How to Generate and use Universal Samplers

Dennis Hofheinz [*]
Karlsruher Institut für Technologie
Dennis.Hofheinz@kit.edu

Tibor Jager
Ruhr-Universität Bochum
Tibor.Jager@rub.de

Dakshita Khurana [†]
UCLA
Center for Encrypted Functionalities
dakshita@cs.ucla.edu

Amit Sahai [†]
UCLA
Center for Encrypted Functionalities
sahai@cs.ucla.edu

Brent Waters [‡]
University of Texas at Austin
Center for Encrypted Functionalities
bwaters@cs.utexas.edu

Mark Zhandry [§]
Stanford University
Center for Encrypted Functionalities
mzhandry@stanford.edu

Abstract

The random oracle is an idealization that allows to model a hash function as an oracle that will output a uniformly random string given an input. We introduce the notion of *universal sampler* scheme as a method sampling securely from *arbitrary* distributions.

We first motivate such a notion by describing several applications including generating the trusted parameters for many schemes from just a single trusted setup. We further demonstrate the versatility of universal sampler by showing how they give rise to applications such as identity-based encryption and multiparty key exchange.

We give a solution in the random oracle model based on indistinguishability obfuscation. At the heart of our construction and proof is a new technique we call "delayed backdoor programming".

[*] Supported by DFG grants GZ HO 4534/2-1 and GZ HO 4534/4-1.

[†] Research 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.

[§] Supported by the DARPA PROCEED program. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of DARPA.

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 [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 trusted parameters,
which can (securely) serve all cryptographic protocols?

[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.

Universal Sampler Schemes To solve the above problem what we need is a cryptographic primitive that allows us to (freshly) sample from an *arbitrary* distribution. We call such a primitive a universal sampler scheme. In such a system there will exist a function, `Sample`, which takes as input a polynomial sized circuit description, d, and outputs a sample $p = d(x)$ for some x. Intuitively, p should "look like" it was freshly sampled from the distribution induced by the function d. That is from an attack algorithm's perspective it should look like a call to the `Sample` algorithm induces a fresh sample by first selecting a random string x and then outputting $d(x)$, but keeping x hidden. (We will return to a formal definition shortly.)

Perhaps the most natural comparison of our notion is to the random oracle model put forth in the seminal work of Bellare and Rogaway [BR93]. Here a function H is (heuristically) modeled as an oracle that when called on a certain input will output a fresh sample of a random string x. The random oracle model has had a tremendous impact on the development of cryptography and several powerful techniques such as "programming" and "rewinding" have been used to leverage its power. However, functions modeled as random oracles are inherently limited to sampling random strings. Our work explores the power of a primitive that is "smarter" and can do this for any distribution.[2] Indeed, the connection to the random oracle model is strong in that our construction makes use of an (indistinguishability) obfuscated function that interacts with the outputs of a function modeled as a random oracle.

Applications of Universal Samplers

Universal setup. What we will see is that our notion of arbitrary sampling is powerful and allows for many applications. For starters let's return to the problem of providing a master setup for all cryptographic protocols. Using a universal sampler scheme this is quite simple. One will just simply publish a function description of `Sample`. Then if subsequently a (hopefully fantastic) new scheme is developed that has a trusted setup algorithm d, everyone can agree to use $p = \texttt{Sample}(d)$ as the scheme's parameters.

We can also use universal sampler schemes as a technical tool to build applications as varied as identity-based encryption (IBE), non-interactive key exchange (NIKE), and broadcast encryption (BE) schemes. We note that our goal is not to claim that our applications below are the "best" realizations of such primitives, but more to demonstrate the different and perhaps surprising ways a universal sampler scheme can be leveraged.

From the public-key to the identity-based setting. We first show how to transport cryptographic schemes from the public-key setting to the identity-based setting. For instance, consider a public-key encryption (PKE) scheme $\mathsf{PKE} = (\mathsf{PKGen}, \mathsf{PKEnc}, \mathsf{PKDec})$. Intuitively, to obtain an IBE scheme IBE from PKE, we use one PKE instance for each identity id of IBE.

[2] We note that random oracles are often used as a tool to help sample from various distributions. For example, we might use them to select a prime. In RSA full domain hash signatures [BR96], they are used to select a group element in $\mathbb{Z}_N^*$. This sampling occurs as a two step process. First, the function H is used to sample a fresh string x *which is completely visible to the attacker*. Then there is some post processing phase such as taking $x \pmod N$ to sample an integer mod N. In the literature this is often described as one function for the sake of brevity. However, the distinction between sampling with a universal sampler scheme and applying post processing to a random oracle output is very important.

A first attempt to do so would be to publish a description of $\texttt{Sample}$ as the master public key of IBE, and then to define a public key pk_{id} for identity id as

$$pk_{id} \ = \ \texttt{Sample}(d_{id}),$$

where d_{id} is the algorithm that first generates a PKE keypair $(pk, sk) \leftarrow \mathsf{PKGen}(1^\lambda)$ and then outputs pk. (Furthermore, to distinguish the keys for different identities, d_{id} contains id as a fixed constant that is built into its code, but not used.) This essentially establishes a "virtual" public-key infrastructure in the identity-based setting.

Encryption to an identity id can then be performed using PKEnc under public key pk_{id}. However, at this point, it is not clear how to derive individual secret keys sk_{id} that would allow to decrypt these ciphertexts. (In fact, this first scheme does not appear to have any master secret key to begin with.)

Hence, as a second attempt, we add a "master PKE public key" pk' from a chosen-ciphertext secure PKE scheme to IBE's master public key. Furthermore, we set

$$(pk_{id}, c'_{id}) \ = \ \texttt{Sample}(d_{id})$$

for the algorithm d_{id} that first samples $(pk, sk) \leftarrow \mathsf{PKGen}(1^\lambda)$, then encrypts sk under pk' via $c' \leftarrow \mathsf{PKEnc}'(pk', sk)$, and finally outputs (pk, c'). This way, we can use sk' as a master trapdoor to extract sk from c'_{id} and thus extract individual user secret keys.

We will show that this construction yields a selectively-secure secure IBE scheme once the used universal sampler scheme is selectively secure and the underlying PKE schemes are secure. Intuitively, during the analysis, we will substitute the user public key pk_{id^*} for the challenge identity id^* with a freshly generated PKE public key, and we will substitute the corresponding c'_{id^*} with a random ciphertext. This allows to embed an externally given PKE public key pk^*, and thus to use PKE's security.

Non-interactive key exchange and broadcast encryption. Furthermore, we provide a very simple construction of a multiparty non-interactive key exchange (NIKE) scheme. In an n-user NIKE scheme, a group of n parties wishes to agree on a shared random key k without any communication. Instead, user i derives k from its own secret key and the public keys of the other parties. (Since we are in the public-key setting, each party chooses its own keypair and publishes its own public key.) Of course, k should look random to any party not in the group.

We construct a conceptually very simple NIKE scheme from a universal sampler scheme and a PKE scheme $\mathsf{PKE} = (\mathsf{PKGen}, \mathsf{PKEnc}, \mathsf{PKDec})$ as follows: the public parameters of a description of $\texttt{Sample}$. Each party chooses a keypair $(pk, sk) \leftarrow \mathsf{PKGen}(1^\lambda)$. Finally, a shared key K among n parties with public keys from the set $S = \{pk_1, \ldots, pk_n\}$ is derived as follows. First, each party computes

$$(c_1, \ldots, c_n) \ = \ \texttt{Sample}(d_S),$$

where d_S is the algorithm that chooses a random key k, and then encrypts it under each pk_i to c_i (i.e., using $c_i \leftarrow \mathsf{PKEnc}(pk_i, k)$). Furthermore, d_S contains a description of the set S, e.g., as a comment. (This ensures that different sets S imply different algorithms d_S and thus different $\texttt{Sample}$ outputs.) Obviously, the party with secret key sk_i can derive k from c_i. On the other hand, we show that k remains hidden to any outsiders, even in an adaptive security model, assuming that the universal sampler scheme is adaptively secure, and the encryption scheme is (IND-CPA) secure.

We also give a variant of the protocol that has no setup at all. Roughly, we follow Boneh and Zhandry [BZ14] and designate one user as the "master party" who generates and publishes the universal sampler parameters along with her public key. Unfortunately, as in [BZ14], the basic conversion is totally broken in the adaptive setting. However, we make a small change to our protocol so that the resulting no-setup scheme *does* have adaptive security. This is in contrast to [BZ14], which required substantial changes to the scheme, achieved only a weaker *semi-static* security, and only obtained security though complexity leveraging. Not only is our scheme the first adaptively-secure multiparty NIKE without any setup, but it is the first to achieve adaptive security even among schemes with trusted setup, and it is the first to achieve *any* security beyond static security without relying on complexity leveraging. One trade-off is that our scheme is only proved secure in the random oracle model, whereas [BZ14] is proved secure in the standard model.

Finally, using an existing transformation of Boneh and Zhandry [BZ14], we also obtain a new distributed broadcast encryption scheme from our NIKE scheme.

Other work leveraging universal sampler schemes. Subsequent, to the initial posting of our paper two other papers have applied universal sampler schemes. Hohenberger, Koppula and Waters [HKW15] used universal samplers to achieve adaptive security without complexity leveraging for a new notion they called universal signature aggregators. Hofheinz, Kamath, Koppula and Waters [HKKW14] showed how to build adaptively secure constrained PRFs [BW13, BGI13, KPTZ13], for any circuits, using universal parameters as a key ingredient. All other such constructions required selective security or complexity leveraging.

Our Technical 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, `Setup`, which is called with a security parameter 1^n and randomness r. The output of this algorithm will be the universal sampler parameters U. In addition, there is a second algorithm `Sample` which takes as input the parameters U and the (circuit) description of a setup algorithm, d, and outputs the induced parameters p_d.[3]

We model security as an ideal/real game. In the real game an attacker will receive the 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 get back $p_i = \texttt{Setup}(U, d_i)$ for $i = 1, \ldots, q$. In the ideal world the attacker will first get the universal sampler 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

[3] Note that compared to our previous informal description, this more formal description of `Sample` takes an additional input U — the actual parameters — as input.

observing their transcripts. [4]

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 sampler 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 `Setup` 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 sampler 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 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 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 sampler 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 the universal sampler 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 sampler 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.

