Breaking the Sub-Exponential Barrier in Obfustopia

Sanjam Garg* Omkant Pandey† Akshayaram Srinivasan‡ Mark Zhandry§

Abstract

Indistinguishability obfuscation ($i\mathcal{O}$) has emerged as a surprisingly powerful notion. Almost all known cryptographic primitives can be constructed from general purpose $i\mathcal{O}$ and other minimalistic assumptions such as one-way functions. The primary challenge in this direction of research is to develop novel techniques for using $i\mathcal{O}$ since $i\mathcal{O}$ by itself offers virtually no protection to secret information in the underlying programs. When dealing with complex situations, often these techniques have to consider an exponential number of hybrids (usually one per input) in the security proof. This results in a *sub-exponential* loss in the security reduction. Unfortunately, this scenario is becoming more and more common and appears to be a fundamental barrier to current techniques.

In this work, we explore the possibility of getting around this *sub-exponential loss barrier* in constructions based on $i\mathcal{O}$ as well as the weaker notion of *functional encryption* ($\mathcal{FE}$). Towards this goal, we achieve the following results:

1. We construct trapdoor one-way permutations from *polynomially-hard* $i\mathcal{O}$ (and standard one-way permutations). This improves upon the recent result of Bitansky, Paneth, and Wichs (TCC 2016) which requires $i\mathcal{O}$ of sub-exponential strength.
2. We present a different construction of trapdoor one-way permutations based on standard, polynomially-secure, public-key functional encryption. This qualitatively improves upon our first result since $\mathcal{FE}$ is a weaker primitive than $i\mathcal{O}$ — it can be based on polynomially-hard assumptions on multi-linear maps whereas $i\mathcal{O}$ inherently seems to requires assumptions of sub-exponential strength.
3. We present a construction of *universal samplers* also based only on polynomially-secure public-key $\mathcal{FE}$. Universal samplers, introduced in the work of Hofheinz, Jager, Khurana, Sahai, Waters and Zhandry (EPRINT 2014), is an appealing notion which allows a *single* trusted setup for *any* protocol. As an application of this result, we construct a *non-interactive multiparty key exchange* (NIKE) protocol for an unbounded number of users without a trusted setup. Prior to this work, such constructions were only known from indistinguishability obfuscation.

In obtaining our results, we build upon and significantly extend the techniques of Garg, Pandey, and Srinivasan (EPRINT 2015) introduced in the context of reducing PPAD-hardness to polynomially-secure $i\mathcal{O}$ and $\mathcal{FE}$.

*University of California, Berkeley, sanjamg@berkeley.edu
†Drexel University, omkant@drexel.edu
‡University of California, Berkeley, akshayaram@berkeley.edu
§MIT, mzhandry@gmail.com

1 Introduction

Indistinguishability obfuscation ($i\mathcal{O}$) [BGI+12, GGH+13b] has emerged as a powerful cryptographic primitive in the past few years. It has proven sufficient to construct almost all cryptographic primitives, many of them for the first time, [SW14, HSW14, BZ14, PPS15, CLP15, HW15, BLR+15, AS16, BPW16]. Recently, $i\mathcal{O}$ also proved instrumental in proving the hardness of complexity class PPAD [BPR15].

The primary challenge in this direction of research stems from the fact that $i\mathcal{O}$ by itself is "too weak" to work with. The standard security of $i\mathcal{O}$ may not even hide any secrets present in the underlying programs. Therefore, the crucial part of most $i\mathcal{O}$-based constructions lies in developing novel techniques for using $i\mathcal{O}$ to obfuscate "programs with secrets."

There are only a limited set of techniques for working with $i\mathcal{O}$. When dealing with complex situations, these techniques often run into what we call the *sub-exponential barrier*. More specifically, the security proof of a typical $i\mathcal{O}$-based construction often ends up considering an exponential number of hybrid experiments in order to make just one change in the underlying obfuscation. The goal is usually to eliminate all "troublesome" inputs, one at a time, that may be affected by the change. Such inputs are usually exponentially many, resulting in a sub-exponential loss in the security reduction. Unfortunately, this situation arises frequently and too many works suffer from the sub-exponential loss.

The problem becomes even worse when instead of $i\mathcal{O}$, one wants to rely on weaker primitives such as *functional encryption* ($\mathcal{FE}$). Although $\mathcal{FE}$ implies $i\mathcal{O}$, this reduction too incurs a sub-exponential loss [AJ15, BV15]. Therefore, one typically seeks a direct construction from $\mathcal{FE}$. Note that $\mathcal{FE}$ can be based on polynomial-hardness assumptions on multilinear maps whereas $i\mathcal{O}$ inherently seems to require assumptions of sub-exponential strength (or exponentially many assumptions of polynomial strength).

Recently, Garg, Pandey, and Srinivasan [GPS15] took upon the issue of sub-exponential loss in $i\mathcal{O}$-based constructions in the context of PPAD hardness. They developed techniques to eliminate the sub-exponential loss in the work of Bitansky, Paneth, and Rosen [BPR15] and reduced the hardness of PPAD to the hardness of standard, polynomially-secure $i\mathcal{O}$ (and injective one-way functions). Garg et al. also presented a new reduction which bases the hardness of PPAD to standard polynomially-secure functional encryption. Together with the result of [GGHZ16], this essentially reduces the hardness of PPAD to simple polynomially-hard assumptions on multi-linear maps [GGH13a].

This work. Our goal is to develop techniques to break the sub-exponential barrier in cryptographic constructions based on $i\mathcal{O}$ and $\mathcal{FE}$. Towards this goal, we build upon and significantly extend the techniques in [GPS15].

