How to Generate and Use Universal Samplers

Dennis Hofheinz[1], Tibor Jager[2], Dakshita Khurana[3](✉), Amit Sahai[3],
Brent Waters[4], and Mark Zhandry[5]

[1] Karlsruher Institut Für Technologie, Karlsruhe, Germany
`Dennis.Hofheinz@kit.edu`
[2] Ruhr-Universität Bochum, Bochum, Germany
`Tibor.Jager@rub.de`
[3] Center for Encrypted Functionalities, UCLA, Los Angeles, USA
`{dakshita,sahai}@cs.ucla.edu`
[4] Center for Encrypted Functionalities, University of Texas at Austin, Austin, USA
`bwaters@cs.utexas.edu`
[5] Princeton University, Princeton, USA
`mzhandry@princeton.edu`

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

We describe several applications that provide a natural motivation for this notion; these include generating the trusted parameters for many schemes from just a single trusted setup. We further demonstrate the versatility of universal samplers by showing how they give rise to simple constructions of identity-based encryption and multiparty key exchange. In particular, we construct adaptively secure non-interactive multiparty key exchange in the random oracle model based on indistinguishability

D. Hofheinz—Supported by DFG grants GZ HO 4534/2-1 and GZ HO 4534/4-1.
D. Khurana and A. Sahai—Research supported in part from a DARPA/ARL SAFEWARE award, NSF Frontier Award 1413955, NSF grants 1228984, 1136174, 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 ARL under Contract W911NF-15-C-0205. 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.
B. Waters—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, DARPA SAFEWARE Award, Google Faculty Research award, the Alfred P. Sloan Fellowship, Microsoft Faculty Fellowship, and Packard Foundation Fellowship.
M. Zhandry—Work done while a graduate student at Stanford University 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.

J.H. Cheon and T. Takagi (Eds.): ASIACRYPT 2016, Part II, LNCS 10032, pp. 715–744, 2016.
DOI: 10.1007/978-3-662-53890-6_24

obfuscation; obtaining the first known construction of adaptively secure NIKE without complexity leveraging.

We give a solution that shows how to transform any random oracle into a universal sampler scheme, based on indistinguishability obfuscation. At the heart of our construction and proof is a new technique we call "delayed backdoor programming" that we believe will have other applications.

1 Introduction

Many cryptographic systems rely on the trusted generation of common parameters to be used by participants. There may be several 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 [10] or batch verifiable signatures [15] require all users to choose their public keys using the same algebraic group structure. Finally, common parameters are sometimes used 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 of security. In other cases, security breaks are more subtle. The issue of trust is exemplified by the recent concern over NSA interference in choosing public parameters for cryptographic schemes [2,27,30].

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 instance, 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 communication costs, the organizational overhead of executing

[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 NIZKs in Sahai-Waters [32] and the recent two-round MPC protocol of Garg et al. [19] 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.

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. We 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?

In this work, we address this question by introducing a new primitive that we call Universal Samplers, and we show how to achieve a strong adaptive notion of security for universal samplers in the random oracle model, using indistinguishability obfuscation (iO). To obtain our result, we introduce a new construction and proof technique called *delayed backdoor programming.* There are only a small handful of known high-level techniques for leveraging iO, and we believe delayed backdoor programming will have other applications in the future.

Universal Sampler Schemes. We want a cryptographic primitive that allows us to (freshly) sample from an *arbitrary* distribution, without revealing the underlying randomness used to generate that sample. 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-size circuit description, d, and outputs a sample $p = d(x)$ for a randomly chosen 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 [5]. In the random oracle model, a function H is 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, our main result is a *transformation*: we show how to transform any ordinary random oracle into a universal sampler scheme, by making use of indistinguishability obfuscation

[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 [6], 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.

applied to a function that *interacts* with the outputs of a random oracle – our construction does not obfuscate a random oracle itself, which would be problematic to model in a theoretically reasonable way.

On Random Oracles, Universal Samplers and Instantiation. We view universal samplers as the next generation of the random oracle model. Universal samplers are an intuitive yet powerful tool: they capture the idea of a trusted box in the sky that can sample from arbitrary user-specified distributions, and provide consistent samples to every user - including providing multiple samples from the same user-specified distribution. Such a trusted box is at least as strong as a random oracle, which is a box in the sky that samples from just the uniform distribution. Our notion formalizes a conversion process in the other direction, from a random oracle to a universal sampler that can sample from arbitrary (possibly adaptively chosen) distributions.

An important issue is how to view universal samplers, given that our strongest security model requires a random oracle for realization. We again turn to the history of the random oracle model for perspective. The random oracle model itself is a well-defined and rigorous model of computation. While it is obvious that a hash function cannot actually be a random oracle, a cryptographic primitive that utilizes a hash function in place of the random oracle, and is analyzed in the random oracle model, might actually lead to a secure realization of that primitive. While it is possible to construct counterexamples [16], there are no natural cryptographic schemes designed in the random oracle model that are known to break when utilizing a cryptographic hash function in place of a random oracle.

In fact, the random oracle model has historically served two roles: (1) for efficiency, and (2) for initial feasibility results. We focus exclusively on the latter role. Our paper shows that for achieving feasibility results, by assuming iO, one can bootstrap the random oracle model to the Universal Sampler Model. And just as random oracle constructions led to standard model constructions in the past, most notably for Identity-Based Encryption, we expect the Universal Sampler Model to be a gateway to new standard model constructions. Indeed, the random-oracle IBE scheme of Boneh-Franklin [9] led to the standard model IBE schemes of Canetti-Halevi-Katz [17], Boneh-Boyen [8], and beyond. It is uncontroverted that these latter constructions owe a lot to Boneh-Franklin [9], even though completely new ideas were needed to remove the random oracle.

Similarly, we anticipate that future standard model constructions will share intuition from universal sampler constructions, but new ideas will be needed as well. Indeed, since the initial publication of our work, this has already happened: for the notion of universal signature aggregators [25], an initial solution was obtained using our universal samplers, and then a standard model notion was obtained using additional ideas, but building upon the intuition conceived in the Universal Sampler Model. We anticipate many other similar applications to arise from our work. Indeed, identifying specific distributions that do not require the full power of iO may allow one to avoid both the random oracle model and iO. But our work would provide the substrate for this exploration.

