\mathcal{A}$ initially takes as input a security parameter and a universal parameter U.*
  - *$\mathcal{A}$ can send a message (RO, x) corresponding to a random oracle query. In response, $\mathcal{A}$ expects to receive the output of the random oracle on input x.*
  - *$\mathcal{A}$ can send a message (params, d), where d is a circuit of length ℓ, taking m bits of input, and outputting k bits. The adversary does not expect any response to this message. Instead, upon sending this message, $\mathcal{A}$ is required to honestly compute $p_d =$ $\texttt{InduceGen}(U, d)$, making use of any additional random oracle queries, and $\mathcal{A}$ appends (d, p_d) to an auxiliary tape.*

Remark. Intuitively, (params, d) messages correspond to an honest party seeking parameters generated by program d. Recall that $\mathcal{A}$ is meant to internalize the behavior of honest parties.

- *Consider the following two experiments:*

  The experiment $\mathtt{Real}(1^\lambda)$ *is as follows:*

  1. *Throughout this experiment, a random oracle* $\mathcal{H}$ *is implemented by assigning random outputs to each unique query made to* $\mathcal{H}$.
  2. $U \leftarrow \mathtt{UniversalGen}^{\mathcal{H}}(1^\lambda)$
  3. $\mathcal{A}(1^\lambda, U)$ *is executed, where every message of the form* (RO, x) *receives the response* $\mathcal{H}(x)$.
  4. *Upon termination of* $\mathcal{A}$*, the output of the experiment is the final output of the execution of* $\mathcal{A}$.

  The experiment $\mathtt{Ideal}(1^\lambda)$ *is as follows:*

  1. *A truly random function* F *that maps* ℓ *bits to* m *bits is implemented by assigning random* m*-bit outputs to each unique query made to* F. *Throughout this experiment, a Parameters Oracle* $\mathcal{O}$ *is implemented as follows: On input* d*, where* d *is a circuit of length* ℓ*, taking* m *bits of input, and outputting* k *bits,* $\mathcal{O}$ *outputs* $d(F(d))$.
  2. $(U, \tau) \leftarrow \mathtt{SimUGen}(1^\lambda)$. *Here,* $\mathtt{SimUGen}$ *can make arbitrary queries to the Parameters Oracle* $\mathcal{O}$.
  3. $\mathcal{A}(1^\lambda, U)$ *and* $\mathtt{SimRO}(\tau)$ *begin simultaneous execution. Messages for* $\mathcal{A}$ *or* $\mathtt{SimRO}$ *are handled as follows:*
     - *Whenever* $\mathcal{A}$ *sends a message of the form* (RO, x)*, this is forwarded to* $\mathtt{SimRO}$*, which produces a response to be sent back to* $\mathcal{A}$.
     - $\mathtt{SimRO}$ *can make any number of queries to the Parameter Oracle* $\mathcal{O}$.
     - *Finally, after* $\mathcal{A}$ *sends any message of the form* (params, d)*, the auxiliary tape of* $\mathcal{A}$ *is examined until an entry of the form* (d, p_d) *is added to it. At this point, if* p_d *is not equal to* $d(F(d))$*, then experiment aborts and we say that an "Honest Parameter Violation" has occurred. Please note that this is the only way that the experiment* $\mathtt{Ideal}$ *can abort. If the adversary "aborts", then we consider this to simply induce an output of zero by the adversary, not an abort of the* $\mathtt{Ideal}$ *experiment.*
  4. *Upon termination of* $\mathcal{A}$*, the output of the experiment is the final output of the execution of* $\mathcal{A}$.

  We require that for every efficient admissible adversary $\mathcal{A}$*, there exists a negligible function* $negl$ *such that the following two conditions hold:*

$$\Pr[\mathtt{Ideal}(1^\lambda) \textit{ aborts}] < negl(\lambda)$$

  and

$$\left|\Pr[\mathtt{Real}(1^\lambda) = 1] - \Pr[\mathtt{Ideal}(1^\lambda) = 1]\right| = negl(\lambda)$$

4 Selective One-Time Universal Parameters

In this section, we prove the following theorem:

Theorem 2 (Selective One-Time Universal Parameters)**.** *If indistinguishability obfuscation and one-way functions exist, then there exists a selectively secure one-time universal parameters scheme, according to Definition 3.*

The required Selective One-Time Universal Parameters Scheme consists of programs `UniversalGen` and `InduceGen`.

- `UniversalGen`(1^λ) first samples the key K for a PRF that takes ℓ bits as input and outputs m bits. It then sets Universal Parameters U to be an indistinguishability obfuscation of the program[8] Selective-Single-Parameters in Figure 1. It outputs U.
- `InduceGen`(U, d) runs the program U on input d to generate and output $U(d)$.

```
                    Selective-Single-Parameters

Constant: PRF key $K$.
Input: Program description $d$.
    1. Output $d(F(K, d))$. Recall that $d$ is a program description which outputs $k$ bits.
```

Figure 1: Program Selective-Single-Parameters

[8]Appropriately padded to the maximum of the size of itself and Program Selective-Single-Parameters: 2

4.1 Hybrids

We prove security by a sequence of hybrid arguments, starting with the original experiment Hybrid_0 in the Real World and replacing the output at d^* with an external Parameters in the final hybrid (Ideal World). We denote changes between subsequent hybrids using red underlined font.

Each hybrid below is an experiment that takes as input 1^λ. The final output of each hybrid experiment is the output produced by the adversary when it terminates.

Hybrid_0:

- The adversary picks protocol description d^* and sends it to the challenger.
- The challenger picks PRF key K and sends the adversary an iO of the program[9] Selective-Single-Parameters in Figure 2.
- The adversary queries the program on input d^* to obtain the Parameters.

Selective-Single-Parameters

Constant: PRF key K.
Input: Program description d.

1. Output $d(F(K, d))$. Recall that d is a program description which outputs k bits.

Figure 2: Program Selective-Single-Parameters

Hybrid_1:

- The adversary picks protocol description d^* and sends it to the challenger.
- The challenger picks PRF key K, sets $f^* = d^*(F(K, d^*))$, punctures K at d^* and sends the adversary an iO of the program[10] Selective-Single-Parameters: 2 in Figure 3.
- The adversary queries the program on input d^* to obtain the Parameters.

Selective-Single-Parameters: 2

Constant: PRF key $K\{d^*\}, d^*, f^*$.
Input: Program description d.

1. If $d = d^*$ output f^*.
2. Output $d(F(K, d))$. Recall that d is a program description which outputs k bits.

Figure 3: Program Selective-Single-Parameters: 2

[9] Appropriately padded to the maximum of the size of itself and Program Selective-Single-Parameters: 2

[10] Appropriately padded to the maximum of the size of itself and Program Selective-Single-Parameters

Hybrid_2:

- The adversary picks protocol description d^* and sends it to the challenger.
- The challenger picks PRF key K, picks $x \leftarrow \{0,1\}^m$, sets $f^* = d^*(x)$, punctures K at d^* and sends the adversary an iO of the program[11] Selective-Single-Parameters: 2 in Figure 4.
- The adversary queries the program on input d^* to obtain the Parameters.

Selective-Single-Parameters: 2

Constant: PRF key $K\{d^*\}$, d^*, f^*.
Input: Program description d.

1. If $d = d^*$ output f^*.
2. Output $d(F(K,d))$. Recall that d is a program description which outputs k bits.

Figure 4: Program Selective-Single-Parameters: 2

Hybrid_3:

- This hybrid describes how `SimUGen` works.
- The adversary picks protocol description d^* and sends it to the challenger.
- The challenger executes $\texttt{SimUGen}(1^\lambda, d^*)$, which does the following: It picks PRF key K, sets $f^* = p_d$ for externally obtained Parameters p_d, punctures K at d^* and outputs an iO of the program[12] Selective-Single-Parameters: 2 in Figure 5. This is then sent to the adversary.
- The adversary queries the program on input d^* to obtain the Parameters.

Selective-Single-Parameters: 2

Constant: PRF key $K\{d^*\}$, d^*, f^*.
Input: Program description d.

1. If $d = d^*$ output f^*.
2. Output $d(F(K,d))$. Recall that d is a program description which outputs k bits.

Figure 5: Program Selective-Single-Parameters: 2

[11] Appropriately padded to the maximum of the size of itself and Program Selective-Single-Parameters
[12] Appropriately padded to the maximum of the size of itself and Program Selective-Single-Parameters

4.2 Indistinguishability of the Hybrids

To prove Theorem 2, it suffices to prove the following claims,

Claim 1. $\mathsf{Hybrid}_0(1^\lambda)$ *and* $\mathsf{Hybrid}_1(1^\lambda)$ *are computationally indistinguishable.*

Proof. Hybrid_0 and Hybrid_1 are indistinguishable by security of iO, since the programs Selective-Single-Parameters and Selective-Single-Parameters: 2 are functionally equivalent.

Suppose not, then there exists a distinguisher $\mathcal{D}_1$ that distinguishes between the two hybrids. This can be used to break security of the iO via the following reduction to distinguisher $\mathcal{D}$.

$\mathcal{D}$ acts as challenger in the experiment of Hybrid_0. He activates the adversary $\mathcal{D}_1$ to obtain input d^* which he passes to the iO Samp algorithm. He also picks PRF key K and passes it to Samp. Samp on input d^* computes $f^* = d^*(F(K, d^*))$. Next, it samples circuit C_0 = Selective-Single-Parameters according to Figure 2 and C_1 = Selective-Single-Parameters: 2 according to Figure 3 with inputs d^*, f^*. He pads the circuits in order to bring them to equal size. It is easy to see that these circuits are functionally equivalent. Next, Samp gives circuit $C_x = iO(C_0)$ or $C_x = iO(C_1)$ to $\mathcal{D}$.

$\mathcal{D}$ continues the experiment of Hybrid_1 except that he sends the obfuscated circuit C_x instead of the obfuscation of Selective-Single-Parameters to the adversary $\mathcal{D}_1$. Since $\mathcal{D}_1$ has significant distinguishing advantage, there exists a polynomial $\mathsf{p}(\cdot)$ such that,

$$\Big|\Pr\big[\mathcal{D}_1(\mathsf{Hybrid}_0) = 1\big] - \Pr\big[\mathcal{D}_1(\mathsf{Hybrid}_1) = 1\big]\Big| \geq 1/\mathsf{p}(\lambda).$$

We note that Hybrid_0 and Hybrid_1 correspond exactly to C_x being C_0 and C_1 respectively, thus we can just have $\mathcal{D}$ echo the output of $\mathcal{D}_1$ such that the following is true, for $\alpha(\cdot) = 1/\mathsf{p}(\cdot)$

$$\Big|\Pr\big[\mathcal{D}(\sigma, iO(n, C_0)) = 1 : (C_0, C_1, \sigma) \leftarrow Samp(1^\lambda)\big]$$
$$-\Pr\big[\mathcal{D}(\sigma, iO(n, C_1)) = 1 : (C_0, C_1, \sigma) \leftarrow Samp(1^\lambda)\big]\Big| \geq \alpha(\lambda)$$

□

Claim 2. $\mathsf{Hybrid}_0(1^\lambda)$ *and* $\mathsf{Hybrid}_1(1^\lambda)$ *are computationally indistinguishable.*

Proof. Hybrid_1 and Hybrid_2 are indistinguishable by security of the punctured PRF $K\{d^*\}$.

Suppose they are not, then consider an adversary $\mathcal{D}_2$ who distinguishes between these hybrids with significant advantage.

This adversary can be used to break *selective* security of the punctured PRF K via the following reduction algorithm to distinguisher $\mathcal{D}$, that first gets the protocol d^* after activating the distinguisher $\mathcal{D}_2$.

The PRF challenger gives the challenge a to the PRF attacker $\mathcal{D}$, which is either the output of the PRF at d^* or is set uniformly at random in $\{0,1\}^m$. $\mathcal{D}$ sets $f^* = d^*(a)$ and continues the experiment of Hybrid_1 against $\mathcal{D}_2$.

Then, there exists polynomial $p(\cdot)$ such that

$$\Big|\Pr\big[\mathcal{D}_2(\mathsf{Hybrid}_2) = 1\big] - \Pr\big[\mathcal{D}_2(\mathsf{Hybrid}_2) = 1\big]\Big| \geq 1/\mathsf{p}(\lambda).$$

If $\mathcal{D}_2$ predicts Hybrid_1, then a is the output of the punctured PRF K at d^*. If $\mathcal{D}_2$ predicts Hybrid_2, then a was chosen uniformly at random. Therefore, we can just have $\mathcal{D}$ echo the output of $\mathcal{D}_2$ such that

$$\Big|\Pr\big[\mathcal{D}(F(K\{d^*\}, d^*)) = 1\big] - \Pr\big[\mathcal{D}(y \leftarrow \{0,1\}^n) = 1\big]\Big| \geq 1/\mathsf{p}(\lambda).$$

□

Claim 3. $\mathsf{Hybrid}_2(1^\lambda)$ *and* $\mathsf{Hybrid}_3(1^\lambda)$ *are identical.*

Proof. Hybrid_2 and Hybrid_3 are identical since x is sampled uniformly at random in $\{0,1\}^n$. □

Claim 4.

$$\Pr[\texttt{InduceGen}(\texttt{SimUGen}(1^\lambda, d, p_d), d) = p_d] = 1$$

Proof. This follows from inspection of our construction, therefore condition (1) in Definition 3 is fulfilled. □

5 Adaptively Secure Universal Parameters

In this section, we prove the following theorem:

Theorem 3 (Adaptively Secure Universal Parameters). *If indistinguishability obfuscation and injective* PRG*s exist, then there exists an adaptively secure universal parameters scheme, according to Definition 4, in the Random Oracle Model.*

We now describe a scheme that fulfills the conditions of Theorem 3. This scheme consists of algorithms `UniversalGen` and `InduceGen`.

- `UniversalGen`(1^λ) first samples PRF keys K_1, K_2, K_2' and then sets Universal Parameters U to be an indistinguishability obfuscation of the program Adaptive-Parameters [13], Figure 6.

  The program takes as input a value u, where $|u| = n^2$ and a value v such that $|v| = n$, both obtained as the output of a random oracle $\mathcal{H}$ on input d. Here, n is defined to be the size of an iO of program[14] P_{K_3} is n. As such, n will be some fixed polynomial in the security parameter λ. The key to our proof is to instantiate the random oracle $\mathcal{H}$ appropriately to generate the Parameters for any protocol description d.

  Denote by $F_1^{(n)} = \{F_1^{1,0}, F_1^{1,1}, F_1^{2,0}, F_1^{2,1} \dots F_1^{n,0}, F_1^{n,1}\}$ a sequence of $2n$ puncturable PRF's that each take n-bit inputs and output n bits. For some key sequence $\{K_1^{1,0}, K_1^{1,1}, K_1^{2,0}, K_1^{2,1} \dots K_1^{n,0}, K_1^{n,1}\}$, denote the combined key by $K_1^{(n)}$. Then, on a n-bit input v_1, denote the combined output of the function $F_1^{(n)}$ using key $K_1^{(n)}$ by $F_1^{(n)}(K_1^{(n)}, v_1)$. Note that the length of this combined output is $2n^2$.

  Denote by F_2 a puncturable PRF that takes inputs of $n^2 + n$ bits and outputs n_1 bits, where n_1 is the size of the key K_3 for the program P_{K_3} in Figure 7. In particular, $n_1 = \lambda$. Denote by F_2' another puncturable PRF that takes inputs of $n^2 + n$ bits and outputs n_2 bits, where n_2 is the size of the randomness r used by the iO given the program P_{K_3} in Figure 7. Denote by F_3 another puncturable PRF that takes inputs of ℓ bits and outputs m bits. Denote by PRG an *injective length-doubling* pseudo-random generator that takes inputs of n bits and outputs $2n$ bits.

  Note m is the size of uniform randomness accepted by $d(\cdot)$, k is the size of parameters generated by $d(\cdot)$.