We start with the construction of trapdoor permutations of Bitanksy, Paneth, and Wichs [BPW16] based on sub-exponentially secure $i\mathcal{O}$. We improve their work by constructing trapdoor permutations based only on *polynomially-secure* $i\mathcal{O}$ (and one-way permutations). We further extend our results and obtain a construction based on standard, polynomial hard, functional encryption (instead of $i\mathcal{O}$). Together with the result of [GGHZ16], this gives us trapdoor permutations based on polynomial-hard assumptions on multi-linear maps.

We then consider the case of *universal samplers*. The notion of universal samplers was put forward by Hofheinz, Jager, Khurana, Sahai, Waters, and Zhandry [HJK+14]. It allows for a single

trusted setup which can be used to sample common parameters for *any* protocol. More specifically, a *universal sampler* is an algorithm that takes as input the description of a sampling procedure (e.g., the sampling procedure for the common parameters of some protocol π) and outputs a sample from that procedure (to be used as the parameters for π). The algorithm is deterministic, so that anyone running the protocol on a given sampling procedure gets the same parameters. Yet the generated parameters should be "as good as" a freshly generated set of parameters. Therefore, the only set of common parameters needed for all protocols is just a single universal sampler. When a group of users wish to engage in a protocol involving a trusted setup, they can each feed the setup procedure of that protocol into the universal sampler, and use the output as the common parameters. Universal samplers is a very appealing notion. Hofheinz et al. construct universal samplers from $i\mathcal{O}$. They also show how to use them to construct *multi-party non-interactive key-exchange* (NIKE) and broadcast encryption.

We consider the task of constructing universal samplers from the weaker notion of only polynomially-secure functional encryption. As noted earlier, we cannot use the generic reduction of [AJ15, BV15] between $\mathcal{FE}$ and $i\mathcal{O}$ since it incurs sub-exponential loss. Intuitively, a fresh approach that is not powerful enough to imply $i\mathcal{O}$ is essential to obtaining a polynomial-time reduction for this task. This is because $i\mathcal{O}$ inherently seems to require assumptions of sub-exponential strength.

We present a new construction of universal samplers directly from $\mathcal{FE}$. We also consider the task of constructing multiparty NIKE for an unbounded number of users based on $\mathcal{FE}$. As detailed later, this turns out to be slightly non-trivial even given the work of Hofheinz et al. This is because the definitions presented in [HJK$^+$14] are not completely suitable to deal with an unbounded number of users. To support unbounded number of users, we strengthen the work of [HJK$^+$14] and introduce a new security notion for universal samplers called *interactive simulation*. We present a construction of universal samplers based on $\mathcal{FE}$ that achieves this notion and gives us multiparty NIKE for unbounded number of users.

An overview of our approach. In the following sections, we present a detailed overview of our approach. We start by describing our construction of trapdoor permutations from poly-hard indistinguishability obfuscator and then discuss the technical challenges that arise in constructing it from poly-hard public-key $\mathcal{FE}$. We then review the notion of universal samplers [HJK$^+$14] and discuss the construction of multiparty NIKE for unbounded number of users. Finally, we describe our construction of universal samplers from functional encryption. Parts of the text in sections 1.1 and 1.2 are taken verbatim from [GPS15].

1.1 Trapdoor Permutations from $i\mathcal{O}$

Recall that a trapdoor permutation is a tuple of efficiently computable algorithms (F, I, Samp). F describes a permutation over some domain D and I (having access to a trapdoor) is the inversion algorithm i.e given a point x in the domain gives the pre-image of x under F. Samp is the sampler algorithm that samples a "random" point from the domain. The (standard) one-wayness property of the trapdoor permutation [GR13] requires that: given $F(x)$ where x is a random point output by Samp, no polynomial time algorithm can find x with non-negligible probability.

BPW idea. Recently Bitansky, Paneth and Wichs [BPW16] gave a construction of trapdoor permutations from indistinguishability obfuscation and one-way functions. We will now recall the main ideas behind their construction. The domain of the trapdoor permutation is a tuple

$(x, \mathsf{PRF}_S(x))$ where x is a κ-bit binary string and $\mathsf{PRF}_S(\cdot)$ is a pseudorandom function with key S. The domain can be thought of as consisting of two components: an index and a "signature" on the index. Intuitively, the permutation can be thought of as a cycle where the point $(x, \mathsf{PRF}_S(x))$ is connected to $(x+1, \mathsf{PRF}_S(x+1))$ and $(1^\kappa, \mathsf{PRF}_S(1^\kappa))$ is connected to $(0^\kappa, \mathsf{PRF}_S(0^\kappa))$.

The public key is an obfuscation of a circuit that on input (x, σ) checks the validity of σ and outputs the next point in the domain if σ is valid. On an invalid point, it outputs a special symbol $\perp$. The trapdoor is the PRF key S. Bitansky, Paneth and Wichs showed that if the underlying indistinguishability obfuscator is sub-exponentially secure then the it is hard to invert the image of a randomly sampled point from the domain. A high level overview of the proof strategy is to first "puncture" the public key at a random point u such that it outputs the special symbol $\perp$ on input $(u, \mathsf{PRF}_S(u))$ instead of $(u+1, \mathsf{PRF}_S(u+1))$. The public key is then iteratively punctured at points $u+1, u+2$ and so on until $i-1$ where $(i, \mathsf{PRF}_S(i))$ is the inversion challenge. The main observation that completes the proof is that once the public key is punctured at $i-1$, no adversary can invert the challenge with non-zero probability. This approach requires sub-exponentially secure indistinguishability obfuacator because it is restricted to puncturing the public key one point at a time in the exponentially large interval $[u+1, i-1]$.