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

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

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 sampler parameters which consists of an obfuscated program U (produced from `Setup`) as well as a hash function H modeled as a random oracle and use these to determine the induced parameters for a particular protocol.

Let's begin by seeing how `Setup` 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 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 sampler parameters U_w are output.

We now show how to generate a set of induced parameters by executing $\texttt{Sample}(\texttt{U},\texttt{d})$. 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

[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.

sampler parameters U are published. The first step uses punctured programming techniques to replace the normal computation of the 1-time universal sampler 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 sampler 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 sampler 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 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*

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

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

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

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

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:*

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

- ***[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*

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

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,* $\mathit{negl}(\cdot)$ *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 sampler schemes. The essential property of a universal sampler scheme is that given the sampler parameters, and given any program d that generates samples from randomness (subject to certain size constraints, see below), it should be possible for any party to use the sampler parameters and the description of d to obtain induced samples that look like the samples 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 sampler 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 samples 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 sampler 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 samples as a "protocol". This is to emphasize that d can be used to generate arbitrary parameters for some protocol.

A universal parameter scheme consists of two algorithms:

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

(2) The second algorithm `Sample` takes as input sampler parameters U and a circuit d of size at most ℓ, and outputs induced samples 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 samples are indistinguishable from honestly generated samples to an arbitrary interactive system of adversarial and honest parties.

[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].

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 samples 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 *sampler* parameters. Parties would use the sampler parameters to derive induced samples 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 sampler parameters is to execute the sample derivation algorithm using the sampler parameters and some program descriptions d (corresponding to the protocols in which they participate) to obtain derived samples, which these honest parties then use in their interactions with the adversary.

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 sampler parameters U to the adversary such that these simulated sampler parameters U actually induce the completely honestly generated samples $d(F(d))$ created by the trusted party: in other words, that $\mathtt{Sample}(U, d) = d(F(d))$. Note that since honest parties are instructed to simply honestly compute induced samples, this ensures that honest parties in the ideal world would obtain these completely honestly generated samples $d(F(d))$. Thus, we do not need to model the honest parties explicitly – the adversary $\mathcal{A}$ can internally simulate any (set of) 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 samples $d^*(F(d^*))$ and the induced samples $\mathtt{Sample}(U, d^*)$. 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 sampler parameters U are set, and then an adversary can potentially adaptively choose a program d for generating samples for some adaptively chosen application scenario. For example, there may be several plausible implementations of a program to generate samples, 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 sampler parameters U of polynomial size, and then with no further interaction with the adversary, force $\mathtt{Sample}(U, d^*) = d^*(F(d^*))$ 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 sampler parameters to be useful to obtain induced samples 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 Samplers

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

Definition 3 (Selectively-Secure One-Time Universal Sampler Scheme)**.** *Let $\ell(\lambda), m(\lambda), k(\lambda)$ be efficiently computable polynomials. A pair of efficient algorithms* $(\texttt{Setup}, \texttt{Sample})$ *where* $\texttt{Setup}(1^\lambda) \to U$*,* $\texttt{Sample}(U, d) \to p_d$*, is a selectively-secure one-time universal sampler scheme if there exists an efficient algorithm* $\texttt{SimUGen}$ *such that:*

- *There exists a negligible function $negl(\cdot)$ 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:*

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

- *For every efficient adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$, where $\mathcal{A}_2$ outputs one bit, there exists a negligible function $negl(\cdot)$ 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* $\mathcal{A}_2(\texttt{Setup}(1^\lambda), \sigma)$.

  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:

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

3.2 Adaptively Secure Universal Samplers

We now proceed to define universal sampler schemes for the adaptive setting in the random oracle model. Our definition will also handle an unbounded number of induced samples 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 Sampler Scheme). *Let $\ell(\lambda), m(\lambda), k(\lambda)$ be efficiently computable polynomials. A pair of efficient oracle algorithms* $(\texttt{Setup}, \texttt{Sample})$ *where* $\texttt{Setup}(1^\lambda) \to U$, $\texttt{Sample}(U, d) \to p_d$, *is an adaptively-secure universal sampler 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 sampler parameters 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 (sample, 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 $p_d = \texttt{Sample}(U, d)$, making use of any additional random oracle queries, and $\mathcal{A}$ appends (d, p_d) to an auxiliary tape.*

    Remark. Intuitively, (sample, d) messages correspond to an honest party seeking a sample generated by program d. Recall that $\mathcal{A}$ is meant to internalize the behavior of honest parties.
- *Consider the following two experiments:*

  The experiment $\texttt{Real}(1^\lambda)$ *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. $U \leftarrow \texttt{Setup}^{\mathcal{H}}(1^\lambda)$
  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 $\texttt{Ideal}(1^\lambda)$ *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 Samples 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. $(U, \tau) \leftarrow \texttt{SimUGen}(1^\lambda)$. *Here,* $\texttt{SimUGen}$ *can make arbitrary queries to the Samples Oracle $\mathcal{O}$.*
  3. *$\mathcal{A}(1^\lambda, U)$ and $\texttt{SimRO}(\tau)$ begin simultaneous execution. Messages for $\mathcal{A}$ or* $\texttt{SimRO}$ *are handled as follows:*
     - *Whenever $\mathcal{A}$ sends a message of the form (RO, x), this is forwarded to* $\texttt{SimRO}$*, which produces a response to be sent back to $\mathcal{A}$.*
     - $\texttt{SimRO}$ *can make any number of queries to the Samples Oracle $\mathcal{O}$.*
     - *Finally, after $\mathcal{A}$ sends any message of the form (sample, 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 Sample*

Violation" has occurred. Please note that this is the only way that the experiment `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* `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 $negl(\cdot)$ such that the following two conditions hold:

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

and

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

3.2.1 One-Time Adaptive Security

While our construction of Section 5 will meet our adaptive definition above, some of our applications will only need a weaker notation of adaptive one-time security.

Definition 5 (Adaptively-Secure Universal Sampler Scheme)**.** *We say that a pair of efficient oracle algorithms* ($\texttt{Setup}, \texttt{Sample}$) *is an adaptively-secure* one-time *universal sampler scheme if they meet the adaptive definition given above, but with the added restriction that an admissible adversary $\mathcal{A}$ is only allowed to send a single message of the form* (sample, d).

4 Selective One-Time Universal Samplers

In this section, we prove the following theorem:

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

The required Selective One-Time Universal Sampler Scheme consists of programs `Setup` and `Sample`.

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

```
**Selective-Single-Samples**

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-Samples

[8] Appropriately padded to the maximum of the size of itself and Program Selective-Single-Samples: 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 samples 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-Samples in Figure 2.
- The adversary queries the program on input d^* to obtain the samples.

```
Selective-Single-Samples

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-Samples

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-Samples: 2 in Figure 3.
- The adversary queries the program on input d^* to obtain the samples.

```
Selective-Single-Samples: 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-Samples: 2

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

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

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-Samples: 2 in Figure 4.
- The adversary queries the program on input d^* to obtain the samples.

Selective-Single-Samples: 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-Samples: 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 samples p_d, punctures K at d^* and outputs an iO of the program[12] Selective-Single-Samples: 2 in Figure 5. This is then sent to the adversary.
- The adversary queries the program on input d^* to obtain the samples.

Selective-Single-Samples: 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-Samples: 2

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

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

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-Samples and Selective-Single-Samples: 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-Samples according to Figure 2 and C_1 = Selective-Single-Samples: 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-Samples to the adversary $\mathcal{D}_1$. Since $\mathcal{D}_1$ has significant distinguishing advantage, there exists a polynomial $\mathsf{p}(\cdot)$ such that,

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

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 $\alpha(\cdot) = 1/\mathsf{p}(\cdot)$

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

□

Claim 2. $\mathsf{Hybrid}_1(1^\lambda)$ *and* $\mathsf{Hybrid}_2(1^\lambda)$ *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

$$\Big|\Pr\big[\mathcal{D}_2(\mathsf{Hybrid}_1) = 1\big] - \Pr\big[\mathcal{D}_2(\mathsf{Hybrid}_2) = 1\big]\Big| \geq 1/p(\lambda).$$

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.

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

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

5 Adaptively Secure Universal Samplers

In this section, we prove the following theorem:

Theorem 3 (Adaptively Secure Universal Samplers)**.** *If indistinguishability obfuscation and injective* PRG*s exist, then there exists an adaptively secure universal sampler 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 `Setup` and `Sample`.

- `Setup`(1^λ) first samples PRF keys K_1, K_2, K_2' and then sets Sampler Parameters U to be an indistinguishability obfuscation of the program Adaptive-Samples [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 samples for any protocol description d.

  Denote by $F_1^{(n)} = \{F_1^{1,0}, F_1^{1,1}, F_1^{2,0}, F_1^{2,1} \ldots F_1^{n,0}, F_1^{n,1}\}$ a sequence of $2n$ puncturable PRF's that each take n-bit inputs and output n bits. For some key sequence $\{K_1^{1,0}, K_1^{1,1}, K_1^{2,0}, K_1^{2,1} \ldots K_1^{n,0}, K_1^{n,1}\}$, 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 samples generated by $d(\cdot)$.

- `Sample`(U, d) queries the random oracle $\mathcal{H}$ to obtain $(u, v) = \mathcal{H}(d)$. It then runs the program U generated by `Setup`(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 samples.

[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-Samples

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-Samples

```
$P_{K_3}$

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 (sample, d) - and construct a modified adversary, such that whenever $\mathcal{A}$ sends message (sample, d), our modified adversary sends message (RO, d) and then sends message (sample, 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 (sample, 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 $\texttt{Setup}$ and $\texttt{Sample}$ 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 $\texttt{SimUGen}, \texttt{SimRO}$ 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-Samples in Figure 8 to be the maximum of the size of itself and all corresponding programs (Adaptive-Samples: 2, Adaptive-Samples: 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-Samples

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-Samples

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}

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

- ○ The challenger pads the program Adaptive-Samples 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 samples 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-Samples

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-Samples

```
                                  $P_{K_3}$

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-Samples: 2, Adaptive-Samples: 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-Samples: 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 samples 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-Samples: 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-Samples

```
$P_{K_3}$

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-Samples: -2.