- `InduceGen`(U, d) queries the random oracle $\mathcal{H}$ to obtain $(u, v) = \mathcal{H}(d)$. It then runs the program U generated by `UniversalGen`(1^λ) on input (u, v) to obtain as output the obfuscated program P. It now runs this program P on input d to obtain the required parameters.

[13]Note that this program must be padded appropriately to equal the maximum of the size of itself and other corresponding programs in various hybrids, as described in the next section.

[14]Appropriately padded to the maximum of the size of itself and P'_{K_3, p_j^*, d_j^*} in future hybrids

Adaptive-Parameters

Constants: PRF keys $K_1^{(n)}$, K_2, K_2'.
Input: Program hash $u = u[1], \ldots, u[n]$, v.

```
1. Compute $F_1(K_1^{(n)}, v) = (y_{1,0}, y_{1,1}), \ldots, (y_{1,0}, y_{1,1})$.
2. For $i = 1, \ldots, n$, if $u[i] = y_{i,0}$ set $x_i = 0$ else if $u[i] = y_{i,1}$ set $x_i = 1$ else set $x_i = \perp$
3. If $x \in \{0,1\}^n$ (i.e. no $\perp$s), output $x$.
4. Else set $K_3 = F_2(K_2, u|v)$, $r = F_2(K_2', u|v)$. Output $P = iO(P_{K_3}; r)$ of the program [a] $P_{K_3}$ of
   Figure 7.
```

[a]Appropriately padded to the maximum of the size of itself and P'_{K_3, p_j^*, d_j^*} in future hybrids

Figure 6: Program Adaptive-Parameters

P_{K_3}

Constant: PRF key K_3.
Input: Program description d.

```
1. Output $d(F_3(K_3, d))$. Recall that $d$ is a program description which outputs $k$ bits.
```

Figure 7: Program P_{K_3}

5.1 Security Game and Hybrids

We first convert the admissible adversary $\mathcal{A}$ - that is allowed to send *any* message (RO, x) or (params, d) - and construct a modified adversary, such that whenever $\mathcal{A}$ sends message (params, d), our modified adversary sends message (RO, d) and then sends message (params, d).

We prove security against this modified adversary. Note that it suffices to prove the security of our scheme with respect to such modified adversaries because this transformation does not change the output of the adversary in any way (that is, the modified adversary is functionally equivalent to the previous adversary). Because the modified adversary always provides protocol description d to the random oracle, our proof will not directly deal with messages of the form (params, d) and it will suffice to deal only with messages (RO, d) sent by the adversary.

We now prove, via a sequence of polynomial hybrids, that algorithms `UniversalGen` and `InduceGen` satisfy the security requirements of Definition 4 in the Random Oracle Model. We begin with Hybrid_0 corresponding to the real world in the security game described above. Suppose the adversary makes $q(\lambda)$ queries to the random oracle $\mathcal{H}$, for some polynomial $q(\cdot)$. The argument proceeds via the sequence of hybrids $\mathsf{Hybrid}_0, \mathsf{Hybrid}_{1,1}, \mathsf{Hybrid}_{1,2}, \ldots \mathsf{Hybrid}_{1,13}, \mathsf{Hybrid}_{2,1}, \mathsf{Hybrid}_{2,2} \ldots \mathsf{Hybrid}_{2,13} \ldots \mathsf{Hybrid}_{q(\lambda),13}$, each of which we prove to be indistinguishable from the previous one. We define $\mathsf{Hybrid}_{0,13}$ for convenience in our proof. The final hybrid $\mathsf{Hybrid}_{q(\lambda),13}$ corresponds to the ideal world in the security game described above, and contains (implicitly) descriptions of `SimUGen`, `SimRO` as required in Definition 4.

We start by describing Hybrid_0 and then describe $\mathsf{Hybrid}_{s,1}$, $\mathsf{Hybrid}_{s,2}, \ldots \mathsf{Hybrid}_{s,13}$ for $s \in [q(\lambda)]$. We denote changes between subsequent hybrids using red underlined font. We also add an intuition behind why the hybrids should be indistinguishable. Detailed proofs can be found in the next section. In the following experiments, the challenger chooses PRF keys $K_1^{(n)}, K_2$ and K_2' for PRF's $F_1^{(n)}, F_2$ and F_2'.

Each hybrid below is an experiment that takes as input 1^λ. The final output of each hybrid experiment is the output produced by the adversary when it terminates.

Hybrid$_0$:

- The challenger pads the program Adaptive-Parameters in Figure 8 to be the maximum of the size of itself and all corresponding programs (Adaptive-Parameters: 2, Adaptive-Parameters: 3) in the following hybrids. Next, he sends the obfuscation of the program in Figure 8 to the adversary.
- Set $j = 0$. While the adversary is making queries to random oracle, increment j and repeat:
  1. Let the adversary query to the random oracle be on protocol description d_j^*.
  2. The challenger sets the output of the random oracle, $u_j^* \leftarrow \{0,1\}^{n^2}, v_j^* \leftarrow \{0,1\}^n$.
- The adversary then outputs a single bit b'.

```
                              Adaptive-Parameters

Constants: PRF keys $K_1^{(n)}$, $K_2, K_2'$.
Input: Program hash $u = u[1], \ldots, u[n]$, $v$.
  1. Compute $F_1(K_1^{(n)}, v) = (y_{1,0}, y_{1,1}), \ldots, (y_{n,0}, y_{n,1})$.
  2. For $i = 1, \ldots, n$, if $u[i] = y_{i,0}$ set $x_i = 0$ else if $u[i] = y_{i,1}$ set $x_i = 1$ else set $x_i = \perp$
  3. If $x \in \{0,1\}^n$ (i.e. no $\perp$s), output $x$.
  4. Else set $K_3 = F_2(K_2, u|v)$, $r = F_2(K_2', u|v)$. Output $iO(P_{K_3}; r)$ of the program[a] $P_{K_3}$ of
     Figure 9.
```

[a]Appropriately appended to the maximum of the size of itself and P'_{K_3,p_j^*,d_j^*} in future hybrids

Figure 8: Program Adaptive-Parameters

```
                              $P_{K_3}$

Constant: PRF key $K_3$.
Input: Program description $d$.
  1. Output $d(F_3(K_3, d))$. Recall that $d$ is a program description which outputs $k$ bits.
```

Figure 9: Program P_{K_3}

Hybrid$_{s-1,13}$:

- The challenger pads the program Adaptive-Parameters in Figure 10 appropriately [15] and sends an iO of the program to the adversary.
- Set $j = 0$. While the adversary is making queries to random oracle, increment j and repeat:
  1. Let the adversary query to the random oracle be on protocol description d_j^*.
  2. If $j < s$, the challenger sets the output of the random oracle, $v_j^* \leftarrow \{0,1\}^n$. He sets $K_3 \leftarrow \{0,1\}^n, e' \leftarrow \{0,1\}^n$. He queries the oracle to obtain the Parameters p_j^* and sets $g = iO(P'_{K_3,p_j^*,d_j^*}, e')$ (See Figure 12). For all $b \in \{0,1\}$ and $i \in [1,n]$, he sets $(y_{1,0}^*, y_{1,1}^*), \ldots, (y_{n,0}^*, y_{n,1}^*) = F_1(K_1^{(n)}, v_j^*)$, $u_j^*[i] = y_{i,g_i}^*$, where g_i is the i^{th} bit of g.
  3. If $j \geq s$, the challenger sets the output of the random oracle, $u_j^* \leftarrow \{0,1\}^{n^2}, v_j^* \leftarrow \{0,1\}^n$.
- The adversary then outputs a single bit b'.

Adaptive-Parameters

Constants: PRF keys $K_1^{(n)}$, K_2, K_2'.
Input: Program hash $u = u[1], \ldots, u[n]$, v.

```
1. Compute $F_1(K_1^{(n)}, v) = (y_{1,0}, y_{1,1}), \ldots, (y_{n,0}, y_{n,1})$.
2. For $i = 1, \ldots, n$, if $u[i] = y_{i,0}$ set $x_i = 0$ else if $u[i] = y_{i,1}$ set $x_i = 1$ else set $x_i = \perp$
3. If $x \in \{0,1\}^n$ (i.e. no $\perp$s), output $x$.
4. Else set $K_3 = F_2(K_2, u|v)$, $r = F_2(K_2', u|v)$. Output $iO(P_{K_3}; r)$ of the program[a] $P_{K_3}$ of Figure 11.
```

[a]Appropriately appended to the maximum of the size of itself and P'_{K_3,p_j^*,d_j^*}

Figure 10: Program Adaptive-Parameters

P_{K_3}

Constant: PRF key K_3.
Input: Program description d.

```
1. Output $d(F_3(K_3, d))$. Recall that $d$ is a program description which outputs $k$ bits.
```

Figure 11: Program P_{K_3}

P'_{K_3,p_j^*,d_j^*}

Constants: PRF key $K_3\{d_j^*\}, d_j^*, p_j^*$.
Input: Program description d.

```
1. If $d = d_j^*$ output $p_j^*$.
2. Else output $d(F_3(K_3, d))$. Recall that $d$ is a program description which outputs $k$ bits.
```

Figure 12: Program P'_{K_3,p_j^*,d_j^*}

[15]To the maximum of the size of itself and all corresponding programs (Adaptive-Parameters: 2, Adaptive-Parameters: 3) in the previous and following hybrids.

$\mathsf{Hybrid}_{s,1}$:

- The challenger sets $v_s^* \leftarrow \{0,1\}^n, u_s^* \leftarrow \{0,1\}^{n^2}$. He sets $(y_{1,0}^*, y_{1,1}^*), \ldots, (y_{n,0}^*, y_{n,1}^*) = F_1(K_1^{(n)}, v_s^*)$. Then, for all $b \in \{0,1\}$, he sets $z_{i,b}^* = \mathsf{PRG}(y_{i,b}^*)$ for $i \in [1,n]$.
  He pads the program Adaptive-Parameters: 2 in Figure 13 appropriately [16] and sends an iO of the program to the adversary.
- Set $j = 0$. While the adversary is making queries to random oracle, increment j and repeat:
  1. Let the adversary query to the random oracle be on protocol description d_j^*.
  2. If $j < s$, the challenger sets the output of the random oracle, $v_j^* \leftarrow \{0,1\}^n$.
     He sets $K_3 \leftarrow \{0,1\}^n, e' \leftarrow \{0,1\}^n$. He queries the oracle to obtain the Parameters p_j^* and sets $g = iO(P'_{K_3,p_j^*,d_j^*}, e')$ (See Figure 15). For all $b \in \{0,1\}$ and $i \in [1,n]$, he sets $(y_{1,0}^*, y_{1,1}^*), \ldots, (y_{n,0}^*, y_{n,1}^*) = F_1(K_1^{(n)}, v_j^*)$, $u_j^*[i] = y_{i,g_i}^*$, where g_i is the i^{th} bit of g.
  3. If $j > s$, the challenger sets the output of the random oracle, $u_j^* \leftarrow \{0,1\}^{n^2}, v_j^* \leftarrow \{0,1\}^n$.
- The adversary then outputs a single bit b'.

Adaptive-Parameters: 2

Constants: v_s^*, PRF key $K_1^{(n)}\{v_s^*\}$, K_2, K_2', $z_{i,b}^*$ for $i \in [1,n]$ and $b \in \{0,1\}$
Input: Program hash $u = u[1], \ldots, u[n]$, v.

```
1. If $v = v_s^*$ then for $i = 1, \ldots, n$ do
   If $\mathsf{PRG}(u[i]) = z_{i,0}^*$ let $x_i = 0$, if $\mathsf{PRG}(u[i]) = z_{i,1}^*$ let $x_i = 1$, else $x_i = \perp$.
   Go to step 4.
2. Compute $F_1(K_1^{(n)}, v) = (y_{1,0}, y_{1,1}), \ldots, (y_{n,0}, y_{n,1})$.
3. For $i = 1, \ldots, n$, if $u[i] = y_{i,0}$ set $x_i = 0$ else if $u[i] = y_{i,1}$ set $x_i = 1$ else set $x_i = \perp$
4. If $x \in \{0,1\}^n$ (i.e. no $\perp$s), output $x$.
5. Else set $K_3 = F_2(K_2, u|v)$, $r = F_2(K_2', u|v)$. Output $iO(P_{K_3}; r)$ of the program[a] $P_{K_3}$ of Figure 14.
```

[a]Appropriately appended to the maximum of the size of itself and P'_{K_3,p_j^*,d_j^*}

Figure 13: Program Adaptive-Parameters

P_{K_3}

Constant: PRF key K_3.
Input: Program description d.

```
1. Output $d(F_3(K_3, d))$. Recall that $d$ is a program description which outputs $k$ bits.
```

Figure 14: Program P_{K_3}

P'_{K_3,p_j^*,d_j^*}

Constants: PRF key $K_3\{d_j^*\}, d_j^*, p_j^*$.
Input: Program description d.

```
1. If $d = d_j^*$ output $p_j^*$.
2. Else output $d(F_3(K_3, d))$. Recall that $d$ is a program description which outputs $k$ bits.
```

Figure 15: Program P'_{K_3,p_j^*,d_j^*}

Intuitively, indistinguishable from $\mathsf{Hybrid}_{s-1,13}$ by iO between Adaptive-Parameters: -2.

[16]To the maximum of the size of itself and all corresponding programs (Adaptive-Parameters, Adaptive-Parameters: 3) in the previous and following hybrids.

$\mathsf{Hybrid}_{s,2}$:

- The challenger sets $v_s^* \leftarrow \{0,1\}^n, u_s^* \leftarrow \{0,1\}^{n^2}$. For all $b \in \{0,1\}$, $i \in [1,n]$, he sets $y_{i,b}^* \leftarrow \{0,1\}^n$.
  Then, for all $b \in \{0,1\}$, he sets $z_{i,b}^* = \mathsf{PRG}(y_{i,b}^*)$ for $i \in [1,n]$.
  He pads the program Adaptive-Parameters: 2 in Figure 16 appropriately [17] and sends an iO of the program to the adversary.
- Set $j = 0$. While the adversary is making queries to random oracle, increment j and repeat:
  1. Let the adversary query to the random oracle be on protocol description d_j^*.
  2. If $j < s$, the challenger sets the output of the random oracle, $v_j^* \leftarrow \{0,1\}^n$.
     He sets $K_3 \leftarrow \{0,1\}^n, e' \leftarrow \{0,1\}^n$. He queries the oracle to obtain the Parameters p_j^* and sets $g = iO(P'_{K_3,p_j^*,d_j^*}, e')$ (See Figure 18). For all $b \in \{0,1\}$ and $i \in [1,n]$, he sets $(y_{1,0}^*, y_{1,1}^*), \ldots, (y_{n,0}^*, y_{n,1}^*) = F_1(K_1^{(n)}, v_j^*)$, $u_j^*[i] = y_{i,g_i}^*$, where g_i is the i^{th} bit of g.
  3. If $j > s$, the challenger sets the output of the random oracle, $u_j^* \leftarrow \{0,1\}^{n^2}, v_j^* \leftarrow \{0,1\}^n$.
- The adversary then outputs a single bit b'.

```
                    Adaptive-Parameters: 2

Constants: $v_s^*$, PRF key $K_1^{(n)}\{v_s^*\}$, $K_2, K_2'$, $z_{i,b}^*$ for $i \in [1,n]$ and $b \in \{0,1\}$
Input: Program hash $u = u[1], \ldots, u[n]$, $v$.
  1. If $v = v_s^*$ then for $i = 1, \ldots, n$ do
     If $\mathsf{PRG}(u[i]) = z_{i,0}^*$ let $x_i = 0$, if $\mathsf{PRG}(u[i]) = z_{i,1}^*$ let $x_i = 1$, else $x_i = \perp$.
     Go to step 4.
  2. Compute $F_1(K_1^{(n)}, v) = (y_{1,0}, y_{1,1}), \ldots, (y_{n,0}, y_{n,1})$.
  3. For $i = 1, \ldots, n$, if $u[i] = y_{i,0}$ set $x_i = 0$ else if $u[i] = y_{i,1}$ set $x_i = 1$ else set $x_i = \perp$
  4. If $x \in \{0,1\}^n$ (i.e. no $\perp$s), output $x$.
  5. Else set $K_3 = F_2(K_2, u|v)$, $r = F_2(K_2', u|v)$. Output $iO(P_{K_3}; r)$ of the program[a] $P_{K_3}$ of
     Figure 17.

[a] Appropriately appended to the maximum of the size of itself and $P'_{K_3,p_j^*,d_j^*}$
```

Figure 16: Program Adaptive-Parameters

```
                    $P_{K_3}$

Constant: PRF key $K_3$.
Input: Program description $d$.
  1. Output $d(F_3(K_3, d))$. Recall that $d$ is a program description which outputs $k$ bits.
```

Figure 17: Program P_{K_3}

```
                    $P'_{K_3,p_j^*,d_j^*}$

Constants: PRF key $K_3\{d_j^*\}, d_j^*, p_j^*$.
Input: Program description $d$.
  1. If $d = d_j^*$ output $p_j^*$.
  2. Else output $d(F_3(K_3, d))$. Recall that $d$ is a program description which outputs $k$ bits.
```

Figure 18: Program P'_{K_3,p_j^*,d_j^*}

Intuitively, indistinguishable from $\mathsf{Hybrid}_{s,1}$ by security of punctured PRF key $K_1^{(n)}\{v_s^*\}$.