We stress that unlike the random oracle model, where heuristic constructions of cryptographic hash functions preceded the random oracle model, before our work there were not even heuristic constructions of universal samplers. Our work goes further, and gives a candidate whose security can be rigorously analyzed in the random oracle model. Moreover, just as iO and UCEs (universal computational extractors) [4] have posited achievable standard-model notions related to ideal models like VBB and random oracles, we anticipate that future work will do so for universal samplers. Our work lays the foundation for this; indeed our bounded-secure notion of universal samplers is already a realizable notion in the standard model, that can be a starting point for such work.

Our work and subsequent work give examples of the power of the universal sampler model. For example, prior to our work obtaining even weak notions of adaptivity for NIKE required extremely cumbersome schemes and proofs, whereas universal samplers give an extremely simple and intuitive solution, detailed in the full version of our paper. Thus, we argue that having universal samplers in the toolkit facilitates the development of new primitives by allowing for very intuitive constructions (as evidenced in subsequent works [7,21,24,25]).

Last, but not least, in settings where only a bounded number of secure samples are required (including a subsequent work [28]), universal samplers are a useful tool for obtaining standard model solutions.

1.1 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 Sect. 3. In our system there is a universal sampler parameter generation algorithm, `Setup`, which is invoked with security parameter 1^λ and randomness r. The output of this algorithm are 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.

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 $p_i = \texttt{Sample}(U, d_i)$ for $i = 1, 2, \ldots, q$.

In the ideal world, the attacker will first get the universal sampler parameters U, as before. Now, when the adversary queries on d_i, a unique true random string r_i is chosen for each distinct d_i, and the adversary gets back $p_i = d_i(r_i)$, as if obtaining a freshly random sample from d_i.

A scheme is secure if no poly-time attacker can distinguish between the real and ideal game with non-negligible advantage after observing their transcripts. Since p_i is a deterministic function of d_i, this strong definition is only achievable in the random oracle model. This strongest definition is formalized in Sect. 3.2.

To make progress toward our eventual solution we begin with a relaxed security notion, which is in fact realizable in the standard model, without random oracles. We relax the definition in two ways: (1) we consider a setting 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 for our long term goals, developing a solution will serve as step towards our final solution and provide insights.

In the selective setting, in the ideal world, it will be possible to program U to contain the output corresponding to the attacker's query. Given this insight, it is straightforward to obtain the selective and bounded notion of security by using indistinguishability obfuscation and applying punctured programming [32] techniques. In our construction we consider setup programs to all come from a polynominal circuit family of size $\ell(\lambda)$, where each setup circuit d takes in input $m(\lambda)$ 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 `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 coins $x \overset{\$}{\leftarrow} \{0,1\}^m$. The universal sampler parameters are created as an obfuscation of a program that on input d computes and outputs $d(F(K,d))$. 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 $z = d(F(K,d))$. 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 sampler 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 in the adaptive unbounded setting will be significantly more difficult. Recall that we consider a setting where a random oracle may be augmented by a program to obtain a universal sampler scheme for arbitrary distributions[3]. Indeed, for uniformly distributed samples, our universal sampler scheme will imply a programmable random oracle.

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 programmable random oracle. This solution is problematic: what does it mean to obfuscate an oracle-aided circuit? It is not clear how to model this notion without yielding an impossibility result *even within the random oracle model*, since the most natural formulation of indistinguishability obfuscation for random-oracle-aided circuits would yield VBB obfuscation, a notion that is known to be impossible to achieve [3]. In particular, Goldwasser and Rothblum [23] also showed a family of random-oracle-aided circuits that are provably impossible to indistinguishably obfuscate. However, these impossibilities only show up when we try to obfuscate circuits that make random oracle calls. Therefore we need to obtain 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 this 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 consist of an obfuscated program U (produced from `Setup`) as well as a hash function H modeled as a random oracle. Users will use these overall parameters to determine the induced samples. We will use the notion of "hidden triggers" [32] that loosely corresponds to information hidden in an otherwise pseudorandom string, that can only be recovered using a secret key.

Let's begin by seeing how `Setup` creates a program, P, that will be obfuscated to create U. The program takes an input w (looking ahead, this input w will be obtained by a user as a result of invoking the random oracle on his input distribution d). The program consists of two main stages. In the first stage, the program checks to see if w encodes a "hidden trigger" using secret key information. If it does, this step will output the "hidden trigger" $x \in \{0,1\}^n$, and the program P will simply output x. However, 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 will be difficult to distinguish an input w containing a hidden trigger value from a uniformly sampled string.

If decoding is unsuccessful, P will move into its second stage. It will compute randomness $r = F(K, w)$ for a puncturable PRF F. Now instead of directly computing the induced samples using r, we add a level of indirection. The program will run the Setup algorithm for a 1-bounded universal parameter generation scheme using randomness r—in particular the program P could call the

[3] Note that once the universal sampler parameters of a fixed polynomial size are given out, it is not possible for a standard model proof to make an unbounded number of parameters consistent with the already-fixed universal sampler parameters.

1-bounded selective scheme we just illustrated above[4]. The program P then outputs the 1-bounded universal sampler parameters U_w.

In order to generate an induced sample by executing $\texttt{Sample}(U, d)$ on an input distribution d, the algorithm first calls the random oracle to obtain $H(d) = w$. Next, it runs the program U to obtain output program $U_w = U(w)$. Finally, it obtains the induced parameters by computing $p_d = U_w(d)$. The extra level of indirection is critical to our proof of security.

We now give an overview of the proof of security. 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 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 the i^{th} transition for input distribution d_i. The first hybrid step uses punctured programming techniques to replace the normal computation of the 1-time universal sampler parameters U_{w_i} inside the program, with a hardwired and randomly sampled value $U_{w_i} = U'$. These techniques require making changes to the universal sampler parameter U. Since U is published *before* the adversary queries the random oracle on distribution d_i, note that we cannot "program" U to specialize to d_i.

The next step[5] involves a "hand-off" operation where we move the source of the one time parameters U' to the trigger that will be hidden inside the random oracle output w_i, instead of using the hardwired value U' inside the program. This step is critical to allowing an unbounded number of samples to be programmed into the universal sampler scheme via the random oracle. Essentially, 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 iO security to return to the situation where U' is no longer hardwired into the program—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 and program the one-time parameters U' ahead of time into U which had to be published well before d_i was known. However, now $U_{w_i} = U'$ comes programmed in to the random oracle output w_i obtained as a result of the

[4] In our construction of Sect. 5 we directly use our 1-bounded scheme inside the construction. However, we believe our construction can be adapted to work for any one bounded scheme.