[16] To the maximum of the size of itself and all corresponding programs (Adaptive-Samples, Adaptive-Samples: 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-Samples: 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 samples 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-Samples: 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-Samples

```
                              $P_{K_3}$

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-Samples, Adaptive-Samples: 3) in the previous and following hybrids.

$\mathsf{Hybrid}_{s,3}$:

- 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-Samples: 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 samples 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-Samples: 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-Samples

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 $\mathsf{Hybrid}_{s,2}$ by security of the PRG.

[18] To the maximum of the size of itself and all corresponding programs (Adaptive-Samples, Adaptive-Samples: 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-Samples: 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 samples 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-Samples: 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-Samples

```
$P_{K_3}$

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-Samples:2-3.

[19] To the maximum of the size of itself and all corresponding programs (Adaptive-Samples, Adaptive-Samples: 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-Samples: 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 samples 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-Samples: 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 $x_i = \bot$.
     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 $x_i = \bot$
  5. If $x \in \{0,1\}^n$ (i.e. no $\bot$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-Samples

```
                                   $P_{K_3}$

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-Samples, Adaptive-Samples: 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-Samples: 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 samples 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-Samples: 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-Samples

```
                                    $P_{K_3}$

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-Samples, Adaptive-Samples: 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-Samples: 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 samples 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-Samples: 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-Samples: 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-Samples:3-2.

[22] To the maximum of the size of itself and all corresponding programs (Adaptive-Samples, Adaptive-Samples: 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 $z_{i,\bar{g}_i}^* = \mathsf{PRG}(y_{i,\bar{g}_i}^*)$, where g_i is the i^{th} bit of g and $\bar{g}_i = 1 - g_i$.

  He pads the program Adaptive-Samples: 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 samples 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-Samples: 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-Samples: 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-Samples, Adaptive-Samples: 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 $(y_{1,0}^*, y_{1,1}^*), \ldots, (y_{n,0}^*, y_{n,1}^*) = F_1(K_1^{(n)}, v_s^*)$, $u_s^*[i] = y_{i,g_i}^*$, $z_{i,b}^* = \mathsf{PRG}(y_{i,b}^*)$, where g_i is the i^{th} bit of g.
  He pads the program Adaptive-Samples: 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 samples 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-Samples: 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-Samples: 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-Samples, Adaptive-Samples: 3) in the previous and following hybrids.

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

- ○ The challenger pads the program Adaptive-Samples 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 samples 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 $(y_{1,0}^*, y_{1,1}^*), \dots, (y_{n,0}^*, y_{n,1}^*) = F_1(K_1^{(n)}, v_j^*)$, $u_j^*[i] = y_{i,g_i}^*$, 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 $(y_{1,0}^*, y_{1,1}^*), \dots, (y_{n,0}^*, y_{n,1}^*) = F_1(K_1^{(n)}, v_j^*)$, $u_j^*[i] = y_{i,g_i}^*$, 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-Samples
Constants: PRF keys $K_1^{(n)}$, $K_2, K_2'$.
Input: Program hash $u = u[1], \dots, u[n]$, $v$.
  1. Compute $F_1(K_1^{(n)}, v) = (y_{1,0}, y_{1,1}), \dots, (y_{n,0}, y_{n,1})$.
  2. For $i = 1, \dots, n$, 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-Samples

```
                         $P_{K_3}$
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-Samples: 2 and Adaptive-Samples.

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

Hybrid$_{s,11}$:

- ○ The challenger pads the program Adaptive-Samples 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 samples 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 $p_j^* = d_j^*(F_3(e, d_j^*)), g = iO(P'_{e,p_j^*,d_j^*}, e')$ (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-Samples

```
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-Samples

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 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-Samples: 2, Adaptive-Samples: 3) in the previous and following hybrids.

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

- ○ The challenger pads the program Adaptive-Samples 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 samples 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 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 $\underline{x' \leftarrow \{0,1\}^m, p_j^* = d_j^*(x')}$, $g = iO(P'_{e,p_j^*,d_j^*}, e')$ (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 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-Samples