[17] To the maximum of the size of itself and all corresponding programs (Adaptive-Parameters, Adaptive-Parameters: 3) in the previous and following hybrids.

Hybrid$_{s,3}$:

- The challenger sets $v_s^* \leftarrow \{0,1\}^n, u_s^* \leftarrow \{0,1\}^{n^2}$. For all $b \in \{0,1\}, i \in [1,n]$, he sets $z_{i,b}^* \leftarrow \{0,1\}^{2n}$.
  He pads the program Adaptive-Parameters: 2 in Figure 19 appropriately [18] and sends an iO of the program to the adversary.
- Set $j = 0$. While the adversary is making queries to random oracle, increment j and repeat:
  1. Let the adversary query to the random oracle be on protocol description d_j^*.
  2. If $j < s$, the challenger sets the output of the random oracle, $v_j^* \leftarrow \{0,1\}^n$.
     He sets $K_3 \leftarrow \{0,1\}^n, e' \leftarrow \{0,1\}^n$. He queries the oracle to obtain the Parameters p_j^* and sets $g = iO(P'_{K_3,p_j^*,d_j^*}, e')$ (See Figure 21). For all $b \in \{0,1\}$ and $i \in [1,n]$, he sets $(y_{1,0}^*, y_{1,1}^*), \ldots, (y_{n,0}^*, y_{n,1}^*) = F_1(K_1^{(n)}, v_j^*)$, $u_j^*[i] = y_{i,g_i}^*$, where g_i is the i^{th} bit of g.
  3. If $j > s$, the challenger sets the output of the random oracle, $u_j^* \leftarrow \{0,1\}^{n^2}, v_j^* \leftarrow \{0,1\}^n$.
- The adversary then outputs a single bit b'.

```
Adaptive-Parameters: 2

Constants: $v_s^*$, PRF keys $K_1^{(n)}\{v_s^*\}$, $K_2, K_2'$, $z_{i,b}^*$ for $i \in [1,n]$ and $b \in \{0,1\}$
Input: Program hash $u = u[1], \ldots, u[n]$, $v$.
  1. If $v = v_s^*$ then for $i = 1, \ldots, n$ do
     If $\mathsf{PRG}(u[i]) = z_{i,0}^*$ let $x_i = 0$, if $\mathsf{PRG}(u[i]) = z_{i,1}^*$ let $x_i = 1$, else $x_i = \perp$.
     Go to step 4.
  2. Compute $F_1(K_1^{(n)}, v) = (y_{1,0}, y_{1,1}), \ldots, (y_{n,0}, y_{n,1})$.
  3. For $i = 1, \ldots, n$, if $u[i] = y_{i,0}$ set $x_i = 0$ else if $u[i] = y_{i,1}$ set $x_i = 1$ else set $x_i = \perp$
  4. If $x \in \{0,1\}^n$ (i.e. no $\perp$s), output $x$.
  5. Else set $K_3 = F_2(K_2, u|v)$, $r = F_2(K_2', u|v)$. Output $iO(P_{K_3}; r)$ of the program[a] $P_{K_3}$ of Figure 20.

[a]Appropriately appended to the maximum of the size of itself and $P'_{K_3,p_j^*,d_j^*}$
```

Figure 19: Program Adaptive-Parameters

```
$P_{K_3}$

Constant: PRF key $K_3$.
Input: Program description $d$.
  1. Output $d(F_3(K_3, d))$. Recall that $d$ is a program description which outputs $k$ bits.
```

Figure 20: Program P_{K_3}

```
$P'_{K_3,p_j^*,d_j^*}$

Constants: PRF key $K_3\{d_j^*\}, d_j^*, p_j^*$.
Input: Program description $d$.
  1. If $d = d_j^*$ output $p_j^*$.
  2. Else output $d(F_3(K_3, d))$. Recall that $d$ is a program description which outputs $k$ bits.
```

Figure 21: Program P'_{K_3,p_j^*,d_j^*}

Intuitively, indistinguishable from Hybrid$_{s,2}$ by security of the PRG.

[18]To the maximum of the size of itself and all corresponding programs (Adaptive-Parameters, Adaptive-Parameters: 3) in the previous and following hybrids.

$\mathsf{Hybrid}_{s,4}$:

- The challenger sets $v_s^* \leftarrow \{0,1\}^n, u_s^* \leftarrow \{0,1\}^{n^2}$. For all $b \in \{0,1\}, i \in [1,n]$, he sets $z_{i,b}^* \leftarrow \{0,1\}^{2n}$. He sets $e = F_2(K_2, u_s^*|v_s^*)$ and $e' = F_2'(K_2', u_s^*|v_s^*)$. Next, he sets $g = iO(P_e, e')$. He pads the program Adaptive-Parameters: 3 in Figure 22 appropriately [19] and sends an iO of the program to the adversary.
- Set $j = 0$. While the adversary is making queries to random oracle, increment j and repeat:
  1. Let the adversary query to the random oracle be on protocol description d_j^*.
  2. If $j < s$, the challenger sets the output of the random oracle, $v_j^* \leftarrow \{0,1\}^n$. He sets $K_3 \leftarrow \{0,1\}^n, e' \leftarrow \{0,1\}^n$. He queries the oracle to obtain the Parameters p_j^* and sets $g = iO(P'_{K_3,p_j^*,d_j^*}, e')$ (See Figure 24). For all $b \in \{0,1\}$ and $i \in [1,n]$, he sets $(y_{1,0}^*, y_{1,1}^*), \ldots, (y_{n,0}^*, y_{n,1}^*) = F_1(K_1^{(n)}, v_j^*)$, $u_j^*[i] = y_{i,g_i}^*$, where g_i is the i^{th} bit of g.
  3. If $j > s$, the challenger sets the output of the random oracle, $u_j^* \leftarrow \{0,1\}^{n^2}, v_j^* \leftarrow \{0,1\}^n$.
- The adversary then outputs a single bit b'.

```
                                   Adaptive-Parameters: 3

Constants: $v_s^*$, $u_s^*, g$, PRF keys $K_1^{(n)}\{v_s^*\}$, $K_2\{u_s^*|v_s^*\}$, $K_2'\{u_s^*|v_s^*\}$, $z_{i,b}^*$ for $i \in [1,n]$ and $b \in \{0,1\}$
Input: Program hash $u = u[1], \ldots, u[n]$, $v$.
  1. If $u = u_s^*$ and $v = v_s^*$ output $g$ and stop.
  2. If $v = v_s^*$ then for $i = 1, \ldots, n$ do
     If $\mathsf{PRG}(u[i]) = z_{i,0}^*$ let $x_i = 0$, if $\mathsf{PRG}(u[i]) = z_{i,1}^*$ let $x_i = 1$, else $x_i = \perp$.
     Go to step 4.
  3. Compute $F_1(K_1^{(n)}, v) = (y_{1,0}, y_{1,1}), \ldots, (y_{n,0}, y_{n,1})$.
  4. For $i = 1, \ldots, n$, if $u[i] = y_{i,0}$ set $x_i = 0$ else if $u[i] = y_{i,1}$ set $x_i = 1$ else set $x_i = \perp$
  5. If $x \in \{0,1\}^n$ (i.e. no $\perp$s), output $x$.
  6. Else set $K_3 = F_2(K_2, u|v)$, $r = F_2(K_2', u|v)$. Output $iO(P_{K_3}; r)$ of the program[a] $P_{K_3}$ of Figure 23.

[a] Appropriately appended to the maximum of the size of itself and $P'_{K_3,p_j^*,d_j^*}$
```

Figure 22: Program Adaptive-Parameters

```
                                   $P_{K_3}$

Constant: PRF key $K_3$.
Input: Program description $d$.
  1. Output $d(F_3(K_3, d))$. Recall that $d$ is a program description which outputs $k$ bits.
```

Figure 23: Program P_{K_3}

```
                                   $P'_{K_3,p_j^*,d_j^*}$

Constants: PRF key $K_3\{d_j^*\}, d_j^*, p_j^*$.
Input: Program description $d$.
  1. If $d = d_j^*$ output $p_j^*$.
  2. Else output $d(F_3(K_3, d))$. Recall that $d$ is a program description which outputs $k$ bits.
```

Figure 24: Program P'_{K_3,p_j^*,d_j^*}

Intuitively, indistinguishable from $\mathsf{Hybrid}_{s,3}$ by iO between Adaptive-Parameters:2-3.

[19] To the maximum of the size of itself and all corresponding programs (Adaptive-Parameters, Adaptive-Parameters: 3) in the previous and following hybrids.

$\mathsf{Hybrid}_{s,5}$:

- The challenger sets $v_s^* \leftarrow \{0,1\}^n, u_s^* \leftarrow \{0,1\}^{n^2}$. For all $b \in \{0,1\}, i \in [1,n]$, he sets $z_{i,b}^* \leftarrow \{0,1\}^{2n}$. He sets $e \leftarrow \{0,1\}^n$ and $e' = F_2'(K_2', u_s^*|v_s^*)$. Next, he sets $g = iO(P_e, e')$.
  He pads the program Adaptive-Parameters: 3 in Figure 25 appropriately [20] and sends an iO of the program to the adversary.
- Set $j = 0$. While the adversary is making queries to random oracle, increment j and repeat:
  1. Let the adversary query to the random oracle be on protocol description d_j^*.
  2. If $j < s$, the challenger sets the output of the random oracle, $v_j^* \leftarrow \{0,1\}^n$.
     He sets $K_3 \leftarrow \{0,1\}^n, e' \leftarrow \{0,1\}^n$. He queries the oracle to obtain the Parameters p_j^* and sets $g = iO(P'_{K_3,p_j^*,d_j^*}, e')$ (See Figure 27). For all $b \in \{0,1\}$ and $i \in [1,n]$, he sets $(y_{1,0}^*, y_{1,1}^*), \ldots, (y_{n,0}^*, y_{n,1}^*) = F_1(K_1^{(n)}, v_j^*)$, $u_j^*[i] = y_{i,g_i}^*$, where g_i is the i^{th} bit of g.
  3. If $j > s$, the challenger sets the output of the random oracle, $u_j^* \leftarrow \{0,1\}^{n^2}, v_j^* \leftarrow \{0,1\}^n$.
- The adversary then outputs a single bit b'.

Adaptive-Parameters: 3

```
Constants: $v_s^*$, $u_s^*$, $g$, PRF keys $K_1^{(n)}\{v_s^*\}$, $K_2\{u_s^*|v_s^*\}$, $K_2'\{u_s^*|v_s^*\}$, $z_{i,b}^*$ for $i \in [1,n]$ and $b \in \{0,1\}$
Input: Program hash $u = u[1], \ldots, u[n]$, $v$.
  1. If $u = u_s^*$ and $v = v_s^*$ output $g$ and stop.
  2. If $v = v_s^*$ then for $i = 1, \ldots, n$ do
     If $\mathsf{PRG}(u[i]) = z_{i,0}^*$ let $x_i = 0$, if $\mathsf{PRG}(u[i]) = z_{i,1}^*$ let $x_i = 1$, else $x_i = \perp$.
     Go to step 4.
  3. Compute $F_1(K_1^{(n)}, v) = (y_{1,0}, y_{1,1}), \ldots, (y_{n,0}, y_{n,1})$.
  4. For $i = 1, \ldots, n$, if $u[i] = y_{i,0}$ set $x_i = 0$ else if $u[i] = y_{i,1}$ set $x_i = 1$ else set $x_i = \perp$
  5. If $x \in \{0,1\}^n$ (i.e. no $\perp$s), output $x$.
  6. Else set $K_3 = F_2(K_2, u|v)$, $r = F_2(K_2', u|v)$. Output $iO(P_{K_3}; r)$ of the program[a] $P_{K_3}$ of Figure 26.
```

[a] Appropriately appended to the maximum of the size of itself and P'_{K_3,p_j^*,d_j^*}

Figure 25: Program Adaptive-Parameters

P_{K_3}

```
Constant: PRF key $K_3$.
Input: Program description $d$.
  1. Output $d(F_3(K_3, d))$. Recall that $d$ is a program description which outputs $k$ bits.
```

Figure 26: Program P_{K_3}

P'_{K_3,p_j^*,d_j^*}

```
Constants: PRF key $K_3\{d_j^*\}, d_j^*, p_j^*$.
Input: Program description $d$.
  1. If $d = d_j^*$ output $p_j^*$.
  2. Else output $d(F_3(K_3, d))$. Recall that $d$ is a program description which outputs $k$ bits.
```

Figure 27: Program P'_{K_3,p_j^*,d_j^*}

Intuitively, indistinguishable from $\mathsf{Hybrid}_{s,4}$ by security of punctured PRF key $K_2\{u_s^*|v_s^*\}$.

[20] To the maximum of the size of itself and all corresponding programs (Adaptive-Parameters, Adaptive-Parameters: 3) in the previous and following hybrids.

$\mathsf{Hybrid}_{s,6}$:

- The challenger sets $v_s^* \leftarrow \{0,1\}^n, u_s^* \leftarrow \{0,1\}^{n^2}$. For all $b \in \{0,1\}, i \in [1,n]$, he sets $z_{i,b}^* \leftarrow \{0,1\}^{2n}$. He sets $e \leftarrow \{0,1\}^n$ and $e' \leftarrow \{0,1\}^n$. Next, he sets $g = iO(P_e, e')$.
  He pads the program Adaptive-Parameters: 3 in Figure 28 appropriately [21] and sends an iO of the program to the adversary.
- Set $j = 0$. While the adversary is making queries to random oracle, increment j and repeat:
  1. Let the adversary query to the random oracle be on protocol description d_j^*.
  2. If $j < s$, the challenger sets the output of the random oracle, $v_j^* \leftarrow \{0,1\}^n$.
     He sets $K_3 \leftarrow \{0,1\}^n, e' \leftarrow \{0,1\}^n$. He queries the oracle to obtain the Parameters p_j^* and sets $g = iO(P'_{K_3,p_j^*,d_j^*}, e')$ (See Figure 30). For all $b \in \{0,1\}$ and $i \in [1,n]$, he sets $(y_{1,0}^*, y_{1,1}^*), \ldots, (y_{n,0}^*, y_{n,1}^*) = F_1(K_1^{(n)}, v_j^*)$, $u_j^*[i] = y_{i,g_i}^*$, where g_i is the i^{th} bit of g.
  3. If $j > s$, the challenger sets the output of the random oracle, $u_j^* \leftarrow \{0,1\}^{n^2}, v_j^* \leftarrow \{0,1\}^n$.
- The adversary then outputs a single bit b'.