GPS idea. Garg, Pandey and Srinivasan [GPS15] described a method to get rid of sub-exponential loss in the security reduction in basing hardness of the complexity class PPAD on indistinguishability obfuscation. We adapt their strategy to construct trapdoor permutation from polynomially hard indistinguishability obfuscation.

The main idea is to consider a domain that allows us to iteratively puncture the public key at a larger interval instead of a single point. The domain now includes κ signatures on the prefixes of the index x instead of a single signature. More formally, every point in the domain is a tuple $(x, \mathsf{PRF}_{S_1}(x_{[1]}), \mathsf{PRF}_{S_2}(x_{[2]}), \cdots, \mathsf{PRF}_{S_\kappa}(x_{[\kappa]}))$ where x is a κ-bit string, $S_1, \cdots, S_\kappa$ are independently chosen PRF keys and $x_{[i]}$ denotes the i-bit prefix of x. The public key is an obfuscation of a circuit that on input $(x, \sigma_1, \cdots, \sigma_\kappa)$ checks the validity of σ_i for all $i \in [\kappa]$ and outputs the next point $(x+1, \cdot, \cdots, \cdot)$ if all the signatures are valid. The trapdoor is given by $S_1, \cdots, S_\kappa$. The sampler is similar to Bitansky et al. construction: it is an obfuscation of a circuit that takes some randomness r, expands it using a pseudorandom generator and signs on all prefixes of the result.

To prove one-wayness of the construction, we start by puncturing the public key of the permutation at a random point u as before and iteratively expand the punctured interval so that it includes $i-1$. The crucial fact that enables a polynomial reduction is: if u has k trailing ones in its bit-representation, then it is possible to puncture the public key on the interval $[u+1, u+2^k]$ in a "single-stroke". This is based on the observations that:

1. The $\kappa - k$ bit prefix of $u+1$ is identical to the $\kappa - k$-bit prefix of all points in the interval $[u+1, u+2^k]$.

2. The signature on $\kappa - k$ bit prefix of $u+1$ is only needed to check for the validity of nodes belonging to the interval $[u+1, u+2^k]$ and need not be explicitly computed. This allows us to perform this check in an "encrypted" form which further enables us to puncture the public key in the interval $[u+1, u+2^k]$.

We then repeat this process by considering $u+2^k$ as the next point and puncture the intervals until we reach $i-1$. The number of intervals that need to be punctured until we reach $i-1$ is polynomially bounded. Metaphorically, the idea of having multiple signatures can be thought of

as "multiple-chains" coming out of every node that connects points at distances 1,2,4 and so on. The number of chains would be equal to one more than the number of trailing 1s in the binary representation of the node. Puncturing an interval is equivalent to cutting a chain of appropriate length.

We now discuss a few technicalities that arise while formalizing the above idea. In the above exposition, we have ignored the fact that an adversary attempting to invert a trapdoor challenge has access to the sampler. Our sampler has the trapdoor $(S_1, \cdots, S_\kappa)$ hardwired in its description which could potentially help the adversary in inverting the challenge. Hence, these keys have to be "punctured" using the punctured-programming approach of [SW14] carefully to enable puncturing the public key in the interval $[u+1, u+2^k]$. Another technical issue is that, the random point u at which the public key is initially punctured should not be "too-far" away from the challenge i. Otherwise, the sampler's image would fall in the range $[u, i-1]$ and we would not be able to puncture the public key on this interval as before. Bitansky, Paneth and Wichs [BPW16] overcame this difficulty by sampling u from a small-but-still-large-enough interval such that the sampler's image does not fall in the interval $[u, i-1]$ with overwhelming probability and we rely on polynomially hard pseudorandom generator. Yet another issue is that, while iteratively puncturing the public key we cannot always cut chains of the longest length as in [GPS15] because it could very well be the case that we overshoot $i-1$. To tackle this, we begin by cutting chains of increasing lengths and at some point we start cutting chains of smaller and smaller lengths until we reach $i-1$. The number of chains that must be cut is still polynomial in the security parameter.

1.2 Trapdoor Permutations from $\mathcal{FE}$

Garg, Pandey and Srinivasan in [GPS15] showed that it is possible to base hardness of PPAD relying on the security of polynomially hard public key functional encryption. We adapt their techniques to construct trapdoor permutation from public key functional encryption.

The domain of the permutation is exactly same as in the previous case i.e it consists of tuples of the form $(x, \mathsf{PRF}_{S_1}(x_{[1]}), \mathsf{PRF}_{S_1}(x_{[2]}), \cdots, \mathsf{PRF}_{S_1}(x_{[\kappa]}))$. The circuit implementing the public key must be able to check the validity of the input signatures and output the signatures on the next point in the domain if the signatures are valid. We give out an "obfuscation" of such a circuit where the obfuscation is emulated using public key functional encryption based techniques from [BV15].

We now give a high level overview of our public key construction. The public key consists of $\kappa+1$ function keys. The first κ function keys implement a bit-extension function i.e on input y outputs a functional encryption of $y\|0$ and $y\|1$. These keys are used to compute a functional encryption of the first component of a domain point i.e x. The last function key implements a function that outputs encryptions of signatures on the next point $(x+1)$ using the signatures of x as the secret key. Intuitively, an evaluator can obtain the valid signatures on the next point $x+1$ if and only if he has valid signatures on x. In order to enable the last function key to compute signatures on the current point (x) as well as next point $(x+1)$, the bit extension functions compute a set of carefully constructed *prefix-punctured* PRF keys [GPS15] that are passed to the next level.