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-Samples

```
                         $P_{K_3}$

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-Samples: 2, Adaptive-Samples: 3) in the previous and following hybrids.

$\mathsf{Hybrid}_{s,13}$:

- The challenger pads the program Adaptive-Samples 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 samples 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 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 $e' \leftarrow \{0,1\}^n$. He queries the oracle to obtain the samples 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 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-Samples

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-Samples

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

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

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

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-Samples and Adaptive-Samples: 2.

It is easy to observe that the programs Adaptive-Samples and Adaptive-Samples: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-Samples and C_1 = Adaptive-Samples: 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 $\Pr[\forall x, C_0(x) = C_1(x) : (C_0, C_1, \sigma) \leftarrow Samp(1^\lambda)] > 1 - \alpha(\lambda)$ 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 $\alpha(\cdot) = 1/\mathsf{p}(\cdot)$

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

In other words, if $\mathcal{D}_1$ predicts $\mathsf{Hybrid}_{s-1,13}$, then the obfuscation C_x is that of Adaptive-Samples, and if it predicts $\mathsf{Hybrid}_{s,1}$, then the obfuscation C_x is that of Adaptive-Samples: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

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

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

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

□

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-Samples: 2 and Adaptive-Samples: 3.

It is easy to see that Adaptive-Samples: 2 and Adaptive-Samples: 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-Samples: 2 except with probability 2^{-n}, since z^* is chosen at random. Therefore, the output of Adaptive-Samples: 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-Samples: 3 (which is g) is therefore the same as that of Adaptive-Samples: 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-Samples: 2 and $C_1 =$ Adaptive-Samples: 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 $\alpha(\cdot) = 1/\mathsf{p}(\cdot)$

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

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

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

□

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

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

□

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-Samples: 2 and Adaptive-Samples: 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-Samples: 3 with the values of z^*, g set as above.

Samp also samples circuit $C_1 =$ Adaptive-Samples: 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 $\Pr[\forall x, C_0(x) = C_1(x) : (C_0, C_1, \sigma) \leftarrow \mathsf{Samp}(1^\lambda)] > 1 - \alpha(\lambda)$ 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 $\alpha(\cdot) = 1/\mathsf{p}(\cdot)$

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

In other words, if $\mathcal{D}_7$ predicts $\mathsf{Hybrid}_{s,6}$, then the obfuscation C_x is that of Adaptive-Samples: 3, and if it predicts $\mathsf{Hybrid}_{s,7}$, then the obfuscation C_x is that of Adaptive-Samples: 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 $i \leq n$, the i^{th} sub-hybrid $\mathsf{Hybrid}_{s,8,i}$, is the same as $\mathsf{Hybrid}_{s,8}$ except that:
For $i < n$, $\forall j \leq i, y_{j,0} = \mathsf{PRF}(K_1^{j,0}, v_s^*)$. Also $\forall i < j \leq n, y_{j,0} \leftarrow \{0,1\}^n$ and $\forall j, y_{j,1} \leftarrow \{0,1\}^n$.

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

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

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

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

□

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-Samples: 2 and Adaptive-Samples.

It is easy to observe that the circuits Adaptive-Samples: 2 and Adaptive-Samples 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-Samples: 2 and C_1 = Adaptive-Samples 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,

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

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 $\alpha(\cdot) = 1/\mathsf{p}(\cdot)$

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

In other words, if $\mathcal{D}_{10}$ predicts $\mathsf{Hybrid}_{s,9}$, then the obfuscation C_x is that of Adaptive-Samples: 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-Samples. □

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,

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

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 $\alpha(\cdot) = 1/\mathsf{p}(\cdot)$

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

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 Sample Violations

Claim 19.

$$\Pr[\mathtt{Ideal}(1^\lambda) \textit{ aborts}] = 0$$

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 samples. This can be verified by a simple observation of `Sample`.

Therefore, because of our construction, an "Honest Sample 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,

$$\Pr[\mathtt{Ideal}(1^\lambda) \text{ aborts}] = 0$$

. □

6 IBE from PKE and Universal Parameters

We now describe further applications of universal samplers. We start with a direct, natural, and very simple construction of identity-based encryption from public-key encryption and a universal sampler scheme. Even though we will consider only public-key encryption, we note that the approach extends easily to other cryptographic public-key primitives. For instance, it can be used in the same way to convert a digital signature scheme into an identity-based signature scheme.

6.1 Basic Definitions

Definition 6 (Secure Public-Key Encryption)**.** *A public-key encryption scheme (PKE) consists of PPT algorithms* $\mathsf{PKE} = (\mathsf{PKGen}, \mathsf{PKEnc}, \mathsf{PKDec})$.

Key generation. PKGen *takes as input security parameter* 1^λ *and returns a key pair* (pk, sk).

Encryption. PKEnc *takes as input public key* pk *and message* m*, and returns a ciphertext* $c \leftarrow \mathsf{PKEnc}(pk, m)$.

Decryption. PKDec *takes as input secret key* sk *and ciphertext* c*, and returns a message* $m \leftarrow \mathsf{PKDec}(sk, c)$.

We require the usual correctness properties.

We say that public-key encryption scheme PKE is *wIND-CCA secure*, if

$$negl(\lambda) \geq \left| \Pr[\mathsf{Exp}^{\mathsf{wcca-0}}_{\mathsf{PKE},\mathcal{A}} \mathcal{A}(\lambda) = 1] - \Pr[\mathsf{Exp}^{\mathsf{wcca-1}}_{\mathsf{PKE},\mathcal{A}}(\lambda) = 1] \right|$$

for some negligible function $negl$ and for all PPT attackers $\mathcal{A}$, where $\mathsf{Exp}^{\mathsf{wcca-b}}_{\mathcal{A},\mathsf{PKE}}(\lambda)$ is the following experiment with scheme PKE and (stateful) attacker $\mathcal{A}$:

1. The experiment runs $(pk, sk) \leftarrow \mathsf{PKGen}(1^\lambda)$.
2. The attacker, on input pk, outputs two messages $(m_0, m_1) \leftarrow \mathcal{A}(1^\lambda, pk)$.
3. The experiment computes $c^* \leftarrow \mathsf{PKEnc}(pk, m_b)$ and returns whatever algorithm $\mathcal{A}^{\mathcal{O}_{\mathsf{wCCA}}}(1^\lambda, c^*)$ returns. Here $\mathcal{O}_{\mathsf{wCCA}}$ is an oracle that on input c returns $\mathsf{PKDec}(sk, c)$ for all $c \neq c^*$.

Note that this is a weakened version of standard IND-CCA security, because the attacker has access to $\mathcal{O}_{\mathsf{wCCA}}$ only after seeing the challenge ciphertext.

We say that public-key encryption scheme PKE is *IND-CPA secure*, if

$$negl(\lambda) \geq \left| \Pr[\mathsf{Exp}^{\mathsf{cpa-0}}_{\mathsf{PKE},\mathcal{A}}(\lambda) = 1] - \Pr[\mathsf{Exp}^{\mathsf{cpa-1}}_{\mathsf{PKE},\mathcal{A}}(\lambda) = 1] \right|$$

for some negligible function $negl$ and for all PPT attackers $\mathcal{A}$, where $\mathsf{Exp}^{\mathsf{cpa-b}}_{\mathcal{A},\mathsf{PKE}}(\lambda)$ denotes an experiment which is identical to $\mathsf{Exp}^{\mathsf{wcca-b}}_{\mathcal{A},\mathsf{PKE}}(\lambda)$, except that $\mathcal{A}$ does not have access to oracle $\mathcal{O}_{\mathsf{wCCA}}$.

Definition 7 (Secure Identity-based Encryption)**.** *An identity-based encryption scheme (IBE) consists of PPT Algorithms* $\mathsf{IBE} = (\mathsf{IDGen}, \mathsf{IDKey}, \mathsf{IDEnc}, \mathsf{IDDec})$*.*

Master key generation. IDGen *takes as input security parameter* 1^λ *and returns a master key pair* (mpk, msk)*.*

User key extraction. IDKey *takes as input an identity* $id \in \{0,1\}^\lambda$ *and* msk*, and returns an identity secret key* sk_{id}*.*

Encryption. IDEnc *takes as input master public key* mpk*, identity* id*, and message* m*, and returns a ciphertext* $c \leftarrow \mathsf{IDEnc}(pk, id, m)$*.*

Decryption. IDDec *takes as input secret key* sk_{id} *and ciphertext* c*, and returns a message* $m \leftarrow \mathsf{PKDec}(sk_{id}, c)$*.*

We require the usual correctness properties.

We say that IBE *is* selectively IND-ID-CPA secure*, if*

$$negl(\lambda) \geq \left| \mathsf{Adv}^{\mathsf{ind\text{-}id\text{-}cpa}}_{\mathsf{IBE},\mathcal{A}}(\lambda) := \Pr[\mathsf{Exp}^{\mathsf{ind\text{-}id\text{-}cpa\text{-}0}}_{\mathsf{IBE},\mathcal{A}}(\lambda) = 1] - \Pr[\mathsf{Exp}^{\mathsf{ind\text{-}id\text{-}cpa\text{-}1}}_{\mathsf{IBE},\mathcal{A}}(\lambda) = 1] \right|$$

for some negligible function negl and for all PPT attackers $\mathcal{A}$*, where* $\mathsf{Exp}^{\mathsf{ind\text{-}id\text{-}cpa\text{-}b}}_{\mathsf{IBE},\mathcal{A}}$ *is the following security experiment with scheme* IBE *and (stateful) attacker* $\mathcal{A}$ *having access to an oracle* $\mathcal{O}_{\mathsf{IBE}}$*.*

1. *The experiment runs* $id^* \leftarrow \mathcal{A}(1^\lambda)$*.*
2. *The experiment runs* $(mpk, msk) \leftarrow \mathsf{IDGen}(1^\lambda)$*.*
3. *The attacker, on input* mpk *outputs two messages* $(m_0, m_1) \leftarrow \mathcal{A}^{\mathcal{O}_{\mathsf{IBE}}}(1^\lambda, pk)$*.*
4. *The experiment computes* $c^* \leftarrow \mathsf{IDEnc}(mpk, id^*, m_b)$ *and returns whatever* $\mathcal{A}^{\mathcal{O}_{\mathsf{IBE}}}(1^\lambda, c^*)$ *returns.*

Here oracle $\mathcal{O}_{\mathsf{IBE}}$ *takes as input* id *such that* $id \neq id^*$*, and returns* $sk_{id} \leftarrow \mathsf{IDKey}(msk, id)$*.*

6.2 Universal Samplers with Additional Input

Recall that algorithm $\texttt{Sample}(U, d)$ of a universal sampler scheme receives as input a sampling algorithm d for some distribution described by program d. Sometimes, for instance in our generic construction of IBE from PKE and a universal sampler scheme, a slightly different definition will be useful, where program d may be fixed, but $\texttt{Sample}$ takes a bit string x as additional input.