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

call to $H(d_i)$. At this point, the program U' needs to be constructed only *after* the oracle call $H(d_i)$ has been made and thus d_i is known to the challenger. We can now use our techniques from the selective setting to force $U'(d_i)$ to output the ideally generated parameters $d_i(r)$ for distribution d_i.

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.

1.2 Applications of Universal Samplers

Universal setup. Our notion of arbitrary sampling 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 simply publish the universal sampler $U \leftarrow \texttt{Setup}(1^\lambda)$, for security parameter λ. Then if subsequently a new scheme is developed that has a trusted setup algorithm d, everyone can agree to use $p = \texttt{Sample}(U, d)$ 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. As a warmup, we show how to transport cryptographic schemes from the public-key to the identity-based setting using universal samplers. 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.

A first attempt to do so would be to publish a description of U as the master public key of IBE, and then to define a public key pk_{id} for identity id as $pk_{id} = \texttt{Sample}(U, d_{id})$, where d_{id} is the algorithm that first generates a PKE key pair $(pk, sk) \leftarrow \mathsf{PKGen}(1^\lambda)$ 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 $(pk_{id}, c'_{id}) = \texttt{Sample}(U, d_{id})$ 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 secret key" to extract sk from c'_{id} – and thus extract individual user secret keys.

We show that this construction yields a selectively-secure IBE scheme once the used universal sampler scheme is selectively secure and the underlying PKE schemes are secure. Intuitively, during the analysis, we substitute the user public key pk_{id^*} for the challenge identity id^* with a freshly generated PKE public key, and we 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. 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. 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 key pair and publishes its public key.) Security demands that k look random to any party not in the group.

We construct a NIKE scheme from a universal sampler scheme and a PKE scheme $\mathsf{PKE} = (\mathsf{PKGen}, \mathsf{PKEnc}, \mathsf{PKDec})$ as follows: the public parameters are the universal samplers U. Each party chooses a keypair $(pk, sk) \leftarrow \mathsf{PKGen}(1^\lambda)$. 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 $(c_1, \ldots, c_n) = \mathtt{Sample}(U, d_S)$, 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 independently random $\mathtt{Sample}$ 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 setting, assuming 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 [12] and designate one user as the "master party" who generates and publishes the universal sampler parameters along with her public key. Unfortunately, as in [12], 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 [12], 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. Subsequent to our work, Rao [31] gave an adaptive multi-party non-interactive key exchange protocol under adaptive assumptions on multilinear maps. One trade-off is that our scheme is only proved secure in the random oracle model, whereas [12,31] are proved secure in the standard model. Nevertheless, we note that adaptively secure NIKE with polynomial loss to underlying assumptions is not known to be achievable outside of the random oracle model unless one makes very strong adaptive (non-falsifiable) assumptions [31].

Finally, using an existing transformation of Boneh and Zhandry [12], we obtain a new adaptive distributed broadcast encryption from our NIKE scheme.

1.3 Subsequent Work Leveraging Universal Sampler Schemes

After the initial posting of our paper, a few other papers have applied universal sampler schemes. Hohenberger, Koppula and Waters [25] used universal samplers to achieve adaptive security without complexity leveraging for a new notion they called universal signature aggregators. Hofheinz, Kamath, Koppula and Waters [24] showed how to build adaptively secure constrained PRFs [11,14,26], for any circuits, using universal parameters as a key ingredient. All previous constructions were only selectively secure, or required complexity leveraging.

Our adaptively secure universal sampler scheme in the random oracle model, also turns out to be a key building block in the construction of proof of human-work puzzles of Blocki and Zhou [7]. Again, the abstraction of universal samplers proved useful for constructing NIKE schemes based on polynomially-hard functional encryption [21].

Another paper that appeared subsequent to ours [18], introduced the notion of explainability compilers and used them to obtain adaptively secure, universally composable MPC in constant rounds based on indistinguishability obfuscation and one-way functions. We note that explainability compilers are related to our notion of selectively secure universal samplers.

1.4 Organization of the Paper

We give an overview of indistinguishability obfuscation and puncturable PRFs, the main technical tools required for our constructions, in Sect. 2. In Sect. 3, we define our notion of universal sampler schemes. We give a realization and proof of security for a 1-bounded selectively secure scheme in Sect. 4. In Sect. 5, we give the construction and security overview for our main notion of an unbounded adaptively secure scheme. The full proof of security of the adaptive unbounded universal sampler scheme is in the full version. Applications of Universal Samplers to IBE and NIKE are also detailed in the full version.

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 can be constructed from one-way functions.

Indistinguishability Obfuscation. The definition below is adapted from [20]:

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

- *For all security parameters $\lambda \in \mathbb{N}$, for all circuits C, for all inputs x, we have that*
$$\Pr[C'(x) = C(x) : C' \leftarrow iO(\lambda, C)] = 1$$
- *For any (not necessarily uniform) PPT adversaries Samp, D, there exists a negligible function α such that the following holds: if $\Pr[|C_0| = |C_1|$ and $\forall x, C_0(x) = C_1(x) : (C_0, C_1, \sigma) \leftarrow Samp(1^\lambda)] > 1 - \alpha(\lambda)$, then we have:*
$$\Big|\Pr\big[D(\sigma, iO(\lambda, C_0)) = 1 : (C_0, C_1, \sigma) \leftarrow Samp(1^\lambda)\big]$$
$$-\Pr\big[D(\sigma, iO(\lambda, C_1)) = 1 : (C_0, C_1, \sigma) \leftarrow Samp(1^\lambda)\big]\Big| \leq \alpha(\lambda)$$

We will sometimes omit λ from the notation whenever convenient and clear from context.

Such indistinguishability obfuscators for circuits were constructed under novel algebraic hardness assumptions in [20].