We note that our construction of public key of the permutation is almost identical [1] to the Garg et al.'s [GPS15] construction of successor circuit and we refer the reader to [GPS15] for a detailed

[1] For readers familiar with the work of Garg et al., the only differences being that instead of outputting the special symbol SOLVED on a valid input $(1^\kappa, \cdot, \cdots, \cdot)$ we output the node $(0^\kappa, \cdot, \cdots, \cdot)$ and in how we are puncturing the permutation function at a random point.

exposition of the construction. We now move on to our construction of the sampler.

Challenge in Sampler Construction. Recall that the sampler in the previous construction first expands the randomness using a length doubling pseudorandom generator PRG and then signs on all prefixes of the result. We first observe that we cannot directly extend the techniques introduced by Garg et al. in [GPS15] to construct the sampler from functional encryption. This is because the techniques crucially depend on passing down an appropriate set of prefix punctured keys that are punctured along the prefixes of the point (that we want to sign) to the lower levels. For the case of the sampler, we need to pass down the set of prefix punctured keys that are punctured along the prefixes of $\mathsf{PRG}(r)$. But the problem is we cannot compute $\mathsf{PRG}(r)$ until we parse r in the encryption tree and thus incurring a sub-exponential loss in the security reduction!

In order to over come this challenge, we modify our sampler construction so that instead of signing on $\mathsf{PRG}(r)$ it now signs on $r\|K_r$ where K is a fresh prefix puncturable key and K_r denotes K prefix punctured at r. The "obfuscated" sampler consists of $\kappa/2+1$ function keys similar to the construction of public key where the first $\kappa/2$ implement the bit extension functions that are used to obtain an functional encryption of r. The bit extension functions pass down a set of prefix puncturings of the keys $S_1, \cdots, S_\kappa$ that are punctured on the prefixes of r. These keys would be used for computing the signatures on the prefixes in the last level. Additionally, it passes down prefix puncturing (punctured along the prefixes of r) of a new PRF key K that maps $\kappa/2$ bits to $\kappa/2$ bits. The last function key (numbered $\kappa/2+1$) implements a function that takes in r, the set of keys $S_1, \cdots, S_\kappa$ that are prefix-punctured at r and the new PRF key K that is also prefix punctured at r and outputs $(r\|K_r, \mathsf{PRF}_{S_1}((r\|K_r)_{[1]}), \mathsf{PRF}_{S_2}((r\|K_r)_{[2]}), \cdots, \mathsf{PRF}_{S_\kappa}((r\|K_r)_{[\kappa]}))$. Observe that the final function can compute this output given r and the set of prefix punctured keys. This sampler "conforms" well with the prefix puncturing techniques of Garg et al. [GPS15] and has the required sparseness of images guarantee that is required to prove one-wayness. But the issue with this construction is that it is unclear on how to show that the sampler samples pseudorandom points.

To fix this issue, we interpret the randomness r that is given as input to the sampler as a public key and output the encryptions of the signatures on prefixes of $r\|\mathsf{PRF}_K(r)$ using r as the public key. That is to sample a point in the domain, we generate a public key pk, secret key pair sk and run the sampler on the public key to obtain the encryptions under pk of the signatures on the prefixes of $pk\|K_{pk}$ which are then decrypted using the secret key sk. Using the prefix puncturing technique of Garg et al. [GPS15] and relying on the semantic security of public key encryption we are able to show the pseudorandomness of samples [2].

Remark 1 *Our construction of TDP from FE is weaker in comparison to our construction from iO (and the construction of Bitansky et al. in [BPW16]) since given K the output of the sampler is not pseudorandom. Thus, our construction of TDP can only be used in applications where the system parameters are generated by honest parties. For example, our TDP cannot be used to guarantee receiver privacy in the Oblivious Transfer (OT) protocol of Even et al. in [EGL85].*

[2] Additionally, we need the public key encryption scheme to have random public keys in order to prove that the output of the sampler is a pseudorandom string.

1.3 Universal Samplers from $\mathcal{FE}$

Recall that universal sampler is an algorithm that takes as input the description of a sampling procedure (say, the sampling procedure for the common parameters of some protocol) and outputs a sample from that procedure (a set of parameters for that protocol). The algorithm is deterministic, so that anyone running the protocol on a given sampling procedure gets the same sampled parameters. Yet the generated parameters should be "as good as" a freshly generated set of parameters.

Unfortunately, defining a satisfactory notion of "as good as" above is non-trivial. Hofheinz et al. give two definitions: a static definition which only remains secure for a bounded number of generated parameters, as well as an adaptive definition that is inherently tied to the random oracle model, but allows for an unbounded number of generated parameters. They show how to use the stronger definitions to realize primitives such as adaptively secure multiparty non-interactive key exchange (NIKE) and broadcast encryption.

In this work, we focus on the standard model, and here we review the static standard-model security definition for universal samplers. Fix some bound k on the number of generated parameters. Intuitively, the k-time static security definition says that up to k freshly generated parameters $s_1, \ldots, s_k$ for sampling algorithms $C_1, \ldots, C_k$ can be embedded into the universal sampler without detection. Thus, if the sampler is used on any of the sampling algorithms C_i, the generated output will be the fresh sample s_i. Formally, there is a simulator Sim that takes as input up to k sampler/sample pairs (C_i, s_i), and outputs a simulated universal sampler $\mathsf{Sampler}$, such that $\mathsf{Sampler}(C_i) = s_i$. As long as the s_i are fresh samples from C_i, the simulated universal sampler will be indistinguishable from a honestly generated sampler.

