Universal Samplers with Fast Verification

Venkata Koppula
University of Texas at Austin
kvenkata@cs.utexas.edu

Andrew Poelstra
Blockstream
apoelstra@blockstream.com

Brent Waters
University of Texas at Austin
bwaters@cs.utexas.edu*

January 9, 2017

Abstract

Recently, Hofheinz, Jager, Khurana, Sahai, Waters and Zhandry [9] proposed a new primitive called *universal samplers* that allows oblivious sampling from arbitrary distributions, and showed how to construct universal samplers using indistinguishability obfuscation ($i\mathcal{O}$) in the ROM.

One important limitation for applying universal samplers in practice is that the constructions are built upon indistinguishability obfuscation. The costs of using current $i\mathcal{O}$ constructions is prohibitively large. We ask is whether the cost of a (universal) sampling could be paid by one party and then shared (soundly) with all other users? We address this question by introducing the notion of universal samplers with verification. Our notion follows the general path of [9], but has additional semantics that allows for validation of a sample.

In this work we define and give a construction for universal samplers with verification. Our verification procedure is simple and built upon one-time signatures, making verification of a sample much faster than computing it. Security is proved under the sub exponential hardness of indistinguishability obfuscation, puncturable pseudorandom functions, and one-time signatures.

1 Introduction

The Random Oracle Model (ROM), introduced by Bellare and Rogaway [3], is a widely used heuristic in cryptography. In the random oracle model a hash function H is modeled as an oracle that when sampled with an input x will output a sample of a fresh random string u. This functionality has been applied in numerous cryptographic applications that have leveraged features of the model such programmability and rewinding. However, one significant limitation of the model is that it can only be used to sample from random strings, whereas in many applications we would like the ability of (obliviously) sample from *arbitrary* distributions.[1]

Recently, Hofheinz, Jager, Khurana, Sahai, Waters and Zhandry [9], addressed this problem. They proposed a new primitive called *universal samplers* that allows oblivious sampling from arbitrary distributions, and showed how to construct universal samplers using indistinguishability obfuscation ($i\mathcal{O}$) in the ROM.

Hofheinz et al. argued that universal samplers can give way to a powerful notion of universal setup. Several cryptographic schemes require the use of a trusted setup to generate common parameters. For example, in an elliptic curve-based public key scheme we might want to generate a common set of curve parameters for everyone to use. However, each such cryptographic scheme proposed will require its users to

*Supported by NSF CNS-1228599 and CNS-1414082, DARPA SafeWare, Microsoft Faculty Fellowship, and Packard Foundation Fellowship.

[1]One could define the random oracle model to provide samples from arbitrary distributions on arbitrary sets. However, such a model no longer heuristically corresponds to real world hash functions.

agree on some trusted user or process for setting up the parameters for the specific scheme. In practice the cost of executing such a setup for every single instance can be quite onerous and might serve as a barrier to adoption. In particular, the effort to get everyone to agree on an authority or gather an acceptable set of parties together to jointly perform (via multiparty computation) the setup process can be difficult. Such "human overhead" is difficult to measure in terms of traditional computational metrics. Using universal parameters, however, one can service several schemes with one universal trusted setup. Here the trusted setup party (or parties) will create a universal sampler. Then if any particular scheme has a setup algorithm described by circuit d, its users can simply universally sample from the distribution d to get a set of parameters for that particular scheme.

In addition to the application of universal setup described above, Hofheinz et al. provided that several applications of universal samplers, non-interactive key exchange and broadcast encryption. Subsequent works [11, 10] used universal parameters to construct universal signature aggregators and constrained pseudorandom functions respectively.

The costs of using universal samplers One important limitation for applying universal samplers in practice is that the constructions are built upon indistinguishability obfuscation. The costs of using current $i\mathcal{O}$ constructions is prohibitively large. Even so we might hope that efforts toward moving the performance of $i\mathcal{O}$ to practice [1, 17, 2] will follow the path of other cryptographic primitives such as multiparty computation and ORAM. Such primitives were once considered way too expensive to even consider, however, sustained algorithmic and engineering efforts (see for example the references in [12]) have gotten reduced the costs by several orders of magnitude and gotten them to the point where many interesting programs or computations can be executed. A central concern though is that even if we assume that the performance costs of obfuscation follow a similar trajectory to other works that the costs will still remain significantly above "traditional" cryptographic primitives such as encryption, signing, etc. that have costs imperceptible to a human.

In the context of universal samplers and a trusted universal setup, it might be acceptable for a well funded party to invest the computation needed to determine a parameter needed for a given scheme, but not acceptable to assume that every single party using the scheme is willing to pay such a high cost.

We ask whether the cost of a (universal) sampling could be paid by one party and then shared (soundly) with all other users. Returning to our elliptic curve example, one could imagine that NIST would run a universal sampler for a particular setup scheme to obtain a set of curve parameters p. Could NIST then share the parameters p with all other users in a manner that convinced them that they were sampled correctly, but where the cost of verification was much smaller than repeating the sampling? We restate this question in terms of universal samplers:

Is it possible to construct a universal sampler that allows for fast verification (that is, verification that uses only traditional cryptography)?

We address this question by introducing the notion of universal samplers with verification. Our notion follows the general path of [9], but has additional semantics that allows for validation of a sample. In our system the Setup outputs a Universal Sampler parameter U as before, but also outputs a verification key VK. [2]

The sampling algorithm Sample as in [9] will maps the sampler parameters U and input circuit $d(\cdot)$ to an element p sampled from d, but also output a certificate σ which can be thought of as a signature on p. Finally, we include an additional algorithm, Check, that takes VK, σ, and the input circuit, and checks whether these are consistent.

We can see now that there are two paths to obtaining a sample from the distribution d. One can call $\mathsf{Sample}(U, d)$ and obtain p. Or one can let another party perform this step and receive p, σ and validate this by calling $\mathsf{Check}(\mathrm{VK}, d, p, \sigma)$.

[2] As in [9] there is a single trusted setup process that runs Setup to produces the sampler parameters. It is then expected to erase the random coins it used. Also as noted by [9] one could employ multi-party computation to distribute this initial setup task among multiple parties.

We require two security properties. The first is the prior indistinguishability of real world and ideal world given in [9]. The second property we require is that it should be computationally infeasible for any poly-time adversary $\mathcal{A}$ to produce a triple d^*, p^*, σ^* such that $\mathsf{Check}(\mathrm{VK}, d^*, p^*, \sigma^*) = 1$ and $\mathsf{Sample}(U, d^*) \neq p^*$. Intuitively, it should be hard to produce a signature that convokes a third party of the "wrong" output.

The first thing we observe is that any standard universal sampler scheme implies one with verification, but in an uninteresting way. To do this we can simply let $\mathrm{VK} = U$ and have the Check algorithm run $\mathsf{Sample}(U, d)$ itself. This will clearly result in a secure universal sampler with verification if the base universal sampler is secure, but not result in any of the savings that motivated our discussion above.

For this reason any scheme of interest must have a verification algorithm Check that is significantly more efficient than running Sample. Ideally, the cost will be close to that of "traditional" cryptographic primitives. We choose not to formalize this final requirement.

Our technical approach We begin our technical exposition by describing what we call *prefix-restricted signature scheme*. This is specialized signature scheme that will we use to sign samples output from our universal sampler. A *prefix-restricted signature scheme* is over a message space $\mathcal{M}_1 \times \mathcal{M}_2$ and differs from an ordinary signature scheme in the following ways:

- A secret key can either be a "master secret key" or admit a "punctured" form at a message (m_1^*, m_2^*) capable of signing any message (m_1, m_2) such that (a) $m_1 \neq m_1^*$ or (b) $(m_1, m_2) = (m_1^*, m_2^*)$.

- In our security game an attacker selectively gives (m_1^*, m_2^*) and receives back a corresponding punctured signing key. No signing queries are allowed. The attacker should be unable to provide a signature on any message (m_1, m_2) where $m_1 = m_1^*$ and $m_2 \neq m_2^*$.

- The scheme is deterministic, even with respect to the master and punctured keys. Moreover, signatures produced by punctured keys (on messages for which this is possible) must be equal to those produced by unpunctured keys on the same messages.

This notion shares a similar flavor to earlier related concepts such as constrained signature[5]. It is actually the last property of matching signature outputs between all key types that is critical for our use and the most tricky to satisfy. Looking ahead, the reason we will need this is to be able to argue that two programs are equivalent when we switch from using a master key to a punctured key in an experiment.

While achieving some form of signature delegation has been considered in other works and transforming a standard signature scheme to a deterministic one can be done by a straightforward application of a PRF [8], forcing such a constrained signature key to output the same signatures as a master key is somewhat more tricky.

We construct a prefix-restricted signature scheme from a deterministic one-time signature scheme (on arbitrary length messages) and a puncturable pseudo random function [4, 6, 13, 15]. Briefly recall that a puncturable PRF is a PRF when one can create a punctured key that allows a keyed function $F(K, \cdot)$ to be evaluated at all but a small number of points.

Let the length of the first message piece, $\mathcal{M}_1$, be n and let m^i be the i-bit prefix of m and $\overline{m}^i$ be the i-bit prefix of m with bit i flipped. To sign a message $m = (m_1, m_2)$. We will first create a Naor-Yung [14] style certificate tree of length n. To create a signature on m for each $i = 1$ to n we first generate a two verify and signing key pairs (one as the 0 key and the other as the 1 key). We denote the keys output in step i as $(\mathrm{SK}_{m^i}, \mathrm{VK}_{m^i}) \leftarrow \mathsf{KeyGen}_1(1^\lambda; F(K, m^i))$ and $(\mathrm{SK}_{\overline{m}^i}, \mathrm{VK}_{\overline{m}^i}) \leftarrow \mathsf{KeyGen}_1(1^\lambda; F(K, \overline{m}^i))$. Importantly, notice that instead of sampling these keys randomly we replace the setup random coins with the output of $F(K, m^i)$ and $F(K, \overline{m}^i)$. Next we create a signature chain by letting σ_i be the signature on $(\mathrm{VK}_{m^{i-1}|0}, (\mathrm{VK}_{m^{i-1}|1})$ with key SK_{m^i}. Finally, at the bottom of the tree we sign the whole message m using the final key SK_{m^i}. Verification is done by verifying the chain and then the signature on the final message.

A punctured key for (m_1^*, m_2^*) can be created by giving out $(\mathrm{SK}_{\overline{m}^i}$ for $i \in [1, n]$, a puncturable PRF key that is punctured as all prefixes of m_1^*, a signature on (m_1^*, m_2^*), and the signature certificates along the path. The fact that the one-time signatures are deterministic coupled with the deterministic process for generating one-time keys allows for corresponding signatures from the master and punctured keys to be the same.

The main construction Now that we have this tool in place we can get back to our universal sampler construction. As mentioned in the work of [9], when using indistinguishability obfuscation in the random oracle model, the hash function(s) modeled as a random oracle *must* be outside the obfuscated circuit(s). Our approach for doing so is different from that of [9], and a remarkable feature of our scheme is its simplicity. The sampler setup algorithm will first generate a prefix restricted signature scheme verification and signing key pair. Next the universal sampler parameters are created as the obfuscation of a program that takes two inputs x, d and outputs $p = d(r)$, where r is computed using a puncturable PRF on input $x||d$. The program also outputs a signature σ (using the signing key) on $(x||d, p)$ using a prefix-restricted signature scheme. The sampler parameters, U, are the obfuscated program and the verification key VK of the universal sampler is the verification key of the prefix restricted signature.

To sample from a distribution d, one computes $x = H(d)$ and runs the sampler output on inputs x, d. Finally, the verification algorithm is used to check that p was the correct output sample for a circuit d when given a prefix restricted signature σ. The verification algorithm first computes $x = H(d)$. Then, it simply checks that the signature σ verifies on the message $m = (m_1, m_2) = (x||d, p)$.

We can now examine the overhead of verification in our sampler which is simply the prefix restricted signature verification on $(x||d, p)$. The cost of performing this will be ℓ one-time signature verifications where ℓ is the bit length of $x||d$. In our construction the bit length of x will be roughly the size of the output size of samples plus a security parameter and the bit length of d corresponds to the string describing the circuit. While the time to verify these ℓ one time signatures is significantly longer than a standard signature scheme, the verification time will be much shorter than running the obfuscated program. Moreover, we would expect it to remain so even as improvements in obfuscation move towards making it realizable.

Proving security The security of our universal sampler with verification is based on subexponential hardness of the underlying building blocks of indistinguishability obfuscation, puncturable pseudorandom functions, and prefix restricted signatures. In addition, the random oracle heuristic is used to prove security.

Let's start by looking at verification security. At a high level our proof proceeds at as a sequence of games. Assume there exists a PPT attacker $\mathcal{A}$ that makes at most q (unique) queries to the random oracle and produces a forgery σ^* of an output p^* on d^*. Our proof starts by guessing both value of d^* and which random oracle query $i \in [q]$ corresponds to d^*. The reduction will abort if the guess is incorrect. It is this complexity leveraging step of guessing over all possible d^* values that requires the use of sub exponential hardness.

Next, suppose that the actual output of the Sample algorithm on input d is out and let $H(d^*) = x^*)$. We change the sampler parameters U to be an obfuscation of a program that uses a restricted key that cannot sign a message $(m_1 = x^*||d^*, m_2)$ if $m_2 \neq \mathsf{out}$. This transition is indistinguishable to the attacker by indistinguishability obfuscation. For this proof step to go through it is critical the signatures produced from the master key and punctured keys are deterministic and consistent so that the corresponding programs are equivalent. Finally, the proof can be completed by invoking the hardness of breaking the prefix restricted signature.

We now turn to the proof of proving existing definition from [9] of the indistinguishability of real world and ideal. Our proof proceeds in a similar manner to theirs in that we switch from generating samples from the obfuscated program to receiving them via "delayed backdoor programming" from the random oracle. One important difference is that our main obfuscated program computes the output of samples directly, whereas the main program of Hofheinz et al. produces a one-time sampler program, which is then itself invoked to produce the actual sample.

In doing things directly we benefit from a more direct construction at the expense of applying complexity leveraging. Our proof will proceeds as a hybrid that programs the outputs of the random oracle one at a time. At each step our reduction must guess the input to the random oracle. Thus, if D is the number of possible circuits, we get a loss of $D \cdot q$ in the reduction. (We emphasize that we avoid a loss of D^q which could not be overcome with complexity leveraging.) Again, this loss is balanced out by the use of sub exponential hardness. We also made our proof steps more modular than those in [9]. One tool in doing so is the introduction of a tool we call a puncturable pseudorandom deterministic encryption scheme.

Other applications of fast verification In addition, to the application of establishing a set of common parameters for a cryptographic scheme [9] give mutliple other applications of universal samplers. Here we sketch how some of these can benefit if the sampler has fast verification.

In the Identity-Based Encryption scheme given in [9] a user performs an encryption to an identity Id by first running $\mathsf{Sample}(U, d_{\mathrm{ID}})$ where d is a circuit that samples and outputs a fresh public key $\mathrm{pk}_{\mathrm{ID}}$. This key is then used to encrypt to the identity. Consider a scenario where more than one party wishes to perform an IBE encryption to the same identity. Using a sampler with fast verification a single party can perform the work of computing $\mathrm{pk}_{\mathrm{ID}}$ and then share this with all other parties (sparing the rest of them from performing the computation). The other parties will be convinced of the authenticity via the certificate and verification procedure.

Another possibility is that instead of multiple parties wishing to perform the computation, there could be a single party running on a machine that has a untrusted procesing environment that is coupled witha trusted, but more expensive environment. Here it would make sense for the untrusted enviorment to perform the sampling and pass on the answer to the more trusted environment to do the rest of the Identity-Based Encryption.

In general these motivational examples will transcend to other applications of universal samplers ranging from non-interactive key exchange [9] to new constructions of constrained PRFs [10]. In particular, adding the fast verification property helps in any mutliparty scenario where multiple (untrusting) parties want to share the output of a call to a sample algorithm. Or where a single party can move the Sample algorithm to an untrusted environment.

1.1 Organization

In Section 2, we introduce some notations and preliminaries. Next, we define our primitive - *universal sampler with verification* in Section 3. To construct a selectively secure universal sampler with (fast) verification, we require the notion of *prefix-restricted signature schemes* defined in Section 4. For the construction, we also require the notion of *puncturable pseudorandom deterministic encryption scheme* defined in Section 5. Finally, in Section 6, we present our fast verification universal sampler scheme.

2 Preliminaries

2.1 Notations

For integers $\ell_{\mathrm{ckt}}, \ell_{\mathrm{inp}}, \ell_{\mathrm{out}}$, let $\mathcal{C}[\ell_{\mathrm{ckt}}, \ell_{\mathrm{inp}}, \ell_{\mathrm{out}}]$ be the set of circuits that have size at most ℓ_{ckt} bits, take ℓ_{inp} bits as input and output ℓ_{out} bits.

2.2 Puncturable Pseudorandom Functions

The notion of constrained PRFs was introduced in the concurrent works of [4, 6, 13]. Punctured PRFs, first termed by [15] are a special class of constrained PRFs.

A PRF $F : \mathcal{K} \times \mathcal{X} \to \mathcal{Y}$ is a puncturable pseudorandom function if there is an additional key space $\mathcal{K}_p$ and three polynomial time algorithms $F.\mathsf{setup}$, $F.\mathsf{eval}$ and $F.\mathsf{puncture}$ as follows:

- $F.\mathsf{setup}(1^\lambda)$ is a randomized algorithm that takes the security parameter λ as input and outputs a description of the key space $\mathcal{K}$, the punctured key space $\mathcal{K}_p$ and the PRF F.
- $F.\mathsf{puncture}(K, x)$ is a randomized algorithm that takes as input a PRF key $K \in \mathcal{K}$ and $x \in \mathcal{X}$, and outputs a key $K\{x\} \in \mathcal{K}_p$.
- $F.\mathsf{eval}(K\{x\}, x')$ is a deterministic algorithm that takes as input a punctured key $K\{x\} \in \mathcal{K}_p$ and $x' \in \mathcal{X}$. Let $K \in \mathcal{K}$, $x \in \mathcal{X}$ and $K\{x\} \leftarrow F.\mathsf{puncture}(K, x)$. For correctness, we need the following

property:

$$F.\mathsf{eval}(K\{x\}, x') = \begin{cases} F(K, x') & \text{if } x \neq x' \\ \perp & \text{otherwise} \end{cases}$$

We will now recall the selective security game for puncturable PRFs. The following definition is equivalent to the one in [15]. Consider a challenger C and adversary A. The security game between C and A consists of two phases.

Challenge Phase: The adversary A sends its challenge string x^*. The challenger chooses a uniformly random PRF key $K \leftarrow \mathcal{K}$. Next, it chooses a bit $b \in \{0,1\}$ and a uniformly random string $y \leftarrow \mathcal{Y}$. It computes $K\{x^*\} \leftarrow F.\mathsf{puncture}(K, x^*)$. If $b = 0$, the challenger outputs $K\{x^*\}$ and $(F(K, x^*), y)$. Else, the challenger outputs $K\{x^*\}$ and $(y, F(K, x^*))$.

Guess: A outputs a guess b' of b.

A wins the security game if $b = b'$. The advantage of A in the security game against F is defined as $\mathsf{Adv}_A^F = \Pr[b = b'] - 1/2$.

Definition 2.1. *The PRF F is a selectively secure puncturable PRF if for all probabilistic polynomial time adversaries A $\mathsf{Adv}_{\mathcal{A}}^F(\lambda)$ is negligible in λ.*

Remark 2.1. *Note the difference between this definition and the one in previous works is in the challenge phase. Here, we require that the challenger output a punctured PRF key and a pair $(y_0, y_1) \in \mathcal{Y}^2$. It chooses a bit b. If $b = 0$, then $y_0 = F(K, x^*)$ and y_1 is chosen uniformly at random. Else, y_0 is chosen uniformly at random and $y_1 = F(K, x^*)$.*

Remark 2.2. *This definition can be extended to handle multiple points being punctured. More formally, we can define the notion of t-puncturable PRFs, where the PRF key K can be punctured at t points. In the selective security game, the adversary chooses the t puncture points, sends them to the challenger. The challenger outputs a key punctured at the t points, along with t output strings, which are either PRF evaluations at the t points or uniformly random strings.*

2.3 Indistinguishability Obfuscation

We recall the definition of indistinguishability obfuscation from [7, 15].