PRF variants. We first consider some simple types of constrained PRFs [11,14,26], 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*
$$\Big|\Pr\big[A_2(\sigma, K_S, S, \mathrm{Eval}_F(K, S)) = 1\big] - \Pr\big[A_2(\sigma, K_S, S, U_{m(\lambda)\cdot|S|}) = 1\big]\Big| = negl(\lambda)$$
where $\mathrm{Eval}_F(K, S)$ *denotes the concatenation of* $\mathrm{Eval}_F(K, x_1)), \ldots, \mathrm{Eval}_F(K, x_k))$ *where $S = \{x_1, \ldots, x_k\}$ is the enumeration of the elements of S in lexicographic order, $negl(\cdot)$ is a negligible function, and U_ℓ denotes the uniform distribution over ℓ bits.*

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

The GGM tree-based construction of PRFs [22] from one-way functions are easily seen to yield puncturable PRFs, as recently observed by [11,14,26]. Thus:

Theorem 1. [11,14,22,26] *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:

- 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[6] 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:

[6] 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 [1,13].

(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.

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 $\texttt{Sample}(U, d^*)$. 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 $\texttt{Sample}(U, d^*) = d^*(F(d^*))$ 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 [29] 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 formally define 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* $\texttt{Setup}(1^\lambda) \rightarrow U, \texttt{Sample}(U, d) \rightarrow p_d$*, 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*

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

where the experiments $\texttt{Real}$ *and* $\texttt{Ideal}$ *are defined below (σ denotes auxiliary information).*

The experiment $\mathtt{Real}(1^\lambda)$ *is as follows:*	*The experiment* $\mathtt{Ideal}(1^\lambda)$ *is as follows:*
– $(d^*, \sigma) \leftarrow \mathcal{A}_1(1^\lambda)$ – *Output* $\mathcal{A}_2(\mathtt{Setup}(1^\lambda), \sigma)$	– $(d^*, \sigma) \leftarrow \mathcal{A}_1(1^\lambda)$ – *Choose* r *uniformly from* $\{0,1\}^m$ – *Let* $p_d = d^*(r)$ – *Output* $\mathcal{A}_2(\mathtt{SimUGen}(1^\lambda, d^*, p_d), \sigma)$

3.2 Adaptively Secure Universal Samplers

We now define universal sampler schemes for the adaptive setting in the random oracle model, handling an unbounded number of induced samples simultaneously. We *do not assume* obfuscation of circuits that call the random oracle. Thus, we allow the random oracle to be used only outside of obfuscated programs.

We consider an adversary that uses a universal sampler to obtain samples on (adaptively chosen) distributions of his choice. We want to guarantee that for any distribution specified by the adversary, the output samples he obtains are indistinguishable from externally generated parameters from the same distribution. In other words, there must exist a simulator that can force the adversary to obtain the externally generated parameters as output of the universal sampler.

Converting this intuition into an actual formal definition turns out to be somewhat complicated. The reason is that in the real world, the adversary must be able to generate samples on his own, using the universal sampler provided to him. However, the simulator which is required to force the external parameters cannot learn the adversary's queries to the sampler program. Such a simulator must observe all of the adversary's queries to the random oracle, and use them to program the output of the samplers, without knowing any of the adversary's actual queries to the sampler program.

Definition 4 (Adaptively-Secure Universal Sampler Scheme). *Let $\ell(\lambda)$, $m(\lambda)$, $k(\lambda)$ be efficiently computable polynomials. A pair of efficient oracle algorithms* $(\mathtt{Setup}, \mathtt{Sample})$ *where* $\mathtt{Setup}^{\mathcal{H}}(1^\lambda) \rightarrow U, \mathtt{Sample}^{\mathcal{H}}(U, d) \rightarrow p_d$ *is an adaptively-secure universal sampler scheme if there exist efficient interactive Turing Machines* $\mathtt{SimUGen}, \mathtt{SimRO}$ *such that for every efficient admissible adversary $\mathcal{A}$, there exists a negligible function $negl(\cdot)$ such that:*

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

where admissible adversaries, the experiments $\mathtt{Real}$ *and* $\mathtt{Ideal}$ *and our (non-standard) notion of the* $\mathtt{Ideal}$ *experiment aborting, are described below.*

- *An* admissible *adversary $\mathcal{A}$ is an efficient interactive Turing Machine that outputs one bit, with the following input/output behavior:*
  - *$\mathcal{A}$ initially takes input security parameter λ and sampler parameters U.*
  - *$\mathcal{A}$ can send a message (RO, x) corresponding to a random oracle query. In response, $\mathcal{A}$ receives 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. $\mathcal{A}$ 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 RO queries, and append (d, p_d) to an auxiliary tape.*

  Remark. Intuitively, (sample, d) corresponds to an honest party seeking a sample generated by program d. Recall that $\mathcal{A}$ is meant to internalize the behavior of honest parties that compute parameters by correctly querying the random oracle and recording the sampler's output[7].

– *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. *The output of the experiment is the final output of the execution of $\mathcal{A}$(which is a bit $b \in \{0, 1\}$).*

– *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 $\mathcal{F}$*[8]*. 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,* SimUGen *can make arbitrary queries to the Samples Oracle $\mathcal{O}$.*
  3. SimRO *corresponds to the output of a programmable random oracle in the ideal world.*
  4. $\mathcal{A}(1^\lambda, U)$ *and* $\texttt{SimRO}(\tau)$ *begin simultaneous execution. Messages for $\mathcal{A}$ or* SimRO *are handled as:*
     - *Whenever $\mathcal{A}$ sends a message of the form* (RO, x)*, this is forwarded to* SimRO*, which produces a response to be sent back to $\mathcal{A}$.*
     - SimRO *can make any number of queries to the Samples Oracle $\mathcal{O}$*[9]*.*
     - *Finally, after $\mathcal{A}$ sends a message of the form* (sample, d)*, the auxiliary tape of $\mathcal{A}$ is examined until $\mathcal{A}$ adds an entry of the form (d, p_d) to it. At this point, if $p_d \neq d(F(d))$, the experiment aborts and we say that an "Honest Sample Violation" has occurred. Note that this*

[7] Note that proving security against such admissible adversaries suffices to capture the intuition behind a universal sampler and in particular suffices for all our applications. This is because honest parties will still use the correctly generated output, and we would like to guarantee that no malicious adversary will be able to distinguish the samples used by **honest parties** from externally generated samples.

[8] $\mathcal{A}$ does not have direct access to $\mathcal{F}$, in fact $\mathcal{A}$ will only have access to SimRO which we define later to model the output of a programmable random oracle.