Application: Multiparty NIKE. From the static definition above, it is straightforward to obtain a statically secure multiparty NIKE protocol analogous to the adaptive protocol of Hofheinz et al. [HJK+14]. Each party simply publishes a public key pk_i for a public key encryption scheme, and keeps the corresponding secret key sk_i hidden. Then to generate the shared group key, all parties run $\mathsf{Sampler}$ on the sampler $C_{\mathsf{pk}_1,\ldots,\mathsf{pk}_n}$. Here, $C_{\mathsf{pk}_1,\ldots,\mathsf{pk}_n}$ is the randomized procedure that generates a random string K, and encrypts K under each of the public keys $\mathsf{pk}_1, \ldots, \mathsf{pk}_n$, resulting in n ciphertexts $c_1, \ldots, c_n$ which it outputs. Then party i decrypts c_i using sk_i. The result is that all parties in the protocol learn K.

Meanwhile, an eavesdropper who does not know any of the secret keys will only have the public keys, the sampler, and thus the ciphertexts c_i outputted by the sampler. The proof that the eavesdropper will not learn K is as follows. First, we consider a hybrid experiment where K is generated uniformly at random, and the universal sampler is simulated on sampler $C_{\mathsf{pk}_1,\ldots,\mathsf{pk}_n}$, and sample $s = (c_1, \ldots, c_n)$, where c_i are fresh encryptions of K under each of the public keys pk_i. 1-time static security of the universal sampler implies that this hybrid is indistinguishable to the adversary from the real world. Next, we change each of the c_i to encrypt 0. Here, indistinguishability follows from the security of the public key encryption scheme. In this final hybrid, the view of the adversary is independent of the shared secret key K, and security follows.

Unbounded multiparty NIKE. One limitation of the protocol above is that the number of users must be a priori bounded. There are several reasons for this, the most notable being that in order to simulate, the universal sampler must be as large as the sample $s = (c_1, \ldots, c_n)$, which grows with n. Thus, once the universal sampler is published, the number of users is capped.

In order to get around this issue, we change the sampling procedure $C_{\mathsf{pk}_1,\ldots,\mathsf{pk}_n}$ fed into the universal sampler. Instead, we feed in circuits of the form $D_{\mathsf{pk},\mathsf{pk}'}$, which generate a new secret and public key $\mathsf{sk}'', \mathsf{pk}''$, encrypt sk'' under both pk and pk', and output both encryptions as well as the new public key pk''. A group of users with public keys $\mathsf{pk}_1, \ldots, \mathsf{pk}_n$ then generates the shared key in an iterative fashion as follows. Run the universal sampler on $D_{\mathsf{pk}_1,\mathsf{pk}_2}$, obtaining a new public key pk'_3, as well as encryptions of the corresponding secret key sk'_3 under both $\mathsf{pk}_1, \mathsf{pk}_2$. Notice that users 1 and 2 can both recover sk'_3 using their secret keys. Then run the universal sampler on $D_{\mathsf{pk}_3,\mathsf{pk}'_3}$, obtaining a new public key pk'_4 and encryptions of the corresponding secret key sk'_4. Notice that user 3 can recover sk'_4 by decrypting the appropriate ciphertext using sk_3, and users 1 and 2 can recover sk'_4 by decrypting the other ciphertext using sk'_3. Continue in this way until public key pk'_{n+1} is generated, and all users 1 through n recover the corresponding secret key sk'_{n+1}. Set sk'_{n+1} to be the shared secret key.

For security, since an eavesdropper does no know any of the secret keys and the ciphertexts are "as good as" fresh ciphertexts, he should not be able to decrypt any of the ciphertexts in the procedure above. However, turning this intuition into a security proof using the static notion of security is problematic. The straightforward approach requires constructing a simulated $\mathsf{Sampler}$ where the outputs on each of the circuits $D_{\mathsf{pk}_i,\mathsf{pk}'_i}$ are fresh samples. Then, each of the ciphertexts in the samples are replaced with encryptions of 0 (instead of the correct secret decryption key). However, as there are n such circuits, a standard incompressibility argument shows that $\mathsf{Sampler}$ must grow linearly in n. Thus again, once the universal sampler is published, the number of users is capped.

Simulating at fewer points. To get around this issue, devise a sequence of hybrids where in each hybrid, we only need replace $\log n$ outputs of the sampler with fresh samples. The core idea is the following. Say that a circuit $D_{\mathsf{pk}_i,\mathsf{pk}'_i}$ has been "treated" if the public key pk'_{i+1} outputted by the universal sampler is freshly sampled and the corresponding ciphertexts are changed to encrypt 0 (instead of the secret key sk'_{i+1}) . We observe that to switch circuit $D_{\mathsf{pk}_i,\mathsf{pk}'_i}$ from untreated to treated, circuit $D_{\mathsf{pk}_{i-1},\mathsf{pk}'_{i-1}}$ needs to currently be treated so that the view of the adversary is independent of the secret key sk'_i. However the status of all the other circuits is irrelevant. Moreover, once we have treated $D_{\mathsf{pk}_i,\mathsf{pk}'_i}$, we can potentially "untreat" $D_{\mathsf{pk}_{i-1},\mathsf{pk}'_{i-1}}$ and reset its ciphertexts to the correct values, assuming $D_{\mathsf{pk}_{i-2},\mathsf{pk}'_{i-2}}$ is currently treated. Our goal is to start from none of the circuits being treated, and arrive at a hybrid where $D_{\mathsf{pk}_n,\mathsf{pk}'_n}$ is treated, which implies that the view of the adversary is independent of the shared secret sk_{n+1}.

This gives rise to an interesting algorithmic problem. The goal is to get a pebble at position n, where the only valid moves are (1) placing or removing a pebble at position 1, or (2) placing or removing a pebble at position i provided there is currently a pebble at position $i-1$. We desire to get a pebble at position n while minimizing the number of pebbles used at any time. The trivial solution is to place a pebble at 1, then 2, and so on, requiring n pebbles. We show a pebbling scheme that gets a pebble to position n using only $O(\log n)$ pebbles by removing certain pebbles as we go. The pebbling scheme is exactly same as the one used in [Ben89] in the context of reversible computation. Moreover, the pebbling scheme is efficient, in the sense that the number of moves is polynomial in n.