Adaptive-Parameters: 3

Constants: v_s^*, u_s^*, g, PRF keys $K_1^{(n)}\{v_s^*\}$, $K_2\{u_s^*|v_s^*\}$, $K_2'\{u_s^*|v_s^*\}$, $z_{i,b}^*$ for $i \in [1,n]$ and $b \in \{0,1\}$
Input: Program hash $u = u[1], \ldots, u[n]$, v.

```
1. If $u = u_s^*$ and $v = v_s^*$ output $g$ and stop.
2. If $v = v_s^*$ then for $i = 1, \ldots, n$ do
   If $\mathsf{PRG}(u[i]) = z_{i,0}^*$ let $x_i = 0$, if $\mathsf{PRG}(u[i]) = z_{i,1}^*$ let $x_i = 1$, else $x_i = \perp$.
   Go to step 4.
3. Compute $F_1(K_1^{(n)}, v) = (y_{1,0}, y_{1,1}), \ldots, (y_{n,0}, y_{n,1})$.
4. For $i = 1, \ldots, n$, if $u[i] = y_{i,0}$ set $x_i = 0$ else if $u[i] = y_{i,1}$ set $x_i = 1$ else set $x_i = \perp$
5. If $x \in \{0,1\}^n$ (i.e. no $\perp$s), output $x$.
6. Else set $K_3 = F_2(K_2, u|v)$, $r = F_2(K_2', u|v)$. Output $iO(P_{K_3}; r)$ of the program[a] $P_{K_3}$ of Figure 29.
```

[a] Appropriately appended to the maximum of the size of itself and P'_{K_3,p_j^*,d_j^*}

Figure 28: Program Adaptive-Parameters

P_{K_3}

Constant: PRF key K_3.
Input: Program description d.

```
1. Output $d(F_3(K_3, d))$. Recall that $d$ is a program description which outputs $k$ bits.
```

Figure 29: Program P_{K_3}

P'_{K_3,p_j^*,d_j^*}

Constants: PRF key $K_3\{d_j^*\}, d_j^*, p_j^*$.
Input: Program description d.

```
1. If $d = d_j^*$ output $p_j^*$.
2. Else output $d(F_3(K_3, d))$. Recall that $d$ is a program description which outputs $k$ bits.
```

Figure 30: Program P'_{K_3,p_j^*,d_j^*}

Intuitively, indistinguishable from $\mathsf{Hybrid}_{s,5}$ by security of punctured PRF key $K_2'\{u_s^*|v_s^*\}$.

[21] To the maximum of the size of itself and all corresponding programs (Adaptive-Parameters, Adaptive-Parameters: 3) in the previous and following hybrids.

$\mathsf{Hybrid}_{s,7}$:

- The challenger sets $v_s^* \leftarrow \{0,1\}^n$. He sets $e \leftarrow \{0,1\}^n$ and $e' \leftarrow \{0,1\}^n$. Next, he sets $g = iO(P_e, e')$. For all $i \in [1,n]$, he sets $y^*_{i,g_i} \leftarrow \{0,1\}^n$, $u_s^*[i] = y^*_{i,g_i}$, $z^*_{i,g_i} = \mathsf{PRG}(y^*_{i,g_i})$ and $z^*_{i,\bar{g}_i} \leftarrow \{0,1\}^{2n}$, where g_i is the i^{th} bit of g and $\bar{g}_i = 1 - g_i$.
  He pads the program Adaptive-Parameters: 2 in Figure 31 appropriately [22] and sends an iO of the program to the adversary.
- Set $j = 0$. While the adversary is making queries to random oracle, increment j and repeat:
  1. Let the adversary query to the random oracle be on protocol description d_j^*.
  2. If $j < s$, the challenger sets the output of the random oracle, $v_j^* \leftarrow \{0,1\}^n$. He sets $K_3 \leftarrow \{0,1\}^n, e' \leftarrow \{0,1\}^n$. He queries the oracle to obtain the Parameters p_j^* and sets $g = iO(P'_{K_3,p_j^*,d_j^*}, e')$ (See Figure 33). For all $b \in \{0,1\}$ and $i \in [1,n]$, he sets $(y^*_{1,0}, y^*_{1,1}), \ldots, (y^*_{n,0}, y^*_{n,1}) = F_1(K_1^{(n)}, v_j^*)$, $u_j^*[i] = y^*_{i,g_i}$, where g_i is the i^{th} bit of g.
  3. If $j > s$, the challenger sets the output of the random oracle, $u_j^* \leftarrow \{0,1\}^{n^2}, v_j^* \leftarrow \{0,1\}^n$.
- The adversary then outputs a single bit b'.

Adaptive-Parameters: 2

```
Constants: $v_s^*$, $g$, PRF keys $K_1^{(n)}\{v_s^*\}$, $K_2$, $K_2'$, $z^*_{i,b}$ for $i \in [1,n]$ and $b \in \{0,1\}$
Input: Program hash $u = u[1], \ldots, u[n]$, $v$.
  1. If $v = v_s^*$ then for $i = 1, \ldots, n$ do
     If $\mathsf{PRG}(u[i]) = z^*_{i,0}$ let $x_i = 0$, if $\mathsf{PRG}(u[i]) = z^*_{i,1}$ let $x_i = 1$, else $x_i = \perp$.
     Go to step 4.
  2. Compute $F_1(K_1^{(n)}, v) = (y_{1,0}, y_{1,1}), \ldots, (y_{n,0}, y_{n,1})$.
  3. For $i = 1, \ldots, n$, if $u[i] = y_{i,0}$ set $x_i = 0$ else if $u[i] = y_{i,1}$ set $x_i = 1$ else set $x_i = \perp$
  4. If $x \in \{0,1\}^n$ (i.e. no $\perp$s), output $x$.
  5. Else set $K_3 = F_2(K_2, u|v)$, $r = F_2(K_2', u|v)$. Output $iO(P_{K_3}; r)$ of the program[a] $P_{K_3}$ of Figure 32.
```

[a] Appropriately appended to the maximum of the size of itself and P'_{K_3,p_j^*,d_j^*}

Figure 31: Program Adaptive-Parameters: 2

P_{K_3}

```
Constant: PRF key $K_3$.
Input: Program description $d$.
  1. Output $d(F_3(K_3, d))$. Recall that $d$ is a program description which outputs $k$ bits.
```

Figure 32: Program P_{K_3}

P'_{K_3,p_j^*,d_j^*}

```
Constants: PRF key $K_3\{d_j^*\}, d_j^*, p_j^*$.
Input: Program description $d$.
  1. If $d = d_j^*$ output $p_j^*$.
  2. Else output $d(F_3(K_3, d))$. Recall that $d$ is a program description which outputs $k$ bits.
```

Figure 33: Program P'_{K_3,p_j^*,d_j^*}

Intuitively, indistinguishable from $\mathsf{Hybrid}_{s,6}$ by iO between Adaptive-Parameters:3-2.

[22] To the maximum of the size of itself and all corresponding programs (Adaptive-Parameters, Adaptive-Parameters: 3) in the previous and following hybrids.

$\mathsf{Hybrid}_{s,8}$:

- The challenger sets $v_s^* \leftarrow \{0,1\}^n$. He sets $e \leftarrow \{0,1\}^n$ and $e' \leftarrow \{0,1\}^n$. Next, he sets $g = iO(P_e, e')$. For all $b \in \{0,1\}, i \in [1,n]$, he sets $y_{i,b}^* \leftarrow \{0,1\}^n$, $u_s^*[i] = y_{i,g_i}^*$, $z_{i,g_i}^* = \mathsf{PRG}(y_{i,g_i}^*)$ and $\underline{z_{i,\bar{g}_i}^* = \mathsf{PRG}(y_{i,\bar{g}_i}^*)}$, where g_i is the i^{th} bit of g and $\bar{g}_i = 1 - g_i$.
  He pads the program Adaptive-Parameters: 2 in Figure 34 appropriately [23] and sends an iO of the program to the adversary.
- Set $j = 0$. While the adversary is making queries to random oracle, increment j and repeat:
  1. Let the adversary query to the random oracle be on protocol description d_j^*.
  2. If $j < s$, the challenger sets the output of the random oracle, $v_j^* \leftarrow \{0,1\}^n$.
     He sets $K_3 \leftarrow \{0,1\}^n, e' \leftarrow \{0,1\}^n$. He queries the oracle to obtain the Parameters p_j^* and sets $g = iO(P'_{K_3,p_j^*,d_j^*}, e')$ (See Figure 36). For all $b \in \{0,1\}$ and $i \in [1,n]$, he sets $(y_{1,0}^*, y_{1,1}^*), \ldots, (y_{n,0}^*, y_{n,1}^*) = F_1(K_1^{(n)}, v_j^*)$, $u_j^*[i] = y_{i,g_i}^*$, where g_i is the i^{th} bit of g.
  3. If $j > s$, the challenger sets the output of the random oracle, $u_j^* \leftarrow \{0,1\}^{n^2}, v_j^* \leftarrow \{0,1\}^n$.
- The adversary then outputs a single bit b'.

```
                         Adaptive-Parameters: 2

Constants: $v_s^*$, $g$, PRF keys $K_1^{(n)}\{v_s^*\}$, $K_2$, $K_2'$, $z_{i,b}^*$ for $i \in [1,n]$ and $b \in \{0,1\}$
Input: Program hash $u = u[1], \ldots, u[n]$, $v$.
  1. If $v = v_s^*$ then for $i = 1, \ldots, n$ do
     If $\mathsf{PRG}(u[i]) = z_{i,0}^*$ let $x_i = 0$, if $\mathsf{PRG}(u[i]) = z_{i,1}^*$ let $x_i = 1$, else $x_i = \perp$.
     Go to step 4.
  2. Compute $F_1(K_1^{(n)}, v) = (y_{1,0}, y_{1,1}), \ldots, (y_{n,0}, y_{n,1})$.
  3. For $i = 1, \ldots, n$, if $u[i] = y_{i,0}$ set $x_i = 0$ else if $u[i] = y_{i,1}$ set $x_i = 1$ else set $x_i = \perp$
  4. If $x \in \{0,1\}^n$ (i.e. no $\perp$s), output $x$.
  5. Else set $K_3 = F_2(K_2, u|v)$, $r = F_2(K_2', u|v)$. Output $iO(P_{K_3}; r)$ of the program[a] $P_{K_3}$ of
     Figure 35.

[a] Appropriately appended to the maximum of the size of itself and $P'_{K_3,p_j^*,d_j^*}$
```

Figure 34: Program Adaptive-Parameters: 2

```
                                $P_{K_3}$

Constant: PRF key $K_3$.
Input: Program description $d$.
  1. Output $d(F_3(K_3, d))$. Recall that $d$ is a program description which outputs $k$ bits.
```

Figure 35: Program P_{K_3}

```
                                $P'_{K_3,p_j^*,d_j^*}$

Constants: PRF key $K_3\{d_j^*\}, d_j^*, p_j^*$.
Input: Program description $d$.
  1. If $d = d_j^*$ output $p_j^*$.
  2. Else output $d(F_3(K_3, d))$. Recall that $d$ is a program description which outputs $k$ bits.
```

Figure 36: Program P'_{K_3,p_j^*,d_j^*}

Intuitively, indistinguishable from $\mathsf{Hybrid}_{s,7}$ by security of the PRG.

[23] To the maximum of the size of itself and all corresponding programs (Adaptive-Parameters, Adaptive-Parameters: 3) in the previous and following hybrids.

$\mathsf{Hybrid}_{s,9}$:

- The challenger sets $v_s^* \leftarrow \{0,1\}^n$. He sets $e \leftarrow \{0,1\}^n$ and $e' \leftarrow \{0,1\}^n$. Next, he sets $g = iO(P_e, e')$. For all $b \in \{0,1\}, i \in [1,n]$, he sets $\underline{(y_{1,0}^*, y_{1,1}^*), \ldots, (y_{n,0}^*, y_{n,1}^*) = F_1(K_1^{(n)}, v_s^*)}$, $u_s^*[i] = y_{i,g_i}^*$, $z_{i,b}^* = \mathsf{PRG}(y_{i,b}^*)$, where g_i is the i^{th} bit of g.
  He pads the program Adaptive-Parameters: 2 in Figure 37 appropriately [24] and sends an iO of the program to the adversary.
- Set $j = 0$. While the adversary is making queries to random oracle, increment j and repeat:
  1. Let the adversary query to the random oracle be on protocol description d_j^*.
  2. If $j < s$, the challenger sets the output of the random oracle, $v_j^* \leftarrow \{0,1\}^n$.
     He sets $K_3 \leftarrow \{0,1\}^n, e' \leftarrow \{0,1\}^n$. He queries the oracle to obtain the Parameters p_j^* and sets $g = iO(P'_{K_3,p_j^*,d_j^*}, e')$ (See Figure 39). For all $b \in \{0,1\}$ and $i \in [1,n]$, he sets $(y_{1,0}^*, y_{1,1}^*), \ldots, (y_{n,0}^*, y_{n,1}^*) = F_1(K_1^{(n)}, v_j^*)$, $u_j^*[i] = y_{i,g_i}^*$, where g_i is the i^{th} bit of g.
  3. If $j > s$, the challenger sets the output of the random oracle, $u_j^* \leftarrow \{0,1\}^{n^2}, v_j^* \leftarrow \{0,1\}^n$.
- The adversary then outputs a single bit b'.

Adaptive-Parameters: 2

Constants: v_s^*, g, PRF keys $K_1^{(n)}\{v_s^*\}$, K_2, K_2', $z_{i,b}^*$ for $i \in [1,n]$ and $b \in \{0,1\}$
Input: Program hash $u = u[1], \ldots, u[n]$, v.

```
1. If $v = v_s^*$ then for $i = 1, \ldots, n$ do
   If $\mathsf{PRG}(u[i]) = z_{i,0}^*$ let $x_i = 0$, if $\mathsf{PRG}(u[i]) = z_{i,1}^*$ let $x_i = 1$, else $x_i = \perp$.
   Go to step 4.
2. Compute $F_1(K_1^{(n)}, v) = (y_{1,0}, y_{1,1}), \ldots, (y_{n,0}, y_{n,1})$.
3. For $i = 1, \ldots, n$, if $u[i] = y_{i,0}$ set $x_i = 0$ else if $u[i] = y_{i,1}$ set $x_i = 1$ else set $x_i = \perp$
4. If $x \in \{0,1\}^n$ (i.e. no $\perp$s), output $x$.
5. Else set $K_3 = F_2(K_2, u|v)$, $r = F_2(K_2', u|v)$. Output $iO(P_{K_3}; r)$ of the program[a] $P_{K_3}$ of Figure 38.
```

[a]Appropriately appended to the maximum of the size of itself and P'_{K_3,p_j^*,d_j^*}

Figure 37: Program Adaptive-Parameters: 2

P_{K_3}

Constant: PRF key K_3.
Input: Program description d.

```
1. Output $d(F_3(K_3, d))$. Recall that $d$ is a program description which outputs $k$ bits.
```

Figure 38: Program P_{K_3}

P'_{K_3,p_j^*,d_j^*}

Constants: PRF key $K_3\{d_j^*\}, d_j^*, p_j^*$.
Input: Program description d.

```
1. If $d = d_j^*$ output $p_j^*$.
2. Else output $d(F_3(K_3, d))$. Recall that $d$ is a program description which outputs $k$ bits.
```

Figure 39: Program P'_{K_3,p_j^*,d_j^*}

Intuitively, indistinguishable from $\mathsf{Hybrid}_{s,8}$ by security of punctured PRF key $K_1^{(n)}\{d_s^*\}$.

[24]To the maximum of the size of itself and all corresponding programs (Adaptive-Parameters, Adaptive-Parameters: 3) in the previous and following hybrids.