Definition 2.2. *(Indistinguishability Obfuscation) Let $\mathcal{C} = \{\mathcal{C}_\lambda\}_{\lambda \in \mathbb{N}}$ be a family of polynomial-size circuits. Let $i\mathcal{O}$ be a uniform PPT algorithm that takes as input the security parameter λ, a circuit $C \in \mathcal{C}_\lambda$ and outputs a circuit C'. $i\mathcal{O}$ is called an indistinguishability obfuscator for a circuit class $\{\mathcal{C}_\lambda\}$ if it satisfies the following conditions:*

- *(Preserving Functionality) For all security parameters $\lambda \in \mathbb{N}$, for all $C \in \mathcal{C}_\lambda$, for all inputs x, we have that $C'(x) = C(x)$ where $C' \leftarrow i\mathcal{O}(1^\lambda, C)$.*
- *(Indistinguishability of Obfuscation) For any (not necessarily uniform) PPT distinguisher $\mathcal{B} = (Samp, \mathcal{D})$, there exists a negligible function $negl(\cdot)$ such that the following holds: if for all security parameters $\lambda \in \mathbb{N}$, $\forall x, C_0(x) = C_1(x) : (C_0; C_1; \sigma) \leftarrow Samp(1^\lambda)$, then*

$$\begin{aligned} &|\Pr[\mathcal{D}(\sigma, i\mathcal{O}(1^\lambda, C_0)) = 1 : (C_0; C_1; \sigma) \leftarrow Samp(1^\lambda)] - \\ &\Pr[\mathcal{D}(\sigma, i\mathcal{O}(1^\lambda, C_1)) = 1 : (C_0; C_1; \sigma) \leftarrow Samp(1^\lambda)]| \\ &\leq negl(\lambda). \end{aligned}$$

In a recent work, [7] showed how indistinguishability obfuscators can be constructed for the circuit class $P/poly$. We remark that $(Samp, \mathcal{D})$ are two algorithms that pass state, which can be viewed equivalently as a single stateful algorithm $\mathcal{B}$. In our proofs we employ the latter approach, although here we state the definition as it appears in prior work.

3 Universal Samplers with Verification

We will now define the syntax and security definitions for universal samplers with verification. In this primitive, as in [9], there is an algorithm Setup which outputs a sampler parameter U as well as a sampling algorithm Sample which maps the sampler parameters and input circuit to an element sampled from the desired distribution. We modify this definition so that Setup also outputs a verification key VK, and Sample also outputs a 'certificate' σ asserting that the sampler output matches the input circuit. An additional algorithm, Check, takes VK, σ, and the input circuit, and checks whether these are consistent.

Syntax Let $\ell_{\mathrm{ckt}}, \ell_{\mathrm{inp}}$ and ℓ_{out} be polynomials. An $(\ell_{\mathrm{ckt}}, \ell_{\mathrm{inp}}, \ell_{\mathrm{out}})$-universal sampler scheme consists of algorithms Setup, Sample and Check defined below.

- $\mathsf{Setup}(1^\lambda)$ takes as input the security parameter λ and outputs the sampler parameters U and a verification key VK.
- $\mathsf{Sample}(U, d)$ takes as input the universal sampler U and a circuit $d \in \mathcal{C}[\ell_{\mathrm{ckt}}(\lambda), \ell_{\mathrm{inp}}(\lambda), \ell_{\mathrm{out}}(\lambda)]$. The output of the function is the induced parameters $p_d \in \{0,1\}^{\ell_{\mathrm{out}}(\lambda)}$ and a certificate σ_d.
- $\mathsf{Check}(\mathrm{VK}, d, p, \sigma)$ takes as input the verification key VK, the circuit $d \in \mathcal{C}[\ell_{\mathrm{ckt}}(\lambda), \ell_{\mathrm{inp}}(\lambda), \ell_{\mathrm{out}}(\lambda)]$, $p \in \{0,1\}^{\ell_{\mathrm{out}}(\lambda)}$ and a certificate σ. It outputs either 0 or 1.

For simplicity of notation, we will drop the dependence of $\ell_{\mathrm{ckt}}, \ell_{\mathrm{inp}}, \ell_{\mathrm{out}}$ on λ when the context is clear.

Correctness For correctness, we require that any honestly generated output and certificate must pass the verification. More formally, for all security parameters λ, $(U, \mathrm{VK}) \leftarrow \mathsf{Setup}(1^\lambda)$, circuit $d \in \mathcal{C}[\ell_{\mathrm{ckt}}, \ell_{\mathrm{inp}}, \ell_{\mathrm{out}}]$,

$$\mathsf{Check}(\mathrm{VK}, d, \mathsf{Sample}(U, d)) = 1.$$

3.1 Security

For security, we require the primitive to satisfy the real vs ideal world definition from [9]. In addition to that, we also need to ensure that no adversary can output 'fake certificates'. This intuition is captured by the following unforgeability definitions. Informally, we require that any PPT adversary should not be able to output a tuple (d^*, p^*, σ^*) such that $\mathsf{Sample}(U, d^*) \neq p^*$ but $\mathsf{Check}(U, d^*, p^*, \sigma^*) = 1$. For clarity of presentation, we chose to present the [9] definitions for real vs ideal world indistinguishability in Appendix 3.2.

The security definition given here is an adaptive game in the random oracle model. One could consider presenting the definition in the standard model. However, as shown in [9], the simulation security definition must involve the random oracle. As a result, we choose to have a random oracle based definition for unforgeability as well.

Definition 3.1. *An $(\ell_{\mathrm{ckt}}, \ell_{\mathrm{inp}}, \ell_{\mathrm{out}})$-universal sampler scheme* $(\mathsf{Setup}, \mathsf{Sample}, \mathsf{Check})$ *is said to be a adaptively secure against forgeries if every PPT adversary $\mathcal{A}$,* $\Pr[\mathcal{A}$ *wins in* $\mathsf{Expt}] \leq$ *negl*(λ)*, where* Expt *is defined as follows.*

1. *The challenger sends* $(U, \mathrm{VK}) \leftarrow \mathsf{Setup}(1^\lambda)$ *to* $\mathcal{A}$.
2. *$\mathcal{A}$ sends random oracle queries* (RO, x)*. For each unique query, the challenger chooses a uniformly random string y and outputs y. It also adds the tuple (x, y) to its table.*
3. *$\mathcal{A}$ sends its output (p^*, σ^*) to the challenger.*

$\mathcal{A}$ wins if $\mathsf{Check}(\mathrm{VK}, d^*, p^*, \sigma^*) = 1$ *and* $\mathsf{Sample}(U, d^*) \neq p^*$.

3.2 Simulation Security - Real vs Ideal World Indistinguishability

In this part, we will recall the adaptive security definition for universal samplers from [9]. As in [9], an *admissible adversary* is an interactive Turing Machine that outputs one bit, with the following input/output behavior:

- $\mathcal{A}$ takes as input security parameter λ and sampler parameters U.
- $\mathcal{A}$ can send a random oracle query (RO, x), and receives the output of the random oracle on input x.
- $\mathcal{A}$ can send a message of the form (params, d) where $d \in \mathcal{C}[\ell_{\text{ckt}}, \ell_{\text{inp}}, \ell_{\text{out}}]$. Upon sending this message, $\mathcal{A}$ is required to honestly compute $p_d = \mathsf{Sample}(U, d)$, making use of any additional random oracle queries, and $\mathcal{A}$ appends (d, p_d) to an auxiliary tape.

Let $\mathsf{SimUGen}$ and SimRO be PPT algorithms. Consider the following two experiments:

$\mathsf{Real}^{\mathcal{A}}(1^\lambda)$:

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

$\mathsf{Ideal}^{\mathcal{A}}_{\mathsf{SimUGen},\mathsf{SimRO}}(1^\lambda)$:

1. A truly random function F that maps ℓ_{ckt} bits to ℓ_{inp} bits is implemented by assigning random ℓ_{inp}-bit outputs to each unique query made to F. Throughout this experiment, a Samples Oracle O is implemented as follows: On input d, where $d \in \mathcal{C}[\ell_{\text{ckt}}, \ell_{\text{inp}}, \ell_{\text{out}}]$, O outputs $d(F(d))$.
2. $(U, \tau) \leftarrow \mathsf{SimUGen}(1^\lambda)$. Here, $\mathsf{SimUGen}$ can make arbitrary queries to the Samples Oracle O.
3. $\mathcal{A}(1^\lambda, U)$ and $\mathsf{SimRO}(\tau)$ begin simultaneous execution.
   - 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 O.
   - Finally, after $\mathcal{A}$ sends any message of the form (params, d), the auxiliary tape of $\mathcal{A}$ is examined until an entry of the form (d, p_d) is added to it. At this point, if p_d is not equal to $d(F(d))$, then experiment aborts, resulting in an *Honest Sample Violation*.
4. Upon termination of $\mathcal{A}$, the output of the experiment is the final output of the execution of $\mathcal{A}$.

Definition 3.2. *A universal sampler scheme $\mathcal{U} = (\mathsf{Setup}, \mathsf{Sample})$, parameterized by polynomials $\ell_{\text{ckt}}, \ell_{\text{inp}}$ and ℓ_{out}, is said to be adaptively secure in the random oracle model if there exist PPT algorithms $\mathsf{SimUGen}$ and SimRO such that for all PPT adversaries $\mathcal{A}$, the following hold:*

$$\Pr[\mathsf{Ideal}^{\mathcal{A}}_{\mathsf{SimUGen},\mathsf{SimRO}}(1^\lambda) \text{ aborts }] = 0^3$$

and

$$\left|\Pr[\mathsf{Real}^{\mathcal{A}}(1^\lambda) = 1] - \Pr[\mathsf{Ideal}^{\mathcal{A}}_{\mathsf{SimUGen},\mathsf{SimRO}}(1^\lambda) = 1]\right| \le negl(\lambda).$$

4 Prefix-Restricted Signatures

In this section we describe a primitive, *prefix-restricted signature schemes*. These are a form of constrained signature[5] which will be used as a building block in the main construction. A prefix-restricted signature schemes is over a message space $\mathcal{M}_1 \times \mathcal{M}_2$ and differs from an ordinary signature scheme in the following ways:

[3] The definition in [9] only requires this probability to be negligible in λ. However, the construction actually achieves zero probability of Honest Sample Violation. Hence, for the simplicity of our proof, we will use this definition

- A secret key can either be a "master secret key" or admit a "punctured" form at a message (m_1^*, m_2^*) capable of signing any message (m_1, m_2) such that (a) $m_1 \neq m_1^*$ or (b) $(m_1, m_2) = (m_1^*, m_2^*)$.
- In our security game an attacker selectively gives (m_1^*, m_2^*) and receives back a corresponding punctured signing key. No signing queries are allowed. The attacker should be unable to provide a signature on any message (m_1, m_2) where $m_1 = m_1^*$ and $m_2 \neq m_2^*$.

  Our security property does not allow the adversary to make signing queries on any message; these are not needed for our purposes.
- The scheme is deterministic, even with respect to punctured keys. That is, signatures produced by punctured keys (on messages for which this is possible) must be equal to those produced by unpunctured keys on the same messages.

This last point is the most important, since this strong determinism is required to obtain the functional equivalence required by indistinguishability obfuscation; it is also the reason that we could not use an existing primitive.

4.1 Definition

Let $\mathcal{M}_1$ and $\mathcal{M}_2$ be two message spaces. We define a prefix-restricted signature scheme for message space $\mathcal{M}_1 \times \mathcal{M}_2$ as a collection of five algorithms:

- $\mathsf{Pre.Setup}(1^\lambda)$ is a randomized algorithm that takes as input the security parameter λ and outputs a master signing key MSK and verification key VK.
- $\mathsf{Pre.Sign}(\mathrm{MSK}, (m_1, m_2))$ is a deterministic algorithm that takes a master signing key MSK and message pair (m_1, m_2), and outputs a signature σ.
- $\mathsf{Pre.Verify}(\mathrm{VK}, (m_1, m_2), \sigma)$ is deterministic and takes a message pair (m_1, m_2), verification key VK and signature σ, and outputs a bit.
- $\mathsf{Pre.Restrict}(\mathrm{MSK}, (m_1^*, m_2^*))$ (possibly randomized) takes a master signing key MSK and message pair (m_1^*, m_2^*), and outputs a restricted key $\mathrm{SK}\{m_1^*, m_2^*\}$.
- $\mathsf{Pre.ResSign}(\mathrm{SK}\{m_1^*, m_2^*\}, (m_1, m_2))$ is deterministic and takes a restricted signing key $\mathrm{SK}\{m_1^*, m_2^*\}$, a message pair (m_1, m_2), and outputs a signature σ.

Correctness We define correctness by the following conditions:

1. For all $(\mathrm{MSK}, \mathrm{VK}) \leftarrow \mathsf{Pre.Setup}(1^\lambda)$ and message pairs $(m_1, m_2) \in \mathcal{M}_1 \times \mathcal{M}_2$,
$$\mathsf{Pre.Verify}(\mathrm{VK}, (m_1, m_2), \mathsf{Pre.Sign}(\mathrm{MSK}, (m_1, m_2))) = 1.$$
2. For all $(\mathrm{MSK}, \mathrm{VK}) \leftarrow \mathsf{Pre.Setup}(1^\lambda)$, $(m_1^*, m_2^*) \in \mathcal{M}_1 \times \mathcal{M}_2$, $\mathrm{SK}\{m_1^*, m_2^*\} \leftarrow \mathsf{Pre.Restrict}(\mathrm{MSK}, (m_1^*, m_2^*))$, and messages $(m_1, m_2) \in \mathcal{M}_1 \times \mathcal{M}_2$ such that either $m_1 \neq m_1^*$ or $(m_1, m_2) = (m_1^*, m_2^*)$,
$$\mathsf{Pre.Sign}(\mathrm{MSK}, (m_1, m_2)) = \mathsf{Pre.ResSign}(\mathrm{SK}\{m_1^*, m_2^*\}, (m_1, m_2)).$$

Security For security, we require that no polynomial time adversary can output a forgery, even after receiving a restricted signing key.

Definition 4.1. *A two message signature scheme is* selectively secure *if every PPT adversary $\mathcal{A}$ has at most negligible advantage in the following security game:*

1. *$\mathcal{A}$ provides a message pair (m_1^*, m_2^*).*

2. *The challenger generates the keys* $(\mathrm{MSK}, \mathrm{VK}) \leftarrow \mathsf{Pre.Setup}(1^\lambda)$ *and* $\mathrm{SK}\{m_1^*, m_2^*\} \leftarrow \mathsf{Pre.Restrict}(\mathrm{MSK}, (m_1^*, m_2^*))$ *and sends the tuple* $(\mathrm{SK}\{m_1^*, m_2^*\}, \mathrm{VK})$ *to* $\mathcal{A}$*.*

3. $\mathcal{A}$ *replies with a message pair* (m_1, m_2) *such that* $m_1 = m_1^*$ *but* $m_2 \neq m_2^*$*, and signature* σ *and wins if it verifies; that is,* $\mathsf{Pre.Verify}(\mathrm{VK}, (m_1, m_2), \sigma) = 1$*.*

We define $\mathcal{A}$*'s advantage to be* $\Pr[\mathcal{A}\ wins]$*.*

4.2 Construction

Next, we construct a restricted-prefix signature scheme from a secure puncturable PRF F and secure deterministic one-time signature scheme $(\mathsf{KeyGen}_1, \mathsf{Sign}_1, \mathsf{Verify}_1)$. Deterministic one-time signature schemes can be constructed using one-way functions.

We consider $m = (m_1, m_2)$ to be a single message; let N be the total length $|m| = |m_1| + |m_2|$ and $n = |m_1|$. Our message space is thus $\{0,1\}^N = \{0,1\}^n \times \{0,1\}^{N-n}$. We further define ℓ to be the bit-length of the verification keys produced by KeyGen_1, and require the domain of $F(K, \cdot)$ to be all bitstrings of length at most n. Assume also that the message space of the one-time signature scheme is all bitstrings of length at most $\max\{N, 2\ell + 1\}$. Finally, ϵ denotes the empty string.

For any message m and $i \in \{1, \ldots, N\}$ we define

$$\begin{aligned} m^i &= \text{ the } i\text{-bit prefix of } m \\ \overline{m}^i &= \text{ the } i\text{-bit prefix of } m \text{ with bit } i \text{ flipped} \\ m[i] &= \text{ the } i\text{th bit of } m \\ \overline{m[i]} &= \text{ the opposite of the } i\text{th bit of } m \end{aligned}$$

Notice that with this notation, if $m = (m_1, m_2)$ that $m_1 = m^n$.

Finally, we also define an operator $\mathsf{switch}_b(x, y)$ as follows:

$$\mathsf{switch}_b(x, y) = \begin{cases} (x, y) & \text{if } b = 0. \\ (y, x) & \text{otherwise} \end{cases}$$

Our algorithms are defined as follows:

- $\mathsf{Pre.Setup}(1^\lambda)$ first generates a puncturable PRF key $K \leftarrow F.\mathsf{setup}(1^\lambda)$, then $(\mathrm{SK}_\epsilon, \mathrm{VK}_\epsilon) \leftarrow \mathsf{KeyGen}_1(1^\lambda; F(K, \epsilon))$. The verification key is VK_ϵ; the secret key is $(K, \mathrm{SK}_\epsilon)$.

- $\mathsf{Pre.Sign}((K, \mathrm{SK}_\epsilon), m)$ For each i from 1 to n compute

$$\begin{aligned} (\mathrm{SK}_{m^i}, \mathrm{VK}_{m^i}) &= \mathsf{KeyGen}_1(1^\lambda; F(K, m^i)) \\ (\mathrm{SK}_{\overline{m}^i}, \mathrm{VK}_{\overline{m}^i}) &= \mathsf{KeyGen}_1(1^\lambda; F(K, \overline{m}^i)) \\ (\mathrm{VK}_i, \mathrm{VK}'_i) &= \mathsf{switch}_{m[i]}(\mathrm{VK}_{m^i}, \mathrm{VK}_{\overline{m}^i}) \\ \sigma_i &= \mathsf{Sign}(\mathrm{SK}_{m^{i-1}}, (\mathrm{VK}_i, \mathrm{VK}'_i)) \end{aligned}$$

  Finally, compute

$$\sigma^* = \mathsf{Sign}(\mathrm{SK}_{m^n}, m)$$

  and output

$$\sigma = \left\{(\mathrm{VK}_i, \mathrm{VK}'_i, \sigma_i)_{i=1}^n, \sigma^*\right\}$$

- $\mathsf{Pre.Verify}(\mathrm{VK}_\epsilon, m, \sigma = \left\{(\mathrm{VK}_i, \mathrm{VK}'_i, \sigma_i)_{i=1}^n, \sigma^*\right\})$ checks that for each i from 0 to $(n-1)$, that

$$\mathsf{Verify}_1(\mathrm{VK}_i, \sigma_{i+1}, (\mathrm{VK}_{i+1}, \mathrm{VK}'_{i+1})) = 1$$

Here we consider $\mathrm{VK}_0 = \mathrm{VK}_\epsilon$. We check also that

$$\mathsf{Verify}_1(\mathrm{VK}_n, \sigma^*, m) = 1$$

We output 1 if the above checks passed; otherwise output 0.