Using our pebbling algorithm, we derive a sequence of hybrids corresponding to each move in the algorithm. Thus we show that the number of circuits that need simulating can be taken to be $O(\log n)$.

A new universal sampler definition. Unfortunately, we run into a problem when trying to base security on the basic static sampler definition of Hofheinz et al. [HJK+14]. The issue stems from the fact that the simulator in the static definition requires knowing all of the circuits $D_{\mathsf{pk}_i,\mathsf{pk}'_i}$ up front. However, in our pebbling approach, some of the pk'_i (and thus the $D_{\mathsf{pk}_i,\mathsf{pk}'_i}$) are determined by the sampler $\mathsf{Sampler}$ - namely, all the pk'_i for which $D_{\mathsf{pk}_{i-1},\mathsf{pk}'_{i-1}}$ is "untreated". Thus we encounter a circularity where we need to know $\mathsf{Sampler}$ to compute the circuit $D_{\mathsf{pk}_i,\mathsf{pk}'_i}$, but we need $D_{\mathsf{pk}_i,\mathsf{pk}'_i}$ in order to simulate the $\mathsf{Sampler}$.

To get around this issue, we devise a new security notion for universal samplers that allows for *interactive simulation*. That is, *before* the simulator outputs $\mathsf{Sampler}$, we are allowed to query it on various inputs, learning what the output of the sampler will be on that input (called as the *read* query). Moreover, we are allowed to feed circuit/sample pairs C, s (called as *write* query) interactively, potentially *after* seeing some of the sample outputs, and the simulator will guarantee that the simulated $\mathsf{Sampler}$ will output s on C. For security, we require that for a statically chosen query index i^* and a circuit C^* the simulator's outputs in the following two cases are computationally indistinguishable:

1. i^{*th} query is a read query on C^*.
2. i^{*th} query is a write query on C^*, s^* where s^* is fresh random sample from C^*.

This new definition allows us to avoid the circularity above and complete the security proof for our NIKE protocol.

Construction. Before we describe our construction from $\mathcal{FE}$, we first describe a construction from $i\mathcal{O}$ that satisfies the above definition of interactive simulation. The universal sampler is an obfuscation of a circuit that has a puncturable PRF key K hardwired in its description and on input C outputs $C(;\mathsf{PRF}_K(C))$ i.e it uses the PRF key to generate the random coins. This is precisely the same construction as given by Hofheinz et al. [HJK+14] for the static security case. To prove that this construction satisfies the stronger definition of interactive simulation we construct a simulator that works as follows. It first samples a fresh PRF key K' and answers the read queries using it. At the end of the simulation, it outputs an obfuscation of a circuit that has the PRF key K' as well as (C_i, s_i) for every write query made by the adversary hardwired in its description. When run on input C where C is one of the write queries, it outputs the corresponding s. On other inputs, it outputs $C(;\mathsf{PRF}_{K'}(C))$.

The security is shown via a hybrid argument. The initial hybrid corresponds to the output of the simulator when the challenge query (made at index i^*) is a write query on (C_{i^*}, s_{i^*}) where s_{i^*} is a fresh random sample from C_{i^*}. We first change the obfuscated circuit to have the PRF key K' punctured at C_{i^*}. This is possible since the circuit does not use K' to compute the output on C_{i^*}. Relying on the security of puncturable PRF we change s_{i^*} from $C_{i^*}(;r)$ where r is random string to $C_{i^*}(;\mathsf{PRF}_{K'}(C_{i^*}))$. We then unpuncture the key K' and finally remove C_{i^*}, s_{i^*} from the hardwired list.

We adapt the above construction from $i\mathcal{O}$ to the $\mathcal{FE}$ setting using techniques from [BV15, GPS15]. Recall that the "obfuscated" universal sampler consists of $\ell+1$ (ℓ is the maximum size of the input circuit) function keys (where each function key computes a bit extension function) along with an initial ciphertext c_ϕ that encrypts the empty string ϕ and a prefix puncturable PRF key K. These bit extension functions form a natural binary tree structure and "parsing" an input circuit

C corresponds to traveling along the path from the root to the leaf labeled C. Each node x along the path from the root to C contains the key K prefix punctured at x. The prefix punctured PRF key appearing at the leaf C is precisely equal to the PRF value at C and we use this to generate a "pseudorandom" sample from C.

We are now ready to describe the construction of our simulator. As in the $i\mathcal{O}$ case the simulator samples a random PRF key K' and uses it to answer the read queries made by the adversary. Recall that for every write query (C_i, s_i) the adversary makes, the simulator must ensure that the sampler on C_i outputs s_i. This is accomplished by "puncturing" the underlying binary tree along path C_i. That is, the simulator "forces" the function keys at every level to output a pre-computed value instead of the bit-extension if the input to the function matches with a prefix of C_i. At the final level, if the input matches C_i then the function outputs s_i. This is enabled by triggering a hidden "trapdoor" thread in the function keys using techniques illustrated in [ABSV15, GPS15]. The set of precomputed values is first encrypted under a symmetric key sk and are hardwired in the description of bit-extension function in each level. The symmetric key sk is encrypted in the initial ciphertext c_ϕ along with the empty string and the PRF key. The trapdoor thread (that is triggered only along the write query paths) uses this secret key sk to decrypt the hardcoded ciphertext and outputs the appropriate pre-computed value. To complete the security proof, we want to prove that we can indistinguishably "puncture" the binary tree along a new path C_i^* and output s_i^* which is a fresh random sample from C_i^* at the leaf. Recall that in the construction of Garg et al. in [GPS15] a single secret key sk is used to for computing the encryptions of pre-computed values along multiple paths. But having a single secret key does not allow us to puncture along a new path C_i^* as this secret key already appears in the initial ciphertext c_ϕ. Hence we cannot rely on the semantic security of symmetric key encryption to augument the pre-computed values to include values along the path C_i^*. In order to get around this issue, we use multiple secret keys: one for each write query [3] which enables use to puncture along a new path C_i^*.