$\mathsf{Hybrid}_{s,10}$:

- The challenger pads the program Adaptive-Parameters in Figure 40 appropriately [25] and sends an iO of the program to the adversary.
- Set $j = 0$. While the adversary is making queries to random oracle, increment j and repeat:
  1. Let the adversary query to the random oracle be on protocol description d_j^*.
  2. If $j < s$, the challenger sets the output of the random oracle, $v_j^* \leftarrow \{0,1\}^n$.
     He sets $K_3 \leftarrow \{0,1\}^n, e' \leftarrow \{0,1\}^n$. He queries the oracle to obtain the Parameters p_j^* and sets $g = iO(P'_{K_3,p_j^*,d_j^*}, e')$ (See Figure 42). For all $b \in \{0,1\}$ and $i \in [1,n]$, he sets $(y_{1,0}^*, y_{1,1}^*), \ldots, (y_{n,0}^*, y_{n,1}^*) = F_1(K_1^{(n)}, v_j^*)$, $u_j^*[i] = y_{i,g_i}^*$, where g_i is the i^{th} bit of g.
  3. If $j = s$, the challenger sets $v_j^* \leftarrow \{0,1\}^n$. He sets $e \leftarrow \{0,1\}^n$ and $e' \leftarrow \{0,1\}^n$. Next, he sets $g = iO(P_e, e')$. For all $i \in [1,n]$, he sets $(y_{1,0}^*, y_{1,1}^*), \ldots, (y_{n,0}^*, y_{n,1}^*) = F_1(K_1^{(n)}, v_j^*)$, $u_j^*[i] = y_{i,g_i}^*$, where g_i is the i^{th} bit of g.
  4. If $j > s$, the challenger sets the output of the random oracle, $u_j^* \leftarrow \{0,1\}^{n^2}, v_j^* \leftarrow \{0,1\}^n$.
- The adversary then outputs a single bit b'.

Adaptive-Parameters

Constants: PRF keys $K_1^{(n)}, K_2, K_2'$.
Input: Program hash $u = u[1], \ldots, u[n]$, v.

```
1. Compute $F_1(K_1^{(n)}, v) = (y_{1,0}, y_{1,1}), \ldots, (y_{n,0}, y_{n,1})$.
2. For $i = 1, \ldots, n$, if $u[i] = y_{i,0}$ set $x_i = 0$ else if $u[i] = y_{i,1}$ set $x_i = 1$ else set $x_i = \perp$
3. If $x \in \{0,1\}^n$ (i.e. no $\perp$s), output $x$.
4. Else set $K_3 = F_2(K_2, u|v)$, $r = F_2(K_2', u|v)$. Output $iO(P_{K_3}; r)$ of the program[a] $P_{K_3}$ of Figure 41.
```

[a]Appropriately appended to the maximum of the size of itself and P'_{K_3,p_j^*,d_j^*}

Figure 40: Program Adaptive-Parameters

P_{K_3}

Constant: PRF key K_3.
Input: Program description d.

```
1. Output $d(F_3(K_3, d))$. Recall that $d$ is a program description which outputs $k$ bits.
```

Figure 41: Program P_{K_3}

P'_{K_3,p_j^*,d_j^*}

Constants: PRF key $K_3\{d_j^*\}, d_j^*, p_j^*$.
Input: Program description d.

```
1. If $d = d_j^*$ output $p_j^*$.
2. Else output $d(F_3(K_3, d))$. Recall that $d$ is a program description which outputs $k$ bits.
```

Figure 42: Program P'_{K_3,p_j^*,d_j^*}

Intuitively, indistinguishable from $\mathsf{Hybrid}_{s,9}$ by iO security between Adaptive-Parameters: 2 and Adaptive-Parameters.

[25]To the maximum of the size of itself and all corresponding programs (Adaptive-Parameters: 2, Adaptive-Parameters: 3) in the previous and following hybrids.

$\mathsf{Hybrid}_{s,11}$:

- The challenger pads the program Adaptive-Parameters in Figure 43 appropriately [26] and sends an iO of the program to the adversary.
- Set $j = 0$. While the adversary is making queries to random oracle, increment j and repeat:
  1. Let the adversary query to the random oracle be on protocol description d_j^*.
  2. If $j < s$, the challenger sets the output of the random oracle, $v_j^* \leftarrow \{0,1\}^n$.
     He sets $K_3 \leftarrow \{0,1\}^n, e' \leftarrow \{0,1\}^n$. He queries the oracle to obtain the Parameters p_j^* and sets $g = iO(P'_{K_3,p_j^*,d_j^*}, e')$ (See Figure 45). For all $b \in \{0,1\}$ and $i \in [1,n]$, he sets $(y_{1,0}^*, y_{1,1}^*), \ldots, (y_{n,0}^*, y_{n,1}^*) = F_1(K_1^{(n)}, v_j^*)$, $u_j^*[i] = y_{i,g_i}^*$, where g_i is the i^{th} bit of g.
  3. If $j = s$, the challenger sets $v_s^* \leftarrow \{0,1\}^n$. He sets $e \leftarrow \{0,1\}^n$ and $e' \leftarrow \{0,1\}^n$. Next, he sets $\underline{p_j^* = d_j^*(F_3(e, d_j^*)), g = iO(P'_{e,p_j^*,d_j^*}, e')}$ (See Figure 45). For all $i \in [1,n]$, he sets $(y_{1,0}^*, y_{1,1}^*), \ldots, (y_{n,0}^*, y_{n,1}^*) = F_1(K_1^{(n)}, v_j^*)$, $u_j^*[i] = y_{i,g_i}^*$, where g_i is the i^{th} bit of g.
  4. If $j > s$, the challenger sets the output of the random oracle, $u_j^* \leftarrow \{0,1\}^{n^2}, v_j^* \leftarrow \{0,1\}^n$.
- The adversary then outputs a single bit b'.

Adaptive-Parameters

Constants: PRF keys $K_1^{(n)}$, K_2, K_2'.
Input: Program hash $u = u[1], \ldots, u[n]$, v.

1. Compute $F_1(K_1^{(n)}, v) = (y_{1,0}, y_{1,1}), \ldots, (y_{n,0}, y_{n,1})$.
2. For $i = 1, \ldots, n$, if $u[i] = y_{i,0}$ set $x_i = 0$ else if $u[i] = y_{i,1}$ set $x_i = 1$ else set $x_i = \perp$
3. If $x \in \{0,1\}^n$ (i.e. no $\perp$s), output x.
4. Else set $K_3 = F_2(K_2, u|v)$, $r = F_2(K_2', u|v)$. Output $iO(P_{K_3}; r)$ of the program[a] P_{K_3} of Figure 44.

[a]Appropriately appended to the maximum of the size of itself and P'_{K_3,p_j^*,d_j^*}

Figure 43: Program Adaptive-Parameters

P_{K_3}

Constant: PRF key K_3.
Input: Program description d.

1. Output $d(F_3(K_3, d))$. Recall that d is a program description which outputs k bits.

Figure 44: Program P_{K_3}

P'_{K_3,p_j^*,d_j^*}

Constants: PRF key $K_3\{d_j^*\}, d_j^*, p_j^*$.
Input: Program description d.

1. If $d = d_j^*$ output p_j^*.
2. Else output $d(F_3(K_3, d))$. Recall that d is a program description which outputs k bits.

Figure 45: Program P'_{K_3,p_j^*,d_j^*}

Intuitively, indistinguishable from $\mathsf{Hybrid}_{s,10}$ by security of iO between programs P_{K_3} and P'_{K_3,d^*,p^*}.

[26]To the maximum of the size of itself and all corresponding programs (Adaptive-Parameters: 2, Adaptive-Parameters: 3) in the previous and following hybrids.

$\mathsf{Hybrid}_{s,12}$:

- The challenger pads the program Adaptive-Parameters in Figure 46 appropriately [27] and sends an iO of the program to the adversary.
- Set $j = 0$. While the adversary is making queries to random oracle, increment j and repeat:
  1. Let the adversary query to the random oracle be on protocol description d_j^*.
  2. If $j < s$, the challenger sets the output of the random oracle, $v_j^* \leftarrow \{0,1\}^n$.
     He sets $K_3 \leftarrow \{0,1\}^n, e' \leftarrow \{0,1\}^n$. He queries the oracle to obtain the Parameters p_j^* and sets $g = iO(P'_{K_3,p_j^*,d_j^*}, e')$ (See Figure 48). For all $b \in \{0,1\}$ and $i \in [1,n]$, he sets $(y_{1,0}^*, y_{1,1}^*), \ldots, (y_{n,0}^*, y_{n,1}^*) = F_1(K_1^{(n)}, v_j^*)$, $u_j^*[i] = y_{i,g_i}^*$, where g_i is the ith bit of g.
  3. If $j = s$, the challenger sets $v_s^* \leftarrow \{0,1\}^n$. He sets $e \leftarrow \{0,1\}^n$ and $e' \leftarrow \{0,1\}^n$. Next, he sets $\underline{x' \leftarrow \{0,1\}^m, p_j^* = d_j^*(x')}$, $g = iO(P'_{e,p_j^*,d_j^*}, e')$ (See Figure 48). For all $i \in [1,n]$, he sets $(y_{1,0}^*, y_{1,1}^*), \ldots, (y_{n,0}^*, y_{n,1}^*) = F_1(K_1^{(n)}, v_j^*)$, $u_j^*[i] = y_{i,g_i}^*$, where g_i is the ith bit of g.
  4. If $j > s$, the challenger sets the output of the random oracle, $u_j^* \leftarrow \{0,1\}^{n^2}, v_j^* \leftarrow \{0,1\}^n$.
- The adversary then outputs a single bit b'.

Adaptive-Parameters

Constants: PRF keys $K_1^{(n)}$, K_2, K_2'.
Input: Program hash $u = u[1], \ldots, u[n]$, v.

```
1. Compute $F_1(K_1^{(n)}, v) = (y_{1,0}, y_{1,1}), \ldots, (y_{n,0}, y_{n,1})$.
2. For $i = 1, \ldots, n$, if $u[i] = y_{i,0}$ set $x_i = 0$ else if $u[i] = y_{i,1}$ set $x_i = 1$ else set $x_i = \perp$
3. If $x \in \{0,1\}^n$ (i.e. no $\perp$s), output $x$.
4. Else set $K_3 = F_2(K_2, u|v)$, $r = F_2(K_2', u|v)$. Output $iO(P_{K_3}; r)$ of the program[a] $P_{K_3}$ of Figure 47.
```

[a] Appropriately appended to the maximum of the size of itself and P'_{K_3,p_j^*,d_j^*}

Figure 46: Program Adaptive-Parameters

P_{K_3}

Constant: PRF key K_3.
Input: Program description d.

```
1. Output $d(F_3(K_3, d))$. Recall that $d$ is a program description which outputs $k$ bits.
```

Figure 47: Program P_{K_3}

P'_{K_3,p_j^*,d_j^*}

Constants: PRF key $K_3\{d_j^*\}, d_j^*, p_j^*$.
Input: Program description d.

```
1. If $d = d_j^*$ output $p_j^*$.
2. Else output $d(F_3(K_3, d))$. Recall that $d$ is a program description which outputs $k$ bits.
```

Figure 48: Program P'_{K_3,p_j^*,d_j^*}

Intuitively, indistinguishable from $\mathsf{Hybrid}_{s,11}$ by security of punctured PRF key $K_3 = e\{d_s^*\}$.

[27] To the maximum of the size of itself and all corresponding programs (Adaptive-Parameters: 2, Adaptive-Parameters: 3) in the previous and following hybrids.

Hybrid$_{s,13}$:

- The challenger pads the program Adaptive-Parameters in Figure 49 appropriately [28] and sends an iO of the program to the adversary.
- Set $j = 0$. While the adversary is making queries to random oracle, increment j and repeat:
  1. Let the adversary query to the random oracle be on protocol description d_j^*.
  2. If $j < s$, the challenger sets the output of the random oracle, $v_j^* \leftarrow \{0,1\}^n$. He sets $K_3 \leftarrow \{0,1\}^n, e' \leftarrow \{0,1\}^n$. He queries the oracle to obtain the Parameters p_j^* and sets $g = iO(P'_{K_3,p_j^*,d_j^*}, e')$ (See Figure 51). For all $b \in \{0,1\}$ and $i \in [1,n]$, he sets $(y_{1,0}^*, y_{1,1}^*), \ldots, (y_{n,0}^*, y_{n,1}^*) = F_1(K_1^{(n)}, v_j^*)$, $u_j^*[i] = y_{i,g_i}^*$, where g_i is the ith bit of g.
  3. If $j = s$, the challenger sets $v_s^* \leftarrow \{0,1\}^n$. He sets $e \leftarrow \{0,1\}^n$ and $e' \leftarrow \{0,1\}^n$. He queries the oracle to obtain the Parameters p_j^* and sets $g = iO(P'_{e,p_j^*,d_j^*}, e')$ (See Figure 51). For all $i \in [1,n]$, he sets $(y_{1,0}^*, y_{1,1}^*), \ldots, (y_{n,0}^*, y_{n,1}^*) = F_1(K_1^{(n)}, v_j^*)$, $u_j^*[i] = y_{i,g_i}^*$, where g_i is the ith bit of g.
  4. If $j > s$, the challenger sets the output of the random oracle, $u_j^* \leftarrow \{0,1\}^{n^2}, v_j^* \leftarrow \{0,1\}^n$.
- The adversary then outputs a single bit b'.

Adaptive-Parameters

Constants: PRF keys $K_1^{(n)}$, K_2, K_2'.
Input: Program hash $u = u[1], \ldots, u[n]$, v.

```
1. Compute $F_1(K_1^{(n)}, v) = (y_{1,0}, y_{1,1}), \ldots, (y_{n,0}, y_{n,1})$.
2. For $i = 1, \ldots, n$, if $u[i] = y_{i,0}$ set $x_i = 0$ else if $u[i] = y_{i,1}$ set $x_i = 1$ else set $x_i = \perp$
3. If $x \in \{0,1\}^n$ (i.e. no $\perp$s), output $x$.
4. Else set $K_3 = F_2(K_2, u|v)$, $r = F_2(K_2', u|v)$. Output $iO(P_{K_3}; r)$ of the program[a] $P_{K_3}$ of Figure 50.
```

[a]Appropriately appended to the maximum of the size of itself and P'_{K_3,p_j^*,d_j^*}

Figure 49: Program Adaptive-Parameters

P_{K_3}

Constant: PRF key K_3.
Input: Program description d.

```
1. Output $d(F_3(K_3, d))$. Recall that $d$ is a program description which outputs $k$ bits.
```

Figure 50: Program P_{K_3}

P'_{K_3,p_j^*,d_j^*}

Constants: PRF key $K_3\{d_j^*\}, d_j^*, p_j^*$.
Input: Program description d.

```
1. If $d = d_j^*$ output $p_j^*$.
2. Else output $d(F_3(K_3, d))$. Recall that $d$ is a program description which outputs $k$ bits.
```

Figure 51: Program P'_{K_3,p_j^*,d_j^*}

This is identical to Hybrid$_{s,12}$.

[28]To the maximum of the size of itself and all corresponding programs (Adaptive-Parameters: 2, Adaptive-Parameters: 3) in the previous and following hybrids.

Note that $\mathsf{Hybrid}_{q(\lambda),13}$ is the Ideal World and it describes how $\texttt{SimUGen}$ and $\texttt{SimRO}$ work in the first and second bullet points above, respectively.

5.2 Indistinguishability of the Hybrids

To establish Theorem 3, it suffices to prove the following claims,

Claim 5. $\mathsf{Hybrid}_0(1^\lambda)$ *and* $\mathsf{Hybrid}_{0,13}(1^\lambda)$ *are identical.*

Proof. Hybrid_0 and $\mathsf{Hybrid}_{s-1,13}$ are identical by inspection. ☐

Claim 6. *For* $s \in [q(\lambda)]$, $\mathsf{Hybrid}_{s-1,13}(1^\lambda)$ *and* $\mathsf{Hybrid}_{s,1}(1^\lambda)$ *are computationally indistinguishable.*

Proof. $\mathsf{Hybrid}_{s-1,13}$ and $\mathsf{Hybrid}_{s,1}$ are indistinguishable by security of iO between Adaptive-Parameters and Adaptive-Parameters: 2.