- $\mathsf{Pre.Restrict}((K, \mathrm{SK}_\epsilon), m)$ computes, for each i from 1 to n,

$$\begin{aligned}(\mathrm{SK}_{m^i}, \mathrm{VK}_{m^i}) &= \mathsf{KeyGen}_1(1^\lambda; F(K, m^i))\\ (\mathrm{SK}_{\overline{m}^i}, \mathrm{VK}_{\overline{m}^i}) &= \mathsf{KeyGen}_1(1^\lambda; F(K, \overline{m}^i))\\ (\mathrm{VK}_i, \mathrm{VK}'_i) &= \mathsf{switch}_{m[i]}(\mathrm{VK}_{m^i}, \mathrm{VK}_{\overline{m}^i})\\ \sigma_i &= \mathsf{Sign}(\mathrm{SK}_{m^{i-1}}, (\mathrm{VK}_i, \mathrm{VK}'_i))\end{aligned}$$

  as well as

$$\sigma^* = \mathsf{Sign}(\mathrm{SK}_{m^n}, m)$$

  It bundles these up into

$$\sigma = \left\{(\mathrm{VK}_i, \mathrm{VK}'_i, \sigma_i)_{i=1}^n, \sigma^*\right\}$$

  Next, it punctures the key K at $\{m^i\}_{i=1}^n \cup \{\epsilon\}$ to obtain a punctured key K'. It outputs the punctured key as

$$\mathrm{SK}\{m\} = \{\sigma, \{\mathrm{SK}_{\overline{m}^i}\}_{i=1}^n, K'\}$$

- $\mathsf{Pre.ResSign}(\mathrm{SK}\{m_*\}, m)$ First, expand $\mathrm{SK}\{m_*\}$ as

$$\mathrm{SK}\{m_*\} = \left\{\sigma = \left\{(\mathrm{VK}_i, \mathrm{VK}'_i, \sigma^*_i)_{i=1}^n, \sigma^*\right\}, \{\mathrm{SK}'_i\}_{i=1}^n, K'\right\}$$

  We have three cases:

  - If $m = m_*$ output σ.
  - Otherwise, if $m^n = m_*^n$ but $m \neq m_*$ output $\perp$.
  - Otherwise, there is some least bit position i^*, $1 \leq i^* < n$ such that $m[i] \neq m^*[i]$. For $1 \leq i \leq i^*$ set $(\mathrm{VK}_i^{\mathsf{res}}, \mathrm{VK}_i'^{\mathsf{res}}, \sigma_i) = (\mathrm{VK}_i, \mathrm{VK}'_i, \sigma^*_i)$. For $i^* < i \leq n$ compute

$$\begin{aligned}(\mathrm{SK}_{m^i}, \mathrm{VK}_{m^i}) &= \mathsf{KeyGen}_1(1^\lambda; F(K', m^i))\\ (\mathrm{SK}_{\overline{m}^i}, \mathrm{VK}_{\overline{m}^i}) &= \mathsf{KeyGen}_1(1^\lambda; F_{K'}(\overline{m}^i))\\ (\mathrm{VK}_i^{\mathsf{res}}, \mathrm{VK}_i'^{\mathsf{res}}) &= \mathsf{switch}_{m[i]}(\mathrm{VK}_{m^i}, \mathrm{VK}_{\overline{m}^i})\\ \sigma_i &= \mathsf{Sign}(\mathrm{SK}_{m^{i-1}}, (\mathrm{VK}_i^{\mathsf{res}}, \mathrm{VK}_i'^{\mathsf{res}}))\end{aligned}$$

    (Notice that since $m^{i-1} \neq m_*^{i-1}$ for all $i > i^*$, we are not evaluating $F_{K'}$ on any punctured points.) Finally compute $\sigma^* = \mathsf{Sign}(\mathrm{SK}_{m^n}, m)$ and output

$$\sigma = \left\{(\mathrm{VK}_i^{\mathsf{res}}, \mathrm{VK}_i'^{\mathsf{res}}, \sigma_i)_{i=1}^n, \sigma^*\right\}$$

Correctness For correctness, we need to show that any signature computed using the master signing key verifies, and any signature computed using the restricted key on an unrestricted message is same as the signature computed using the master signing key. The first property is immediate, and follows from the correctness of the one-time deterministic signature scheme.

To prove the second correctness condition, let m be any N bit message, and let $(K, \mathrm{SK}_\epsilon)$ be any master signing key output by $\mathsf{Pre.Setup}$. The restricted key $\mathrm{SK}\{m\}$ consists of a signature $\sigma = \{(\mathrm{VK}_j, \mathrm{VK}'_j, \sigma_j)_{j \leq n}, \sigma^*\}$, n secret keys $\{\mathrm{SK}_{\overline{m}^i}\}_{i \leq n}$ and a PRF key K' punctured at $\{\epsilon \cup \{m^i\}\}$. The restricted secret key $\mathrm{SK}\{m\}$

can be used to sign m and any message $\widetilde{m}$ such that $m^n \neq \widetilde{m}^n$. Clearly, $\mathsf{Pre.ResSign}(\mathrm{SK}\{m\}, m) = \mathsf{Pre.Sign}(\mathrm{SK}, m) = \sigma$.

Consider any message $\widetilde{m}$ such that $m^n \neq \widetilde{m}^n$. Let $i \leq n$ be the first index such that $m[i] \neq \widetilde{m}[i]$, and let $\widetilde{\sigma} = \mathsf{Sign}(\mathrm{SK}, \widetilde{m})$, $\widetilde{\sigma^{\mathsf{res}}} = \mathsf{ResSign}(\mathrm{SK}\{m\}, \widetilde{m})$, where $\widetilde{\sigma} = \{(\widetilde{\mathrm{VK}_j}, \widetilde{\mathrm{VK}'_j}, \widetilde{\sigma_j})_{j \leq n}, \widetilde{\sigma^*}\}$ and $\widetilde{\sigma^{\mathsf{res}}} = \{(\widetilde{\mathrm{VK}^{\mathsf{res}}_j}, \widetilde{\mathrm{VK}'^{\mathsf{res}}_j}, \widetilde{\sigma^{\mathsf{res}}_j})_{j \leq n}, \widetilde{\sigma^{*\mathsf{res}}}\}$. We need to show that $\widetilde{\sigma} = \widetilde{\sigma^{\mathsf{res}}}$.

From the definition of $\mathsf{Pre.ResSign}$, it follows that for $j \leq i$, $(\widetilde{\mathrm{VK}^{\mathsf{res}}_j}, \widetilde{\mathrm{VK}'^{\mathsf{res}}_j}, \widetilde{\sigma^{\mathsf{res}}_j}) = (\widetilde{\mathrm{VK}_j}, \widetilde{\mathrm{VK}'_j}, \widetilde{\sigma_j})$ for all $j \leq i$. Similarly, from the definition of $\mathsf{Pre.Sign}$, it follows that $(\widetilde{\mathrm{VK}_j}, \widetilde{\mathrm{VK}'_j}, \widetilde{\sigma_j}) = (\widetilde{\mathrm{VK}_j}, \widetilde{\mathrm{VK}'_j}, \widetilde{\sigma_j})$ for all $j \leq i$ (this is because for $j < i, m^j = \widetilde{m}^j$, and for $j = i$, $(\mathrm{VK}_j, \mathrm{VK}'_j) = (\widetilde{\mathrm{VK}_j}, \widetilde{\mathrm{VK}_j})$).

Finally, for all $j > i$, the punctured PRF key K' can be used to compute the correct secret key/verification key pair, since $\widetilde{m}^j \neq m^j$ for all $j > i$. Therefore, the signature components for $j > i$ are same for both $\widetilde{\sigma}$ and $\widetilde{\sigma^{\mathsf{res}}}$. This concludes our correctness proof.

Security We prove security of this construction in the following theorem.

Theorem 4.1. *Assuming F is a selectively secure puncturable PRF and* $(\mathsf{Setup}_1, \mathsf{KeyGen}_1, \mathsf{Sign}_1, \mathsf{Verify}_1)$ *is a secure one time signature scheme, the prefix-restricted signature scheme described above is secure against forgeries as described in Definition 4.1.*

Proof. To prove this theorem, we will first define a sequence of hybrid experiments.

Hybrid Hyb_0 This is identical to the security game for the prefix-restricted signature scheme.

1. $\mathcal{A}$ sends a message m^* of length N.
2. The challenger chooses a puncturable PRF $K \leftarrow F.\mathsf{setup}(1^\lambda)$.
   Next, it computes $(\mathrm{SK}_\epsilon, \mathrm{VK}_\epsilon) = \mathsf{Setup}_1(1^\lambda; F(K, \epsilon))$.
3. It computes a signature σ for message m^*. Let $\mathrm{SK}_0 = \mathrm{SK}_\epsilon$. For $i = 1$ to n, do the following:
   (a) It computes the keys $(\mathrm{SK}_{m^{*i}}, \mathrm{VK}_{m^{*i}}) = \mathsf{Setup}_1(1^\lambda; F(K, m^{*i}))$, $(\mathrm{SK}_{\overline{m^{*i}}}, \mathrm{VK}_{\overline{m^{*i}}}) = \mathsf{Setup}_1(1^\lambda; F(K, \overline{m^*}^i))$.
   (b) Next, it computes $(\mathrm{VK}_i, \mathrm{VK}'_i) = \mathsf{switch}_{m^*[i]}(\mathrm{VK}_{m^{*i}}, \mathrm{VK}_{\overline{m^{*i}}})$ and $\sigma_i = \mathsf{Sign}_1(\mathrm{SK}_{m^{*(i-1)}}, (\mathrm{VK}_i, \mathrm{VK}'_i))$ for $1 \leq i \leq n$.
   (c) Finally, it signs m^* using $\mathrm{SK}_{m^{*n}}$, that is, it computes $\sigma^* = \mathsf{Sign}_1(\mathrm{SK}_{m^{*n}}, m^*)$. It sets $\sigma = \{(\mathrm{VK}_i, \mathrm{VK}'_i, \sigma_i)\}, \sigma^*\}$.
4. It computes a punctured key $K' \leftarrow F.\mathsf{puncture}(K, \{\{m^{*i}\}_{i \leq n} \cup \epsilon\})$ and sets $\mathrm{SK}\{m^*\} = \{\sigma, \{\mathrm{SK}_{\overline{m^{*i}}}\}_{i \leq n}, K'\}$.
5. Finally, the challenger sends $\mathrm{VK}_\epsilon, \mathrm{SK}\{m^*\}$ to $\mathcal{A}$.
6. $\mathcal{A}$ responds with a forgery $\widetilde{\sigma} = \{\{(\widetilde{\mathrm{VK}_i}, \widetilde{\mathrm{VK}'_i}, \widetilde{\sigma_i})\}, \widetilde{\sigma^*}\}$ and wins if
   (a) For all $1 \leq i \leq n$, $\mathsf{Verify}_1(\widetilde{\mathrm{VK}_{i-1}}, (\widetilde{\mathrm{VK}_i}, \widetilde{\mathrm{VK}'_i}), \widetilde{\sigma_i}) = 1$, where $\widetilde{\mathrm{VK}_0} = \mathrm{VK}_\epsilon$.
   (b) $\mathsf{Verify}_1(\widetilde{\mathrm{VK}_n}, m^*, \widetilde{\sigma^*}) = 1$.

Hybrid Hyb_1 In this experiment, the challenger chooses $(\mathrm{SK}_{m^{*i}}, \mathrm{VK}_{m^{*i}})$ using true randomness, instead of the pseudorandom string given by $F(K, m^{*i})$.

1. $\mathcal{A}$ sends a message m^* of length N.
2. The challenger chooses a puncturable PRF $K \leftarrow F.\mathsf{setup}(1^\lambda)$.
   Next, it computes $\underline{(\mathrm{SK}_\epsilon, \mathrm{VK}_\epsilon) = \mathsf{Setup}_1(1^\lambda)}$.
3. It computes a signature σ for message m^*. Let $\mathrm{SK}_0 \leftarrow \mathrm{SK}_\epsilon$. For $i = 1$ to n, do the following:
   (a) It computes the keys $\underline{(\mathrm{SK}_{m^{*i}}, \mathrm{VK}_{m^{*i}}) \leftarrow \mathsf{Setup}_1(1^\lambda)}$, $(\mathrm{SK}_{\overline{m^{*i}}}, \mathrm{VK}_{\overline{m^{*i}}}) = \mathsf{Setup}_1(1^\lambda; F(K, \overline{m^*}^i))$.
   (b) Next, it computes $(\mathrm{VK}_i, \mathrm{VK}'_i) = \mathsf{switch}_{m^*[i]}(\mathrm{VK}_{m^{*i}}, \mathrm{VK}_{\overline{m^{*i}}})$ and $\sigma_i = \mathsf{Sign}_1(\mathrm{SK}_{m^{*(i-1)}}, (\mathrm{VK}_i, \mathrm{VK}'_i))$ for $1 \leq i \leq n$.

(c) Finally, it signs m^* using $\mathrm{SK}_{m^{*n}}$, that is, it computes $\sigma^* = \mathsf{Sign}_1(\mathrm{SK}_{m^{*n}}, m^*)$. It sets $\sigma = \{(\mathrm{VK}_i, \mathrm{VK}'_i, \sigma_i)\}, \sigma^*\}$.

4. It computes a punctured key $K' \leftarrow F.\mathsf{puncture}(K, \{\{m^{*i}\}_{i \le n} \cup \epsilon\})$ and sets $\mathrm{SK}\{m^*\} = \{\sigma, \{\mathrm{SK}_{\overline{m^*}^i}\}_{i \le n}, K'\}$.
5. Finally, the challenger sends $\mathrm{VK}_\epsilon, \mathrm{SK}\{m^*\}$ to $\mathcal{A}$.
6. $\mathcal{A}$ responds with a forgery $\widetilde{\sigma} = \{\{(\widetilde{\mathrm{VK}_i}, \widetilde{\mathrm{VK}'_i}, \widetilde{\sigma_i})\}, \widetilde{\sigma^*}\}$ and wins if

   (a) For all $1 \le i \le n$, $\mathsf{Verify}_1(\widetilde{\mathrm{VK}_{i-1}}, (\widetilde{\mathrm{VK}_i}, \widetilde{\mathrm{VK}'_i}), \widetilde{\sigma_i}) = 1$, where $\widetilde{\mathrm{VK}_0} = \mathrm{VK}_\epsilon$.
   (b) $\mathsf{Verify}_1(\widetilde{\mathrm{VK}_n}, m^*, \widetilde{\sigma^*}) = 1$.

Hybrid Hyb_2 In the previous hybrid, the challenger sends VK_ϵ and n verification keys $\mathrm{VK}_{m^{*i}}$ for $1 \le i \le n$ as part of the signature σ. In the forgery, the adversary sends n tuples $(\widetilde{\mathrm{VK}_i}, \widetilde{\mathrm{VK}'_i}, \sigma_i)$. In this game, the challenger guesses the first i such that $\mathrm{VK}_{m^{*i}} \ne \widetilde{\mathrm{VK}_i}$. It chooses $i \leftarrow \{1, \ldots, n+1\}$, where $i = n+1$ indicates the guess that $\mathrm{VK}_{m^{*i}} = \widetilde{\mathrm{VK}_i}$ for all i. The attacker wins if its forgery verifies and this guess is correct.

1. $\mathcal{A}$ sends a message m^* of length N.
2. The challenger first chooses $i^* \leftarrow \{1, \ldots, n+1\}$.
3. It chooses a puncturable PRF $K \leftarrow F.\mathsf{setup}(1^\lambda)$.
   Next, it computes $(\mathrm{SK}_\epsilon, \mathrm{VK}_\epsilon) = \mathsf{Setup}_1(1^\lambda)$.
4. It computes a signature σ for message m^*. Let $\mathrm{SK}_0 \leftarrow \mathrm{SK}_\epsilon$. For $i = 1$ to n, do the following:

   (a) It computes the keys $(\mathrm{SK}_{m^{*i}}, \mathrm{VK}_{m^{*i}}) \leftarrow \mathsf{Setup}_1(1^\lambda)$, $(\mathrm{SK}_{\overline{m^*}^i}, \mathrm{VK}_{\overline{m^*}^i}) = \mathsf{Setup}_1(1^\lambda; F(K, \overline{m^*}^i))$.
   (b) Next, it computes $(\mathrm{VK}_i, \mathrm{VK}'_i) = \mathsf{switch}_{m^*[i]}(\mathrm{VK}_{m^{*i}}, \mathrm{VK}_{\overline{m^*}^i})$ and $\sigma_i = \mathsf{Sign}_1(\mathrm{SK}_{m^{*(i-1)}}, (\mathrm{VK}_i, \mathrm{VK}'_i))$ for $1 \le i \le n$.
   (c) Finally, it signs m^* using $\mathrm{SK}_{m^{*n}}$, that is, it computes $\sigma^* = \mathsf{Sign}_1(\mathrm{SK}_{m^{*n}}, m^*)$. It sets $\sigma = \{(\mathrm{VK}_i, \mathrm{VK}'_i, \sigma_i)\}, \sigma^*\}$.

5. It computes a punctured key $K' \leftarrow F.\mathsf{puncture}(K, \{\{m^{*i}\}_{i \le n} \cup \epsilon\})$ and sets $\mathrm{SK}\{m^*\} = \{\sigma, \{\mathrm{SK}_{\overline{m^*}^i}\}_{i \le n}, K'\}$.
6. Finally, the challenger sends $\mathrm{VK}_\epsilon, \mathrm{SK}\{m^*\}$ to $\mathcal{A}$.
7. $\mathcal{A}$ responds with a forgery $\widetilde{\sigma} = \{\{(\widetilde{\mathrm{VK}_i}, \widetilde{\mathrm{VK}'_i}, \widetilde{\sigma_i})\}, \widetilde{\sigma^*}\}$ and wins if

   (a) For all $i < i^*$, $\mathrm{VK}_{m^{*i}} = \widetilde{\mathrm{VK}_i}$ and $\mathrm{VK}_{m^{*i^*}} \ne \widetilde{\mathrm{VK}_{i^*}}$.
   (b) For all $1 \le i \le n$, $\mathsf{Verify}_1(\widetilde{\mathrm{VK}_{i-1}}, (\widetilde{\mathrm{VK}_i}, \widetilde{\mathrm{VK}'_i}), \widetilde{\sigma_i}) = 1$, where $\widetilde{\mathrm{VK}_0} = \mathrm{VK}_\epsilon$.
   (c) $\mathsf{Verify}_1(\widetilde{\mathrm{VK}_n}, m^*, \widetilde{\sigma^*}) = 1$.

Analysis We will now analyse the probability of an adversary's success in each of these hybrids. Let $\mathsf{Prob}^i_{\mathcal{A}}$ denote the probability of adversary $\mathcal{A}$ winning in hybrid Hyb_i.

Lemma 4.1. *Assuming F is a selectively secure puncturable pseudorandom function, for any PPT adversary $\mathcal{A}$, $|\mathsf{Prob}^0_{\mathcal{A}} - \mathsf{Prob}^1_{\mathcal{A}}| \le negl(\lambda)$.*

Proof. Suppose there exists a PPT adversary $\mathcal{A}$ such that $|\mathsf{Prob}^0_{\mathcal{A}} - \mathsf{Prob}^1_{\mathcal{A}}| = \gamma$. We will construct a PPT algorithm $\mathcal{B}$ that uses $\mathcal{A}$ to break the selective PPRF security of F. $\mathcal{B}$ works as follows.

1. $\mathcal{B}$ receives message m^* from $\mathcal{A}$. $\mathcal{B}$ then requests the PPRF challenger for a key punctured at the set $\{\{m^{*i}\}_{i \le n} \cup \epsilon\}$ along with the $n+1$ evaluations at $(\epsilon, m^{*1}, \ldots, m^{*n})$. It receives a punctured key K' and the $n+1$ strings $(y_0, \ldots, y_n)$, where y_i is either the PRF evaluation at m^{*i} or a uniformly random string.
2. Using K', it computes the PRF evaluations at $\overline{m^*}^i$ for all $i \le n$, that is, it sets $\overline{y}_i = F(K', \overline{m^*}^i)$.
3. $\mathcal{B}$ first computes $(\mathrm{SK}_\epsilon, \mathrm{VK}_\epsilon) = \mathsf{KeyGen}_1(1^\lambda; y_0)$.
4. It then computes, for $1 \le i \le n$, $(\mathrm{SK}_{m^{*i}}, \mathrm{VK}_{m^{*i}}) = \mathsf{KeyGen}_1(1^\lambda; y_i)$, $(\mathrm{SK}_{\overline{m^*}^i}, \mathrm{VK}_{\overline{m^*}^i}) = \mathsf{KeyGen}_1(1^\lambda; \overline{y}_i)$.
5. Next, it computes, for $1 \le i \le n$, $(\mathrm{VK}_i, \mathrm{VK}'_i) = \mathsf{switch}_{m^*[i]}(\mathrm{VK}_{m^{*i}}, \mathrm{VK}_{\overline{m^*}^i})$, $\sigma_i = \mathsf{Sign}_1(\mathrm{SK}_{m^{*i}}, (\mathrm{VK}_i, \mathrm{VK}'_i))$ and $\sigma^* = \mathsf{Sign}_1(\mathrm{SK}_{m^{*n}}, m)$. It sets $\sigma = \{(\mathrm{VK}_i, \mathrm{VK}'_i, \sigma_i)_{i \le n}, \sigma^*\}$.