2 Preliminaries

κ denotes the security parameter. A function $\mu(\cdot) : \mathbb{N} \to \mathbb{R}^+$ is said to be negligible if for all polynomials $\mathsf{poly}(\cdot)$, $\mu(\kappa) < \frac{1}{\mathsf{poly}(\kappa)}$ for large enough κ. For a probabilistic algorithm $\mathcal{A}$, we denote by $\mathcal{A}(x; r)$ the output of $\mathcal{A}$ on input x with the content of the random tape being r. We will omit r when it is implicit from the context. We denote $y \leftarrow \mathcal{A}(x)$ as the process of sampling y from the output distribution of $\mathcal{A}(x)$ with a uniform random tape. For a finite set S, we denote $x \xleftarrow{\$} S$ as the process of sampling x uniformly from the set S. We model non-uniform adversaries $\mathcal{A} = \{\mathcal{A}_\kappa\}$ as circuits such that for all κ, $\mathcal{A}_\kappa$ is of size $p(\kappa)$ where $p(\cdot)$ is a polynomial. We will drop the subscript κ from the adversary's description when it is clear from the context. We will also assume that all algorithms are given the unary representation of security parameter 1^κ as input and will not mention this explicitly when it is clear from the context. We will use PPT to denote Probabilistic Polynomial Time algorithm. We denote $[\kappa]$ to be the set $\{1, \cdots, k\}$. We will use $\mathsf{negl}(\cdot)$ to denote an unspecified negligible function and $\mathsf{poly}(\cdot)$ to denote an unspecified polynomial.

A binary string $x \in \{0,1\}^\kappa$ is represented as $x_1 \cdots x_\kappa$. x_1 is the most significant (or the highest order bit) and x_κ is the least significant (or the lowest order bit). The i-bit prefix $x_1 \cdots x_i$ of the

[3] In the security definition, the number of write queries that an adversary could make is apriori bounded. On the otherhand, the adversary could make an unbounded number of read queries. Thus, we can fix the number of secret keys to be sampled at the time of setup.

binary string x is denoted by $x_{[i]}$. We use $x\|y$ to denote concatenation of binary strings x and y. We say that a binary string y is a prefix of x if and only if there exists a string $z \in \{0,1\}^*$ such that $x = y\|z$.

Injective Pseudo Random Generator. We give the definition of an injective Pseudo Random Generator PRG.

Definition 2 *An injective pseudo random generator* PRG *is a deterministic polynomial time algorithm with the following properties:*

- ***Expansion:*** *There exists a polynomial $\ell(\cdot)$ (called as the expansion factor) such that for all κ and $x \in \{0,1\}^\kappa$, $|\mathsf{PRG}(x)| = \ell(\kappa)$.*
- ***Pseudorandomness:*** *For all κ and for all poly sized adversaries $\mathcal{A}$,*

$$|\Pr[\mathcal{A}(\mathsf{PRG}(U_\kappa)) = 1] - \Pr[\mathcal{A}(U_{\ell(\kappa)}) = 1]| \leq \mathsf{negl}(\kappa)$$

  where U_i denotes the uniform distribution on $\{0,1\}^i$.

- ***Injectivity:*** *For every κ and for all $x, x' \in \{0,1\}^\kappa$ such that $x \neq x'$, $\mathsf{PRG}(x) \neq \mathsf{PRG}(x')$.*

We in fact need an additional property from an injective PRG. Let us consider PRG where the expansion factor (or the output length) is given by $2 \cdot \ell(\cdot)$. Let us denote the first $\ell(\cdot)$ bits of the output of the PRG by the function PRG_0 and the next $\ell(\cdot)$ bits of the output of the PRG by PRG_1.

Definition 3 *A pseudorandom generator* PRG *is said to be left half injective if for every κ and for all $x, x' \in \{0,1\}^\kappa$ such that $x \neq x'$, $\mathsf{PRG}_0(x) \neq \mathsf{PRG}_0(x')$.*

Note that left half injective PRG is also an injective PRG. We note that the standard construction of pseudorandom generator for arbitrary polynomial stretch from one-way permutations is left half injective. For completeness, we state the construction:

Lemma 4 *Assuming the existence of one-way permutations, there exists a pseudorandom generator that is left half injective.*

Proof Let $f : \{0,1\}^\kappa \to \{0,1\}^\kappa$ be a one-way permutation with hardcore predicate $B : \{0,1\}^\kappa \to \{0,1\}$ [GL89]. Let G be an algorithm defined as follows: On input $x \in \{0,1\}^\kappa$, $G(x) = f^n(x)\|B(x)\|$ $B(f(x)) \cdots B(f^{n-1}(x))$ where $n = 2\ell(\kappa) - \kappa$. Clearly, $|G(x)| = 2\ell(\kappa)$. The pseudorandomness property of $G(\cdot)$ follows from the security of hardcore bit. The left half injectivity property follows from the observation that f^n is a permutation. ■

Puncturable pseudorandom Function. We recall the notion of puncturable pseudorandom function from [SW14]. The construction of pseudorandom function given in [GGM86] satisfies the following definition [BW13, KPTZ13, BGI14].