[9] Looking ahead, in our proof, SimRO will use the output of queries to $\mathcal{O}$ to generate a programmed output of the Random Oracle.

corresponds to a correctness requirement in the ideal world, and is the only way that the experiment `Ideal` *can abort*[10]*. In this case, if the adversary itself "aborts", we consider this to be an output of zero by the adversary, not an abort of the experiment itself.*

5. *The output of the experiment is the final output of the execution of $\mathcal{A}$ (which is a bit $b \in \{0, 1\}$).*

Remark 1. We note that indistinguishability of the real and ideal worlds also implies that: $\Pr[\texttt{Ideal}(1^\lambda) \text{ aborts}] < negl(\lambda)$

4 Selective One-Time Universal Samplers

In this section, we show the following:

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`.

- $\texttt{Setup}(1^\lambda)$ 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[11] Selective-Single-Samples in Figure 1. It outputs U.
- $\texttt{Sample}(U, d)$ 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.

Fig. 1. Program Selective-Single-Samples

[10] Recall that an admissible adversary only honestly computes samples and adds them to its tape – i.e., an admissible adversary always writes $p_d = \texttt{Sample}^{\mathcal{H}}(U, d)$ as the honest output of the sampler program. Thus, an honest sample violation in the ideal world indicates that the simulator did not force the correct samples $d(F(d))$ obtained externally from a trusted party, into the output of the sampler program.

[11] Appropriately padded to the maximum of the size of itself and Program Selective-Single-Samples: 2 in Fig. 2.

4.1 Overview of Security Proof

The proof follows straightforwardly from the puncturing techniques of [32] and we give a brief overview before giving the full proof. In the real world, the adversary commits to his input d^* and then the challenger gives the Selective-Single-Samples program to the adversary. In the first hybrid, we puncture the PRF key K at value d^*, and hardwire the output $f^* = d^*(PRF(K, d^*))$ into the program, arguing security by iO of the functionally equivalent programs. In the next hybrid, $PRF(K, d^*)$ can be replaced with a random value x, setting $f^* = d^*(x)$ and arguing security because of the puncturable PRF. Finally, the value f^* can be replaced with the external sample p_d.

4.2 Hybrids

We prove security by a sequence of hybrids, starting with the original experiment Hybrid_0 in the Real World and replacing the output at d^* with an external sample in the final hybrid (Ideal World). Each hybrid is an experiment that takes as input 1^λ. The output of each hybrid is the adversary's output when it terminates. We denote changes between subsequent hybrids using red underlined font.

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[12] Selective-Single-Samples in Fig. 1.
- The adversary queries the program on input d^* to obtain the sample.

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[13] Selective-Single-Samples: 2 in Fig. 2.
- The adversary queries the program on input d^* to obtain the sample.

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[14] Selective-Single-Samples: 2 in Fig. 2.
- The adversary queries the program on input d^* to obtain the sample.

[12] Padded to the maximum of the size of itself and Selective-Single-Samples: 2.

[13] Padded to the maximum of the size of itself and Selective-Single-Samples.

[14] Padded to the maximum of the size of itself and Selective-Single-Samples.

Selective-Single-Samples: 2

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

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

Fig. 2. 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 sample p_d, punctures K at d^* and outputs an iO of the program[15] Selective-Single-Samples: 2 in Fig. 2. This is then sent to the adversary.
- The adversary queries the program on input d^* to obtain the sample.

4.3 Indistinguishability of the Hybrids

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

Claim. $\mathsf{Hybrid}_0(1^\lambda)$ 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^*, and computes $f^* = d^*(F(K, d^*))$, to obtain circuits C_0 = Selective-Single-Samples according to Fig. 1 and C_1 = Selective-Single-Samples: 2 according to Fig. 2 with inputs d^*, f^*. He gives C_0, C_1 to the iO challenger.

The iO challenger pads these circuits in order to bring them to equal size. It is easy to see that these circuits are functionally equivalent. Next, the iO challenger 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, $\Big|\Pr\big[\mathcal{D}_1(\mathsf{Hybrid}_0) = 1\big] - \Pr\big[\mathcal{D}_1(\mathsf{Hybrid}_1) = 1\big]\Big| \geq 1/\mathsf{p}(\lambda)$.

[15] Padded to the maximum of the size of itself and Selective-Single-Samples.

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

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

Claim. $\mathsf{Hybrid}_1(1^\lambda)$ 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 punctured PRF K along with 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, $\Big|\Pr\big[\mathcal{D}_2(\mathsf{Hybrid}_1) = 1\big] - \Pr\big[\mathcal{D}_2(\mathsf{Hybrid}_2) = 1\big]\Big| \geq 1/p(\lambda)$ for some polynomial $p(\cdot)$.

If a is the output of the punctured PRF K at d^*, then we are in Hybrid_1. If a was chosen uniformly at random, then we are in Hybrid_2. 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. $\mathsf{Hybrid}_2(1^\lambda)$ and $\mathsf{Hybrid}_3(1^\lambda)$ are identical.

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

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

Proof. It follows from inspection of our construction that the program always outputs the external samples in the ideal world, therefore condition (1) in Definition 3 is fulfilled.

5 Adaptively Secure Universal Samplers

Theorem 3 (Adaptively Secure Universal Samplers). *If indistinguishability obfuscation and one way functions exist, then there exists an adaptively secure universal sampler scheme, according to Definition 4, in the Random Oracle Model.*

Our scheme consists of algorithms `Setup` and `Sample`, defined below. We rely on injective PRGs and indistinguishability obfuscation.

- $\texttt{Setup}(1^\lambda, r)$ first samples PRF keys K_1, K_2, K_2' and then sets Sampler Parameters U to be an indistinguishability obfuscation of the program AdaptiveSamples [16], Figure 3. The first three steps in the program look for "hidden triggers" and extract an output if a trigger is found, the final step represents the normal operation of the program (when no triggers are found).

  The program takes as input a value u, where $|u| = n^2$ and v where $|v| = n$, such that $u||v$ is obtained as the output of a random oracle $\mathcal{H}$ on input d. Here, n is the size of an iO of program[17] P_{K_3}(Figure 4). 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 sample for any input 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 Fig. 4. 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 Fig. 4. 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.

  Here m is the size of uniform randomness accepted by $d(\cdot)$, k is the size of samples generated by $d(\cdot)$.
- $\texttt{Sample}(U, d)$ queries the random oracle $\mathcal{H}$ to obtain $(u, v) = \mathcal{H}(d)$. It then runs the program U generated by $\texttt{Setup}(1^\lambda)$ 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.