6. $\mathcal{B}$ sets the restricted key $\mathrm{SK}\{m^*\} = \{\sigma, \{\mathrm{SK}_{\overline{m^{*i}}}\}_{i \le n}, K'\}$ and sends $\mathrm{SK}\{m^*\}, \mathrm{VK}_\epsilon$ to $\mathcal{A}$.
7. Finally, $\mathcal{A}$ sends a forgery. If the forgery verifies, $\mathcal{B}$ sends $b' = 0$, indicating the evaluations $y_0, \ldots, y_n$ were pseudorandom; else it sends $b' = 1$.

To analyse $\mathcal{B}$'s advantage in the PPRF security game, let b denote the bit chosen by challenger. Then $\Pr[b' = 1|b = 0] = \mathsf{Prob}^0_{\mathcal{A}}$ and $\Pr[b' = 1|b = 1] = \mathsf{Prob}^1_{\mathcal{A}}$. Therefore, if $|\mathsf{Prob}^0_{\mathcal{A}} - \mathsf{Prob}^1_{\mathcal{A}}|$ is non-negligible, then so is $\mathcal{B}$'s advantage in the PPRF security game.

■

Claim 4.1. *For any adversary $\mathcal{A}$, $\mathsf{Prob}^2_{\mathcal{A}} = \mathsf{Prob}^1_{\mathcal{A}}/(q + 1)$.*

Proof. This follows directly from the description of the hybrid experiments Hyb_1 and Hyb_2. The challenger's choice of i^* is independent of $\mathcal{A}$'s view. Therefore, $\Pr[\mathcal{A}$ wins in $\mathsf{Hyb}_2] = \Pr[i^*$ is correct guess$] \Pr[\mathcal{A}$ wins in $\mathsf{Hyb}_1]$.

■

Lemma 4.2. *Assuming $\mathcal{S}_1 = (\mathsf{KeyGen}_1, \mathsf{Sign}_1, \mathsf{Verify}_1)$ is a one-time secure deterministic signature scheme, $\mathsf{Prob}^2_{\mathcal{A}}$ is negligible in λ.*

Proof. We will construct an algorithm $\mathcal{B}$ that breaks the one-time security of $\mathcal{S}_1$ with probability $\mathsf{Prob}^2_{\mathcal{A}}$. $\mathcal{B}$ is defined as follows.

1. $\mathcal{B}$ chooses $i^* \leftarrow \{1, \ldots, q + 1\}$. It receives verification key VK^* from the $\mathcal{S}_1$ challenger.
2. $\mathcal{A}$ sends the challenge message m^*.
3. For all $i \ne (i^* - 1)$, it chooses $(\mathrm{SK}_{m^{*i}}, \mathrm{VK}_{m^{*i}}) \leftarrow \mathsf{KeyGen}_1(1^\lambda)$ and sets $\mathrm{VK}_{m^{*i^*-1}} = \mathrm{VK}^*$. It also computes $(\mathrm{SK}_{\overline{m^{*i}}}, \mathrm{VK}_{\overline{m^{*i}}}) = \mathsf{KeyGen}_1(1^\lambda; F(K, \overline{m^{*}}^{i}))$.
4. Next, it must compute signatures on the verification key pairs. For all $i \ne i^*$, it computes $\sigma_i = \mathsf{Sign}_1(\mathrm{SK}_{m^{*(i-1)}}, \mathsf{switch}_{m^*[i]}(\mathrm{VK}_{m^{*i}}, \mathrm{VK}_{\overline{m^{*i}}}))$. For $i = i^*$, if $i^* \ne n + 1$, it sends as signature query the tuple $\mathsf{switch}_{m^*[i^*]}(\mathrm{VK}_{m^{*i^*}}, \mathrm{VK}_{\overline{m^{*i^*}}})$ to the $\mathcal{S}_1$ challenger; if $i^* = n + 1$, it sends m as the signature query. It receives σ^* in response. Therefore, $\mathcal{B}$ can perfectly simulate the signature σ on m^*.
5. To compute the restricted signing key, it computes $K' \leftarrow F.\mathsf{puncture}(K, \{\{m^{*i}\} \cup \epsilon\})$. It has all the required signing keys $\mathrm{SK}_{\overline{m^{*i}}}$. Therefore, it sends VK_ϵ and $\mathrm{SK}\{m^*\} = \{\{\mathrm{SK}_{\overline{m^{*i}}}\}, K', \sigma\}$.
6. $\mathcal{A}$ finally sends a forgery. If $\mathcal{A}$ wins in Hyb_2, then it must send $(\widetilde{\mathrm{VK}_{i^*}}, \widetilde{\mathrm{VK}_{i^*}}) \ne (\mathrm{VK}_{m^{*i^*}}, \mathrm{VK}_{\overline{m^{*i^*}}})$ but $\mathsf{Verify}_1(\mathrm{VK}_{m^{*(i^*-1)}}, (\widetilde{\mathrm{VK}_{i^*}}, \widetilde{\mathrm{VK}_{i^*}})) = 1$. Therefore $\mathcal{B}$ sends $(\widetilde{\mathrm{VK}_{i^*}}, \widetilde{\mathrm{VK}_{i^*}})$ as forgery to $\mathcal{S}_1$ challenger, and wins with the same probability as $\mathcal{A}$.

■

■

5 Pseudorandom Puncturable Deterministic Encryption (PPDE)

In this section we describe another primitive, *puedorandom puncturable deterministic encryption schemes.* This is a variation of puncturable deterministic encryption as put forth by Waters[16].

In this scheme, there is a setup algorithm $\mathsf{PPDE.Setup}$ which generates a key K, as well as a deterministic encryption algorithm $\mathsf{PPDE.Enc}$ which takes the key K and message m. Since encryption is deterministic, the security property cannot by IND-CPA; instead we introduce a "puncturing algorithm" $\mathsf{PPDE.Puncture}$ which inputs a key K and message m and outputs a punctured key $K\{m\}$; the security property is that the encryption of m appears uniformly random to an adversary in possession of $K\{m\}$.

The actual construction uses techniques very similar to the "hidden trigger" mechanism using puncturable PRF's, as described in [15]; this is also used by [16].

5.1 Definition

Let $\mathcal{M}$ be the message space. A *pseudorandom puncturable deterministic encryption scheme* (or *PPDE scheme*) for $\mathcal{M}$ and ciphertext space $\mathcal{CT} \subseteq \{0,1\}^\ell$ (for some polynomial ℓ), is defined to be a collection of four algorithms.

- $\mathsf{PPDE.Setup}(1^\lambda)$ takes the security parameter and generates a key K in keyspace $\mathcal{K}$. This algorithm is randomized.
- $\mathsf{PPDE.Enc}(K, m)$ takes a key $K \in \mathcal{K}$ and message $m \in \mathcal{M}$ and produces a ciphertext $\mathrm{ct} \in \mathcal{CT}$. This algorithm is deterministic.
- $\mathsf{PPDE.Dec}(K, \mathrm{ct})$ takes a key $K \in \mathcal{K}$ and ciphertext $\mathrm{ct} \in \mathcal{CT}$ and outputs $m \in \mathcal{M} \cup \{\perp\}$. This algorithm is deterministic.
- $\mathsf{PPDE.Puncture}(K, m)$ takes a key $K \in \mathcal{K}$ and message $m \in \mathcal{M}$ and produces a *punctured key* $K\{m\} \in \mathcal{K}$ and $y \in \{0,1\}^\ell$. This algorithm may be randomized.

Correctness A PPDE scheme is correct if it satisfies the following conditions.

1. **Correct Decryption** For all messages m and keys $K \leftarrow \mathcal{K}$, we require
$$\mathsf{PPDE.Dec}(K, \mathsf{PPDE.Enc}(K, m)) = m.$$
2. **Correct Decryption Using Punctured Key** For all distinct messages m, for all keys $K \leftarrow \mathcal{K}$,
$$\Pr\left[\begin{array}{c} \#\{\mathrm{ct} : \mathsf{Decrypt}(K\{m\}, \mathrm{ct}) \neq \mathsf{Decrypt}(K, \mathrm{ct})\} > 1 \Big| \\ (K\{m\}, y) \leftarrow \mathsf{Puncture}(K, m) \end{array} \right]$$
is less than $\mathrm{negl}(\lambda)$, where all probabilities are taken over the coins of $\mathsf{PPDE.Puncture}$.
3. For all messages $m^* \in \mathcal{M}$ and keys $K \leftarrow \mathcal{K}$,
$$\left\{ y \;\middle|\; (K\{m^*\}, y) \leftarrow \mathsf{PPDE.Puncture}(K, m^*) \right\} \approx U_\ell$$
where U_ℓ denotes the uniform distribution over $\{0,1\}^\ell$.

Definition 5.1. *A PPDE scheme is* selectively secure *if no PPT algorithm $\mathcal{A}$ can determine the bit b in the following game except with probability negligibly close to $\frac{1}{2}$:*

1. *$\mathcal{A}$ chooses a message m^* to send to the challenger.*
2. *The challenger chooses $K \leftarrow \mathsf{PPDE.Setup}(1^\lambda)$ and computes $(K\{m^*\}, y) \leftarrow \mathsf{PPDE.Puncture}(K, m^*)$ and $\mathrm{ct} = \mathsf{PPDE.Enc}(K, m^*)$. Next, it chooses $b \leftarrow \{0,1\}$. If $b = 0$, it sends $(K\{m^*\}, (\mathrm{ct}, y))$; otherwise it sends $(K\{m^*\}, (y, \mathrm{ct}))$.*
3. *$\mathcal{A}$ outputs a guess b' for b.*

5.2 Construction

Next, we construct a secure PPDE scheme using a pair F_1, F_2 of selectively secure puncturable PRFs. Here $F_1 : \{0,1\}^m \to \{0,1\}^n$ and $F_2 : \{0,1\}^n \to \{0,1\}^m$, where m and n are polynomials in the security parameter λ. Additionally, we require F_1 to be statistically injective.

Our keyspace $\mathcal{K}$ will be the product of the keyspaces of F_1 and F_2; the message space $\mathcal{M} = \{0,1\}^m$ and ciphertext space is $\mathcal{CT} = \{0,1\}^{m+n}$.

Our algorithms are defined as follows:

- $\mathsf{PPDE.Setup}(1^\lambda)$ runs the setup algorithms for F_1 and F_2 to obtain keys K_1, K_2 respectively. It outputs $K = (K_1, K_2)$.
- $\mathsf{PPDE.Enc}((K_1, K_2), m)$ computes $A = F_1(K_1, m)$ and outputs
$$\mathrm{ct} = (A, F_2(K_2, A) \oplus m)$$
- $\mathsf{PPDE.Dec}((K_1, K_2), (\mathrm{ct}_1, \mathrm{ct}_2))$ computes the message $m = F_2(K_2, \mathrm{ct}_1) \oplus \mathrm{ct}_2$. It then checks that $F_1(K_1, m) = \mathrm{ct}_1$; if so it outputs m, otherwise it outputs $\perp$.
- $\mathsf{PPDE.Puncture}((K_1, K_2), m)$ chooses $y = (y_1, y_2) \in \mathcal{CT}$ uniformly randomly. It computes $A = F_1(K_1, m)$, then punctures K_1 at m to obtain $K_1\{m\}$ and K_2 at $\{A, y_1\}$ to produce $K_2\{A, y_1\}$. It outputs
$$K\{m\} = (K_1\{m\}, K_2\{A, y_1\}), y = (y_1, y_2).$$

Correctness We observe that as long as F_1 is injective (which occurs except with negligible probability in the coins of $\mathsf{PPDE.Setup}$), decryption will be correct on all inputs using the punctured key. Here "correct" means: identical to the behavior at the punctured key on all points except the encryption of the punctured message, where the output is changed to $\perp$. (If F_1 were not injective, the puncturing of K_2 at the output of F_1 may cause other PRF outputs to be changed to $\perp$, violating the requirement that the set of changed outputs have size at most 1.)

Correctness of decryption using non-punctured keys is immediate.

Security We argue security through a series of hybrids.

Theorem 5.1. *Suppose that no PPT adversary has advantage greater than ϵ_1 in the selective security game against F_1 or greater than ϵ_2 in the selective security game against F_2. Then no PPT adversary has advantage greater than $\epsilon_1 + \epsilon_2$ in the selective security game as defined in Definition 5.1.*

Proof. Let $\mathcal{A}$ be an arbitrary PPT adversary. We start by defining a sequence of hybrids.

Hyb_0 This hybrid is identical to the original security game with $b = 0$.

1. $\mathcal{A}$ chooses a message m^* to send to the challenger.
2. The challenger produces $(K_1, K_2) = \mathsf{PPDE.Setup}(1^\lambda)$. He computes the punctured key $(K\{m^*\}, (y_1, y_2)) \leftarrow \mathsf{PPDE.Puncture}((K_1, K_2), m^*)$ and sends $K\{m^*\}$ to $\mathcal{A}$. He also computes $A = F_1(K_1, m^*)$ and sends $\mathrm{ct} = (A, F_2(K_2, A) \oplus m^*)$.

Hyb_1 This hybrid is same as the previous one, except that A is replaced by y_1.

1. $\mathcal{A}$ chooses a message m^* to send to the challenger.
2. The challenger produces $(K_1, K_2) = \mathsf{PPDE.Setup}(1^\lambda)$. He computes the punctured key $(K\{m^*\}, (y_1, y_2)) \leftarrow \mathsf{PPDE.Puncture}((K_1, K_2), m^*)$ and sends $K\{m^*\}$ to $\mathcal{A}$.
He sends $\underline{\mathrm{ct} = (y_1, F_2(K_2, y_1) \oplus m^*)}$ as the ciphertext.

Hyb_2 This hybrid is the same as the previous one, except that $F_2(K_2, A)$ is replaced by y_2. The ciphertext is now $(y_1, y_2 \oplus m^*)$.

1. $\mathcal{A}$ chooses a message m^* to send to the challenger.
2. The challenger produces $(K_1, K_2) = \mathsf{PPDE.Setup}(1^\lambda)$. He computes the punctured key $(K\{m^*\}, (y_1, y_2)) \leftarrow \mathsf{PPDE.Puncture}((K_1, K_2), m^*)$ and sends $K\{m^*\}$ to $\mathcal{A}$.
He sends $\underline{\mathrm{ct} = (y_1, y_2 \oplus m^*)}$ as the ciphertext.

We see that Hyb_2 is the original security game with $b = 1$, except for the presence of $y_2 \oplus m^*$ in place of y_2, which does not affect an attacker's advantage. We need only now to argue that these hybrids are indistinguishable.

Hyb_0 to Hyb_1 We claim that an attacker $\mathcal{A}$ which can distinguish between Hyb_0 and Hyb_1 with advantage ϵ can be used by a simulator $\mathcal{B}$ to win the selective security game against F_1 with advantage ϵ.

$\mathcal{B}$ acts as follows:

1. $\mathcal{A}$ sends a message m^* to $\mathcal{B}$, who gives it to the PRF challenger. The challenger replies with a punctured key $K_1(m^*)$ and a challenge pair (x_1, x_2) consisting of $F_1(K_1, m^*)$ and a uniformly random element.

2. $\mathcal{B}$ computes $K_2 = \mathsf{Setup}_{F_2}(1^\lambda)$ and $K_2(x_1, x_2) = \mathsf{Puncture}_{F_2}(K_2, \{x_1, x_2\})$. He sets $K(m^*) = (K_1(m^*), K_2(x_1, x_2))$, $\mathrm{ct} = (x_1, F_2(K_2, x_1))$, and sends these to $\mathcal{A}$.

3. $\mathcal{A}$ outputs a guess b that he is in Hyb_b.

We see that if $\mathcal{A}$ is in Hyb_0, this is exactly the case that the PRF challenger set $x_1 = F_1(K_1, m^*)$; Hyb_1 is the case when $x_2 = F_1(K_1, m^*)$. Thus $\mathcal{A}$'s guess can be translated into a guess for which of $\{x_1, x_2\}$ is equal to $F_1(K_1, m^*)$ which is correct exactly when $\mathcal{A}$ is, so that $\mathcal{A}$'s advantage can be at most ϵ_{F_1}.

Hyb_1 to Hyb_2 We claim that an attacker $\mathcal{A}$ which can distinguish between Hyb_1 and Hyb_2 with advantage ϵ can be used by a simulator $\mathcal{B}$ to win the selective security game against F_2 with advantage ϵ.

$\mathcal{B}$ acts as follows:

1. $\mathcal{A}$ sends a message m^* to $\mathcal{B}$. $\mathcal{B}$ computes $K_1 = \mathsf{Setup}_{F_1}(1^\lambda)$ and chooses (y_1, y_2) uniformly at random. It computes $A = F_1(K_1, m^*)$ and submits $\{y_1, A\}$ to the challenger as his selective challenge.

2. The challenger replies with a punctured key $K_2(A, y_1)$ and a pair (x_1, x_2) consisting of both $F_2(K_2, A)$ and a uniformly random element. (In fact, the challenger also provides a pair consisting of $F_2(K_2, y_1)$, but we do not need this and ignore it.)

3. $\mathcal{B}$ sets $K(m^*) = (K_1(m^*), K_2(A, y_1))$ and sends this to $\mathcal{A}$. He also sends $\mathrm{ct} = (A, x_1 \oplus m^*)$.

4. $\mathcal{A}$ outputs a guess b that he is in Hyb_{b+1}.

We see that if $\mathcal{A}$ is in Hyb_1, this is exactly the case that the PRF challenger set $x_1 = F_2(K_2, A)$; Hyb_2 is exactly the case that the challenger set $x_2 = F_2(K_2, A)$. We conclude that $\mathcal{A}$'s advantage can be at most ϵ_{F_2}.

Conclusion Summing the attacker's maximum advantage in distinguishing the hybrids and winning in the game of Hyb_2, we see that the maximum advantage in the selective security game for the PPDE scheme is $\epsilon_{F_1} + \epsilon_{F_2}$. ■

6 Signed Universal Samplers

In this section, we will describe our construction for a signed universal sampler scheme. We will show that it is both simulation secure (as per Definitions 3.2) and secure against forgeries (as per Definition 3.1).

A remarkable feature of our scheme is its simplicity. The sampler setup algorithm will first generate a prefix restricted signature scheme verification and signing key pair. Next the universal sampler parameters are created as the obfuscation of a program that takes two inputs x, d and outputs $p = d(r)$, where r is computed using a puncturable PRF on input $x||d$. The program also outputs a signature σ (using the signing key) on $(x||d, p)$ using a prefix-restricted signature scheme. The sampler parameters, U, are the obfuscated program and the verification key VK of the universal sampler is the verification key of the prefix restricted signature.

To sample from a distribution d, one computes $x = H(d)$ and runs the sampler output on inputs x, d. Finally, the verification algorithm is used to check that p was the correct output sample for a circuit d when given a prefix restricted signature σ. The verification algorithm first computes $x = H(d)$. Then, it simply checks that the signature σ verifies on the message $m = (m_1, m_2) = (x||d, p)$.