It is easy to observe that the programs Adaptive-Parameters and Adaptive-Parameters:2 are functionally equivalent for inputs $v \neq v_s^*$. Moreover, even on input $v = v_s^*$, such that $(z_{1,0}^*, z_{1,1}^*), \ldots, (z_{n,0}^*, z_{n,1}^*) = \mathsf{PRG}(F(K_1^{(n)}, v_s^*))$, the functionality of both circuits is identical if the PRG is injective.

Therefore, the obfuscated circuits must be indistinguishable by security of iO. Suppose they are not, then consider an adversary $\mathcal{D}_1$ who distinguishes between these hybrids with significant advantage.

$\mathcal{D}_1$ can be used to break *selective* security of the indistinguishability obfuscation (according to Definition 1) via the following reduction to iO distinguisher $\mathcal{D}$. $\mathcal{D}$ acts as challenger in the experiment of $\mathsf{Hybrid}_{s-1,13}$. The iO challenger $\mathsf{Samp}(1^\lambda)$ first activates the distinguisher $\mathcal{D}$, which samples input $v_s^* \leftarrow \{0,1\}^n$ and passes it to Samp.

The iO challenger on input v_s^* samples circuits C_0 = Adaptive-Parameters and C_1 = Adaptive-Parameters: 2 with $(z_{1,0}^*, z_{1,1}^*), \ldots, (z_{n,0}^*, z_{n,1}^*) = \mathsf{PRG}(F(K_1^{(n)}, v_s^*))$. We note that the condition $\Pr[\forall x, C_0(x) = C_1(x) : (C_0, C_1, \sigma) \leftarrow \mathit{Samp}(1^\lambda)] > 1 - \alpha(\lambda)$ is trivially satisfied for all auxiliary information σ and all negligible functions $\alpha(\cdot)$, since the circuits are always functionally equivalent.

The iO challenger then sends $C_x = iO(n, C_0)$ or $C_x = iO(n, C_1)$ to the adversary $\mathcal{D}$. $\mathcal{D}$ then acts as challenger against $\mathcal{D}_1$ in the distinguishing game between $\mathsf{Hybrid}_{s-1,13}$ and $\mathsf{Hybrid}_{s,1}$. He follows the $\mathsf{Hybrid}_{s-1,13}$ game, such that he sets the circuit to the obfuscated circuit C_x. Since $\mathcal{D}_1$ has significant distinguishing advantage, there exists a polynomial $\mathsf{p}(\cdot)$ such that,

$$\Big|\Pr\big[\mathcal{D}_1(\mathsf{Hybrid}_{s-1,13}) = 1\big] - \Pr\big[\mathcal{D}_1(\mathsf{Hybrid}_{s,1}) = 1\big]\Big| \geq 1/\mathsf{p}(\lambda).$$

We note that $\mathsf{Hybrid}_{s-1,13}$ and $\mathsf{Hybrid}_{s,1}$ correspond exactly to C_x being C_0 and C_1 respectively, thus we can just have $\mathcal{D}$ echo the output of $\mathcal{D}_1$ such that the following is true, for $\alpha(\cdot) = 1/\mathsf{p}(\cdot)$

$$\Big|\Pr\big[\mathcal{D}(\sigma, iO(n, C_0)) = 1 : (C_0, C_1, \sigma) \leftarrow \mathit{Samp}(1^\lambda)\big]$$
$$-\Pr\big[\mathcal{D}(\sigma, iO(n, C_1)) = 1 : (C_0, C_1, \sigma) \leftarrow \mathit{Samp}(1^\lambda)\big]\Big| \geq \alpha(\lambda)$$

In other words, if $\mathcal{D}_1$ predicts $\mathsf{Hybrid}_{s-1,13}$, then the obfuscation C_x is that of Adaptive-Parameters, and if it predicts $\mathsf{Hybrid}_{s,1}$, then the obfuscation C_x is that of Adaptive-Parameters:2 with $(z_{1,0}^*, z_{1,1}^*), \ldots, (z_{n,0}^*, z_{n,1}^*) = \mathsf{PRG}(F(K_1^{(n)}, v_s^*))$. ☐

Claim 7. *For $s \in [q(\lambda)]$, $\mathsf{Hybrid}_{s,1}(1^\lambda)$ and $\mathsf{Hybrid}_{s,2}(1^\lambda)$ are computationally indistinguishable.*

Proof. $\mathsf{Hybrid}_{s,1}$ and $\mathsf{Hybrid}_{s,2}$ are indistinguishable by security of puncturable PRF $K_1^{(n)}$.

Suppose they are not, then consider an adversary $\mathcal{D}_2$ who distinguishes between these hybrids with significant advantage. This adversary can be used to break *selective* security of the punctured PRF $K_1^{(n)}$(more precisely, at least *one* of the punctured PRF's in the sequence $K_1^{(n)}$) via the following reduction algorithm, that first gets the protocol hash v_s^* from the distinguisher $\mathcal{D}_2$.

Consider a sequence of $2n+1$ sub-hybrids, such for $i \le n$, the i^{th} sub-hybrid $\mathsf{Hybrid}_{s,1,i}$, is the same as $\mathsf{Hybrid}_{s,1}$ except that:
For $i < n$, $\forall j \le i, y_{j,0} \leftarrow \{0,1\}^n$. Also $\forall i < j \le n, y_{j,0} = \mathsf{PRF}(K_1^{j,0}, v_s^*)$ and $\forall j > n, y_{j,1} = \mathsf{PRF}(K_1^{j,0}, v_s^*)$.

For $i > n$, $\forall j \le n, y_{j,0} \leftarrow \{0,1\}^n$, $\forall n < j \le i, y_{j-n,1} \leftarrow \{0,1\}^n$ and $\forall j > i, y_{j-n,1} = \mathsf{PRF}(K_1^{j,1}, v_s^*)$.

Note that $\mathsf{Hybrid}_{s,1,0} \equiv \mathsf{Hybrid}_{s,1}$ and $\mathsf{Hybrid}_{s,1,2n} \equiv \mathsf{Hybrid}_{s,2}$.

Then, there exists some $j \in [0, 2n-1]$ such that $\mathcal{D}_2$ distinguishes between $\mathsf{Hybrid}_{s,1,j}$ and $\mathsf{Hybrid}_{s,1,j+1}$ with significant advantage.

Assume without loss of generality that $j < n$ (arguments for $j > n$ will follow similarly), then $\mathcal{D}_2$ can be used to break *selective* security of the punctured PRF $K_1^{j+1,0}$ via the following reduction algorithm, that first gets the protocol hash v_s^* from the distinguisher $\mathcal{D}_2$.

The PRF attacker submits v_s^* to the PRF challenger and receives the punctured PRF $K_1^{j+1,0}(\{v_s^*\})$ and the challenge a, which is either chosen uniformly at random or is the output of the PRF at v_s^*. The PRF attacker continues the experiment of $\mathsf{Hybrid}_{s,1,j}$ as challenger, except that he sets $y_{j+1,0}^* = a$.

Then, there exists polynomial $p(\cdot)$ such that

$$\Big|\Pr\big[\mathcal{D}_2(\mathsf{Hybrid}_{s,1,j}) = 1\big] - \Pr\big[\mathcal{D}_2(\mathsf{Hybrid}_{s,1,j+1}) = 1\big]\Big| \ge 1/2n\mathsf{p}(\lambda).$$

If $\mathcal{D}_2$ predicts $\mathsf{Hybrid}_{s,1,j}$, then a is the output of the PRF $K_1^{j+1,0}$ at v_s^*. If $\mathcal{D}_2$ predicts $\mathsf{Hybrid}_{s,1,j+1}$, then a was chosen uniformly at random.

Therefore, we can just have $\mathcal{D}$ echo the output of $\mathcal{D}_2$ such that

$$\Big|\Pr\big[\mathcal{D}(y = \mathsf{PRF}(K_1^{j+1,0}\{v_s^*\}, v_s^*)) = 1\big] - \Pr\big[\mathcal{D}(y \leftarrow \{0,1\}^n) = 1\big]\Big| \ge 1/2n\mathsf{p}(\lambda).$$

□

Claim 8. *For $s \in [q(\lambda)]$, $\mathsf{Hybrid}_{s,2}(1^\lambda)$ and $\mathsf{Hybrid}_{s,3}(1^\lambda)$ are computationally indistinguishable.*

Proof. $\mathsf{Hybrid}_{s,2}$ and $\mathsf{Hybrid}_{s,3}$ are indistinguishable by security of the PRG.

Suppose they are not, then consider an adversary $\mathcal{D}_3$ who distinguishes between these hybrids with significant advantage.

Now, consider a sequence of $2n+1$ sub-hybrids, where the i^{th} sub-hybrid $\mathsf{Hybrid}_{s,2,i}$ is identical to $\mathsf{Hybrid}_{s,2}$ except that:
For $i \leq n$, then $\forall j \leq i, z^*_{j,0} \leftarrow \{0,1\}^n$, $\forall i < j \leq n, z^*_{j,0} = \mathsf{PRG}(y^*)$ for $y^* \leftarrow \{0,1\}^n$, $\forall j > n, z^*_{j-n,1} = \mathsf{PRG}(y^*)$ for $y^* \leftarrow \{0,1\}^n$.
For $i > n$, then $\forall j < n, z^*_{j,0}, \leftarrow \{0,1\}^n$, $\forall n < j < i, z^*_{j-n,1} \leftarrow \{0,1\}^n$ and $\forall j \geq i, z^*_{j-n,1} = \mathsf{PRG}(y^*)$ for $y^* \leftarrow \{0,1\}^n$. Note that $\mathsf{Hybrid}_{s,2,0} \equiv \mathsf{Hybrid}_{s,2}$ and $\mathsf{Hybrid}_{s,2,2n} \equiv \mathsf{Hybrid}_{s,3}$.

Then, there exists some $j \in [0, 2n-1]$ such that $\mathcal{D}_3$ distinguishes between $\mathsf{Hybrid}_{s,2,j}$ and $\mathsf{Hybrid}_{s,2,j+1}$ with significant advantage. But we show that if this is true, then $\mathcal{D}_3$ can be used to break security of the PRG via the following reduction.

$\mathcal{D}$ is a distinguisher of the PRG security game which takes a PRG challenge a, setting $z^*_{j+1,0} = a$ if $j < n$ and $z^*_{j+1-n,1} = a$ if $j \geq n$. It then continues the rest of the experiment of $\mathsf{Hybrid}_{s,2,j}$ as the challenger for $\mathcal{D}_3$.

Then, there exists polynomial $p(\cdot)$ such that

$$\left|\Pr\left[\mathcal{D}_3(\mathsf{Hybrid}_{s,2,j}) = 1\right] - \Pr\left[\mathcal{D}_3(\mathsf{Hybrid}_{s,2,j+1}) = 1\right]\right| \geq 1/2n\mathsf{p}(\lambda).$$

If a was the output of a PRG, then we are in $\mathsf{Hybrid}_{s,2,j}$. If a was chosen as a random string, then we are in $\mathsf{Hybrid}_{s,2,j+1}$.

Therefore, we can just have $\mathcal{D}$ echo the output of $\mathcal{D}_{10}$ such that

$$\left|\Pr\left[\mathcal{D}(\mathsf{PRG}(y) \text{ for } y \leftarrow \{0,1\}^n) = 1\right] - \Pr\left[\mathcal{D}(y \leftarrow \{0,1\}^{2n}) = 1\right]\right| \geq 1/2n\mathsf{p}(\lambda).$$

□

Claim 9. *For $s \in [q(\lambda)]$, $\mathsf{Hybrid}_{s,3}(1^\lambda)$ and $\mathsf{Hybrid}_{s,4}(1^\lambda)$ are computationally indistinguishable.*

Proof. $\mathsf{Hybrid}_{s,3}$ and $\mathsf{Hybrid}_{s,4}$ are indistinguishable by security of iO between Adaptive-Parameters: 2 and Adaptive-Parameters: 3.

It is easy to see that Adaptive-Parameters: 2 and Adaptive-Parameters: 3 are functionally equivalent on all inputs other than (u_s^*, v_s^*). Moreover, on input v_s^*, note that the condition in Step 1 is never satisfied in Adaptive-Parameters: 2 except with probability 2^{-n}, since z^* is chosen at random. Therefore, the output of Adaptive-Parameters: 2 on input (u_s^*, v_s^*) is an iO of the program $P_{\mathsf{PRF}(K_2, u_s^* | v_s^*)}$ using randomness $\mathsf{PRF}(K_2', u_s^* | v_s^*)$. On input (u_s^*, v_s^*), the output of Adaptive-Parameters: 3 (which is g) is therefore the same as that of Adaptive-Parameters: 2.

Since their functionality is exactly identical on all inputs, both obfuscated circuits must be indistinguishable by security of iO. Suppose they are not, then consider an adversary $\mathcal{D}_4$ who distinguishes between these hybrids with significant advantage.

This adversary can be used to break *selective* security of the indistinguishability obfuscation (according to Definition 1) via the following reduction to iO distinguisher $\mathcal{D}$. $\mathsf{Samp}(1^\lambda)$ first activates the distinguisher $\mathcal{D}$. $\mathcal{D}$ picks (u_s^*, v_s^*) uniformly at random and passes them to Samp.

The iO challenger $\mathsf{Samp}(1^\lambda)$ on input (u_s^*, v_s^*) picks $z_{i,b}^* \leftarrow \{0,1\}^{2n}$ for all $i \in [1,n], b \in \{0,1\}$. He then samples circuits $C_0 =$ Adaptive-Parameters: 2 and $C_1 =$ Adaptive-Parameters: 3 setting g appropriately. We note that the condition $\Pr[\forall x, C_0(x) = C_1(x) : (C_0, C_1, \sigma) \leftarrow \mathsf{Samp}(1^\lambda)] > 1 - \alpha(\lambda)$ is trivially satisfied for $\alpha(\lambda) = 2^{-n}$.

The iO challenger then sends $C_x = iO(n, C_0)$ or $C_x = iO(n, C_1)$ to the adversary $\mathcal{D}$. $\mathcal{D}$ then acts as challenger against $\mathcal{D}_4$ in the distinguishing game between $\mathsf{Hybrid}_{s,3}$ and $\mathsf{Hybrid}_{s,4}$. He follows the $\mathsf{Hybrid}_{s,3}$ game, such that he sets the circuit to the obfuscated circuit C_x. Since $\mathcal{D}_4$ has significant distinguishing advantage, there exists a polynomial $\mathsf{p}(\cdot)$ such that,

$$\Big| \Pr\big[\mathcal{D}_4(\mathsf{Hybrid}_{s,3}) = 1\big] - \Pr\big[\mathcal{D}_4(\mathsf{Hybrid}_{s,4}) = 1\big] \Big| \geq 1/\mathsf{p}(\lambda).$$

We note that $\mathsf{Hybrid}_{s,3}$ and $\mathsf{Hybrid}_{s,4}$ correspond exactly to C_x being C_0 and C_1 respectively, thus we can just have $\mathcal{D}$ echo the output of $\mathcal{D}_4$ such that the following is true, for $\alpha(\cdot) = 1/\mathsf{p}(\cdot)$

$$\Big| \Pr\big[\mathcal{D}(\sigma, iO(n, C_0)) = 1 : (C_0, C_1, \sigma) \leftarrow Samp(1^\lambda)\big] \\ - \Pr\big[\mathcal{D}(\sigma, iO(n, C_1)) = 1 : (C_0, C_1, \sigma) \leftarrow Samp(1^\lambda)\big] \Big| \geq \alpha(\lambda)$$

In other words, if $\mathcal{D}_4$ predicts $\mathsf{Hybrid}_{s,3}$, then the obfuscation C_x is that of Adaptive-Parameters: 2, and if it predicts $\mathsf{Hybrid}_{s,4}$, then the obfuscation C_x is that of Adaptive-Parameters: 3.