5.1 Overview of the Security Game and Hybrids

We convert any admissible adversary $\mathcal{A}$ - that is allowed to send *any* message (RO, x) or (params, d) - and construct a modified adversary, such that whenever $\mathcal{A}$ sends message (params, d), our modified adversary sends message (RO, d) and then sends message (params, d). It suffices to prove the security of our scheme with respect to such modified adversaries because this modified adversary is functionally equivalent to the admissible adversary. Because the modified adversary always provides protocol description d to the random oracle, our proof will

[16] This program must be padded appropriately to maximum of the size of itself and other corresponding programs in various hybrids, as described in the next section.

[17] 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 $F_1(K_1^{(n)}, v) = (y_{1,0}, y_{1,1}), \ldots, (y_{n,0}, y_{n,1})$.
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 $P =$
   $iO(P_{K_3}; r)$ of the program [a] $P_{K_3}$ of Figure 4.

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

Fig. 3. 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.
```

Fig. 4. Program P_{K_3}

not directly deal with messages of the form (params, d) and it will suffice to handle only messages (RO, d) sent by the adversary.

We prove via a sequence of hybrids, that algorithms `Setup` and `Sample` satisfy the security requirements of Definition 4 in the Random Oracle Model. Hybrid_0 corresponds 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 $\mathsf{Hybrid}_0, \mathsf{Hybrid}_{1,1}, \mathsf{Hybrid}_{1,2}, \ldots \mathsf{Hybrid}_{1,13}$, $\mathsf{Hybrid}_{2,1}$, $\ldots \mathsf{Hybrid}_{2,13} \ldots \mathsf{Hybrid}_{q(\lambda),13}$, each of which we prove to be indistinguishable from the previous one. We define $\mathsf{Hybrid}_0 \equiv \mathsf{Hybrid}_{0,13}$ for convenience. The final hybrid $\mathsf{Hybrid}_{q(\lambda),13}$ corresponds to the ideal world in the security game described above, and contains (implicitly) descriptions of `SimUGen`, `SimRO` as required in Definition 4. For brevity, we only describe Hybrid_0 and $\mathsf{Hybrid}_{s,13}$ for a generic $s \in q(\lambda)$ in this section. We also give a short overview of how the sequence of hybrids progresses. The complete sequence of hybrids along with complete indistinguishability arguments, beginning with Hybrid_0 and then $\mathsf{Hybrid}_{s,1}$, $\mathsf{Hybrid}_{s,2}, \ldots \mathsf{Hybrid}_{s,13}$ for a generic $s \in [q(\lambda)]$, can be found in the next sections.

In the following experiments, the challenger chooses PRF keys $K_1^{(n)}, K_2$ and K_2' for PRFs $F_1^{(n)}, F_2$ and F_2'. Each hybrid is an experiment that takes input 1^λ. The output of any hybrid experiment denotes the output of the adversary upon termination. Changes between hybrids are denoted using red underlined font.

Hybrid_0:

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

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

- The challenger pads the program Adaptive-Samples in Fig. 5 appropriately [18] and sends an iO of the program to the adversary.
- Set $j = 0$. While the adversary queries the RO, increment j and repeat:
  1. Let the adversary query the random oracle on protocol description d_j^*.
  2. If $j \leq s$, 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 sample p_j^* and sets $g = iO(P'_{K_3,p_j^*,d_j^*}, e')$ (See Fig. 7). 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 $j > s$, challenger sets the RO output, $(u_j^*, v_j^*) \leftarrow \{0,1\}^{n^2+n}$.
- The adversary then outputs a single bit b'.

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.

From $\mathsf{Hybrid}_{s-1,13}$ to $\mathsf{Hybrid}_{s,13}$.

We now outline a series of sub-hybrids from $\mathsf{Hybrid}_{s-1,13}$ to $\mathsf{Hybrid}_{s,13}$ for a generic $s \in [1,q]$, where we program the universal sampler U to output external parameters on the s^{th} query of the adversary. Our proof comprises of two main steps: the first step consists in hardwiring a fresh single-use program into the random oracle output for the s^{th} query – this is done by first hardwiring values into the obfuscated program, then changing the output of the random oracle, and then un-hardwiring these values from the obfuscated program.

Once this is done, the second step comprises of hardwiring the external parameters into this single-use program. The complete hybrids and indistinguishability arguments are in the next subsection.

[18] To the maximum of the size of itself and all corresponding programs in the other 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 $F_1(K_1^{(n)}, v) = (y_{1,0}, y_{1,1}), \ldots, (y_{n,0}, y_{n,1})$.
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 6.
```

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

Fig. 5. Program Adaptive-Samples

P_{K_3}

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

1. Output $d(F_3(K_3, d))$.
```

Fig. 6. 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 $d(F_3(K_3, d))$.
```

Fig. 7. Program P'_{K_3,p_j^*,d_j^*}

First step. $\mathsf{Hybrid}_{s,1}$: Let the s^{th} random oracle query of the adversary be on input d_s^*. We first use punctured programming to hardwire computation corresponding to input d_s^* into the Adaptive-Samples program.