We note that is straightforward to extend $\texttt{Sample}$, without requiring a new construction or security analysis. To this end, for any program d, we let d_x denote the program d extended with an additional comment containing string x. This allows to alter the description of d without changing its functionality in any way. We require that this extension is performed in some standard and deterministic way, for instance by always adding the comment at the very beginning of the code. We will write d_x to denote the program d with comment x. To obtain an universal sampler scheme with additional input, we simply set

$$\texttt{Sample}(U, d, x) := \texttt{Sample}(U, d_x)$$

Hashing to arbitrary sets. Universal sampler schemes can also be used to construct hash functions that map into arbitrary efficiently samplable sets, while allowing a limited form of "programmability" that allows to map certain inputs to certain outputs in an indistinguishable way. For instance, imagine a (programmable) hash function that maps strings into

- group elements $g^x \in \mathbb{G}$ for a group $\mathbb{G}$ with known generator g (but unknown exponent x), or
- Diffie-Hellman pairs of group elements (g^x, h^x) for publicly known g, h (but unknown x), or
- vectors of the form $\mathbf{v} = \mathbf{A} \cdot \mathbf{x} + \mathbf{e}$ for a known matrix $\mathbf{A}$, but unknown $\mathbf{x}$ and "short" error vector $\mathbf{e}$ (i.e., vectors $\mathbf{v}$ close to the lattice spanned by the columns of $\mathbf{A}$).

We note that the respective sets that the universal sampler scheme maps into should be easy to sample: for instance, group elements g^x can be sampled by choosing x randomly and then computing g^x from g and x. However, the function H with $\mathsf{H}(x) := g^x$ would not be very interesting for most cryptographic purposes, since it would directly reveal x. (In most applications, knowing trapdoor information like x to hashed images g^x would allow to break the constructed scheme.)

Universal sampler schemes with additional input allow to construct such hash functions easily. Let $U \leftarrow \texttt{Setup}(1^\lambda)$ and let S be an efficiently samplable set with sampling algorithm d. Then the function H defined by (U, d) as

$$\mathsf{H}(x) := \texttt{Sample}(U, d_x),$$

forms a hash function that maps into S.

If $(\texttt{Setup}, \texttt{Sample})$ is a selective one-time universal sampler scheme, then this hash function can be "programmed" as follows. Let $s^* \in S$ be sampled according to distribution d, and let x^* be an element of the domain of H. Computing sampler parameters as $U' \leftarrow \texttt{SimUGen}(1^\lambda, d_{x^*}, s^*)$ defines a programmed map H' with $\mathsf{H}(x) := \texttt{Sample}(U', d_x)$ such that $\mathsf{H}'(x^*) = s^*$, and U' is computationally indistinguishable from $U \leftarrow \texttt{Setup}(1^\lambda)$ by the selective one-time security of the universal sampler scheme.

6.3 Generic Construction

Let $\mathsf{PKE} = (\mathsf{PKGen}, \mathsf{PKEnc}, \mathsf{PKDec})$ and $\mathsf{PKE}_{\mathsf{wCCA}} = (\mathsf{PKGen}_{\mathsf{wCCA}}, \mathsf{PKEnc}_{\mathsf{wCCA}}, \mathsf{PKDec}_{\mathsf{wCCA}})$ be public-key encryption schemes (Definition 6), and let $(\texttt{Setup}, \texttt{Sample})$ be a (selective, one-time secure) universal sampler scheme with algorithm $\texttt{SimUGen}$ for the "programmed" generation of sampler parameters U.

Remark. In the sequel it will sometimes be helpful to make the random coins of an algorithm explicit. Let $\rho = (\rho_0, \rho_1) \in \{0,1\}^{\ell_d^{\mathsf{in}}}$ with $\rho_0, \rho_1 \in \{0,1\}^{\ell_d^{\mathsf{in}}/2}$, then we will write $\mathsf{PKGen}(1^\lambda; \rho_0)$ and $\mathsf{PKEnc}_{\mathsf{wCCA}}(pk_{\mathsf{wCCA}}, m; \rho_1)$ to denote the deterministic execution of PKGen and $\mathsf{PKEnc}_{\mathsf{wCCA}}$, respectively, on input randomness ρ_0 and ρ_1.

Here we assume for simplicity that the randomness space of PKGen and $\mathsf{PKEnc}_{\mathsf{wCCA}}$ is $\{0,1\}^{\ell_d^{\mathsf{in}}/2}$. Longer randomness can easily be generated from $\rho \in \{0,1\}^{\ell_d^{\mathsf{in}}}$ by applying a pseudorandom generator.

Consider the following IBE-scheme $\mathsf{IBE} = (\mathsf{IDGen}, \mathsf{IDKey}, \mathsf{IDEnc}, \mathsf{IDDec})$.

IDGen: On input 1^λ, the master key generation algorithm works as follows.

1. It runs $(pk_{\mathsf{wCCA}}, sk_{\mathsf{wCCA}}) \leftarrow \mathsf{PKGen}(1^\lambda)$ to generate a key pair for $\mathsf{PKE}_{\mathsf{wCCA}}$.
2. Then it creates a program d that, on input randomness $\rho = (\rho_0, \rho_1) \leftarrow \{0,1\}^{\ell_d^{\mathsf{in}}}$, computes a key pair $(pk, sk) \leftarrow \mathsf{PKGen}(1^\lambda; \rho_0)$, ciphertext $c^* \leftarrow \mathsf{PKEnc}_{\mathsf{wCCA}}(sk; \rho_1)$, and outputs (pk, c^*).
3. Finally it computes $U \leftarrow \mathtt{Setup}(1^\lambda)$.

The master public key is $mpk := (U, d)$, the master secret key is $msk := sk_{\mathsf{wCCA}}$.

Recall that we write d_x to denote program d deterministically extended with a comment field containing x (cf. Section 6.2). If the comment field takes values in $\{0,1\}^\lambda$, then (U, d) define a map $\mathtt{Sample}(U, d) : \{0,1\}^\lambda \to \mathcal{S}$, where

$$\mathcal{S} = \left\{(pk, \mathsf{PKEnc}_{\mathsf{wCCA}}(sk)) : (pk, sk) \leftarrow \mathsf{PKGen}(1^\lambda)\right\}.$$

In the sequel we will write $\mathsf{H}(x)$ to abbreviate $\mathtt{Sample}(U, d_x)$.

IDKey: The user key extraction algorithm receives as input $id \in \{0,1\}^\lambda$ and msk. It computes $(pk_{id}, c_{id}) \leftarrow \mathsf{H}(id)$ and returns $sk_{id} := \mathsf{PKDec}_{\mathsf{wCCA}}(msk, c_{id})$.

IDEnc: The encryption algorithm receives as input mpk, id, and message m. It computes $(pk_{id}, c_{id}) \leftarrow \mathsf{H}(id)$ and returns $c \leftarrow \mathsf{PKEnc}(pk, m)$.

IDDec: The decryption algorithm receives as input sk_{id} and ciphertext c. It computes and returns $m \leftarrow \mathsf{PKDec}(sk, c)$.

The correctness of this scheme follows immediately from the correctness of PKE, $\mathsf{PKE}_{\mathsf{wCCA}}$, and $(\mathtt{Setup}, \mathtt{Sample})$.

Theorem 4. *Algorithms* $\mathsf{IBE} = (\mathsf{IDGen}, \mathsf{IDKey}, \mathsf{IDEnc}, \mathsf{IDDec})$ *form a secure IBE scheme in the sense of Definition 7.*

The concept of selective, one-time universal sampler schemes makes the proof extremely simple and natural. Essentially, we first use the programmed generation algorithm $\mathtt{SimUGen}$ to define H such that $\mathsf{H}(id^*) = (pk^*, c^*)$ where $c^* = \mathsf{PKEnc}_{\mathsf{wCCA}}(pk_{\mathsf{wCCA}}, sk^*)$. Then we use the wCCA-security of $\mathsf{PKE}_{\mathsf{wCCA}}$ to replace c^* with an encryption of 1^λ (the wCCA decryption oracle is used to answer $\mathcal{O}_{\mathsf{IBE}}$-queries). From this point on we do not need to know sk^* anymore, which makes the reduction to the IND-CPA security (with challenge public key pk^*) of PKE straightforward.

Proof. We proceed in a sequence of hybrid games $H_0, \ldots, H_4$, where Hybrid H_0 corresponds to the IND-ID-CPA security experiment $\mathsf{Exp}_{\mathsf{IBE},\mathcal{A}}^{\mathsf{ind\text{-}id\text{-}cpa\text{-}0}}(\lambda)$ and Hybrid H_4 corresponds to experiment $\mathsf{Exp}_{\mathsf{IBE},\mathcal{A}}^{\mathsf{ind\text{-}id\text{-}cpa\text{-}1}}(\lambda)$.

Hybrid 0. This is the original $\mathsf{Exp}_{\mathsf{IBE},\mathcal{A}}^{\mathsf{ind\text{-}id\text{-}cpa\text{-}0}}(\lambda)$ security experiment with scheme IBE. By definition we have

$$\Pr[H_0 = 1] = \Pr[\mathsf{Exp}_{\mathsf{IBE},\mathcal{A}}^{\mathsf{ind\text{-}id\text{-}cpa\text{-}0}}(\lambda) = 1]$$

Hybrid 1. This hybrid is identical to Hybrid 0, except for the following. The experiment generates sampler parameters as $U \leftarrow \texttt{SimUGen}(1^\lambda, d_{(id^*)}, (pk^*, c^*_{\mathsf{wCCA}}))$, where $(pk^*, c^*_{\mathsf{wCCA}}) \leftarrow d(\rho)$ for uniformly random $\rho \leftarrow \{0,1\}^{\ell_d^{\mathsf{in}}}$ and id^* is the challenge identity chosen by $\mathcal{A}$. Note that this programs the function H such that $\mathsf{H}(id^*) = (pk^*, c^*_{\mathsf{wCCA}})$.

Note that in Hybrid 0, the sampler parameters U are generated exactly as in the $\texttt{Real}$-experiment from Definition 3, while in Hybrid 1, the sampler parameters are generated as in the $\texttt{Ideal}$-experiment. Thus, by the selective security of the one-time secure sampler scheme, we have

$$|\Pr[H_1 = 1] - \Pr[H_0 = 1]| \leq negl(\lambda)$$

for some negligible function $negl$.

Hybrid 2. This hybrid is identical to Hybrid 1, except for the following. In Hybrid 1 we have $(pk^*, c^*_{\mathsf{wCCA}}) \leftarrow d_{(id^*)}(\rho)$, where $\rho = (\rho_0, \rho_1) \leftarrow \{0,1\}^{\ell_d^{\mathsf{in}}}$, and $(pk^*, sk^*) \leftarrow \mathsf{PKGen}(1^\lambda; \rho_0)$, and $c^*_{\mathsf{wCCA}} \leftarrow \mathsf{PKEnc}_{\mathsf{wCCA}}(pk_{\mathsf{wCCA}}, sk^*; \rho_1)$. In Hybrid 2 we replace c^*_{wCCA} with an encryption of 1^λ.

We construct an attacker $\mathcal{B}_{\mathsf{wCCA}}$ against the wIND-CCA security of $\mathsf{PKE}_{\mathsf{wCCA}}$ from any attacker $\mathcal{A}$ that distinguishes Hybrid 2 from Hybrid 1. $\mathcal{B}_{\mathsf{wCCA}}$ runs $\mathcal{A}$ as a subroutine by simulating the IBE security experiment for $\mathcal{A}$ as follows.

1. At the beginning, $\mathcal{B}_{\mathsf{wCCA}}$ receives as input a challenge public key pk^*_{wCCA} from the IND-CCA experiment. Then it starts $\mathcal{A}$ and receives a challenge identity id^* from $\mathcal{A}$.
2. $\mathcal{B}_{\mathsf{wCCA}}$ runs $(pk^*, sk^*) \leftarrow \mathsf{PKGen}(1^\lambda; \rho_0)$ for $\rho_0 \leftarrow \{0,1\}^{\ell_d^{\mathsf{in}}/2}$ to generate a key pair, and outputs $(sk^*, 1^\lambda)$ to the IND-CCA experiment. It receives in response a ciphertext c^*_{wCCA}.
3. $\mathcal{B}_{\mathsf{wCCA}}$ runs $U \leftarrow \texttt{SimUGen}(1^\lambda, d_{(id^*)}, (pk^*, c^*_{\mathsf{wCCA}}))$ and sets $mpk := (U, d)$ and $msk := \perp$.
4. $\mathcal{O}_{\mathsf{IBE}}$ is simulated by $\mathcal{B}$ as follows. When $\mathcal{A}$ makes a query $\mathcal{O}_{\mathsf{IBE}}(id)$, then $\mathcal{B}_{\mathsf{wCCA}}$ computes $(pk_{id}, c_{id}) = \mathsf{H}(id)$ and returns whatever $\mathcal{O}_{\mathsf{wCCA}}(c_{id})$ returns.

If c^*_{wCCA} is an encryption of sk^*, then this is a perfect simulation of Hybrid 1, while if c^*_{wCCA} is an encryption of 1^λ, then this is a perfect simulation of Hybrid 2. It follows from the wIND-CCA security of $\mathsf{PKE}_{\mathsf{wCCA}}$ that

$$|\Pr[H_2 = 1] - \Pr[H_1 = 1]| \leq negl(\lambda)$$

for some negligible function $negl$.