Our Construction Let ($\mathsf{Pre.Setup}$, $\mathsf{Pre.Sign}$, $\mathsf{Pre.Verify}$, $\mathsf{Restrict}$, $\mathsf{ResSign}$) be a restructed-prefix signature scheme, F a puncturable PRF with algorithms $F.\mathsf{setup}$, $F.\mathsf{puncture}$ and $F.\mathsf{eval}$, PPDE = ($\mathsf{PPDE.Setup}$, $\mathsf{PPDE.Enc}$, $\mathsf{PPDE.Dec}$, $\mathsf{PPDE.Puncture}$) a puncturable deterministic encryption scheme with pseudorandom ciphertexts.

Our (ℓ_{ckt}, ℓ_{rnd}, ℓ_{out})-signed universal sampler scheme consists of the following algorithms.

- $\mathsf{Setup}(1^\lambda)$ The setup algorithm first chooses a signing and verification key for the restricted-prefix signature scheme; it computes $(\text{SK}_{\text{pre}}, \text{VK}_{\text{pre}}) \leftarrow \mathsf{Pre.Setup}(1^\lambda)$. Next, it chooses a puncturable PRF key $K_F \leftarrow F.\mathsf{setup}(1^\lambda)$ and sets U to be an obfuscation of the program $\mathsf{USampler}$[4] defined in Figure 1; that is, $U \leftarrow i\mathcal{O}(\mathsf{USampler})$ and $\text{VK} = \text{VK}_{\text{pre}}$. It outputs (U, VK).

```
USampler

Inputs $x \in \{0,1\}^{\ell_1}$, $d \in \{0,1\}^{\ell_{\text{ckt}}}$.

Constants Puncturable PRF key $K_F$, prefix-restricted signing key $\text{SK}_{\text{pre}}$.

  Compute $r = F(K, (x||d))$.
  Compute $\mathsf{out} = d(r)$.
  Compute $\sigma = \mathsf{Pre.Sign}(\text{SK}_{\text{pre}}, (x||d, \mathsf{out}))$.
  Output $(\mathsf{out}, \sigma)$.
```

Figure 1: Program $\mathsf{USampler}$

- $\mathsf{Sample}(U, d)$ The sample generation algorithm computes $x = H(d)$ and $(p_d, \sigma) = U(x, d)$. It outputs (p_d, σ).

- $\mathsf{Verify}(\text{VK}, d, p_d, \sigma)$ The verification algorithm computes $x = H(d)$ and then outputs $\mathsf{Pre.Verify}(\text{VK}, (x||d, p_d), \sigma)$.

6.1 Proof of Unforgeability

We will define a sequence of hybrids to show that the construction satisfies the adaptive unforgeability definition.

Without loss of generality, let us assume the adversary $\mathcal{A}$ makes q unique random oracle queries before submitting the forgery corresponding to one of the queries.

Proof Intuition This proof is fairly straightforward. The challenger first guesses the random oracle query which corresponds to the forgery. Let this query be d^*. The challenger then modifies the obfuscated program $\mathsf{USampler}$ to use a restricted signing key. Once the program has a restricted signing key, we can use the security of our special signature scheme to argue that the adversary cannot forge a signature corresponding to d^*.

Hybrid Hyb_0 Hyb_0 is the real security game between an adversary $\mathcal{A}$ and challenger.

1. Challenger computes universal samplers. It chooses $K_F \leftarrow F.\mathsf{setup}(1^\lambda)$, $(\text{SK}_{\text{pre}}, \text{VK}_{\text{pre}}) \leftarrow \mathsf{Pre.Setup}(1^\lambda)$ and computes $U \leftarrow i\mathcal{O}(\mathsf{USampler}\{K_F, \text{SK}_{\text{pre}}\})$.
   It sends $(U, \text{VK}_{\text{pre}})$ to $\mathcal{A}$.

[4]Padded to be of the same size as the corresponding programs in the proof.

2. $\mathcal{A}$ sends q random oracle queries. For i^{th} query d_i, the challenger chooses uniformly random strings $x_i \leftarrow \{0,1\}^{\ell_1}$, sets $H_1(d_i) = x_i$; it sends $H_1(d_i)$ to $\mathcal{A}$.
3. $\mathcal{A}$ finally sends the forgery (d^*, p^*, σ^*) and wins if
   (a) $d^* = d_i$ for some $i \in [q]$,
   (b) $\mathsf{Sample}(U, d^*)_1 \neq p^*$; that is, $x^* = H_1(d^*)$, $(\mathsf{out}, \sigma) = U(x^*, d^*)$ and $\mathsf{out} \neq p^*$,
   (c) $\mathsf{Verify}(\mathrm{VK}_{\mathrm{pre}}, (x^* || d^*, p^*), \sigma^*) = 1$.

Hybrid Hyb_1 In this experiment, the challenger guesses the random oracle query which will correspond to the forgery. If this guess is incorrect, the challenger aborts.

1. Challenger first chooses $i^* \leftarrow [q]$.
2. Challenger computes universal samplers. It chooses $K_F \leftarrow F.\mathsf{setup}(1^\lambda)$, $(\mathrm{SK}_{\mathrm{pre}}, \mathrm{VK}_{\mathrm{pre}}) \leftarrow \mathsf{Pre.Setup}(1^\lambda)$ and computes $U \leftarrow i\mathcal{O}(\mathsf{USampler}\{K_F, \mathrm{SK}_{\mathrm{pre}}\})$.
   It sends $(U, \mathrm{VK}_{\mathrm{pre}})$ to $\mathcal{A}$.
3. $\mathcal{A}$ sends q random oracle queries. For i^{th} query d_i, the challenger chooses uniformly random strings $x_i \leftarrow \{0,1\}^{\ell_1}$, sets $H_1(d_i) = x_i$; it sends $H_1(d_i)$ to $\mathcal{A}$.
4. $\mathcal{A}$ finally sends the forgery (d^*, p^*, σ^*) and wins if
   (a) $d^* = d_i$,
   (b) $\mathsf{Sample}(U, d^*)_1 \neq p^*$; that is, $x^* = H_1(d^*)$, $(\mathsf{out}, \sigma) = U(x^*, d^*)$ and $\mathsf{out} \neq p^*$,
   (c) $\mathsf{Verify}(\mathrm{VK}_{\mathrm{pre}}, (x^* || d^*, p^*), \sigma^*) = 1$.

Hybrid Hyb_2 In this experiment, the challenger guesses the circuit sent as the $(i^*)^{th}$ random oracle query. If this guess is incorrect, the challenger aborts.

1. Challenger first chooses $i^* \leftarrow [q]$.
2. Challenger chooses $d' \leftarrow \{0,1\}^{\ell_{\mathrm{ckt}}}$, $x' \leftarrow \{0,1\}^{\ell_1}$ and sets $H_1(d') = x'$.
3. Challenger computes universal samplers. It chooses $K_F \leftarrow F.\mathsf{setup}(1^\lambda)$, $(\mathrm{SK}_{\mathrm{pre}}, \mathrm{VK}_{\mathrm{pre}}) \leftarrow \mathsf{Pre.Setup}(1^\lambda)$ and computes $U \leftarrow i\mathcal{O}(\mathsf{USampler}\{K_F, \mathrm{SK}_{\mathrm{pre}}\})$.
   It sends $(U, \mathrm{VK}_{\mathrm{pre}})$ to $\mathcal{A}$.
4. $\mathcal{A}$ sends q random oracle queries. For i^{th} query d_i, if $i \neq i^*$, the challenger chooses uniformly random strings $x_i \leftarrow \{0,1\}^{\ell_1}$, sets $H_1(d_i) = x_i$; it sends $H_1(d_i)$ to $\mathcal{A}$.
   If $i = i^*$ and $d_i = d'$, it sends x' to $\mathcal{A}$, else it aborts.
5. $\mathcal{A}$ finally sends the forgery (d^*, p^*, σ^*) and wins if
   (a) $d^* = d'$,
   (b) $(\mathsf{out}, \sigma) = U(x', d')$ and $\mathsf{out} \neq p^*$,
   (c) $\mathsf{Verify}(\mathrm{VK}_{\mathrm{pre}}, (x' || d', p^*), \sigma^*) = 1$.

Hybrid Hyb_3 In this experiment, the challenger outputs the obfuscation of $\mathsf{USampler}'$ (defined in 2) instead of $\mathsf{USampler}$. The only difference between $\mathsf{USampler}$ and $\mathsf{USampler}'$ is that $\mathsf{USampler}'$ uses a restricted signing key.

1. Challenger first chooses $i^* \leftarrow [q]$.
2. Challenger chooses $d' \leftarrow \{0,1\}^{\ell_{\mathrm{ckt}}}$, $x' \leftarrow \{0,1\}^{\ell_1}$ and sets $H_1(d') = x'$.
3. Challenger computes universal samplers. It chooses $K_F \leftarrow F.\mathsf{setup}(1^\lambda)$, $(\mathrm{SK}_{\mathrm{pre}}, \mathrm{VK}_{\mathrm{pre}}) \leftarrow \mathsf{Pre.Setup}(1^\lambda)$.
   It computes $r' = F(K_F, x' || d')$, $\mathsf{out}' = d(r')$.
   It computes $\mathrm{SK}\{(x' || d', \mathsf{out}')\} \leftarrow \mathsf{Restrict}(\mathrm{SK}_{\mathrm{pre}}, (x' || d', \mathsf{out}'))$.
   It sets $U \leftarrow i\mathcal{O}(\mathsf{USampler}'\{K_F, \mathrm{SK}\{x' || d', \mathsf{out}'\}\})$.
   It sends $(U, \mathrm{VK}_{\mathrm{pre}})$ to $\mathcal{A}$.

4. $\mathcal{A}$ sends q random oracle queries. For i^{th} query d_i, if $i \neq i^*$, the challenger chooses uniformly random strings $x_i \leftarrow \{0,1\}^{\ell_1}$, sets $H_1(d_i) = x_i$; it sends $H_1(d_i)$ to $\mathcal{A}$.
   If $i = i^*$ and $d_i = d'$, it sends x' to $\mathcal{A}$, else it aborts.
5. $\mathcal{A}$ finally sends the forgery (d^*, p^*, σ^*) and wins if
   (a) $d^* = d'$,
   (b) $(\mathsf{out}, \sigma) = U(x', d')$ and $\mathsf{out} \neq p^*$,
   (c) $\mathsf{Verify}(\mathrm{VK}_{\mathrm{pre}}, (x'||d', p^*), \sigma^*) = 1$.

```
                                USampler'

Inputs $x \in \{0,1\}^{\ell_1}$, $d \in \{0,1\}^{\ell_{\mathrm{ckt}}}$.

Constants Puncturable PRF key $K_F$, prefix-restricted signing key $\underline{\mathrm{SK}\{(x'||d', \mathsf{out}')\}}$.

  Compute $r = F(K, (x||d))$.
  Compute $\mathsf{out} = d(r)$.
  Compute $\sigma = \mathsf{ResSign}(\mathrm{SK}\{(x'||d', \mathsf{out}')\}, (x||d, \mathsf{out}))$.
  Output $(\mathsf{out}, \sigma)$.
```

Figure 2: Program $\mathsf{USampler}'$

Next, we need to analyse the adversary's advantage in each of these games. This analysis is included in Appendix A.

6.2 Proof of Simulation Security

Let us assume the adversary $\mathcal{A}$ queries the random oracle by sending a message (RO, d) before sending a message (params, d). Without loss of generality, let q be the number of queries made by $\mathcal{A}$. We will define a sequence of hybrid experiments, and then show that any PPT adversary cannot distinguish between the hybrid experiments with advantage non-negligible in the security parameter λ.

Proof Intuition First, we give a high level intuition of our proof strategy. The main idea is to gradually change the random oracle query responses from uniformly random strings to more structured strings which will allow simulation. First, the challenger modifies the program $\mathsf{USampler}$ in order to allow trapdoors. The program, instead of computing $r = F(K_F, x||d)$ and $p = d(r)$, first decrypts the string x. It also has a string α hardwired. If the decryption is successful, and the output message is $(\tilde{d}, a, m)$ where $d = \tilde{d}$, $\mathrm{PRG}(a) = \alpha$, then the program simply outputs m as the sampled parameter. Due to the security of PRG, we can argue that the adversary cannot notice the difference. Now, the challenger can modify the random oracle queries. For a query corresponding to circuit d, the challenger outputs an encryption of $(d, a, d(t))$ where t is a uniformly random string. This looks like a uniformly random string due to the property of PPDE ciphertexts. However, note that the obfuscated program has the decryption key hardwired. Using the techniques from punctured programming, we show how to transform the random oracle responses from truly random strings to PPDE encryptions.

Experiment Expt_0 This experiment corresponds to the real world. The challenger runs the universal sampler setup honestly to compute U, and sends it to the adversary $\mathcal{A}$. Next, for each random oracle query, it outputs a uniformly random string.

1. Challenger computes universal samplers. It chooses $K_F \leftarrow F.\mathsf{setup}(1^\lambda)$, $(\mathrm{SK}_{\mathrm{pre}}, \mathrm{VK}_{\mathrm{pre}}) \leftarrow \mathsf{Pre.Setup}(1^\lambda)$.
   It computes $U \leftarrow i\mathcal{O}(\mathsf{USampler}\{K_F, \mathrm{SK}_{\mathrm{pre}}\})$.
   It sends $(U, \mathrm{VK}_{\mathrm{pre}})$ to $\mathcal{A}$.
2. $\mathcal{A}$ sends q random oracle queries. For j^{th} query d_j,

- The challenger chooses uniformly random strings $x_j \leftarrow \{0,1\}^{\ell_1}$, sets $H_1(d_j) = x_j$; it sends $H_1(d_j)$ to $\mathcal{A}$.

3. $\mathcal{A}$ finally sends a bit b.

The output of this experiment is b.

Experiment Expt_1 In this experiment, the challenger outputs an obfuscation of $\mathsf{USampler\text{-}1}$ (defined in Figure 3) as the universal sampler program output during setup. This new program has a PPDE key hardwired, and it uses this key to decrypt the input string. If the decryption is successful (and some additional checks are satisfied), the program outputs the decrypted string. Else, its output is the same as in previous experiment.

1. Challenger computes universal samplers. It chooses $K_F \leftarrow F.\mathsf{setup}(1^\lambda)$, $(\mathrm{SK}_{\mathrm{pre}}, \mathrm{VK}_{\mathrm{pre}}) \leftarrow \mathsf{Pre.Setup}(1^\lambda)$. It chooses K_{PPDE} and $\alpha \leftarrow \{0,1\}^{2\lambda}$.
   It computes $U \leftarrow i\mathcal{O}(\mathsf{USampler\text{-}1}\{K_F, \mathrm{SK}_{\mathrm{pre}}, K_{\mathrm{PPDE}}, \alpha\})$ and sends $(U, \mathrm{VK}_{\mathrm{pre}})$ to $\mathcal{A}$.
2. $\mathcal{A}$ sends q random oracle queries. For j^{th} query d_j,
   - The challenger chooses uniformly random strings $x_j \leftarrow \{0,1\}^{\ell_1}$, sets $H_1(d_j) = x_j$; it sends $H_1(d_j)$ to $\mathcal{A}$.
3. $\mathcal{A}$ finally sends a bit b.

```
                                USampler-1

Inputs $x \in \{0,1\}^{\ell_1}$, $d \in \{0,1\}^{\ell_{\mathrm{ckt}}}$.

Constants Puncturable PRF key $K_F$, PPDE key $K_{\mathrm{PPDE}}$, $\alpha \in \{0,1\}^{2\lambda}$, prefix-restricted signing key $\mathrm{SK}_{\mathrm{pre}}$.

  Compute $m = \mathsf{PPDE.Dec}(K_{\mathrm{PPDE}}, x)$. If $m \neq \perp$, let $m = (\tilde{d}, a, y) \in \{0,1\}^\lambda \times \{0,1\}^{\ell_{\mathrm{out}}}$.
  if $m \neq \perp$ and $\tilde{d} = d$ and $\alpha = \mathrm{PRG}(a)$ then
      Set $\mathsf{out} = y$.
  else
      Compute $r = F(K, (x||d))$.
      Compute $\mathsf{out} = d(r)$.
  end if
  Compute $\sigma = \mathsf{Pre.Sign}(\mathrm{SK}_{\mathrm{pre}}, (x||d, \mathsf{out}))$.
  Output $(\mathsf{out}, \sigma)$.
```

Figure 3: Program $\mathsf{USampler\text{-}1}$

Experiment Expt_2 In this experiment, the string α hardwired in the program is a pseudorandom string, computed using PRG.

1. Challenger computes universal samplers. It chooses $K_F \leftarrow F.\mathsf{setup}(1^\lambda)$, $(\mathrm{SK}_{\mathrm{pre}}, \mathrm{VK}_{\mathrm{pre}}) \leftarrow \mathsf{Pre.Setup}(1^\lambda)$. It chooses a puncturable PPDE key K_{PPDE},
   $a \leftarrow \{0,1\}^\lambda$ and sets $\alpha = \mathrm{PRG}(a)$.
   It computes $U \leftarrow i\mathcal{O}(\mathsf{USampler\text{-}1}\{K_F, \mathrm{SK}_{\mathrm{pre}}, K_{\mathrm{PPDE}}, \alpha\})$ and sends $(U, \mathrm{VK}_{\mathrm{pre}})$ to $\mathcal{A}$.
2. $\mathcal{A}$ sends q random oracle queries. For j^{th} query d_j,
   - The challenger chooses uniformly random strings $x_j \leftarrow \{0,1\}^{\ell_1}$, sets $H_1(d_j) = x_j$; it sends $H_1(d_j)$ to $\mathcal{A}$.
3. $\mathcal{A}$ finally sends a bit b.

The output of this experiment is b

Next, we will have q hybrid experiments $\mathsf{Expt}_{2,i}$ for $0 \le i \le q$. In each hybrid, the challenger changes the response to the random oracle queries. Instead of sending uniformly random strings, it sends encryptions computed using $\mathsf{PPDE.Enc}(\cdot,\cdot)$.

Experiment $\mathsf{Expt}_{2,i}$ In this experiment, the challenger queries the Parameters Oracle to compute the response for the first i random oracle queries. For the remaining queries, it outputs a uniformly random string.

1. Challenger computes universal samplers. It chooses $K_F \leftarrow F.\mathsf{setup}(1^\lambda)$, $(\mathrm{SK}_{\mathrm{pre}}, \mathrm{VK}_{\mathrm{pre}}) \leftarrow \mathsf{Pre.Setup}(1^\lambda)$.
   It chooses a puncturable PPDE key K_{PPDE}, $a \leftarrow \{0,1\}^\lambda$ and sets $\alpha = \mathrm{PRG}(a)$.
   It computes $U \leftarrow i\mathcal{O}(\mathsf{USampler\text{-}1}\{K_F, \mathrm{SK}_{\mathrm{pre}}, K_{\mathrm{PPDE}}, \alpha\})$ and sends $(U, \mathrm{VK}_{\mathrm{pre}})$ to $\mathcal{A}$.
2. $\mathcal{A}$ sends q random oracle queries. For j^{th} query d_j,
   - $\underline{\text{if } j \le i\text{, the challenger queries the Parameter Oracle.}}$
     $\underline{\text{On input } d_j\text{, it receives } p_j \text{ in response.}}$
     $\underline{\text{It sets } H_1(d_j) = \mathsf{PPDE.Enc}(K_{\mathrm{PPDE}}, p_j)}$ and sends $H_1(d_j)$ to $\mathcal{A}$.
   - $\underline{\text{if } j > i}$, the challenger chooses uniformly random strings $x_j \leftarrow \{0,1\}^{\ell_1}$, sets $H_1(d_j) = x_j$; it sends $H_1(d_j)$ to $\mathcal{A}$.
3. $\mathcal{A}$ finally sends a bit b.

The output of this experiment is b.

Clearly, $\mathsf{Expt}_{2,0}$ is identical to experiment Expt_2, while $\mathsf{Expt}_{2,q}$ corresponds to the ideal world. We now need to show that any PPT adversary has almost identical advantage in each of the experiments described above. Due to space constraints, the full analysis is included in the appendix. Here, we give an outline of the proof.