To do this, in $\mathsf{Hybrid}_{s,1}$ the challenger picks v_s^* uniformly at random as the output of the random oracle on input d_s^*. 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 $b \in \{0,1\}, i \in [n]$ he sets $z_{i,b}^* = \mathsf{PRG}(y_{i,b}^*)$. Next, he adds a check at the beginning of the main program such that for $v = v_s^*$, if $u[i] = z_{i,b}^*$, the program sets $x_i = b$. The program Adaptive-Samples of $\mathsf{Hybrid}_{s-1,13}$ is replaced by the program Adaptive-Samples: 2 illustrated in Fig. 8. This is indistinguishable from the previous hybrid by the security of indistinguishability obfuscation, because the programs Adaptive-Samples and Adaptive-Samples: 2 are functionally equivalent.

```
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 $\mathsf{PRG}(u[i]) = z_{i,1}^*$ $x_i = 1$, 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 6.

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

Fig. 8. Program Adaptive-Samples: 2

$\mathsf{Hybrid}_{s,2}$: In $\mathsf{Hybrid}_{s,2}$, the output of PRF F_1 on input v_s^* is replaced with random. That is for all $b \in \{0,1\}, i \in [n]$, he sets $y_{i,b} \xleftarrow{\$} \{0,1\}^n$. This hybrid is indistinguishable from $\mathsf{Hybrid}_{s,1}$ by security of the puncturable PRF.

$\mathsf{Hybrid}_{s,3}$: Next, the string z^* is set uniformly at random. That is, for each $i \in [n], b \in \{0,1\}$, instead of setting $z_{i,b}^* = \mathsf{PRG}(y_{i,b}^*)$, the challenger sets $z_{i,b}^* \xleftarrow{\$} \{0,1\}^{2\lambda}$. This hybrid is indistinguishable from $\mathsf{Hybrid}_{s,2}$ by security of the PRG. Note that this step "deactivates" the extra check we had added in $\mathsf{Hybrid}_{s,1}$, because with overwhelming probability, z^* will lie outside the image of the PRG.

$\mathsf{Hybrid}_{s,4}$: Once this is done, for u_s^* and v_s^* both fixed uniformly at random as random oracle response to query d_s^*, in $\mathsf{Hybrid}_{s,4}$ the challenger sets $e = F_2(K_2, u_s^*|v_s^*)$, $e' = F_2'(K_2', u_s^*|v_s^*)$, $g = iO(P_e, e')$ and adds an initial check in the main program: if input $u = u_s^*$ and $v = v_s^*$, then output g and exit. Simultaneously, the challenger punctures the keys K_2 and K_2' in the main program. The modified program Adaptive-Samples: 3 is depicted in Fig. 9. At this point, we have hardwired Adaptive-Samples: 3 to output g on input values (u_s^*, v_s^*), obtained from the RO on input d_s^*. This is indistinguishable from $\mathsf{Hybrid}_{s,3}$ by the security of indistinguishability obfuscation, because the programs Adaptive-Samples: 3 and Adaptive-Samples: 2 are functionally equivalent.

$\mathsf{Hybrid}_{s,5}$: In this hybrid, the challenger generates e uniformly at random instead of the output of the punctured PRF F_2.

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 6.
```

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

Fig. 9. Program Adaptive-Samples: 3

$\mathsf{Hybrid}_{s,6}$: In this hybrid, the challenger generates e' uniformly at random instead of the output of the punctured PRF F_2'. This will be needed in the next few hybrids when we start programming the single-use parameters.

$\mathsf{Hybrid}_{s,7}$: Since the (bounded size) program Adaptive-Samples: 3 must remain programmable for an unbounded number of samples, we now move the hardwired single-use paramters g from the Adaptive-Samples: 3 program to a hidden trigger encoding in the output of the random oracle, u_s^*. Specifically, this is done by setting for all $i \in [1,n], z_{i,g_i}^* = \mathsf{PRG}(u_s^*[i])$ in $\mathsf{Hybrid}_{s,7}$. This is made possible also by injectivity of the PRG. Once u_s^* has been programmed appropriately to encode the value g, hardwiring g into the program becomes redundant, and it is possible to replace Adaptive-Samples: 3 with the previous program Adaptive-Samples: 2.

At this point, we can *seal* back the punctured keys, un-hardwire g from the program and return to the original program Adaptive-Samples in a sequence of hybrids, $\mathsf{Hybrid}_{s,8}$ to $\mathsf{Hybrid}_{s,10}$ which reverse our sequence of operations from $\mathsf{Hybrid}_{s,1}$ to $\mathsf{Hybrid}_{s,3}$. More specifically, $\mathsf{Hybrid}_{s,8}$ involves generating $z_{i,b}^*$ for all $i \in [n], b \in \{0,1\}$ as outputs of a PRG, and this is indistinguishable by security of the PRG. Then $\mathsf{Hybrid}_{s,9}$ involves generating $(y_{1,0}^*, y_{1,1}^* \ldots y_{n,0}^*, y_{n,1}^*$ as the output of $F_1(K_1^{(n)}, v_s^*)$, and this is indistinguishable by security of the puncturable PRF.

At this point, hardwiring the z^* values becomes redundant, and it is possible to go back to program Adaptive-Samples, in $\mathsf{Hybrid}_{s,10}$ arguing indistinguishability via indistinguishability obfuscation.

Now, $\mathsf{Hybrid}_{s,10}$ becomes identical to $\mathsf{Hybrid}_{s-1,13}$ except for a *trapdoor* that has been programmed into the random oracle output u_s^*, which outputs specific selective single-use parameters.

Second Step. Now, it is straightforward (following the same sequence of hybrids as the selective single-use case) to force the single-use parameters that were programmed into u_s^* to output external parameters p_s^*, in hybrids $\mathsf{Hybrid}_{s,11}$ through $\mathsf{Hybrid}_{s,13}$. Please refer to the full version for a more detailed proof.

No honest sample violations. At this point, in the final hybrid, 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 p_s^*. Thus, the correctness requirement in the ideal world is always met, and there are no honest sample violations according to Definition 4.

Acknowledgements. The authors would like to thank the anonymous Asiacrypt 2016 reviewers for their helpful comments, and in particular for pointing out the contents of Remark 1.

References

1. Ananth, P., Boneh, D., Garg, S., Sahai, A., Zhandry, M.: Differing-inputs obfuscation and applications. IACR Cryptology ePrint Archive 2013, p. 689 (2013)
2. Ball, J., Borger, J., Greenwald, G.: Revealed: how US and UK spy agencies defeat internet privacy and security. The Guardian (2013). http://www.theguardian.com/world/2013/sep/05/nsa-gchq-encryption-codes-security
3. Barak, B., Goldreich, O., Impagliazzo, R., Rudich, S., Sahai, A., Vadhan, S., Yang, K.: On the (Im)possibility of obfuscating programs. In: Kilian, J. (ed.) CRYPTO 2001. LNCS, vol. 2139, pp. 1–18. Springer, Heidelberg (2001). doi:10.1007/3-540-44647-8_1
4. Bellare, M., Hoang, V.T., Keelveedhi, S.: Instantiating random Oracles via UCEs. In: Canetti, R., Garay, J.A. (eds.) CRYPTO 2013. LNCS, vol. 8043, pp. 398–415. Springer, Heidelberg (2013). doi:10.1007/978-3-642-40084-1_23
5. Bellare, M., Rogaway, P.: Random Oracles are practical: a paradigm for designing efficient protocols. In: CCS 1993, Proceedings of the 1st ACM Conference on Computer and Communications Security, Fairfax, Virginia, USA, 3–5 November 1993, pp. 62–73 (1993)
6. Bellare, M., Rogaway, P.: The exact security of digital signatures - how to sign with RSA and Rabin. In: Proceeding Advances in Cryptology - EUROCRYPT 1996, International Conference on the Theory and Application of Cryptographic Techniques, Saragossa, Spain, 12–16 May 1996, pp. 399–416 (1996)
7. Blocki, J., Zhou, H.: Designing proof of human-work puzzles for cryptocurrency and beyond. IACR Cryptology ePrint Archive 2016, p. 145 (2016). http://eprint.iacr.org/2016/145
8. Boneh, D., Boyen, X.: Efficient selective identity-based encryption without random oracles. J. Cryptology **24**(4), 659–693 (2011). http://dx.doi.org/10.1007/s00145-010-9078-6