Hybrid 3. This hybrid is identical to Hybrid 2, except for the following. In Hybrid 2 the experiment creates the challenge ciphertext c^* as $c^* \leftarrow \mathsf{PKEnc}(pk^*, m_0)$, where (m_0, m_1) are the messages chosen by $\mathcal{A}$. In this game c^* is created as $c^* \leftarrow \mathsf{PKEnc}(pk^*, m_1)$.

We construct an attacker $\mathcal{B}_{\mathsf{CPA}}$ against the IND-CPA security of PKE from any attacker that distinguishes Hybrid 2 from Hybrid 3. $\mathcal{B}_{\mathsf{CPA}}$ runs $\mathcal{A}$ as a subroutine by simulating the IBE security experiment as follows.

1. At the beginning, $\mathcal{B}_{\mathsf{CPA}}$ receives as input a challenge public key pk^* from the IND-CCA experiment and generates a key pair $(pk_{\mathsf{wCCA}}, sk_{\mathsf{wCCA}}) \leftarrow \mathsf{PKGen}_{\mathsf{wCCA}}(1^\lambda)$. Then it starts $\mathcal{A}$ and receives a challenge identity id^* from $\mathcal{A}$.
2. $\mathcal{B}_{\mathsf{CPA}}$ runs $U \leftarrow \mathtt{SimUGen}(1^\lambda, d_{(id^*)}, (pk^*, c_{\mathsf{wCCA}}))$, where $c_{\mathsf{wCCA}} \leftarrow \mathsf{PKEnc}(pk_{\mathsf{wCCA}}, 1^\lambda)$, and sets $mpk := (U, d)$ and $msk := sk_{\mathsf{wCCA}}$. Note that $\mathcal{B}_{\mathsf{CPA}}$ can simulate $\mathcal{O}_{\mathsf{IBE}}$, since it knows msk.
3. When $\mathcal{A}$ outputs two messages (m_0, m_1), then $\mathcal{B}$ forwards these messages to the IND-CPA security experiment, and receives in response a challenge ciphertext $c^* \leftarrow \mathsf{PKEnc}(pk^*, m_b)$. Ciphertext c^* is forwarded by $\mathcal{B}_{\mathsf{CPA}}$ to $\mathcal{A}$.
4. Finally, when $\mathcal{A}$ terminates then $\mathcal{B}_{\mathsf{CPA}}$ outputs whatever $\mathcal{A}$ returns.

Note that if c^* is an encryption of m_0, then the view of $\mathcal{A}$ is identical to Hybrid 2, while if c^* encrypts m_1, then the view is identical to Hybrid 3. The IND-CPA security of PKE thus implies that

$$|\Pr[H_3 = 1] - \Pr[H_2 = 1]| \leq negl(\lambda)$$

for some negligible function $negl$.

Hybrid 4. This hybrid is identical to Hybrid 3, except for the following. In Hybrid 3, we have $\mathsf{H}(id^*) = (pk^*, c^*_{\mathsf{wCCA}})$, where c^*_{wCCA} is an encryption of 1^λ. In this hybrid we replace c^*_{wCCA} with an encryption of sk^*. With the same arguments as in Hybrid H_2 we have

$$|\Pr[H_4 = 1] - \Pr[H_3 = 1]| \leq negl(\lambda)$$

Hybrid 5. This hybrid is identical to Hybrid 4, except that regular sampler parameters are generated. That is, the experiment generates U as $U \leftarrow \mathtt{Setup}(1^\lambda, d)$. As in Hybrid H_1, we have

$$|\Pr[H_5 = 1] - \Pr[H_4 = 1]| \leq negl(\lambda)$$

Note also that Hybrid 5 is identical to $\mathsf{Exp}^{\mathsf{ind\text{-}id\text{-}cpa\text{-}1}}_{\mathsf{IBE},\mathcal{A}}(\lambda)$. Thus we have

$$\left|\mathsf{Exp}^{\mathsf{ind\text{-}id\text{-}cpa\text{-}0}}_{\mathsf{IBE},\mathcal{A}}(\lambda) - \mathsf{Exp}^{\mathsf{ind\text{-}id\text{-}cpa\text{-}1}}_{\mathsf{IBE},\mathcal{A}}(\lambda)\right| \leq negl(\lambda).$$

□

6.4 Extensions

From selective to adaptive security. The generic IBE construction from PKE and selective one-time universal sampler schemes achieves *selective* security. It is possible to extend this to *adaptive* security, where the IND-ID-CPA attacker selects the challenge identity id^* not at the beginning, but together with the messages (m_0, m_1).

One obvious approach is to use an adaptively secure universal sampler scheme. However, we prefer a simpler, more direct method. One can replace the function $\mathsf{H}(\cdot)$ in the above construction with a function $\mathsf{H}(\mathsf{RO}(\cdot))$, where $\mathsf{RO} : \{0,1\}^\lambda \to \{0,1\}^\lambda$ is a cryptographic hash function. If RO is modeled as a random oracle, then this construction can be proven adaptively secure using standard arguments (as in [BF01], for instance).

Extension to other primitives. Note that the "hashing-identities-to-public-keys" approach used in the above construction extends easily to other cryptographic primitives. For instance, it is straightforward to use the same approach to construct identity-based signature scheme from public-key signatures. We note that for this construction again a selective *one-time* secure universal sampler scheme is sufficient (essentially, because in the classical EUF-CMA security experiment only one challenge public-key is considered). Thus, one-time secure universal sampler schemes can be seen as a generic tool to make certain public-key primitives identity-based.

There are however examples of applications where selective *q-time* universal sampler schemes with $q > 1$ are required for the "public-key-to-identity-based" conversion. For instance, in order to convert a 2-party non-interactive key exchange protocol into a 2-party ID-based NIKE protocol. While the application of universal sampler schemes is analogous to the PKE-to-IBE-setting, we will need a selective 2-*time* universal sampler scheme here. This is essentially because there are two challenge public keys in the 2-NIKE security experiment.

We did not make the notion of selective *q-time* universal sampler schemes explicit in this paper, but note that their definition and construction (for small q) are simple extensions of selective one-time universal sampler schemes.

7 Multiparty Key Exchange from PKE and Universal Samplers

In this section, we use universal sampler schemes together with a public-key encryption (PKE) scheme to construct non-interactive multiparty key exchange (NIKE) schemes. We also show how our construction implies a new class of adaptively secure broadcast encryption schemes.

7.1 Definitions

Definition 8 (Secure Non-interactive Multiparty Key Exchange)**.** *A multiparty non-interactive key exchange protocol (NIKE) consists of PPT algorithms* $\mathsf{NIKE} = (\mathsf{KESetup}, \mathsf{KEPub}, \mathsf{KeyGen})$.

Trusted Setup. $\mathsf{KESetup}$ *takes as input the security parameter* 1^κ *and an upper bound* 1^n *on the number of users. It returns the public parameters* PP.

Publish. KEPub *takes as input the public parameters* PP*, a unique identifier id, and produces a user secret key sk and user public key pk. The user publishes pk, keeping sk secret.*

Key Generation. KeyGen *takes as input the public parameters* PP*, a list of up to* n *user identifiers* $S, |S| \leq n$ *and corresponding public keys* $\{pk_{id}\}_{j \in S}$*, and one user secret key* $sk_{id}, id \in S$*. It outputs a shared secret key* K_S *for the user set* S.

For correctness, we require that for any set S of identifiers, if $(sk_{id}, pk_{id}) \leftarrow \mathsf{KEPub}(\mathsf{PP})$ for each $id \in S$, then for each $id_1 \neq id_2$, the following holds:

$$\mathsf{KeyGen}(\mathsf{PP}, S, \{pk_{id}\}_{id \in S}, sk_{id_1}) = \mathsf{KeyGen}(\mathsf{PP}, S, \{pk_{id}\}_{id \in S}, sk_{id_2})$$

For security, we follow [FHKP13, BZ14] and define active security by the following game between adversary and challenger, parameterized by security parameter λ and bound $n(\lambda)$ on the largest number of users that can compute a shared secret key. The challenger flips a bit $b \leftarrow \{0, 1\}$. The adversary receives $\mathsf{PP} \leftarrow \mathsf{KESetup}(1^\lambda, 1^n)$, and then is allowed to make the following queries:

- Register Honest User: These are queries of the form (Reg, id) for an arbitrary string id that has not appeared before in a register honest user or register corrupt user (see below) query. The challenger runs $(sk_{id}, pk_{id}) \leftarrow \mathsf{KEPub}(\mathsf{PP}, id)$, records the tuple (id, sk_{id}, pk_{id}), and returns pk_{id} to the challenger.

- Register Corrupt User: These are queries of the form $(\mathsf{RegCor}, id, pk_{id}, honest)$ for an arbitrary string id that has not been used in a register honest or corrupt user query, and adversarially chosen public key pk_{id}. The challenger records the tuple $(id, \perp, pk_{id}, corrupt)$. The adversary does not expect a reply

- Extract: These are queries of the form (Ext, id), where id corresponds to a user that was registered honest, and that $id \notin S^*$, where S^* is the challenge set (see below). The challenger looks for the tuple $(id, sk_{id}, pk_{id}, honest)$, changes it to $(id, sk_{id}, pk_{id}, corrupt)$, and returns sk_{id} to the adversary.

- Reveal: These are queries of the form (Rev, S, id) where S is a subset of identities of size at most n, and $id \in S$ is an identity. We require that id is registered as honest. We also require that S is not equal to the challenge set S^* (see below). The challenger runs $k_S \leftarrow \mathsf{KeyGen}(\mathsf{PP}, S, \{pk_{id}\}_{id \in S}, sk_{id})$, and gives k_S to the adversary. In other words, the challenger computes the shared secret key k_S for the set S, as computed by id. The challenger records the set S.

- Challenge: These are queries of the form (Chal, S^*) where S^* is a subset of identities, called the challenge set. We require that S^* consists of identities that are registered as honest, have not since been corrupted, and that S^* is distinct from all sets S queried in reveal queries (though S^* may have arbitrary overlaps with these sets). The challenger chooses an arbitrary $id \in S^*$, and runs $k_0^* \leftarrow \mathsf{KeyGen}(\mathsf{PP}, S^*, \{pk_{id}\}_{id \in S^*}, sk_{id})$ (correctness shows that the choice of id does not matter). It also chooses a random k_1^*. Then, the challenger responds with k_b^*.

  For simplicity we restrict the adversary to making a single challenge query — a simple hybrid argument shows that this implies security for multiple challenge queries.

Finally, the adversary produces a guess b' for b. The advantage of the adversary is defined as $\Pr[b' = b] - 1/2$.

Definition 9. *We say that a tuple of algorithms* $\mathsf{NIKE} = (\mathsf{KESetup}, \mathsf{KEPub}, \mathsf{KeyGen})$ *is an adaptively-secure multiparty non-interactive key exchange protocol if the advantage of any PPT adversary in the above experiment is negligible.*

7.2 Construction

We show how universal sampler schemes give a simple instantiation of multiparty key agreement. We actually give two constructions, one which requires a trusted setup, and a second slightly more complicated scheme that requires no setup at all. Our second scheme is derived from the first similarly to Boneh and Zhandry [BZ14] by designating one of the parties as the "master party" who runs the setup algorithm and publishes both the sampler parameters and her own public value.

Our scheme is very simple. A trusted setup generates sampler parameters. Each user publishes a public key for a public key encryption scheme, keeping the corresponding secret key as their private input. The "induced samples" for a set of users is then an encryption of a random key k to each of the public keys in that set of users. To generate the shared secret key, each user will run the universal sampler to obtain such induced samples, and then decrypt the ciphertext component corresponding to their public key to obtain k. The correctness of the universal sampler scheme ensures that all parties obtain k. Universal sampler security implies that the induced samples look like a tuple of freshly generated ciphertexts, and the CPA security of the encryption scheme then shows that k is hidden to the adversary. We now describe our first scheme NIKE:

- $\mathsf{KESetup}(1^\kappa, 1^n)$: run $U \leftarrow \mathtt{Setup}(1^\lambda, 1^\ell)$ where $\ell = \ell(\lambda, n)$ is chosen so that the circuits used below in the protocol will always have size at most ℓ. Output $\mathsf{PP} = U$
- $\mathsf{KEPub}(U, id)$: run $(sk_{id}, pk_{id}) \leftarrow \mathsf{PKGen}(1^\lambda)$, where $\mathsf{PKE} = (\mathsf{PKGen}, \mathsf{PKEnc}, \mathsf{PKDec})$ is a public key encryption scheme.
- $\mathsf{KeyGen}(U, S, \{pk_{id}\}_{id\in S}, sk_{id})$: Let d_S be a circuit that samples a random k, and outputs

  $$\{\mathsf{PKEnc}(pk_{id}, k)\}_{id\in S}$$

  We will include the set S as a comment in d_S to ensure that different sets S have different circuits d_S, even if the public keys pk_{id} are the same[29]. Then run $\{c_j\}_{j\in S} \leftarrow \mathtt{Sample}(U, d_S)$. Finally, run $k \leftarrow \mathsf{PKDec}(sk_{id}, c_{id})$, and output k as the shared secret key.

For correctness, observe that the circuit d_S computed in KeyGen only depends on the public values, and not the user secret key. Therefore, all users in S will compute the same d_S. Therefore, the sets $\{c_{id}\}_{id\in S}$ computed will be the same for each user, and since each c_{id} encrypts the same key k, all users will arrive at k.

Before proceeding, we observe that we can actually modify the above construction to allow users to individually choose their own potentially different public-key encryption scheme; the description of the encryption algorithm will be part of pk, and d_S will contain the descriptions of all the encryption schemes for the various users. With this modification, users can choose their desired CPA-secure scheme, and can even use their existing keys.

A scheme with no setup. We now give our second variant NIKE', which requires no setup phase. Roughly, analogously to Boneh and Zhandry [BZ14], we have each user carry out the trusted setup

[29] A malicious user may set his pk to be identical to an existing party's pk. Without the commenting, this could result in two different sets S, S' having the same d_S. This would lead to an attack on the scheme

from the scheme NIKE, obtaining separate sampler parameters U_{id}. Then for each group of users, a cannonical user is chosen, and that user's sampler parameters will be used for key generation.

Unfortunately, this simple approach breaks down in the adaptive setting. We describe a high-level attack. The adversary chooses an arbitrary group S^* as the challenge group, and obtains the "parameters" $C_{S^*} = \{c_{S^*,id}\}_{id \in S^*}$ corresponding to this set. Notice that if the adversary can decrypt *any* of the $c_{S^*,id}$ ciphertexts, he can learn the group key and break the scheme. Next the adversary chooses a set S containing at least one user id^* from S^*, as well as one corrupt user id_{cor}. The adversary sets up the malicious parameters for id_{cor} so that, on input d_S, the ciphertext component corresponding to id^* will be exactly c_{S^*,id^*}. Moreover, he chooses S and id^* in a way so that his sampler parameters are chosen for key generation. The security definition of universal sampler schemes guarantees nothing when the sampler parameters are generated adversarially. Moreover, in our universal sampler scheme construction, the sampler parameters are obfuscated programs, so it is very reasonable for the adversary to hard-code a particular output into the program without detection.

At this point, the adversary performs a reveal query on the set S and user id^*. id^* will compute what he thinks is the shared group key for S as the decryption of c_{S^*,id^*}, which he will then output to the adversary. Since this is exactly the actually shared group key for S^*, the adversary successfully breaks the scheme.

We note that while the simple scheme is broken, the break is much weaker than the break in Boneh and Zhandry [BZ14]. In their scheme, the malicious obfuscated program is run on user secrets themselves, and so the attack can actually leak the entire user secret. In our case, the obfuscated program — namely, the sampler parameters — is only run on public circuits. The only way the adversary interacts with the secret key is by learning the decryptions of outputs of the universal sampler. In effect, this gives the adversary a decryption oracle, but nothing more.

We therefore propose two changes to the protocol above. First, to block the attack above, we bind each ciphertext outputted by d_S to the set S. We do this by encrypting the set S along with k. Then during key generation, the user will decrypt, and check that the resulting set S' is actually equal to S. If the sets do not match, the user knows the ciphertext was not generated properly, so he throws away the shared key and aborts. However, if the sets match, then the user accepts.

Now the adversary can no longer insert the components of the challenge into later reveal queries since the sets will necessarily not match. However, if the scheme is malleable, he may be able to maul the ciphertexts in the challenge to change the underlying set. Moreover, the adversary can still potentially embed other ciphertexts of his choice into his parameters, thus maintaining the decryption oracle. Therefore, our second modification is to require that the encryption scheme is CCA secure. We show that this is sufficient for adaptive security.

We now give the scheme. Since there is no setup, the publish step must now take as input the security parameter λ, and bound n on the number of users:

- ○ $\mathsf{KEPub}'(1^\lambda, 1^n, id)$: run $(sk_{id}, pk_{id}) \leftarrow \mathsf{PKGen}(1^\lambda)$, where $\mathsf{PKE} = (\mathsf{PKGen}, \mathsf{PKEnc}, \mathsf{PKDec})$ is a public key encryption scheme. Associate the random oracle $\mathcal{H}_{id}(\cdot) = \mathcal{H}(id, \cdot)$ with the user id, and run $U_{id} \leftarrow \mathtt{Setup}(1^\lambda, 1^\ell)$, using $\mathcal{H}_{id}$ as the random oracle. $\ell = \ell(\lambda, n)$ is chosen so that the circuits used below in the protocol will always have size at most ℓ. Output sk as the user secret, and (pk, U) as the user published value.

- $\mathsf{KeyGen}'(S, \{pk_{id}, U_{id}\}_{id \in S}, sk_{id})$: Let d_S be a circuit that samples a random k and outputs

  $$\{\mathsf{PKEnc}(pk_{id},\ (S, k)\ )\}_{id \in S}\ .$$

  Notice that S itself is already a part of the description of d_S, so we do not need to include S in comments as in NIKE. Let U be the U_{id} that is lexicographically smallest. Then run $\{c_{id}\}_{id \in S} \leftarrow \mathtt{Sample}(U, d_S)$, again using $\mathcal{H}_{id}(\cdot) = \mathcal{H}(id, \cdot)$ as the random oracle. Finally, run $(S', k) \leftarrow \mathsf{PKDec}(sk_{id}, c_{id})$ and check that $S' = S$. If the check passes, output k as the shared secret key. Otherwise abort and output $\perp$.

For security, we have the following theorem:

Theorem 5. *If* $(\mathtt{Setup}, \mathtt{Sample})$ *is an adaptively one-time secure universal sampler scheme and* PKE *is a CPA-secure encryption scheme, then* NIKE *above is adaptively secure. If in addition* PKE *is CCA-secure, then* NIKE' *is also adaptively secure.*

Proof. We prove the security of NIKE', the proof of NIKE being similar. Let $\mathcal{A}$ be an adversary for NIKE. We may assume, taking only a polynomial loss in security, that $\mathcal{A}$ commits to a user id^* whose sampler parameters will be used to generate the challenge key. This clearly commits id^* to be in the challenge set, and in particular, id^* must be an honest party.

Consider the following derived adversary $\mathcal{B}$ for the universal sampler $(\mathtt{Setup}, \mathtt{Sample})$:

- Upon receiving the sampler parameters U, $\mathcal{B}$ chooses a random bit b and simulates $\mathcal{B}$, responding to $\mathcal{A}$'s queries as follows:
  - Random oracle queries to $\mathcal{H}_{id}(x)$. If $id = id^*$, $\mathcal{B}$ forwards the query to its own random oracle. If $id \neq id^*$, $\mathcal{B}$ responds with a fresh random value y, recording (id, x, y) for future use to ensure it always responds to $\mathcal{H}_{id}(x)$ queries consistently.
  - Register honest queries (Reg, id). $\mathcal{B}$ runs $(sk_{id}, pk_{id}) \leftarrow \mathsf{PKGen}(1^\lambda)$. If $id \neq id^*$, $\mathcal{B}$ runs $U_{id} \leftarrow \mathtt{Setup}(1^\lambda, 1^\ell)$ using $\mathcal{H}_{id}(\cdot) = \mathcal{H}(id, \cdot)$ as the random oracle, while if $id = id^*$, $\mathcal{B}$ sets $U_{id} = U$. Note that since $\mathcal{A}$ committed to id^* being honest, $\mathcal{A}$ will at some point register id^* as honest. Then $\mathcal{B}$ gives pk_{id}, U_{id} to $\mathcal{A}$. $\mathcal{B}$ records $(id, sk_{id}, (pk_{id}, U_{id}), honest)$.
  - Register corrupt queries $(\mathsf{RegCor}, id, (pk_{id}, U_{id}))$. $\mathcal{B}$ records $(id, \perp, (pk_{id}, U_{id}), corrupt)$.
  - Extract (Ext, id). $\mathcal{B}$ responds with sk_{id}, and updates the tuple containing id to $(id, sk_{id}, (pk_{id}, U_{id}), corrupt)$.
  - Reveal (Rev, S, id). Let id' be the identity whose sampler parameters will be used to generate the group key. $\mathcal{B}$ computes the circuit d_S that samples a random k, and outputs

    $$\{\mathsf{PKEnc}(pk_{id},\ (S, k)\ )\}_{id \in S}\ .$$

    Then $\mathcal{B}$ runs $C_S = \{c_{S,j}\}_{j \in S} \leftarrow \mathtt{Sample}(U_{id'}, d_S)$ using the oracle $\mathcal{H}_{id'}$ (regardless of if id' is an honest or corrput user).

    Finally, $\mathcal{B}$ decrypts $c_{S,id}$ using sk_{id} obtaining (S', k_S). $\mathcal{B}$ checks that $S' = S$. If the check fails, respond with $\perp$. If the check passes, give the key k_S in the decryption to $\mathcal{A}$.

  - Challenge Chal, S^*. $\mathcal{B}$ computes $d_{S^*}, C_{S^*}, k_{S^*}$ as in a reveal query, choosing an arbitrary $id \in S^*$ for decryption (correctness of the universal sampler implying that the choice does not matter). Since the adversary guarantees that $U = U_{id^*}$ will be used to generate the shared key, $\mathcal{B}$ sends a query $(\mathsf{sample}, d_{S^*})$ and records C_{S^*} to the auxillary tape as well. $\mathcal{B}$ sets $k_0^* = k_{S^*}$. It also chooses a random k_1^*. Finally, responds with k_b^*.
  - Random oracle queries. In addition to the regular NIKE queries, $\mathcal{A}$ is also allowed to make random oracle queries, which $\mathcal{B}$ forwards to its random oracle.

- Finally, $\mathcal{B}$ receives a bit b' from $\mathcal{A}$. If $b = b'$, $\mathcal{B}$ outputs 1, otherwise it outputs 0.