□

Claim 10. *For $s \in [q(\lambda)]$, $\mathsf{Hybrid}_{s,4}(1^\lambda)$ and $\mathsf{Hybrid}_{s,5}(1^\lambda)$ are computationally indistinguishable.*

Proof. $\mathsf{Hybrid}_{s,4}$ and $\mathsf{Hybrid}_{s,5}$ are indistinguishable by security of the puncturable PRF K_2. Suppose they are not, then consider an adversary $\mathcal{D}_5$ who distinguishes between these hybrids with significant advantage. This adversary can be used to break *selective* security of the punctured PRF K_2 via the following reduction algorithm to distinguisher $\mathcal{D}$, that first gets the protocol hash (u_s^*, v_s^*) after activating the distinguisher $\mathcal{D}_5$.

The PRF attacker $\mathcal{D}$ gives (u_s^*, v_s^*) to the PRF challenger. The attacker receives the punctured PRF key $K_2\{u_s^*|v_s^*\}$ and the challenge a, which is either chosen uniformly at random or is the output of the PRF at $u_s^*|v_s^*$. The PRF attacker continues the experiment of $\mathsf{Hybrid}_{s,4}$ as challenger, except that he uses the same (u_s^*, v_s^*) and sets $e = a$.

Then, there exists polynomial $p(\cdot)$ such that

$$\Big|\Pr\big[\mathcal{D}_{5a}(\mathsf{Hybrid}_{s,4}) = 1\big] - \Pr\big[\mathcal{D}_{5a}(\mathsf{Hybrid}_{s,5}) = 1\big]\Big| \geq 1/\mathsf{p}(\lambda).$$

If $\mathcal{D}_5$ predicts $\mathsf{Hybrid}_{s,4}$, then a is the output of the punctured PRF K_2 at $u_s^*|v_s^*$. If $\mathcal{D}_5$ predicts $\mathsf{Hybrid}_{s,5}$, then a was chosen uniformly at random. Therefore, we can just have $\mathcal{D}$ echo the output of $\mathcal{D}_5$ such that

$$\Big|\Pr\big[\mathcal{D}(\mathsf{PRF}(K_2(\{v_s^*|u_s^*\}), v_s^*|u_s^*)) = 1\big] - \Pr\big[\mathcal{D}(y \leftarrow \{0,1\}^{n_1}) = 1\big]\Big| \geq 1/\mathsf{p}(\lambda).$$

$\square$

Claim 11. *For $s \in [q(\lambda)]$, $\mathsf{Hybrid}_{s,5}(1^\lambda)$ and $\mathsf{Hybrid}_{s,6}(1^\lambda)$ are computationally indistinguishable.*

Proof. $\mathsf{Hybrid}_{s,5}$ and $\mathsf{Hybrid}_{s,6}$ are indistinguishable by security of the puncturable PRF K_2'. Suppose they are not, then consider an adversary $\mathcal{D}_6$ who distinguishes between these hybrids with significant advantage. This adversary can be used to break *selective* security of the punctured PRF K_2' via the following reduction algorithm to distinguisher $\mathcal{D}$, that first gets the protocol hash (u_s^*, v_s^*) after activating the distinguisher $\mathcal{D}_6$.

The PRF attacker $\mathcal{D}$ gives (u_s^*, v_s^*) to the PRF challenger. The attacker receives the punctured PRF key $K_2'(\{u_s^*|v_s^*\})$ and the challenge a, which is either chosen uniformly at random or is the output of the PRF at $u_s^*|v_s^*$. The PRF attacker continues the experiment of $\mathsf{Hybrid}_{s,5}$ as challenger, except that he uses the same u_s^*, v_s^* and sets $e' = a$.

Then, there exists polynomial $p(\cdot)$ such that

$$\Big|\Pr\big[\mathcal{D}_6(\mathsf{Hybrid}_{s,5}) = 1\big] - \Pr\big[\mathcal{D}_6(\mathsf{Hybrid}_{s,6}) = 1\big]\Big| \geq 1/\mathsf{p}(\lambda).$$

If $\mathcal{D}_6$ predicts $\mathsf{Hybrid}_{s,5}$, then a is the output of the punctured PRF K_2 at $u_s^*|v_s^*$. If $\mathcal{D}_6$ predicts $\mathsf{Hybrid}_{s,6}$, then a was chosen uniformly at random. Therefore, we can just have $\mathcal{D}$ echo the output of $\mathcal{D}_6$ such that

$$\Big|\Pr\big[\mathcal{D}(\mathsf{PRF}(K_2(\{v_s^*|u_s^*\}), v_s^*|u_s^*) = 1\big] - \Pr\big[\mathcal{D}(y \leftarrow \{0,1\}^{n_2}) = 1\big]\Big| \geq 1/\mathsf{p}(\lambda).$$

$\square$

Claim 12. *For $s \in [q(\lambda)]$, $\mathsf{Hybrid}_{s,6}(1^\lambda)$ and $\mathsf{Hybrid}_{s,7}(1^\lambda)$ are computationally indistinguishable.*

Proof. $\mathsf{Hybrid}_{s,6}$ and $\mathsf{Hybrid}_{s,7}$ are indistinguishable by security of iO between Adaptive-Parameters: 2 and Adaptive-Parameters: 3.

Suppose they are not, then consider an adversary $\mathcal{D}_7$ who distinguishes between these hybrids with significant advantage. We will use $\mathcal{D}_7$ to break security of iO via the following reduction to distinguisher $\mathcal{D}$, which acts as challenger for $\mathcal{D}_7$.

$\mathsf{Samp}(1^\lambda)$ first activates the distinguisher $\mathcal{D}$. $\mathcal{D}$ sets $u_s^* \leftarrow \{0,1\}^n$, v_s^* according to $\mathsf{Hybrid}_{s,7}$ and gives (u_s^*, v_s^*) to Samp. $\mathcal{D}$ also gives punctured PRF keys K_1, K_2, K_2' at points $v_s^*, u_s^*|v_s^*$ respectively, along with $e, e' \leftarrow \{0,1\}^n, g = iO(P_e; e')$ and $z_{i,b}^* \leftarrow \{0,1\}^{2n}$ for all $i \in [1,n], b \in \{0,1\}$. Samp then samples circuit $C_0 =$ Adaptive-Parameters: 3 with the values of z^*, g set as above.

Samp also samples circuit $C_1 =$ Adaptive-Parameters: 2 except by setting $z_{i,g_i}^* = \mathsf{PRG}(u_s^*[i])$ (but setting $z_{i,\bar{g}_i}^* \leftarrow \{0,1\}^{2n}$).

The circuits C_0 and C_1 are easily seen to be functionally equivalent for $v \neq v_s^*$ and for $(u = u_s^*, v = v_s^*)$. We note that if z^* are chosen uniformly at random, the condition in step 2 is possibly satisfied in circuit C_0 with probability only 2^{-n} by security of the length-doubling PRG. Moreover, even in circuit C_1, this condition will only be satisfied on input u_s^* corresponding to v_s^* except with probability 2^{-n} by security of the length-doubling PRG and by injectivity of the PRG. Therefore, the condition $\Pr[\forall x, C_0(x) = C_1(x) : (C_0, C_1, \sigma) \leftarrow Samp(1^\lambda)] > 1 - \alpha(\lambda)$ is met for all auxiliary information σ and $\alpha(\lambda) = 2^{-(n-1)}$.

The iO adversary $\mathcal{D}$ obtains challenge circuit $C_x = iO(n, C_0)$ or $C_x = iO(n, C_1)$ from the iO challenger.

$\mathcal{D}$ then acts as challenger against $\mathcal{D}_7$ in the distinguishing game between $\mathsf{Hybrid}_{s,6}$ and $\mathsf{Hybrid}_{s,7}$. He follows the $\mathsf{Hybrid}_{s,7}$ game, such that he sends to $\mathcal{D}_7$, the obfuscated circuit C_x.

Since $\mathcal{D}_7$ has significant distinguishing advantage, there exists a polynomial $\mathsf{p}(\cdot)$ such that,

$$\Big|\Pr\big[\mathcal{D}_7(\mathsf{Hybrid}_{s,6}) = 1\big] - \Pr\big[\mathcal{D}_7(\mathsf{Hybrid}_{s,7}) = 1\big]\Big| \geq 1/\mathsf{p}(\lambda).$$

We note that $\mathsf{Hybrid}_{s,6}$ and $\mathsf{Hybrid}_{s,7}$ correspond exactly to C_x being C_0 and C_1 respectively, thus we can just have $\mathcal{D}$ echo the output of $\mathcal{D}_7$ such that the following is true, for $\alpha(\cdot) = 1/\mathsf{p}(\cdot)$

$$\Big|\Pr\big[\mathcal{D}(\sigma, iO(n, C_0)) = 1 : (C_0, C_1, \sigma) \leftarrow Samp(1^\lambda)\big]$$
$$-\Pr\big[\mathcal{D}(\sigma, iO(n, C_1)) = 1 : (C_0, C_1, \sigma) \leftarrow Samp(1^\lambda)\big]\Big| \geq \alpha(\lambda)$$

In other words, if $\mathcal{D}_7$ predicts $\mathsf{Hybrid}_{s,6}$, then the obfuscation C_x is that of Adaptive-Parameters: 3, and if it predicts $\mathsf{Hybrid}_{s,7}$, then the obfuscation C_x is that of Adaptive-Parameters: 2. ☐

Claim 13. *For $s \in [q(\lambda)]$, $\mathsf{Hybrid}_{s,7}(1^\lambda)$ and $\mathsf{Hybrid}_{s,8}(1^\lambda)$ are computationally indistinguishable.*

Proof. $\mathsf{Hybrid}_{s,7}$ and $\mathsf{Hybrid}_{s,8}$ are indistinguishable by security of the PRG.

Suppose they are not, then consider an adversary $\mathcal{D}_8$ that distinguishes between these hybrids with significant advantage.

Now, consider a sequence of $n+1$ sub-hybrids, where the i^{th} sub-hybrid $\mathsf{Hybrid}_{s,7,i}$ for $i \in [0,n]$ is identical to $\mathsf{Hybrid}_{s,7}$ except that:
For all $j \le i$, $z^*_{j,\bar{g_j}} = \mathsf{PRG}(y^*)$ for $y^* \leftarrow \{0,1\}^n$, and for all $j > i$, $z^*_{j,\bar{g_j}} \leftarrow \{0,1\}^{2n}$.
Note that $\mathsf{Hybrid}_{s,7,0} \equiv \mathsf{Hybrid}_{s,7}$ and $\mathsf{Hybrid}_{s,7,n} \equiv \mathsf{Hybrid}_{s,8}$.

Then, there exists some $j \in [0, n-1]$ such that $\mathcal{D}_8$ distinguishes between $\mathsf{Hybrid}_{s,7,j}$ and $\mathsf{Hybrid}_{s,7,j+1}$ with significant advantage. But we show that if this is true, then $\mathcal{D}_8$ can be used to break security of the PRG via the following reduction.

$\mathcal{D}$ is a distinguisher of the PRG security game which takes a PRG challenge a, setting $z^*_{j+1,\bar{g_{j+1}}} = a$. Note that he can do this since the seed of the PRG, $y^*_{j,\bar{g_j}}$ is not used anywhere else. He then continues the rest of the experiment of $\mathsf{Hybrid}_{s,7,j}$ as the challenger for $\mathcal{D}_8$.

Then, there exists polynomial $p(\cdot)$ such that

$$\Big|\Pr\big[\mathcal{D}_8(\mathsf{Hybrid}_{s,7,j}) = 1\big] - \Pr\big[\mathcal{D}_8(\mathsf{Hybrid}_{s,7,j+1}) = 1\big]\Big| \ge 1/2n\mathsf{p}(\lambda).$$

If a was the output of a PRG, then we are in $\mathsf{Hybrid}_{s,7,j}$. If a was chosen as a random string, then we are in $\mathsf{Hybrid}_{s,7,j+1}$.

Therefore, we can just have $\mathcal{D}$ echo the output of $\mathcal{D}_8$ such that

$$\Big|\Pr\big[\mathcal{D}(\mathsf{PRG}(y) \text{ for } y \leftarrow \{0,1\}^n) = 1\big] - \Pr\big[\mathcal{D}(y \leftarrow \{0,1\}^{2n}) = 1\big]\Big| \ge 1/n\mathsf{p}(\lambda).$$

□

Claim 14. *For $s \in [q(\lambda)]$, $\mathsf{Hybrid}_{s,8}(1^\lambda)$ and $\mathsf{Hybrid}_{s,9}(1^\lambda)$ are computationally indistinguishable.*

Proof. $\mathsf{Hybrid}_{s,8}$ and $\mathsf{Hybrid}_{s,9}$ are indistinguishable by security of the puncturable PRF $K_1^{(n)}$.

Suppose they are not, then consider an adversary $\mathcal{D}_9$ who distinguishes between these hybrids with significant advantage.

Now consider a sequence of $2n+1$ sub-hybrids, such for $i \le n$, the i^{th} sub-hybrid $\mathsf{Hybrid}_{s,8,i}$, is the same as $\mathsf{Hybrid}_{s,8}$ except that:
For $i < n$, $\forall j \le i, y_{j,0} = \mathsf{PRF}(K_1^{j,0}, v_s^*)$. Also $\forall i < j \le n, y_{j,0} \leftarrow \{0,1\}^n$ and $\forall j, y_{j,1} \leftarrow \{0,1\}^n$.

For $i > n$, $\forall j, y_{j,0} = \mathsf{PRF}(K_1^{j,0}, v_s^*)$, $\forall j \le i, y_{j-n,1} = \mathsf{PRF}(K_1^{j,1}, v_s^*)$ and $\forall j > i, y_{j-n,1} \leftarrow \{0,1\}^n$.

Note that $\mathsf{Hybrid}_{s,8,0} \equiv \mathsf{Hybrid}_{s,8}$ and $\mathsf{Hybrid}_{s,8,2n} \equiv \mathsf{Hybrid}_{s,9}$.

Then, there exists some $j \in [0, 2n-1]$ such that $\mathcal{D}_9$ distinguishes between $\mathsf{Hybrid}_{s,8,j}$ and $\mathsf{Hybrid}_{s,8,j+1}$ with significant advantage.

Assume without loss of generality that $j < n$ (arguments for $j > n$ will follow similarly), then $\mathcal{D}_9$ can be used to break *selective* security of the punctured PRF $K_1^{j+1,0}$ via the following reduction algorithm, that first gets the protocol v_s^* from the distinguisher $\mathcal{D}_9$.

The PRF attacker $\mathcal{D}$ submits v_s^* to the PRF challenger and receives the punctured PRF $K_1^{j+1,0}(\{v_s^*\})$ and the challenge a, which is either chosen uniformly at random or is the output of the PRF at v_s^*. Then $\mathcal{D}$ continues the experiment of $\mathsf{Hybrid}_{s,8,j}$ as challenger, except that he sets $y_{j+1,0}^* = a$, and programs $u_s^*[j+1]$ to $y_{j+1,0}^*$ if $p_{s,j+1}^* = 0$.

Then, there exists polynomial $p(\cdot)$ such that

$$\Big|\Pr\big[\mathcal{D}_9(\mathsf{Hybrid}_{s,8,j}) = 1\big] - \Pr\big[\mathcal{D}_9(\mathsf{Hybrid}_{s,8,j+1}) = 1\big]\Big| \ge 1/2n\mathsf{p}(\lambda).$$

If $\mathcal{D}_9$ predicts $\mathsf{Hybrid}_{s,8,j}$, then a was chosen uniformly at random. If $\mathcal{D}_9$ predicts $\mathsf{Hybrid}_{s,8,j+1}$, then a is the output of the PRF $K_1^{j+1,0}$ at v_s^*. Therefore, we can just have $\mathcal{D}$ echo the output of $\mathcal{D}_9$ such that

$$\Big|\Pr\big[\mathcal{D}(y = \mathsf{PRF}(K_1^{j+1,0}\{v_s^*\}, v_s^*)) = 1\big] - \Pr\big[\mathcal{D}(y \leftarrow \{0,1\}^n) = 1\big]\Big| \ge 1/2n\mathsf{p}(\lambda).$$

□

Claim 15. *For $s \in [q(\lambda)]$, $\mathsf{Hybrid}_{s,9}(1^\lambda)$ and $\mathsf{Hybrid}_{s,10}(1^\lambda)$ are computationally indistinguishable.*

Proof. $\mathsf{Hybrid}_{s,9}$ and $\mathsf{Hybrid}_{s,10}$ are indistinguishable by security of iO between circuits Adaptive-Parameters: 2 and Adaptive-Parameters.