9. Boneh, D., Franklin, M.: Identity-based encryption from the weil pairing. In: Kilian, J. (ed.) CRYPTO 2001. LNCS, vol. 2139, pp. 213–229. Springer, Heidelberg (2001). doi:10.1007/3-540-44647-8_13
10. Boneh, D., Gentry, C., Lynn, B., Shacham, H.: Aggregate and verifiably encrypted signatures from bilinear maps. In: Biham, E. (ed.) EUROCRYPT 2003. LNCS, vol. 2656, pp. 416–432. Springer, Heidelberg (2003). doi:10.1007/3-540-39200-9_26
11. Boneh, D., Waters, B.: Constrained pseudorandom functions and their applications. IACR Cryptology ePrint Archive 2013, p. 352 (2013)
12. Boneh, D., Zhandry, M.: Multiparty key exchange, efficient traitor tracing, and more from indistinguishability obfuscation. In: Garay, J.A., Gennaro, R. (eds.) CRYPTO 2014. LNCS, vol. 8616, pp. 480–499. Springer, Heidelberg (2014). doi:10.1007/978-3-662-44371-2_27
13. Boyle, E., Chung, K.-M., Pass, R.: On extractability obfuscation. In: Lindell, Y. (ed.) TCC 2014. LNCS, vol. 8349, pp. 52–73. Springer, Heidelberg (2014). doi:10.1007/978-3-642-54242-8_3
14. Boyle, E., Goldwasser, S., Ivan, I.: Functional signatures and pseudorandom functions. IACR Cryptology ePrint Archive 2013, p. 401 (2013)
15. Camenisch, J., Hohenberger, S., Pedersen, M.Ø.: Batch verification of short signatures. J. Cryptology **25**(4), 723–747 (2012)
16. Canetti, R., Goldreich, O., Halevi, S.: The random oracle methodology, revisited. J. ACM **51**(4), 557–594 (2004)
17. Canetti, R., Halevi, S., Katz, J.: A forward-secure public-key encryption scheme. J. Cryptology **20**(3), 265–294 (2007)
18. Dachman-Soled, D., Katz, J., Rao, V.: Adaptively secure, universally composable, multiparty computation in constant rounds. In: Dodis, Y., Nielsen, J.B. (eds.) TCC 2015. LNCS, vol. 9015, pp. 586–613. Springer, Heidelberg (2015). doi:10.1007/978-3-662-46497-7_23
19. Garg, S., Gentry, C., Halevi, S., Raykova, M.: Two-round secure MPC from indistinguishability obfuscation. In: Lindell, Y. (ed.) TCC 2014. LNCS, vol. 8349, pp. 74–94. Springer, Heidelberg (2014). doi:10.1007/978-3-642-54242-8_4
20. Garg, S., Gentry, C., Halevi, S., Raykova, M., Sahai, A., Waters, B.: Candidate indistinguishability obfuscation and functional encryption for all circuits. In: FOCS (2013)
21. Garg, S., Pandey, O., Srinivasan, A., Zhandry, M.: Breaking the sub-exponential barrier in obfustopia. IACR Cryptology ePrint Archive 2016, p. 102 (2016). http://eprint.iacr.org/2016/102
22. Goldreich, O., Goldwasser, S., Micali, S.: How to construct random functions (extended abstract). In: FOCS, pp. 464–479 (1984)
23. Goldwasser, S., Rothblum, G.N.: On best-possible obfuscation. In: Vadhan, S.P. (ed.) TCC 2007. LNCS, vol. 4392, pp. 194–213. Springer, Heidelberg (2007). doi:10.1007/978-3-540-70936-7_11
24. Hofheinz, D., Kamath, A., Koppula, V., Waters, B.: Adaptively secure constrained pseudorandom functions. IACR Cryptology ePrint Archive 2014, p. 720 (2014)
25. Hohenberger, S., Koppula, V., Waters, B.: Universal signature aggregators. In: Oswald, E., Fischlin, M. (eds.) EUROCRYPT 2015. LNCS, vol. 9057, pp. 3–34. Springer, Heidelberg (2015). doi:10.1007/978-3-662-46803-6_1
26. Kiayias, A., Papadopoulos, S., Triandopoulos, N., Zacharias, T.: Delegatable pseudorandom functions and applications. IACR Cryptology ePrint Archive 2013, p. 379 (2013)

27. Larson, J., Perlroth, N., Shane, S.: Revealed: The NSA's secret campaign to crack, undermine internet security. Pro-Publica (2013). http://www.propublica.org/article/the-nsas-secret-campaign-to-crack-undermine-internet-encryption
28. Liang, B., Li, H., Chang, J.: The generic transformation from standard signatures to identity-based aggregate signatures. In: Lopez, J., Mitchell, C.J. (eds.) ISC 2015. LNCS, vol. 9290, pp. 21–41. Springer, Heidelberg (2015). doi:10.1007/978-3-319-23318-5_2
29. Nielsen, J.B.: Separating Random Oracle Proofs from Complexity Theoretic Proofs: the non-committing encryption case. In: Yung, M. (ed.) CRYPTO 2002. LNCS, vol. 2442, pp. 111–126. Springer, Heidelberg (2002). doi:10.1007/3-540-45708-9_8
30. Perlroth, N., Larson, J., Shane, S.: N.S.A. able to foil basic safeguards of privacy on web. Internation New York Times (2013). http://www.nytimes.com/2013/09/06/us/nsa-foils-much-internet-encryption.html
31. Rao, V.: Adaptive multiparty non-interactive key exchange without setup in the standard model. Cryptology ePrint Archive, Report 2014/910 (2014). http://eprint.iacr.org/
32. Sahai, A., Waters, B.: How to use indistinguishability obfuscation: deniable encryption, and more. In: STOC, pp. 475–484 (2014)