In the first hybrid, the challenger replaces the program $\mathsf{USampler}$ with program $\mathsf{USampler\text{-}1}$. The only difference between these two programs is that $\mathsf{USampler\text{-}1}$ first decrypts the input x using PPDE key. If the decryption is successful and can be parsed as $(\tilde{d}, a, m)$, then the program checks if $d = \tilde{d}$ and $\mathrm{PRG}(a) = \alpha$, where α is a uniformly random string. As a result, this step is never executed, and hence the two programs are identical. Therefore, using security of $i\mathcal{O}$, the hybrids are computationally indistinguishable.

Next, the challenger replaces α with a pseudorandom string. It chooses a string a and sets $\alpha = \mathrm{PRG}(a)$. This step is indistinguishable due to the security of PRG.

Now, the first step of the program is "Decrypt x. If decryption is successful, and outputs $(\tilde{d}, a, m)$ and $d = \tilde{d}$ and $\mathrm{PRG}(a) = \alpha$, then output m". This gives the challenger a 'trapdoor'. Now, the adversary sends encryption of $(d, a, d(t))$ as the response for $\mathsf{RO}(d)$. To prove that the adversary cannot distinguish between the encryptions and random strings, we define q hybrids. In the i^{th} hybrid, the first i responses are encryptions, while the remaining are random strings. We now need to show that the i^{th} and $(i+1)^{th}$ hybrids are indistinguishable. For this, the main idea is to first puncture the PPDE key, and then switch the random RO responses to ciphertexts. However, to puncture the PPDE key, we will need to know the 'puncture point' in advance, resulting in a subexponential security loss. Here, note that the security loss is $q \cdot 2^{\ell_{\mathrm{ckt}}}$, not $2^{q\ell_{\mathrm{ckt}}}$. This allows us to use complexity leveraging with subexponential security for $i\mathcal{O}$, PRG and F.

References

[1] Ananth, P.V., Gupta, D., Ishai, Y., Sahai, A.: Optimizing obfuscation: Avoiding barrington's theorem. In: Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security, Scottsdale, AZ, USA, November 3-7, 2014. pp. 646–658 (2014)

[2] Applebaum, B., Brakerski, Z.: Obfuscating circuits via composite-order graded encoding. In: Theory of Cryptography - 12th Theory of Cryptography Conference, TCC 2015, Warsaw, Poland, March 23-25, 2015, Proceedings, Part II. pp. 528–556 (2015)

[3] Bellare, M., Rogaway, P.: Random oracles are practical: A paradigm for designing efficient protocols. In: ACM Conference on Computer and Communications Security. pp. 62–73 (1993)

[4] Boneh, D., Waters, B.: Constrained pseudorandom functions and their applications. In: ASIACRYPT. pp. 280–300 (2013)

[5] Boneh, D., Zhandry, M.: Multiparty key exchange, efficient traitor tracing, and more from indistinguishability obfuscation. In: CRYPTO (2014)

[6] Boyle, E., Goldwasser, S., Ivan, I.: Functional signatures and pseudorandom functions. In: Public-Key Cryptography - PKC 2014 - 17th International Conference on Practice and Theory in Public-Key Cryptography, Buenos Aires, Argentina, March 26-28, 2014. Proceedings. pp. 501–519 (2014)

[7] Garg, S., Gentry, C., Halevi, S., Raykova, M., Sahai, A., Waters, B.: Candidate indistinguishability obfuscation and functional encryption for all circuits. In: FOCS (2013)

[8] Goldreich, O.: Two remarks concerning the goldwasser-micali-rivest signature scheme. In: Advances in Cryptology - CRYPTO '86, Santa Barbara, California, USA, 1986, Proceedings. pp. 104–110 (1986)

[9] Hofheinz, D., Jager, T., Khurana, D., Sahai, A., Waters, B., Zhandry, M.: How to generate and use universal parameters. In: ASIACRYPT (2016)

[10] Hofheinz, D., Kamath, A., Koppula, V., Waters, B.: Adaptively secure constrained pseudorandom functions. Cryptology ePrint Archive, Report 2014/720 (2014), `http://eprint.iacr.org/`

[11] Hohenberger, S., Koppula, V., Waters, B.: Universal signature aggregators. In: Advances in Cryptology - EUROCRYPT 2015 - 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part II. pp. 3–34 (2015)

[12] Huang, Y., Katz, J., Evans, D.: Efficient secure two-party computation using symmetric cut-and-choose. In: Advances in Cryptology - CRYPTO 2013 - 33rd Annual Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2013. Proceedings, Part II. pp. 18–35 (2013)

[13] Kiayias, A., Papadopoulos, S., Triandopoulos, N., Zacharias, T.: Delegatable pseudorandom functions and applications. In: ACM Conference on Computer and Communications Security. pp. 669–684 (2013)

[14] Naor, M., Yung, M.: Universal one-way hash functions and their cryptographic applications. In: Proceedings of the 21st Annual ACM Symposium on Theory of Computing, May 14-17, 1989, Seattle, Washigton, USA. pp. 33–43 (1989)

[15] Sahai, A., Waters, B.: How to use indistinguishability obfuscation: deniable encryption, and more. In: STOC. pp. 475–484 (2014)

[16] Waters, B.: A punctured programming approach to adaptively secure functional encryption. In: Advances in Cryptology - CRYPTO 2015 - 35th Annual Cryptology Conference, Santa Barbara, CA, USA, August 16-20, 2015, Proceedings, Part II. pp. 678–697 (2015)

[17] Zimmerman, J.: How to obfuscate programs directly. In: Advances in Cryptology - EUROCRYPT 2015 - 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part II. pp. 439–467 (2015)

A Proof of Unforgeability

Analysis Let $\mathsf{Adv}^i_{\mathcal{A}}$ denote the advantage of adversary $\mathcal{A}$ in hybrid experiment Hyb_i.

Lemma A.1. *For any adversary $\mathcal{A}$, $\mathsf{Adv}^1_{\mathcal{A}} = \mathsf{Adv}^0_{\mathcal{A}}/q$.*

Proof. This follows directly from the description of the experiment. The challenger's guess i^* is correct with probability $1/q$, which implies that if $\mathcal{A}$ wins with probability ϵ in Hyb_0, then it wins with probability ϵ/q in Hyb_1. ■

Lemma A.2. *For any adversary $\mathcal{A}$, $\mathsf{Adv}^2_{\mathcal{A}} = \mathsf{Adv}^1_{\mathcal{A}}/2^{\ell_{\mathrm{ckt}}}$.*

Proof. This proof also follows directly from the description of the experiment. The challenger's guess d' should be equal to the $(i^*)^{th}$ random oracle query, and is correct with probability $1/2^{\ell_{\mathrm{ckt}}}$, which implies that if $\mathcal{A}$ wins with probability ϵ in Hyb_1, then it wins with probability $\epsilon/2^{\ell_{\mathrm{ckt}}}$ in Hyb_2. ■

Lemma A.3. *Assuming $i\mathcal{O}$ secure indistinguishability obfuscator, for any adversary $\mathcal{A}$, $|\mathsf{Adv}^3_{\mathcal{A}} - \mathsf{Adv}^2_{\mathcal{A}}| \leq negl(\lambda)$.*

Proof. Here we change from $\mathsf{USampler}$ to $\mathsf{USampler}'$. The only difference between $\mathsf{USampler}$ and $\mathsf{USampler}'$ is that $\mathsf{USampler}$ uses the (master) secret key $\mathrm{SK}_{\mathrm{pre}}$, while $\mathsf{USampler}'$ uses a restricted secret key $\mathrm{SK}\{(x'||d', \mathsf{out}')\}$, where $x' = H_1(d')$, $r' = F(K_F, x'||d')$ and $\mathsf{out}' = d'(r')$. For all $x||d \neq x'||d'$, $\mathsf{Pre.Sign}(\mathrm{SK}_{\mathrm{pre}}, (x||d, \mathsf{out})) = \mathsf{ResSign}(\mathrm{SK}\{(x'||d', \mathsf{out}')\})$, and therefore, the programs have identical functionality if $x||d \neq x'||d'$. On input x', d', note that the program only signs the pair $(x'||d', \mathsf{out}')$, and $\mathsf{Pre.Sign}(\mathrm{SK}_{\mathrm{pre}}, (x'||d', \mathsf{out}')) = \mathsf{ResSign}(\mathrm{SK}\{(x'||d', \mathsf{out}')\}, (x'||d', \mathsf{out}'))$. As a result, $\mathsf{USampler}$ and $\mathsf{USampler}'$ are functionally identical on all inputs. Hence, if there exists a PPT adversary $\mathcal{A}$ such that $|\mathsf{Adv}^3_{\mathcal{A}} - \mathsf{Adv}^2_{\mathcal{A}}| = \epsilon$, then there exists a PPT algorithm $\mathcal{B}$ that breaks the security of indistinguishability obfuscation with advantage ϵ. This concludes our proof. ■

Lemma A.4. *Assuming $\mathcal{S} = (\mathsf{Pre.Setup}, \mathsf{Pre.Sign}, \mathsf{Pre.Verify}, \mathsf{Restrict}, \mathsf{ResSign})$ is a selectively secure prefix-restricted signature scheme, for any PPT adversary $\mathcal{A}$, $\mathsf{Adv}^3_{\mathcal{A}} \leq negl(\lambda)$.*

Proof. Suppose there exists a PPT adversary $\mathcal{A}$ such that $\mathsf{Adv}^3_{\mathcal{A}} = \epsilon$. We will construct a PPT algorithm $\mathcal{B}$ that uses $\mathcal{A}$ to break the security of prefix-restricted signature scheme $\mathcal{S}$ with probability ϵ. $\mathcal{B}$ is defined as follows.

1. $\mathcal{B}$ chooses $i^* \leftarrow [q]$, $d' \leftarrow \{0,1\}^{lckt}$, $x' \leftarrow \{0,1\}^{\ell_1}$ and $K_F \leftarrow Fsetup(1^\lambda)$. It computes $\mathsf{out}' = d'(F(K_F, (x'||d')))$ and sends $(x'||d', \mathsf{out}')$ to the signature scheme challenger. It receives $\mathrm{SK}\{(x'||d', \mathsf{out}')\}$, $\mathrm{VK}_{\mathrm{pre}}$ from the challenger. $\mathcal{B}$ computes $U \leftarrow i\mathcal{O}(\mathsf{USampler}'\{K_F, \mathrm{SK}\{(x'||d', \mathsf{out}')\}\})$ and sends $U, \mathrm{VK}_{\mathrm{pre}}$ to $\mathcal{A}$.
2. $\mathcal{B}$ then receives random oracle queries from $\mathcal{A}$. For each random oracle query $d_i \neq d'$, it chooses a uniformly random string $r_i \leftarrow \{0,1\}^{\ell_1}$ and sets $H_1(d_i) = r_i$. For the $(i^*)^{th}$ query d', it sends x'.
3. Finally, $\mathcal{A}$ sends a forgery (d', p^*, σ^*). $\mathcal{B}$ forwards message $(x'||d', p^*)$ and signature σ^* as forgery to the challenger.

Clearly, if $\mathcal{A}$ has advantage ϵ, then so does $\mathcal{B}$. ■

From the above lemmas, it follows that if any PPT adversary has at most $\epsilon_{\mathcal{S}}$ advantage in the prefix-restricted signature security game and at most $\epsilon_{i\mathcal{O}}$ in the security game against $i\mathcal{O}$, then any PPT adversary has advantage at most $q \cdot 2^{\ell_{\mathrm{ckt}}} \cdot (\epsilon_{i\mathcal{O}} + \epsilon_{\mathcal{S}})$ in the adaptive unforgeability game of the universal sampler described above. Therefore, assuming sub-exponential security of $i\mathcal{O}$ and $\mathcal{S}$ (that is, assuming $q \cdot 2^{\ell_{\mathrm{ckt}}} \cdot (\epsilon_{i\mathcal{O}} + \epsilon_{\mathcal{S}}) \leq \mathrm{negl}(\lambda)$), our universal sampler satisfies the adaptive unforgeability definition (Definition 3.1).

B Analysis of Adversary's Advantage in Simulation Security Proof

Let $\mathsf{Prob}^x_{\mathcal{A}}$ denote the probability that $\mathcal{A}$ outputs 1 in experiment Exp_x.

Claim B.1. *Assuming $i\mathcal{O}$ is a secure indistinguishability obfuscator, for any PPT adversary $\mathcal{A}$, $|\mathsf{Prob}^0_{\mathcal{A}} - \mathsf{Prob}^1_{\mathcal{A}}| \leq \epsilon_{i\mathcal{O}}$.*

Proof. To prove this claim, we need to show that the programs $\mathsf{USampler}$ and $\mathsf{USampler}$-1 are functionally identical. Recall the only difference between $\mathsf{USampler}$ and $\mathsf{USampler}$-1 is that $\mathsf{USampler}$-1 has PPDE key K_{PPDE} and a uniformly random string α hardwired. Consider any input x, d. $\mathsf{USampler}$-1 first decrypts x to compute m. If m is of the form $(\tilde{d}, a, y)$ and $d = \tilde{d}$ and $\text{PRG}(a) = \alpha$, then it sets $\mathsf{out} = y$. However, since α is chosen uniformly at random, except with negligible probability, there exists no a such that $\text{PRG}(a) = \alpha$. As a result, on all inputs (x, d), the first check of $\mathsf{USampler}$-1 fails and therefore, $\mathsf{USampler}$ and $\mathsf{USampler}$-1 have identical functionality. This implies that if there exists a PPT adversary $\mathcal{A}$ such that $|\mathsf{Prob}^0_{\mathcal{A}} - \mathsf{Prob}^1_{\mathcal{A}}| = \epsilon$, then there exists a PT algorithm $\mathcal{B}$ that breaks the security of $i\mathcal{O}$ with advantage ϵ.

■

Claim B.2. *Assuming* PRG *is a secure pseudorandom generator, for any PPT adversary $\mathcal{A}$, $|\mathsf{Prob}^1_{\mathcal{A}} - \mathsf{Prob}^2_{\mathcal{A}}| \leq \epsilon_{\text{PRG}}$.*

Proof. Suppose there exists a PPT adversary $\mathcal{A}$ such that $|\mathsf{Prob}^1_{\mathcal{A}} - \mathsf{Prob}^2_{\mathcal{A}}| = \epsilon$. We can use $\mathcal{A}$ to construct a PPT algorithm $\mathcal{B}$ that breaks the PRG security with advantage ϵ. $\mathcal{B}$ is defined as follows.

1. $\mathcal{B}$ chooses PRF key K_F, PPDE key K_{PPDE} and prefix-restricted signature key pair $(\text{SK}_{\text{pre}}, \text{VK}_{\text{pre}})$. It receives α from the PRG challenger. Using $K_F, K_{\text{PPDE}}, \text{SK}_{\text{pre}}$ and α, it computes the sampler parameters $U \leftarrow i\mathcal{O}(\mathsf{USampler}\text{-}1\{K_F, K_{\text{PPDE}}, \text{SK}_{\text{pre}}, \alpha\})$ and sends $U, \text{VK}_{\text{pre}}$ to $\mathcal{A}$.
2. $\mathcal{A}$ sends random oracle queries. For each unique query d_i, $\mathcal{B}$ chooses a uniformly random ℓ_1 bit string x_i and sets $H_1(d_i) = x_i$; it sends x_i to $\mathcal{A}$.
3. Finally, after q random oracle queries, $\mathcal{A}$ sends a bit b'. If $b' = 0$, $\mathcal{B}$ output 0 (indicating α is a truly random string), else it outputs 1, indicating α is pseudorandom.

Clearly, from the reduction, it follows that depending on whether α is pseudorandom or truly random, $\mathcal{A}$ participates in either experiment Expt_1 or Expt_2. Therefore, if $|\mathsf{Prob}^1_{\mathcal{A}} - \mathsf{Prob}^2_{\mathcal{A}}| = \epsilon$, then $\mathcal{B}$ breaks the PRG security with advantage ϵ.

■

Lemma B.1. *For any adversary $\mathcal{A}$ and any $i \in [q]$, $|\mathsf{Prob}^{2,i}_{\mathcal{A}} - \mathsf{Prob}^{2,i+1}_{\mathcal{A}}| \leq 2^{\ell_{\text{ckt}}}(5\epsilon_{i\mathcal{O}} + \epsilon_{\text{PPDE}} + \epsilon_F + \epsilon_{\text{PRG}})$.*

Proof. The proof of this hybrid involves multiple hybrid experiments $H_0, \ldots, H_{12}$, where H_0 corresponds to $\mathsf{Expt}_{2,i}$ and H_{12} corresponds to $\mathsf{Expt}_{2,i+1}$.

Experiment H_0: This corresponds to $\mathsf{Expt}_{2,i}$.

Experiment H_1: In this experiment, the challenger guesses the $(i+1)^{th}$ query. It chooses $d^* \leftarrow \{0,1\}^{\ell_{\text{ckt}}}$, and the output of the experiment is 1 if and only if the adversary's output is 1 and the $(i+1)^{th}$ query is d^*.

1. Challenger first chooses a circuit $d^* \leftarrow \{0,1\}^{\ell_{\text{ckt}}}$.
   Next, it computes universal samplers. It chooses $K_F \leftarrow F.\mathsf{setup}(1^\lambda)$, $(\text{SK}_{\text{pre}}, \text{VK}_{\text{pre}}) \leftarrow \mathsf{Pre.Setup}(1^\lambda)$. It chooses a puncturable PPDE key K_{PPDE}, $a \leftarrow \{0,1\}^\lambda$ and sets $\alpha = \text{PRG}(a)$.
   It computes $U \leftarrow i\mathcal{O}(\mathsf{USampler}\text{-}1\{K_F, \text{SK}_{\text{pre}}, K_{\text{PPDE}}, \alpha\})$ and sends $(U, \text{VK}_{\text{pre}})$ to $\mathcal{A}$.
2. $\mathcal{A}$ sends q random oracle queries. For j^{th} unique query d_j,
   - if $j \leq i$, the challenger queries the Parameter Oracle on input d_j and receives p_j in response. It sets $H_1(d_j) = \mathsf{PPDE.Enc}(K_{\text{PPDE}}, p_j)$ and sends $H_1(d_j)$ to $\mathcal{A}$.

- if $j > i$, the challenger chooses uniformly random strings $x_j \leftarrow \{0,1\}^{\ell_1}$, sets $H_1(d_j) = x_j$; it sends $H_1(d_j)$ to $\mathcal{A}$.

3. $\mathcal{A}$ finally sends a bit b.

The output of this experiment is 1 if and only if $b = 1$ and $d_{i+1} = d^*$.

Experiment H_2**:** This experiment is identical to the previos one, except that the challenger modifies the response to $(i+1)^{th}$ random oracle query. Instead of choosing $H(d_{i+1}) = y^*$ uniformly at random, it first computes the PPDE punctured key. Recall that $\mathsf{PPDE.Puncture}$ also outputs a 'random-looking' string y^*. The challenger sends this string as the response to the $(i+1)^{th}$ random oracle query. The obfuscated sampler program still uses the unpunctured PPDE key.

1. Challenger first chooses a circuit $d^* \leftarrow \{0,1\}^{\ell_{\text{ckt}}}$.
   Next, it computes universal samplers. It chooses $K_F \leftarrow F.\mathsf{setup}(1^\lambda)$, $(\mathrm{SK}_{\text{pre}}, \mathrm{VK}_{\text{pre}}) \leftarrow \mathsf{Pre.Setup}(1^\lambda)$.
   It chooses a puncturable PPDE key K_{PPDE}, $a \leftarrow \{0,1\}^\lambda$ and sets $\alpha = \mathrm{PRG}(a)$.
   Next, it queries the parameters oracle on input d^* and receives p^*.
   It computes $(K'_{\text{PPDE}}, y^*) \leftarrow \mathsf{PPDE.Puncture}(K_{\text{PPDE}}, (d^*, a, p^*))$.
   It computes $U \leftarrow i\mathcal{O}(\mathsf{USampler\text{-}1}\{K_F, \mathrm{SK}_{\text{pre}}, K_{\text{PPDE}}, \alpha\})$ and sends $(U, \mathrm{VK}_{\text{pre}})$ to $\mathcal{A}$.
2. $\mathcal{A}$ sends q random oracle queries. For j^{th} unique query d_j,
   - if $j \leq i$, the challenger queries the Parameter Oracle on input d_j and receives p_j in response. It sets $H_1(d_j) = \mathsf{PPDE.Enc}(K_{\text{PPDE}}, p_j)$ and sends $H_1(d_j)$ to $\mathcal{A}$.
   - if $j = i+1$, the challenger sets $H_1(d_j) = y^*$ and sends y^* to the challenger.
   - if $j > i+1$, the challenger chooses uniformly random strings $x_j \leftarrow \{0,1\}^{\ell_1}$, sets $H_1(d_j) = x_j$; it sends $H_1(d_j)$ to $\mathcal{A}$.
3. $\mathcal{A}$ finally sends a bit b.