It is easy to observe that the circuits Adaptive-Parameters: 2 and Adaptive-Parameters are functionally equivalent on all inputs $v \neq v^*$. Moreover, even on input $v = v_s^*$, such that $(z_{1,0}^*, z_{1,1}^*), \ldots, (z_{n,0}^*, z_{n,1}^*) = \mathsf{PRG}(F(K_1^{(n)}, v_s^*))$, the functionality of both circuits is identical if the PRG is injective.

The, the iO of both circuits must be indistinguishable. Suppose not, then consider an adversary $\mathcal{D}_{10}$ who distinguishes between these hybrids with significant advantage.

This adversary can be used to break *selective* security of the indistinguishability obfuscation (according to Definition 1) via the following reduction to distinguisher $\mathcal{D}$, which acts as challenger to distinguisher $\mathcal{D}_{10}$. $\mathcal{D}$ samples $v_s^* \leftarrow \{0,1\}^n$ and gives v_s^*, $(z_{1,0}^*, z_{1,1}^*), \ldots, (z_{1,0}^*, z_{1,1}^*) = \mathsf{PRG}(F_1(K_1^{(n)}, v_s^*))$ to the iO challenger $\mathsf{Samp}(1^\lambda)$.

Samp on input v_s^* samples circuits C_0 = Adaptive-Parameters: 2 and C_1 = Adaptive-Parameters with $(z_{1,0}^*, z_{1,1}^*), \ldots, (z_{n,0}^*, z_{n,1}^*) = \mathsf{PRG}(F_1(K_1^{(n)}, v_s^*))$. We note that the condition $\Pr[\forall x, C_0(x) = C_1(x) : (C_0, C_1, \sigma) \leftarrow \mathsf{Samp}(1^\lambda)] > 1 - \alpha(\lambda)$ is trivially satisfied for all auxiliary information σ and all negligible functions $\alpha(\cdot)$, since the circuits are always functionally equivalent.

The iO challenger then sends $C_x = iO(n, C_0)$ or $C_x = iO(n, C_1)$ to the adversary $\mathcal{D}$. $\mathcal{D}$ then acts as challenger against $\mathcal{D}_{10}$ in the distinguishing game between $\mathsf{Hybrid}_{s,9}$ and $\mathsf{Hybrid}_{s,10}$. He follows the $\mathsf{Hybrid}_{s,9}$ game, such that he sets the circuit to the obfuscated circuit C_x. Since $\mathcal{D}_{10}$ has significant distinguishing advantage, there exists a polynomial $\mathsf{p}(\cdot)$ such that,

$$\Big|\Pr\big[\mathcal{D}_1(\mathsf{Hybrid}_{s,9}) = 1\big] - \Pr\big[\mathcal{D}_1(\mathsf{Hybrid}_{s,10}) = 1\big]\Big| \geq 1/\mathsf{p}(\lambda).$$

We note that $\mathsf{Hybrid}_{s,9}$ and $\mathsf{Hybrid}_{s,10}$ correspond exactly to C_x being C_0 and C_1 respectively, thus we can just have $\mathcal{D}$ echo the output of $\mathcal{D}_1$ such that the following is true, for $\alpha(\cdot) = 1/\mathsf{p}(\cdot)$

$$\Big|\Pr\big[\mathcal{D}(\sigma, iO(n, C_0)) = 1 : (C_0, C_1, \sigma) \leftarrow Samp(1^\lambda)\big]$$
$$-\Pr\big[\mathcal{D}(\sigma, iO(n, C_1)) = 1 : (C_0, C_1, \sigma) \leftarrow Samp(1^\lambda)\big]\Big| \geq \alpha(\lambda)$$

In other words, if $\mathcal{D}_{10}$ predicts $\mathsf{Hybrid}_{s,9}$, then the obfuscation C_x is that of Adaptive-Parameters: 2 with $(z_{1,0}^*, z_{1,1}^*), \ldots, (z_{n,0}^*, z_{n,1}^*) = \mathsf{PRG}(F(K_1^{(n)}, v_s^*))$, and if it predicts $\mathsf{Hybrid}_{s,10}$, then the obfuscation C_x is that of Adaptive-Parameters. □

Claim 16. *For $s \in [q(\lambda)]$, $\mathsf{Hybrid}_{s,10}(1^\lambda)$ and $\mathsf{Hybrid}_{s,11}(1^\lambda)$ are computationally indistinguishable.*

Proof. $\mathsf{Hybrid}_{s,10}$ and $\mathsf{Hybrid}_{s,11}$ are indistinguishable by security of iO between circuits P_{K_3} and P'_{K_3,p_s^*,d_s^*}, if $p_s^* = d_s^*(\mathsf{PRF}(K_3, d_s^*))$. Note that the circuits are functionally equivalent for this setting of p_s^*.

Suppose these hybrids are not indistinguishable, then consider an adversary $\mathcal{D}_{11}$ who distinguishes between these hybrids with significant advantage.

This adversary can be used to break *selective* security of the indistinguishability obfuscation (according to Definition 1) via the following reduction to distinguisher $\mathcal{D}$, which acts as challenger in the experiment of $\mathsf{Hybrid}_{s,10}$ until it obtains d_s^* from the distinguisher $\mathcal{D}_{11}$ which it passes to the iO challenger, along with $p_s^* = d_s^*(\mathsf{PRF}(K_3, d_s^*))$.

$\mathsf{Samp}(1^\lambda)$ on input d_s^*, p_s^* samples circuits $C_0 = P_{K_3}$ and $C_1 = P'_{K_3,p_s^*,d_s^*}$.

We note that the condition $\Pr[\forall x, C_0(x) = C_1(x) : (C_0, C_1, \sigma) \leftarrow \mathsf{Samp}(1^\lambda)] > 1 - \alpha(\lambda)$ is always met since the circuits are functionally equivalent.

The iO adversary $\mathcal{D}$ then obtains $C_x = iO(n, C_0)$ or $C_x = iO(n, C_1)$. He continues as challenger in the distinguishing game between $\mathsf{Hybrid}_{s,10}$ and $\mathsf{Hybrid}_{s,11}$. He follows the $\mathsf{Hybrid}_{s,10}$ game, except that he sets g to the obfuscated circuit C_x. Since $\mathcal{D}_{11}$ has significant distinguishing advantage, there exists a polynomial $\mathsf{p}(\cdot)$ such that,

$$\left|\Pr\left[\mathcal{D}_{11}(\mathsf{Hybrid}_{s,10}) = 1\right] - \Pr\left[\mathcal{D}_{11}(\mathsf{Hybrid}_{s,11}) = 1\right]\right| \geq 1/\mathsf{p}(\lambda).$$

We note that $\mathsf{Hybrid}_{s,10}$ and $\mathsf{Hybrid}_{s,11}$ correspond exactly to C_x being C_0 and C_1 respectively, we can just have $\mathcal{D}$ echo the output of $\mathcal{D}_{11}$ such that the following is true, for $\alpha(\cdot) = 1/\mathsf{p}(\cdot)$

$$\Big|\Pr\left[\mathcal{D}(\sigma, iO(n, C_0)) = 1 : (C_0, C_1, \sigma) \leftarrow Samp(1^\lambda)\right]$$
$$-\Pr\left[\mathcal{D}(\sigma, iO(n, C_1)) = 1 : (C_0, C_1, \sigma) \leftarrow Samp(1^\lambda)\right]\Big| \geq \alpha(\lambda)$$

In other words, if $\mathcal{D}_{11}$ predicts $\mathsf{Hybrid}_{s,10}$, then the obfuscation C_x is that of P_{K_3}, and if it predicts $\mathsf{Hybrid}_{s,11}$, then the obfuscation C_x is that of P'_{K_3,p_s^*,d_s^*}. □

Claim 17. *For $s \in [q(\lambda)]$, $\mathsf{Hybrid}_{s,11}(1^\lambda)$ and $\mathsf{Hybrid}_{s,12}(1^\lambda)$ are computationally indistinguishable.*

Proof. $\mathsf{Hybrid}_{s,11}$ and $\mathsf{Hybrid}_{s,12}$ are indistinguishable by security of the puncturable PRF key $K_3 = e$.

Suppose they are not, then consider an adversary $\mathcal{D}_{12}$ who distinguishes between these hybrids with significant advantage. This adversary can be used to break *selective* security of the punctured PRF key K_3 via the following reduction to distinguisher $\mathcal{D}$.

The PRF attacker $\mathcal{D}$ begins the experiment of $\mathsf{Hybrid}_{s,11}$ and continues it until the hybrid adversary makes a random oracle query d_s^*. $\mathcal{D}$ passes d_s^* to the PRF challenger. The PRF challenger gives $\mathcal{D}$ the punctured PRF key $K_3(\{d_s^*\})$ and the challenge a, which is either chosen uniformly at random or is the output of the PRF at d_s^*. The PRF attacker continues the experiment of $\mathsf{Hybrid}_{s,11}$ as challenger, except that he sets $p_s^* = d_s^*(a)$.

Then, there exists polynomial $p(\cdot)$ such that

$$\Big|\Pr\big[\mathcal{D}_{12}(\mathsf{Hybrid}_{s,11}) = 1\big] - \Pr\big[\mathcal{D}_{12}(\mathsf{Hybrid}_{s,12}) = 1\big]\Big| \geq 1/\mathsf{p}(\lambda).$$

If $\mathcal{D}_{12}$ predicts $\mathsf{Hybrid}_{s,11}$, then a is the output of the punctured PRF K_3 at d_s^*. If $\mathcal{D}_{12}$ predicts $\mathsf{Hybrid}_{s,12}$, then a was chosen uniformly at random. Therefore, we can just have $\mathcal{D}$ echo the output of $\mathcal{D}_{12}$ such that

$$\Big|\Pr\big[\mathcal{D}(\mathsf{PRF}(K_3(\{d_s^*\}), d_s^*)) = 1\big] - \Pr\big[\mathcal{D}(y \leftarrow \{0,1\}^m) = 1\big]\Big| \geq 1/\mathsf{p}(\lambda).$$

□

Claim 18. *For $s \in [q(\lambda)]$, $\mathsf{Hybrid}_{s,12}(1^\lambda)$ and $\mathsf{Hybrid}_{s,13}(1^\lambda)$ are identical.*

Proof. $\mathsf{Hybrid}_{s,12}$ and $\mathsf{Hybrid}_{s,13}$ are identical when x' is sampled uniformly at random in $\{0,1\}^m$. □

5.3 No Honest Parameter Violations

Claim 19.
$$\Pr[\texttt{Ideal}(1^\lambda) \text{ aborts}] = 0$$

Proof. Note that whenever the adversary queries $\mathcal{H}$ on any input d, in the final hybrid we set $(u, v) = \mathcal{H}(d)$ to output the externally specified parameters. This can be verified by a simple observation of `InduceGen`.

Therefore, because of our construction, an "Honest Parameter Violation" never occurs in the ideal world for any d sent to the random oracle. That is, condition (1) in Definition 4 is always satisfied. In other words,
$$\Pr[\texttt{Ideal}(1^\lambda) \text{ aborts}] = 0$$
. □

References

[ABG$^+$13] Prabhanjan Ananth, Dan Boneh, Sanjam Garg, Amit Sahai, and Mark Zhandry. Differing-inputs obfuscation and applications. *IACR Cryptology ePrint Archive*, 2013:689, 2013. 7

[BBG13] James Ball, Julian Borger, and Glenn Greenwald. Revealed: how us and uk spy agencies defeat internet privacy and security. *The Guardian*, 2013. Online: http://www.theguardian.com/world/2013/sep/05/nsa-gchq-encryption-codes-security. 1

[BCP14] Elette Boyle, Kai-Min Chung, and Rafael Pass. On extractability obfuscation. In *TCC*, pages 52–73, 2014. 7

[BF03] Dan Boneh and Matthew K. Franklin. Identity-based encryption from the weil pairing. *SIAM J. Comput.*, 32(3):586–615, 2003. 2

[BGI13] Elette Boyle, Shafi Goldwasser, and Ioana Ivan. Functional signatures and pseudorandom functions. *IACR Cryptology ePrint Archive*, 2013:401, 2013. 5, 6

[BGLS03] Dan Boneh, Craig Gentry, Ben Lynn, and Hovav Shacham. Aggregate and verifiably encrypted signatures from bilinear maps. In *EUROCRYPT*, pages 416–432, 2003. 1

[BW13] Dan Boneh and Brent Waters. Constrained pseudorandom functions and their applications. *IACR Cryptology ePrint Archive*, 2013:352, 2013. 5, 6

[CFN$^+$14] Stephen Checkoway, Matt Fredrikson, Ruben Niederhagen, Matt Green, Tanja Lange, Tom Ristenpart, Daniel J. Bernstein, Jake Maskiewicz, and Hovav Shacham. On the practical exploitability of dual ec in tls implementations. In *USENIX Security*, 2014. 1

[CHP12] Jan Camenisch, Susan Hohenberger, and Michael Østergaard Pedersen. Batch verification of short signatures. *J. Cryptology*, 25(4):723–747, 2012. 1

[GGH$^+$13] Sanjam Garg, Craig Gentry, Shai Halevi, Mariana Raykova, Amit Sahai, and Brent Waters. Candidate indistinguishability obfuscation and functional encryption for all circuits. In *FOCS*, 2013. 5

[GGHR14] Sanjam Garg, Craig Gentry, Shai Halevi, and Mariana Raykova. Two-round secure mpc from indistinguishability obfuscation. In *TCC*, pages 74–94, 2014. 1

[GGM84] Oded Goldreich, Shafi Goldwasser, and Silvio Micali. How to construct random functions (extended abstract). In *FOCS*, pages 464–479, 1984. 6

[Hof14] Dennis Hofheinz. Fully secure constrained pseudorandom functions using random oracles. Cryptology ePrint Archive, Report 2014/372, 2014. http://eprint.iacr.org/. 3

[KPTZ13] Aggelos Kiayias, Stavros Papadopoulos, Nikos Triandopoulos, and Thomas Zacharias. Delegatable pseudorandom functions and applications. *IACR Cryptology ePrint Archive*, 2013:379, 2013. 5, 6

[LPS13] Jeff Larson, Nicole Perlroth, and Scott Shane. Revealed: The nsa's secret campaign to crack, undermine internet security. *Pro-Publica*, 2013. Online: http://www.propublica.org/article/the-nsas-secret-campaign-to-crack-undermine-internet-encryption. 1

[Nie02] Jesper Buus Nielsen. Separating random oracle proofs from complexity theoretic proofs: The non-committing encryption case. In *CRYPTO*, pages 111–126, 2002. 3, 8

[NIS12] NIST. Special publication 800-90: Recommendation for random number generation using deterministic random bit generators. *National Institue of Standards and Technology.*, 2012. Online: http://csrc.nist.gov/publications/PubsSPs.html#800-90A. 1

[PLS13] Nicole Perlroth, Jeff Larson, and Scott Shane. N.s.a. able to foil basic safeguards of privacy on web. *Internation New York Times*, 2013. Online: http://www.nytimes.com/2013/09/06/us/nsa-foils-much-internet-encryption.html. 1

[Sha85] Adi Shamir. Identity-based cryptosystems and signature schemes. In *Proceedings of CRYPTO 84 on Advances in cryptology*, pages 47–53, New York, NY, USA, 1985. Springer-Verlag New York, Inc. 2

[SW14] Amit Sahai and Brent Waters. How to use indistinguishability obfuscation: deniable encryption, and more. In *STOC*, pages 475–484, 2014. 2