By the adaptive security of the universal sampler, the probability $\mathcal{B}$ outputs 1 in the real world (where $U \leftarrow \mathtt{Setup}^{\mathcal{H}}(1^\lambda)$ where $\mathcal{H}$ is a random oracle) is negligibly close to the probability it outputs 1 in the ideal world, where $U \leftarrow \mathtt{SimUGen}(1^\lambda)$, and $\mathcal{H}$ is simulated using $\mathtt{SimRO}$. Thus, in the ideal world, $\mathcal{A}$ still guesses b correctly with non-negligible probability.

In the ideal world, the view of $\mathcal{A}$ is as follows: a random function F is chosen, and a Samples Oracle $\mathcal{O}$ is implemented as $\mathcal{O}(d) = d(F(d))$. $\mathcal{A}$ can make all of the NIKE queries and random oracle queries, whose responses are the same as in the real world with the following exceptions:

- When registering id^* as honest, $\mathcal{A}$ receives $U = U_{id^*}$ generated as $(U_{id^*}, \tau) \leftarrow \mathtt{SimUGen}(1^\lambda)$ (note that while $\mathcal{A}$ may wait to register id^*, (U_{id^*}, τ) are sampled at the very beginning).
- The random oracles queries sent to the oracle $\mathcal{H}_{id^*}$ are answered using $\mathtt{RandRO}(\tau)$, which makes oracle queries to the Samples Oracle $\mathcal{O}$. Random oracle queries to $\mathcal{H}_{id}$ for $id \neq id^*$ are still answered with fresh random values.

The result is that the k_{S^*} in the challenge query (which uses U_{id^*}) is chosen independently at random, and the c_{S^*,id^*} are fresh encryptions of k_{S^*} under the public key pk_{id^*}.

We will assume wlog that $\mathcal{A}$ actually computes $C_{S^*} = \{c_{S^*,j}\}_{j \in S^*} \leftarrow \mathtt{Sample}(U, d_{S^*})$ upon making the challenge query.

Let q be the number of $\mathcal{O}$ queries that are ultimately made in the idea world. Now consider the following set of hybrids H_ℓ, parameterized by $\ell \in \{0, \dots, n\}$. We first pick random k_0^*, k_1^*, as well as a random $i \in [q]$. Then we simulate the idea world view of $\mathcal{A}$, sampling $\mathcal{O}$ on the fly. For every query d to $\mathcal{O}$ other than the ith query, we answer by choosing a random sample from d. For the ith query, we check that d has the form d_{S^*} for some set S^*. If the check fails, we abort and output a random bit. Otherwise, we encrypt k_1^* to the first ℓ public keys in d_{S^*}, and k_0^* to the remaining (at most) $n - \ell$ public keys.

Finally, when $\mathcal{A}$ finishes and outputs a guess b', we check S^* was actually the set queried in the challenge query (which in particular implies that all the users in S^* where registered as honest). If the check fails, we abort and output a random bit. Otherwise, we output b'.

Now notice that, conditioned on not aborting, when $\ell = 0$, we perfectly simulate the view of $\mathcal{A}$ in the ideal word in the case $b = 0$, and when $\ell = n$, we perfectly simulate the view of $\mathcal{A}$ in the ideal world in the case $b = 1$. Since $\mathcal{A}$ does eventually compute C_{S^*}, $\mathcal{O}$ will at some point be queried on d_{S^*}. Therefore, with probability $1/q$, we will correctly guess which query to $\mathcal{O}$ corresponds to the challenge, and in this case we will not abort. Thus $\mathcal{A}$ distinguishes H_0 from H_ℓ with non-negligible probability.

It remains to argue that $H_{\ell-1}$ and H_ℓ are computationally indisitnugishable. Indeed, suppose $\mathcal{A}$ distinguished the two hybrids with non-negligible probability. We will construct an adversary $\mathcal{C}$ that breaks the CCA security of PKE. Let r be an upper bound on the number of honest users registered. $\mathcal{C}$, upon receiving a public key pk, picks a random $j \in [r]$, and simulates the view of $\mathcal{A}$ as in hybrid $H_{\ell-1}$, with the following exceptions.

- In the jth register honest query for user id', $\mathcal{C}$ sets $pk_{id} = pk$, the given public key.
- On the ith Samples oracle query, $\mathcal{C}$ runs all the checks as before, plus checks that id' is the ℓth user in S^*. If the checks fail, output a random bit and abort. Otherwise, construct C_{S^*} as follows: encrypt (S^*, k_1^*) to the first $\ell - 1$ public keys and (S^*, k_0^*) to the remaining $n - \ell$ public keys. Finally, make a challenge query on $((S^*, k_0^*), (S^*, k_1^*))$, setting the resulting challenge ciphertext c^* to be the ciphertext for the ℓth public key (which is pk). Set $\mathcal{O}$ to output C_{S^*} for this query.
- On any reveal query for a set S and user id, if $id \neq id'$, decrypt $c_{S,id}$ as in the regular scheme, and output the resulting key (or $\perp$ if the public keys in the decryption are incorrect).

  If $id = id'$, and if the ith Samples oracle query has already occured, check that $c_{S,id'}$ was not the challenge ciphertext c^*. If it was, output $\perp$. Otherwise, make a CCA query to decrypt $c_{S,id'}$. Recall that c^* is an encryption of (S^*, k_b^*) for some b, and $S^* \neq S$. Therefore, if $c_{S,id'} = c^*$ and we were to decrypt and check that the plaintext contains S, we would also abort. Moreover, if the ith Samples oracle query has not occured, then with overwhelming probability c^* will not equal $c_{S,id'}$. Therefore, in the case $id = id'$, the reveal query is handled correctly.

Conditioned on no aborts, if c^* is an encryption of k_0^*, then this simulates hybrid $H_{\ell-1}$ and if c^* is an encryption of k_1^*, this simulates hybrid H_ℓ. With probability $1/r$, we will correctly guess the ℓth user in S and will not abort. Therefore, if $H_{\ell-1}$ and H_ℓ were distinguishable, we would successfully break the CCA security of PKE, a contradiction.

Putting it all together, Hybrid H_0 is indisitnguishable from Hybrid H_n. However, Hybrid H_0 is the case where $\mathcal{A}$ receives the correct group key, and H_n is the case where $\mathcal{A}$ receives a random group key. Therefore, $\mathcal{A}$ has negligible probability of distinguishing in the ideal world. This implies a negligible probability of distinguishing in the real world.

Security of NIKE. The security proof for NIKE is basically the same as above with many simplifications because there is only a single sampler parameter U, and it is trusted. However, obtaining security from CPA-secure encryption (as opposed to CCA-secure encryption) requires a minor change. Since there is only a single sampler parameter, in the view of $\mathcal{A}$ in the ideal world, all ciphertext tuples C_S in reveal queries are generated by the Samples oracle, and are thus fresh encryptions of a fresh key. When simulating this view, we will draw the fresh key for ourselves and therefore will know it without having to perform a decryption. Therefore, we can eliminate CCA queries altogether and base the scheme on CPA security. □

7.3 Adaptively Secure Broadcast Encryption

Boneh and Zhandry [BZ14] give a simple conversion from any multiparty non-interactive key exchange protocol into a distributed broadcast encryption scheme (that is, a broadcast scheme where users determine their own secret keys). The ciphertexts consist of a single user public key pk, secret keys are user secret keys sk, and the public parameters are exactly the public parameters of the underlying key exchange protocol. Therefore, applying to our scheme gives a distributed broadcast scheme with short ciphertexts and secret keys. Moreover, Boneh and Zhandry show that, if the underlying key exchange protocol is adaptively secure, then the derived broadcast scheme is adaptively secure as well.

We note that while Boneh and Zhandry gave the conversion and proved adaptive security, they were unable to supply an adaptively secure key exchange protocol to instantiate the broadcast scheme with. Therefore, we obtain the first adaptively secure distributed broadcast scheme with short ciphertexts and secret keys. An interesting direction for future work is to also get short public parameters while preserving the distributed property.

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. 9

[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. 9

[BF01] Dan Boneh and Matthew K. Franklin. Identity-based encryption from the weil pairing. In *Proceedings of the 21st Annual International Cryptology Conference on Advances in Cryptology*, CRYPTO '01, 2001. 55

[BGI13] Elette Boyle, Shafi Goldwasser, and Ioana Ivan. Functional signatures and pseudorandom functions. *IACR Cryptology ePrint Archive*, 2013:401, 2013. 4, 8

[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

[BR93] Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *CCS '93, Proceedings of the 1st ACM Conference on Computer and Communications Security, Fairfax, Virginia, USA, November 3-5, 1993.*, pages 62–73, 1993. 2

[BR96] Mihir Bellare and Phillip Rogaway. The exact security of digital signatures - how to sign with RSA and rabin. In *Advances in Cryptology - EUROCRYPT '96, International Conference on the Theory and Application of Cryptographic Techniques, Saragossa, Spain, May 12-16, 1996, Proceeding*, pages 399–416, 1996. 2

[BW13] Dan Boneh and Brent Waters. Constrained pseudorandom functions and their applications. *IACR Cryptology ePrint Archive*, 2013:352, 2013. 4, 8

[BZ14] Dan Boneh and Mark Zhandry. Multiparty key exchange, efficient traitor tracing, and more from indistinguishability obfuscation. In JuanA. Garay and Rosario Gennaro, editors, *Advances in Cryptology – CRYPTO 2014*, volume 8616 of *Lecture Notes in Computer Science*, pages 480–499. Springer Berlin Heidelberg, 2014. 4, 56, 57, 58, 62

[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

[FHKP13] EduardaS.V. Freire, Dennis Hofheinz, Eike Kiltz, and Kenneth G. Paterson. Non-interactive key exchange. In Kaoru Kurosawa and Goichiro Hanaoka, editors, *Public-Key Cryptography – PKC 2013*, volume 7778 of *Lecture Notes in Computer Science*, pages 254–271. Springer Berlin Heidelberg, 2013. 56

[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. 7, 8

[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. 8

[HKKW14] Dennis Hofheinz, Akshay Kamath, Venkata Koppula, and Brent Waters. Adaptively secure constrained pseudorandom functions. *IACR Cryptology ePrint Archive*, 2014:720, 2014. 4

[HKW15] Susan Hohenberger, Venkata Koppula, and Brent Waters. Universal signature aggregators. In *EUROCRYPT*, 2015. 4

[KPTZ13] Aggelos Kiayias, Stavros Papadopoulos, Nikos Triandopoulos, and Thomas Zacharias. Delegatable pseudorandom functions and applications. *IACR Cryptology ePrint Archive*, 2013:379, 2013. 4, 8

[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.

[Nie02] Jesper Buus Nielsen. Separating random oracle proofs from complexity theoretic proofs: The non-committing encryption case. In *CRYPTO*, pages 111–126, 2002. 5, 10

[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

[SW14] Amit Sahai and Brent Waters. How to use indistinguishability obfuscation: deniable encryption, and more. In *STOC*, pages 475–484, 2014. 5