The output of this experiment is 1 if and only if $b = 1$ and $d_{i+1} = d^*$.

Experiment H_3**:** In this experiment, the challenger replaces $\mathsf{USampler\text{-}1}$ with $\mathsf{USampler\text{-}2}$ (defined in Figure 4). The only difference between these two programs is that on input (d^*, y^*), $\mathsf{USampler\text{-}2}$ outputs a hardwired value p^*, which is set to be $d(F(K_F, (y^*||d^*)))$. Also, the program uses a punctured PRF key.

1. Challenger first chooses a circuit $d^* \leftarrow \{0,1\}^{\ell_{\text{ckt}}}$.
   Next, it computes universal samplers. It chooses $K_F \leftarrow F.\mathsf{setup}(1^\lambda)$, $(\mathrm{SK}_{\text{pre}}, \mathrm{VK}_{\text{pre}}) \leftarrow \mathsf{Pre.Setup}(1^\lambda)$.
   It chooses a puncturable PPDE key K_{PPDE}, $a \leftarrow \{0,1\}^\lambda$ and sets $\alpha = \mathrm{PRG}(a)$.
   Next, it queries the parameters oracle on input d^* and receives p^*.
   It computes $(K'_{\text{PPDE}}, y^*) \leftarrow \mathsf{PPDE.Puncture}(K_{\text{PPDE}}, (d^*, a, p^*))$.
   It sets $p^* = d^*(F(K_F, y^*||d^*))$ and $K'_F \leftarrow F.\mathsf{puncture}(K_F, y^*||d^*)$.
   It computes $U \leftarrow i\mathcal{O}(\mathsf{USampler\text{-}2}\{K'_F, \mathrm{SK}_{\text{pre}}, K_{\text{PPDE}}, \alpha, p^*, y^*\})$; sends $(U, \mathrm{VK}_{\text{pre}})$ to $\mathcal{A}$.
2. $\mathcal{A}$ sends q random oracle queries. For j^{th} unique query d_j,
   - if $j \leq i$, the challenger queries the Parameter Oracle on input d_j and receives p_j in response. It sets $H_1(d_j) = \mathsf{PPDE.Enc}(K_{\text{PPDE}}, p_j)$ and sends $H_1(d_j)$ to $\mathcal{A}$.
   - if $j = i+1$, the challenger sets $H_1(d_j) = y^*$ and sends y^* to the challenger.
   - if $j > i+1$, the challenger chooses uniformly random strings $x_j \leftarrow \{0,1\}^{\ell_1}$, sets $H_1(d_j) = x_j$; it sends $H_1(d_j)$ to $\mathcal{A}$.
3. $\mathcal{A}$ finally sends a bit b.

The output of this experiment is 1 if and only if $b = 1$ and $d_{i+1} = d^*$.

```
USampler-2

Inputs $x \in \{0,1\}^{\ell_1}$, $d \in \{0,1\}^{\ell_{\mathrm{ckt}}}$.

Constants Punctured PRF key $K'_F$, PPDE key $K_{\mathrm{PPDE}}$, $\alpha \in \{0,1\}^{2\lambda}$, $p^* \in \{0,1\}^{\ell_{\mathrm{out}}}$, prefix-restricted
signing key $\mathrm{SK_{pre}}$.

  if $x = y^*$ and $d = d^*$ then
      Set $\mathsf{out} = p^*$.
  else
      Compute $m = \mathsf{PPDE.Dec}(K_{\mathrm{PPDE}}, x)$. If $m \neq \perp$, let $m = (\tilde{d}, a, y) \in \{0,1\}^{\lambda} \times \{0,1\}^{\ell_{\mathrm{out}}}$.
      if $m \neq \perp$ and $\tilde{d} = d$ and $\alpha = \mathrm{PRG}(a)$ then
          Set $\mathsf{out} = y$.
      else
          Compute $r = F(K'_F, (x||d))$.
          Compute $\mathsf{out} = d(r)$.
      end if
  end if
  Compute $\sigma = \mathsf{Pre.Sign}(\mathrm{SK_{pre}}, (x||d, \mathsf{out}))$.
  Output $(\mathsf{out}, \sigma)$.
```

Figure 4: Program $\mathsf{USampler\text{-}2}$

Experiment H_4: In this experiment, the challenger replaces $F(K, y^*||d^*)$ with a truly random string. The indistinguishability of the hybrids follows from the security of puncturable PRFs.

1. Challenger first chooses a circuit $d^* \leftarrow \{0,1\}^{\ell_{\mathrm{ckt}}}$.
   Next, it computes universal samplers. It chooses $K_F \leftarrow F.\mathsf{setup}(1^\lambda)$, $(\mathrm{SK_{pre}}, \mathrm{VK_{pre}}) \leftarrow \mathsf{Pre.Setup}(1^\lambda)$.
   It chooses a puncturable PPDE key K_{PPDE}, $a \leftarrow \{0,1\}^\lambda$ and sets $\alpha = \mathrm{PRG}(a)$.
   Next, it queries the parameters oracle on input d^* and receives p^*.
   It computes $(K'_{\mathrm{PPDE}}, y^*) \leftarrow \mathsf{PPDE.Puncture}(K_{\mathrm{PPDE}}, (d^*, a, p^*))$.
   It chooses a uniformly random string ρ, sets $p^* = d^*(\rho)$ and $K'_F \leftarrow F.\mathsf{puncture}(K_F, y^*||d^*)$.
   It computes $U \leftarrow i\mathcal{O}(\mathsf{USampler\text{-}2}\{K'_F, \mathrm{SK_{pre}}, K_{\mathrm{PPDE}}, \alpha, p^*, y^*\})$; sends $(U, \mathrm{VK_{pre}})$ to $\mathcal{A}$.
2. $\mathcal{A}$ sends q random oracle queries. For j^{th} unique query d_j,
   - if $j \leq i$, the challenger queries the Parameter Oracle on input d_j and receives p_j in response. It sets $H_1(d_j) = \mathsf{PPDE.Enc}(K_{\mathrm{PPDE}}, p_j)$ and sends $H_1(d_j)$ to $\mathcal{A}$.
   - if $j = i+1$, the challenger sets $H_1(d_j) = y^*$ and sends y^* to the challenger.
   - if $j > i+1$, the challenger chooses uniformly random strings $x_j \leftarrow \{0,1\}^{\ell_1}$, sets $H_1(d_j) = x_j$; it sends $H_1(d_j)$ to $\mathcal{A}$.
3. $\mathcal{A}$ finally sends a bit b.

The output of this experiment is 1 if and only if $b = 1$ and $d_{i+1} = d^*$.

Experiment H_5: In this experiment, $\mathsf{USampler\text{-}2}$ is replaced by $\mathsf{USampler\text{-}3}$ (defined in Figure 5). These two programs are identical, except that $\mathsf{USampler\text{-}3}$ uses an unpunctured PRF key. Note that the two programs have identical functionality because the PRF evaluation step is not executed for $x = y^*, d = d^*$.

1. Challenger first chooses a circuit $d^* \leftarrow \{0,1\}^{\ell_{\mathrm{ckt}}}$.
   Next, it computes universal samplers. It chooses $K_F \leftarrow F.\mathsf{setup}(1^\lambda)$, $(\mathrm{SK_{pre}}, \mathrm{VK_{pre}}) \leftarrow \mathsf{Pre.Setup}(1^\lambda)$.
   It chooses a puncturable PPDE key K_{PPDE}, $a \leftarrow \{0,1\}^\lambda$ and sets $\alpha = \mathrm{PRG}(a)$.
   Next, it queries the parameters oracle on input d^* and receives p^*.
   It computes $(K'_{\mathrm{PPDE}}, y^*) \leftarrow \mathsf{PPDE.Puncture}(K_{\mathrm{PPDE}}, (d^*, a, p^*))$.
   It chooses a uniformly random string ρ, sets $p^* = d^*(\rho)$ and $K'_F \leftarrow F.\mathsf{puncture}(K_F, y^*||d^*)$.
   It computes $U \leftarrow i\mathcal{O}(\mathsf{USampler\text{-}3}\{K_F, \mathrm{SK_{pre}}, K_{\mathrm{PPDE}}, \alpha, p^*, y^*\})$; sends $(U, \mathrm{VK_{pre}})$ to $\mathcal{A}$.

2. $\mathcal{A}$ sends q random oracle queries. For j^{th} unique query d_j,
   - if $j \leq i$, the challenger queries the Parameter Oracle on input d_j and receives p_j in response. It sets $H_1(d_j) = \mathsf{PPDE.Enc}(K_{\text{PPDE}}, p_j)$ and sends $H_1(d_j)$ to $\mathcal{A}$.
   - if $j = i + 1$, the challenger sets $H_1(d_j) = y^*$ and sends y^* to the challenger.
   - if $j > i + 1$, the challenger chooses uniformly random strings $x_j \leftarrow \{0,1\}^{\ell_1}$, sets $H_1(d_j) = x_j$; it sends $H_1(d_j)$ to $\mathcal{A}$.
3. $\mathcal{A}$ finally sends a bit b.

The output of this experiment is 1 if and only if $b = 1$ and $d_{i+1} = d^*$.

USampler-3

Inputs $x \in \{0,1\}^{\ell_1}$, $d \in \{0,1\}^{\ell_{\text{ckt}}}$.

Constants Puncturable PRF key K_F, PPDE key K_{PPDE}, $\alpha \in \{0,1\}^{2\lambda}$, $p^* \in \{0,1\}^{\ell_{\text{out}}}$, prefix-restricted signing key SK_{pre}.

```
if $x = y^*$ and $d = d^*$ then
    Set $\mathsf{out} = p^*$.
else
    Compute $m = \mathsf{PPDE.Dec}(K_{\text{PPDE}}, x)$. If $m \neq \perp$, let $m = (\tilde{d}, a, y) \in \{0,1\}^{\lambda} \times \{0,1\}^{\ell_{\text{out}}}$.
    if $m \neq \perp$ and $\tilde{d} = d$ and $\alpha = \text{PRG}(a)$ then
        Set $\mathsf{out} = y$.
    else
        Compute $r = F(K_F, (x||d))$.
        Compute $\mathsf{out} = d(r)$.
    end if
end if
Compute $\sigma = \mathsf{Pre.Sign}(\text{SK}_{\text{pre}}, (x||d, \mathsf{out}))$.
Output $(\mathsf{out}, \sigma)$.
```

Figure 5: Program USampler-3

Experiment H_6: In this experiment, the challenger sends an obfuscation of USampler-4 (defined in Figure 6) instead of USampler-3 . The new program USampler-4 uses a punctured PPDE key for decryption. As a result, to maintain functionality, it has the encryption of punctured point hardwired.

1. Challenger first chooses a circuit $d^* \leftarrow \{0,1\}^{\ell_{\text{ckt}}}$.
   Next, it computes universal samplers. It chooses $K_F \leftarrow F.\mathsf{setup}(1^\lambda)$, $(\text{SK}_{\text{pre}}, \text{VK}_{\text{pre}}) \leftarrow \mathsf{Pre.Setup}(1^\lambda)$.
   It chooses a puncturable PPDE key K_{PPDE}, $a \leftarrow \{0,1\}^\lambda$ and sets $\alpha = \text{PRG}(a)$.
   Next, it queries the parameters oracle on input d^* and receives p^*.
   It computes $(K'_{\text{PPDE}}, y^*) \leftarrow \mathsf{PPDE.Puncture}(K_{\text{PPDE}}, (d^*, a, p^*))$ and the ciphertext $\mathsf{ct}^* = \mathsf{PPDE.Enc}(K_{\text{PPDE}}, (d^*, a, p^*))$.
   It chooses $p^* \leftarrow \{0,1\}^{\ell_{\text{out}}}$ and $K\{y^*||d^*\} \leftarrow F.\mathsf{puncture}(K, y^*||d^*)$.
   It computes $U \leftarrow i\mathcal{O}(\mathsf{USampler\text{-}4}\{K_F, \text{SK}_{\text{pre}}, K'_{\text{PPDE}}, \alpha, p^*, y^*, \mathsf{ct}^*\})$; sends $(U, \text{VK}_{\text{pre}})$ to $\mathcal{A}$.
2. $\mathcal{A}$ sends q random oracle queries. For j^{th} unique query d_j,
   - if $j \leq i$, the challenger queries the Parameter Oracle on input d_j and receives p_j in response. It sets $H_1(d_j) = \mathsf{PPDE.Enc}(K_{\text{PPDE}}, p_j)$ and sends $H_1(d_j)$ to $\mathcal{A}$.
   - if $j = i + 1$, the challenger sets $H_1(d_j) = y^*$ and sends y^* to the challenger.
   - if $j > i + 1$, the challenger chooses uniformly random strings $x_j \leftarrow \{0,1\}^{\ell_1}$, sets $H_1(d_j) = x_j$; it sends $H_1(d_j)$ to $\mathcal{A}$.
3. $\mathcal{A}$ finally sends a bit b.

The output of this experiment is 1 if and only if $b = 1$ and $d_{i+1} = d^*$.

```
USampler-4

Inputs $x \in \{0,1\}^{\ell_1}$, $d \in \{0,1\}^{\ell_{\rm ckt}}$.

Constants Puncturable PRF key $K_F$, PPDE key $K_{\rm PPDE}$, $\alpha \in \{0,1\}^{2\lambda}$, $p^* \in \{0,1\}^{\ell_{\rm out}}$,
PPDE ciphertext $\mathsf{ct}^*$, prefix-restricted signing key $\mathrm{SK}_{\rm pre}$.

  if $x = y^*$ and $d = d^*$ then
    Set $\mathsf{out} = p^*$.
  else if $x = \mathsf{ct}^*$ and $d = d^*$ then
    Set $\mathsf{out} = p^*$.
  else
    Compute $m = \mathsf{PPDE.Dec}(K_{\rm PPDE}, x)$. If $m \neq \perp$, let $m = (\tilde{d}, a, y) \in \{0,1\}^\lambda \times \{0,1\}^{\ell_{\rm out}}$.
    if $m \neq \perp$ and $\tilde{d} = d$ and $\alpha = \mathrm{PRG}(a)$ then
      Set $\mathsf{out} = y$.
    else
      Compute $r = F(K_F, (x||d))$.
      Compute $\mathsf{out} = d(r)$.
    end if
  end if
  Compute $\sigma = \mathsf{Pre.Sign}(\mathrm{SK}_{\rm pre}, (x||d, \mathsf{out}))$.
  Output $(\mathsf{out}, \sigma)$.
```

Figure 6: Program $\mathsf{USampler\text{-}4}$

Experiment H_7: In this experiment, the challenger swaps y^* and ct^* using the PPDE security. As a result, the response to $(i+1)^{th}$ random oracle query is a PPDE ciphertext. The obfuscated program is now $\mathsf{USampler\text{-}5}$ (defined in Figure 7).

1. Challenger first chooses a circuit $d^* \leftarrow \{0,1\}^{\ell_{\rm ckt}}$.
   Next, it computes universal samplers. It chooses $K_F \leftarrow F.\mathsf{setup}(1^\lambda)$, $(\mathrm{SK}_{\rm pre}, \mathrm{VK}_{\rm pre}) \leftarrow \mathsf{Pre.Setup}(1^\lambda)$.
   It chooses a puncturable PPDE key $K_{\rm PPDE}$, $a \leftarrow \{0,1\}^\lambda$ and sets $\alpha = \mathrm{PRG}(a)$.
   Next, it queries the parameters oracle on input d^* and receives p^*.
   It computes $(K'_{\rm PPDE}, y^*) \leftarrow \mathsf{PPDE.Puncture}(K_{\rm PPDE}, (d^*, a, p^*))$.
   It chooses $p^* \leftarrow \{0,1\}^{\ell_{\rm out}}$ and $K\{y^*||d^*\} \leftarrow F.\mathsf{puncture}(K, y^*||d^*)$.
   It computes $U \leftarrow i\mathcal{O}(\mathsf{USampler\text{-}4}\{K_F, \mathrm{SK}_{\rm pre}, K'_{\rm PPDE}, \alpha, p^*, \mathsf{ct}^*, y^*\})$; sends $(U, \mathrm{VK}_{\rm pre})$ to $\mathcal{A}$.
2. $\mathcal{A}$ sends q random oracle queries. For j^{th} unique query d_j,
   - if $j \leq i$, the challenger queries the Parameter Oracle on input d_j and receives p_j in response. It sets $H_1(d_j) = \mathsf{PPDE.Enc}(K_{\rm PPDE}, p_j)$ and sends $H_1(d_j)$ to $\mathcal{A}$.
   - if $j = i+1$, the challenger sets $H_1(d_j) = \mathsf{ct}^*$ and sends ct^* to the challenger.
   - if $j > i+1$, the challenger chooses uniformly random strings $x_j \leftarrow \{0,1\}^{\ell_1}$, sets $H_1(d_j) = x_j$ and sends $H_1(d_j)$ to the adversary.
3. $\mathcal{A}$ finally sends a bit b.

The output of this experiment is 1 if and only if $b = 1$ and $d_{i+1} = d^*$.

Remark: Note that y^* and ct^* have been swapped in the hardwiring of program $\mathsf{USampler\text{-}4}$. As a result, the first two lines of the program are as follows:

- If $x = \mathsf{ct}^*$ and $d = d^*$ set $\mathsf{out} = p^*$.
- Else if $x = y^*$ set $\mathsf{out} = p^*$.
- Else compute $m = \mathsf{PPDE.Dec}(K_{\rm PPDE}, x)$. ...

Experiment H_8: In this experiment, the challenger outputs an obfuscation of USampler-5 (defined in Figure 7). This program uses an unpunctured PPDE key. As a result, it does not have the clause 'if $x = \mathsf{ct}^*$ and $d = d^*$ set $\mathsf{out} = p^*$' since the PPDE decryption will set $\mathsf{out} = p^*$ if $x = \mathsf{ct}^*$. Also, instead of checking if $x = y^*$, it checks if $\mathrm{PRG}(x) = z^*$, where z^* is hardwired to be $\mathrm{PRG}(y^*)$. Note that the PRG is injective, and therefore these two checks are equivalent.

1. Challenger first chooses a circuit $d^* \leftarrow \{0,1\}^{\ell_{\mathrm{ckt}}}$.
   Next, it computes universal samplers. It chooses $K_F \leftarrow F.\mathsf{setup}(1^\lambda)$, $(\mathrm{SK}_{\mathrm{pre}}, \mathrm{VK}_{\mathrm{pre}}) \leftarrow \mathsf{Pre.Setup}(1^\lambda)$.
   It chooses a puncturable PPDE key K_{PPDE}, $a \leftarrow \{0,1\}^\lambda$ and sets $\alpha = \mathrm{PRG}(a)$.
   Next, it queries the parameters oracle on input d^* and receives p^*.
   It computes $(K'_{\mathrm{PPDE}}, y^*) \leftarrow \mathsf{PPDE.Puncture}(K_{\mathrm{PPDE}}, (d^*, a, p^*))$ and $\underline{z^* = \mathrm{PRG}(y^*)}$.
   It chooses $p^* \leftarrow \{0,1\}^{\ell_{\mathrm{out}}}$ and $K\{y^*||d^*\} \leftarrow F.\mathsf{puncture}(K, y^*||d^*)$.
   It computes $\underline{U \leftarrow i\mathcal{O}(\mathsf{USampler\text{-}5}\{K_F, \mathrm{SK}_{\mathrm{pre}}, K_{\mathrm{PPDE}}, \alpha, p^*, \mathsf{ct}^*, y^*, z^*\})}$; sends $(U, \mathrm{VK}_{\mathrm{pre}})$ to $\mathcal{A}$.
2. $\mathcal{A}$ sends q random oracle queries. For j^{th} unique query d_j,
   - if $j \le i$, the challenger queries the Parameter Oracle on input d_j and receives p_j in response. It sets $H_1(d_j) = \mathsf{PPDE.Enc}(K_{\mathrm{PPDE}}, p_j)$ and sends $H_1(d_j)$ to $\mathcal{A}$.
   - if $j = i + 1$, the challenger sets $H_1(d_j) = \mathsf{ct}^*$ and sends ct^* to the challenger.
   - if $j > i + 1$, the challenger chooses uniformly random strings $x_j \leftarrow \{0,1\}^{\ell_1}$, sets $H_1(d_j) = x_j$ and sends $H_1(d_j)$ to the adversary.
3. $\mathcal{A}$ finally sends a bit b.

The output of this experiment is 1 if and only if $b = 1$ and $d_{i+1} = d^*$.

USampler-5

Inputs $x \in \{0,1\}^{\ell_1}$, $d \in \{0,1\}^{\ell_{\mathrm{ckt}}}$.

Constants Puncturable PRF key K_F, PPDE key K_{PPDE}, $\alpha \in \{0,1\}^{2\lambda}$, $p^* \in \{0,1\}^{\ell_{\mathrm{out}}}$, PPDE ciphertext ct^*, $\underline{z^* \in \{0,1\}^{2\ell_1}}$, prefix-restricted signing key $\mathrm{SK}_{\mathrm{pre}}$.

```
if $\underline{\mathrm{PRG}(x) = z^* \text{ and } d = d^*}$ then
    $\underline{\text{Set } \mathsf{out} = p^*}$.
else
    Compute $m = \mathsf{PPDE.Dec}(K_{\mathrm{PPDE}}, x)$. If $m \neq \perp$, let $m = (\tilde{d}, a, y) \in \{0,1\}^\lambda \times \{0,1\}^{\ell_{\mathrm{out}}}$.
    if $m \neq \perp$ and $\tilde{d} = d$ and $\alpha = \mathrm{PRG}(a)$ then
        Set $\mathsf{out} = y$.
    else
        Compute $r = F(K, (x||d))$.
        Compute $\mathsf{out} = d(r)$.
    end if
end if
Compute $\sigma = \mathsf{Pre.Sign}(\mathrm{SK}_{\mathrm{pre}}, (x||d, \mathsf{out}))$.
Output $(\mathsf{out}, \sigma)$.
```

Figure 7: Program USampler-5

Experiment H_9: In this experiment, the challenger chooses y^* uniformly at random. As a result, it no longer needs to compute $\mathsf{PPDE.Puncture}(\cdot,\cdot)$.

1. Challenger first chooses a circuit $d^* \leftarrow \{0,1\}^{\ell_{\mathrm{ckt}}}$.
   Next, it computes universal samplers. It chooses $K_F \leftarrow F.\mathsf{setup}(1^\lambda)$, $(\mathrm{SK}_{\mathrm{pre}}, \mathrm{VK}_{\mathrm{pre}}) \leftarrow \mathsf{Pre.Setup}(1^\lambda)$.

It chooses a puncturable PPDE key K_{PPDE}, $a \leftarrow \{0,1\}^\lambda$ and sets $\alpha = \text{PRG}(a)$.
Next, it queries the parameters oracle on input d^* and receives p^*.
It $\underline{\text{chooses } y^* \leftarrow \{0,1\}^{\ell_1}}$ and $z^* = \text{PRG}(y^*)$.
It chooses $p^* \leftarrow \{0,1\}^{\ell_{\text{out}}}$ and $K\{y^*||d^*\} \leftarrow F.\mathsf{puncture}(K, y^*||d^*)$.
It computes $U \leftarrow i\mathcal{O}(\mathsf{USampler\text{-}5}\{K_F, \text{SK}_{\text{pre}}, K_{\text{PPDE}}, \alpha, p^*, \mathsf{ct}^*, y^*, z^*\})$; sends $(U, \text{VK}_{\text{pre}})$ to $\mathcal{A}$.

2. $\mathcal{A}$ sends q random oracle queries. For j^{th} unique query d_j,
   - if $j \le i$, the challenger queries the Parameter Oracle on input d_j and receives p_j in response. It sets $H_1(d_j) = \mathsf{PPDE.Enc}(K_{\text{PPDE}}, p_j)$ and sends $H_1(d_j)$ to $\mathcal{A}$.
   - if $j = i+1$, the challenger sets $H_1(d_j) = \mathsf{ct}^*$ and sends ct^* to the challenger.
   - if $j > i+1$, the challenger chooses uniformly random strings $x_j \leftarrow \{0,1\}^{\ell_1}$, sets $H_1(d_j) = x_j$ and sends $H_1(d_j)$ to the adversary.
3. $\mathcal{A}$ finally sends a bit b.

The output of this experiment is 1 if and only if $b = 1$ and $d_{i+1} = d^*$.

Experiment H_{10}: In this experiment, the challenger chooses z^* uniformly at random. This change is indistinguishable due to the security of PRG.

1. Challenger first chooses a circuit $d^* \leftarrow \{0,1\}^{\ell_{\text{ckt}}}$.
   Next, it computes universal samplers. It chooses $K_F \leftarrow F.\mathsf{setup}(1^\lambda)$, $(\text{SK}_{\text{pre}}, \text{VK}_{\text{pre}}) \leftarrow \mathsf{Pre.Setup}(1^\lambda)$.
   It chooses a puncturable PPDE key K_{PPDE}, $a \leftarrow \{0,1\}^\lambda$ and sets $\alpha = \text{PRG}(a)$.
   Next, it queries the parameters oracle on input d^* and receives p^*.
   It chooses $y^* \leftarrow \{0,1\}^{\ell_1}$ and $\underline{z^* \leftarrow \{0,1\}^{2\ell_1}}$.
   It chooses $p^* \leftarrow \{0,1\}^{\ell_{\text{out}}}$ and $K\{y^*||d^*\} \leftarrow F.\mathsf{puncture}(K, y^*||d^*)$.
   It computes $U \leftarrow i\mathcal{O}(\mathsf{USampler\text{-}5}\{K_F, \text{SK}_{\text{pre}}, K_{\text{PPDE}}, \alpha, p^*, \mathsf{ct}^*, y^*, z^*\})$; sends $(U, \text{VK}_{\text{pre}})$ to $\mathcal{A}$.
2. $\mathcal{A}$ sends q random oracle queries. For j^{th} unique query d_j,
   - if $j \le i$, the challenger queries the Parameter Oracle on input d_j and receives p_j in response. It sets $H_1(d_j) = \mathsf{PPDE.Enc}(K_{\text{PPDE}}, p_j)$ and sends $H_1(d_j)$ to $\mathcal{A}$.
   - if $j = i+1$, the challenger sets $H_1(d_j) = \mathsf{ct}^*$ and sends ct^* to the challenger.
   - if $j > i+1$, the challenger chooses uniformly random strings $x_j \leftarrow \{0,1\}^{\ell_1}$, sets $H_1(d_j) = x_j$ and sends $H_1(d_j)$ to the adversary.
3. $\mathcal{A}$ finally sends a bit b.

The output of this experiment is 1 if and only if $b = 1$ and $d_{i+1} = d^*$.

Experiment H_{11}: In this experiment, the challenger outputs an obfuscation of $\mathsf{USampler\text{-}1}$. This hybrid is identical to $\mathsf{Hyb}_{2,i+1}$, except that the challenger also guesses d^*. Note that the only difference between $\mathsf{USampler\text{-}5}$ and $\mathsf{USampler\text{-}1}$ is the clause 'if $\text{PRG}(x) = z^*$ then output p^*'. However, this statement is never executed for a uniformly random z^*. As a result, these two programs are functionally identical, and hence their obfuscations are computationally indistinguishable.

1. Challenger first chooses a circuit $d^* \leftarrow \{0,1\}^{\ell_{\text{ckt}}}$.
   Next, it computes universal samplers. It chooses $K_F \leftarrow F.\mathsf{setup}(1^\lambda)$, $(\text{SK}_{\text{pre}}, \text{VK}_{\text{pre}}) \leftarrow \mathsf{Pre.Setup}(1^\lambda)$.
   It chooses a puncturable PPDE key K_{PPDE}, $a \leftarrow \{0,1\}^\lambda$ and sets $\alpha = \text{PRG}(a)$.
   Next, it queries the parameters oracle on input d^* and receives p^*. It computes $\mathsf{ct}^* = \mathsf{PPDE.Enc}(K_{\text{PPDE}}, (d^*, a, p^*))$
   It computes $\underline{U \leftarrow i\mathcal{O}(\mathsf{USampler\text{-}1}\{K_F, \text{SK}_{\text{pre}}, K_{\text{PPDE}}, \alpha\})}$; sends $(U, \text{VK}_{\text{pre}})$ to $\mathcal{A}$.
2. $\mathcal{A}$ sends q random oracle queries. For j^{th} unique query d_j,

- if $j \le i$, the challenger queries the Parameter Oracle on input d_j and receives p_j in response. It sets $H_1(d_j) = \mathsf{PPDE.Enc}(K_{\text{PPDE}}, p_j)$ and sends $H_1(d_j)$ to $\mathcal{A}$.
- if $j = i+1$, the challenger sets $H_1(d_j) = \mathsf{ct}^*$ and sends ct^* to the challenger.
- if $j > i+1$, the challenger chooses uniformly random strings $x_j \leftarrow \{0,1\}^{\ell_1}$, sets $H_1(d_j) = x_j$ and sends $H_1(d_j)$ to the adversary.

3. $\mathcal{A}$ finally sends a bit b.

The output of this experiment is 1 if and only if $b = 1$ and $d_{i+1} = d^*$.

Experiment H_{12}**:** This corresponds to $\mathsf{Hyb}_{2,i+1}$.

We will now analyze the probability of j^{th} hybrid H_j outputting 1. Let $p^j_{\mathcal{A}}$ denote the probability that the outcome of H_j is 1.

Claim B.3. *For any adversary* $\mathcal{A}$, $p^1_{\mathcal{A}} = p^0_{\mathcal{A}}/2^{\ell_{\text{ckt}}}$.

Proof. This follows directly from the definition of H_0 and H_1, the fact that d^* is chosen uniformly at random. ■

Claim B.4. *For any adversary* $\mathcal{A}$, $|p^1_{\mathcal{A}} - p^2_{\mathcal{A}}| \le negl(\lambda)$.

Proof. This follows from Correctness Property 3 of PPDE schemes. Recall this property states that for all messages (a, p^*) and PPDE keys K_{PPDE}, if $(K'_{\text{PPDE}}, y^*) \leftarrow \mathsf{PPDE.Puncture}(K_{\text{PPDE}}, (d^*, a, p^*))$, then y^* is statistically close to a uniformly random ℓ bit string, even when given K, a and p^*. ■

Claim B.5. *Assuming* $i\mathcal{O}$ *is a secure indistinguishability obfuscator, for any PPT adversary* $\mathcal{A}$, $|p^2_{\mathcal{A}} - p^3_{\mathcal{A}}| \le \epsilon_{i\mathcal{O}}$.

Proof. The proof of this claim follows from the fact that $\mathsf{USampler\text{-}1}$ and $\mathsf{USampler\text{-}2}$ are functionally identical. The only difference between the two programs is that one uses an unpunctured PRF key, while the other uses a punctured key. However, $\mathsf{USampler\text{-}2}$ uses the hardwired value p^* on input (y^*, d^*). As a result, both programs have the same output on (y^*, d^*). On all other inputs, due to the correctness of puncturable PRFs, both programs have identical behavior. As a result, their obfuscations are computationally indistinguishable. ■

Claim B.6. *Assuming* F *is a selectively secure puncturable PRF, for any PPT adversary* $\mathcal{A}$, $|p^3_{\mathcal{A}} - p^4_{\mathcal{A}}| \le \epsilon_F$.

Proof. This proof follows from the security of puncturable PRFs. Consider any PPT adversary $\mathcal{A}$. We will construct a reduction algorithm $\mathcal{B}$ that breaks the selective security of F with advantage $|p^2_{\mathcal{A}} - p^3_{\mathcal{A}}|$. The reduction algorithm chooses d^*, y^* uniformly at random and sends $(y^* || d^*)$ to the PRF challenger. It receives a punctured key K'_F and a string r^*, which is either a PRF evaluation or a truly random string. The reduction algorithm sets $p^* = d^*(r^*)$. It then chooses the PPDE key and signing/verification keys and computes an obfuscation of $\mathsf{USampler\text{-}2}$. Note that $\mathcal{B}$ does not require p^* for answering the random oracle queries. Finally, the adversary sends a bit b. If $b = 1$, $\mathcal{B}$ guesses that r^* was pseudorandom, else it guesses that it was truly random. This concludes our proof. ■

Claim B.7. *Assuming* $i\mathcal{O}$ *is a secure indistinguishability obfuscator, for any PPT adversary* $\mathcal{A}$, $|p^4_{\mathcal{A}} - p^5_{\mathcal{A}}| \le \epsilon_{i\mathcal{O}}$.

Proof. The proof of this claim follows from the fact that $\mathsf{USampler}$-2 and $\mathsf{USampler}$-3 are functionally identical. The only difference between the two programs is that one uses a punctured PRF key (punctured at $y^*||d^*$), while the other uses an unpunctured key. However, both programs do not execute the PRF evaluation step for $x = y^*, d = d^*$. This is because for input $x = y^*, d = d^*$, both programs output the hardwired value p^*. As a result, the two programs are functionally identical, and therefore their obfuscations are computationally indistinguishable. ∎

Claim B.8. *Assuming $i\mathcal{O}$ is a secure indistinguishability obfuscator, for any PPT adversary $\mathcal{A}$, $|p^5_{\mathcal{A}} - p^6_{\mathcal{A}}| \leq \epsilon_{i\mathcal{O}}$.*

Proof. The only difference between the experiments H_5 and H_6 is that one uses $\mathsf{USampler}$-3, and the other uses $\mathsf{USampler}$-4. The program $\mathsf{USampler}$-4 uses a punctured PPDE key, and has the output hardwired for the punctured point. As a result, both programs have the same output for $\mathsf{ct}^* = \mathsf{PPDE.Enc}(K_{\text{PPDE}}, (d^*, a, p^*))$. For all other inputs, the two programs are identical since the PPDE scheme is correct (decryptions using K'_{PPDE} are identical to decryptions using K_{PPDE} for all inputs not equal to ct^*). As a result, their obfuscations are computationally indistinguishable. ∎

Claim B.9. *Assuming* PPDE *is a secure puncturable pseudorandom deterministic encryption scheme, for any PPT adversary $\mathcal{A}$, $|p^6_{\mathcal{A}} - p^7_{\mathcal{A}}| \leq \epsilon_{\text{PPDE}}$.*

Proof. Recall the PPDE security states that given a PPDE key K'_{PPDE} punctured at input γ, no PPT adversary can distinguish between (ct, r) and (r, ct), where ct is the encryption of γ and r is a uniformly random string. Let $\mathcal{A}$ be any PPT adversary. We will construct a reduction algorithm $\mathcal{B}$ that breaks the security of PPDE with advantage $|p^5_{\mathcal{A}} - p^6_{\mathcal{A}}|$ using $\mathcal{A}$. The reduction algorithm first chooses a PRF key K_F and signing/verification keys. Next, it chooses a, d^*, y^*, r^* uniformly at random and sets $p^* = d^*(r^*)$. It sends (d^*, a, p^*) to the PPDE challenger, and receives a PPDE key K'_{PPDE} and a tuple (y_1, y_2), where one of them is the encryption of (d^*, a, p^*), and the other is a uniformly random string. The reduction algorithm computes an obfuscation of $\mathsf{USampler}\text{-}4\{K_F, \text{SK}_{\text{pre}}, K'_{\text{PPDE}}, \alpha, p^*, y_1, y_2\}$. Finally, when responding to random oracle queries, for the $(i + 1)^{th}$ query, it sends y_1. The adversary outputs a bit b, which $\mathcal{B}$ forwards to the PPDE challenger.

Now, if y_1 is a random string, then this corresponds to H_6, else it corresponds to H_7. This concludes our proof. ∎

Claim B.10. *Assuming $i\mathcal{O}$ is a secure indistinguishability obfuscator, for any PPT adversary $\mathcal{A}$, $|p^7_{\mathcal{A}} - p^8_{\mathcal{A}}| \leq \epsilon_{i\mathcal{O}}$.*

Proof. To prove this claim, we need to argue that $\mathsf{USampler}$-4 and $\mathsf{USampler}$-5 are functionally identical. There are two differences between these two programs: first, $\mathsf{USampler}$-5 has $z^* = \text{PRG}(y^*)$ hardwired and it checks if $\text{PRG}(x) = z^*$ instead of checking if $x = y^*$. This does not change functionality since the pseudorandom generator is injective. Next, it uses an unpunctured PPDE key, while $\mathsf{USampler}$-4 uses a punctured PPDE key and has the output hardwired for input (ct^*, d^*). However, note that for this input, both programs have identical outputs. This implies that both programs are functionally identical, and as a result, their obfuscations are computationally indistinguishable. ∎

Claim B.11. *For any adversary $\mathcal{A}$, $|p^8_{\mathcal{A}} - p^9_{\mathcal{A}}| \leq negl(\lambda)$.*

Proof. This follows from Correctness Property 3 of PPDE schemes, similar to the proof of Claim B.4. ∎

Claim B.12. *Assuming* PRG *is a secure pseudorandom generator, for any PPT adversary $\mathcal{A}$, $|p^9_{\mathcal{A}} - p^{10}_{\mathcal{A}}| \leq \epsilon_{\text{PRG}}$.*

Proof. The proof of this claim follows from the security of PRG. Note that the only difference between the two hybrids is that in H_9, the challenger sets $z^* = \mathrm{PRG}(y^*)$, while in the other one, it chooses $z^* \leftarrow \{0,1\}^{2\ell_1}$. In both hybrids, y^* is not used anywhere else. As a result, we can construct a reduction algorithm $\mathcal{B}$ that breaks the PRG security with advantage $|p^9_{\mathcal{A}} - p^{10}_{\mathcal{A}}|$. ∎

Claim B.13. *Assuming $i\mathcal{O}$ is a secure indistinguishability obfuscator, for any PPT adversary $\mathcal{A}$, $|p^{10}_{\mathcal{A}} - p^{11}_{\mathcal{A}}| \leq \epsilon_{i\mathcal{O}}$.*

Proof. As in the other proofs involving $i\mathcal{O}$, we will show that the two programs involved are functionally identical. Let us consider the differences between USampler-5 and USampler-1. The only difference between the two programs is the 'if $\mathrm{PRG}(x) = z^*$, set $\mathsf{out} = p^*$' clause. However, since z^* is chosen uniformly at random, with high probability, for all inputs x, this part will never be executed. Hence, the two programs are functionally identical. As a result, using the security of $i\mathcal{O}$, we can argue that $|p^{10}_{\mathcal{A}} - p^{11}_{\mathcal{A}}| \leq \epsilon_{i\mathcal{O}}$. ∎

Claim B.14. *For any PPT adversary $\mathcal{A}$, $p^{11}_{\mathcal{A}} = p^{12}_{\mathcal{A}}/2^{\ell_{\mathrm{ckt}}}$.*

Proof. This is immediate since d^* is chosen uniformly at random in H_{11}. ∎

Summing up, we get that $|p^0_{\mathcal{A}} - p^{12}_{\mathcal{A}}| \leq 2^{\ell_{\mathrm{ckt}}}(5\epsilon_{i\mathcal{O}} + \epsilon_{\mathrm{PRG}} + \epsilon_F + \epsilon_{\mathrm{PPDE}})$ ∎