Definition 5 *A puncturable pseudorandom function $\mathcal{PRF}$ is a tuple of PPT algorithms* $(\mathsf{KeyGen}_{\mathcal{PRF}}, \mathsf{PRF}, \mathsf{Punc})$ *with the following properties:*

- ***Efficiently Computable:*** *For all κ and for all $S \leftarrow \mathsf{KeyGen}_{\mathcal{PRF}}(1^\kappa)$, $\mathsf{PRF}_S : \{0,1\}^{\mathsf{poly}(\kappa)} \rightarrow \{0,1\}^\kappa$ is polynomial time computable.*

- ***Functionality is preserved under puncturing:*** *For all κ, for all $y \in \{0,1\}^\kappa$ and $\forall x \neq y$,*

$$\Pr[\mathsf{PRF}_{S\{y\}}(x) = \mathsf{PRF}_S(x)] = 1$$

  where $S \leftarrow \mathsf{KeyGen}_{\mathcal{PRF}}(1^\kappa)$ and $S\{y\} \leftarrow \mathsf{Punc}(S, y)$.

- ***pseudorandomness at punctured points:*** *For all κ, for all $y \in \{0,1\}^\kappa$, and for all poly sized adversaries $\mathcal{A}$*

$$|\Pr[\mathcal{A}(\mathsf{PRF}_S(y), S\{y\}) = 1] - \Pr[\mathcal{A}(U_\kappa, S\{y\}) = 1]| \leq \mathsf{negl}(\kappa)$$

  where $S \leftarrow \mathsf{KeyGen}_{\mathcal{PRF}}(1^\kappa)$, $S\{y\} \leftarrow \mathsf{Punc}(S, y)$ and U_κ denotes the uniform distribution over $\{0,1\}^\kappa$.

Indistinguishability Obfuscator. We now define Indistinguishability obfuscator from [BGI+12, GGH+13b].

Definition 6 *A PPT algorithm $i\mathcal{O}$ is an indistinguishability obfuscator for a family of circuits $\{C_\kappa\}_\kappa$ that satisfies the following properties:*

- ***Correctness:*** *For all κ and for all $\mathcal{C} \in C_\kappa$ and for all x,*

$$\Pr[i\mathcal{O}(\mathcal{C})(x) = \mathcal{C}(x)] = 1$$

  where the probability is over the random choices of $i\mathcal{O}$.

- ***Security:*** *For all $\mathcal{C}_0, \mathcal{C}_1 \in C_\kappa$ such that for all x, $\mathcal{C}_0(x) = \mathcal{C}_1(x)$ and for all poly sized adversaries $\mathcal{A}$,*

$$|\Pr[\mathcal{A}(i\mathcal{O}(\mathcal{C}_0)) = 1] - \Pr[\mathcal{A}(i\mathcal{O}(\mathcal{C}_1)) = 1]| \leq \mathsf{negl}(\kappa)$$

Functional Encryption. We recall the notion of functional encryption with selective indistinguishability based security [BSW11, O'N10].

A functional encryption $\mathcal{FE}$ is a tuple of PPT algorithms $(\mathsf{FE.Setup}, \mathsf{FE.Enc}, \mathsf{FE.KeyGen}, \mathsf{FE.Dec})$ with the message space $\{0,1\}^*$ having the following syntax:

- $\mathsf{FE.Setup}(1^\kappa)$: Takes as input the unary encoding of the security parameter κ and outputs a public key PK and a master secret key MSK.

- $\mathsf{FE.Enc}_{PK}(m)$: Takes as input a message $m \in \{0,1\}^*$ and outputs an encryption C of m under the public key PK.

- $\mathsf{FE.KeyGen}(MSK, f)$: Takes as input the master secret key MSK and a function f (given as a circuit) as input and outputs the function key FSK_f.

- $\mathsf{FE.Dec}(\mathsf{FSK}_f, C)$: Takes as input the function key FSK_f and the ciphertext C and outputs a string y.

Definition 7 (Correctness) *The functional encryption scheme $\mathcal{FE}$ is correct if for all κ and for all messages $m \in \{0,1\}^*$,*

$$\Pr\left[y = f(m) \middle| \begin{array}{l} (PK, MSK) \leftarrow \mathsf{FE.Setup}(1^\kappa) \\ C \leftarrow \mathsf{FE.Enc}_{PK}(m) \\ \mathsf{FSK}_f \leftarrow \mathsf{FE.KeyGen}(MSK, f) \\ y \leftarrow \mathsf{FE.Dec}(\mathsf{FSK}_f, C) \end{array} \right] = 1$$

Definition 8 (Selective Security) *For all κ and for all poly sized adversaries $\mathcal{A}$,*

$$\left|\Pr[\mathsf{Expt}_{1^\kappa,0,\mathcal{A}} = 1] - \Pr[\mathsf{Expt}_{1^\kappa,1,\mathcal{A}} = 1]\right| \leq \mathsf{negl}(\kappa)$$

where $\mathsf{Expt}_{1^\kappa,b,\mathcal{A}}$ is defined below:

- ***Challenge Message Queries:*** *The adversary $\mathcal{A}$ outputs two messages m_0, m_1 such that $|m_0| = |m_1|$ to the challenger.*
- *The challenger samples $(PK, MSK) \leftarrow \mathsf{FE.Setup}(1^\kappa)$ and generates the challenge ciphertext $C \leftarrow \mathsf{FE.Enc}_{PK}(m_b)$. It then sends (PK, C) to $\mathcal{A}$.*
- ***Function Queries****: $\mathcal{A}$ submits function queries f to the challenger. The challenger responds with $\mathsf{FSK}_f \leftarrow \mathsf{FE.KeyGen}(MSK, f)$.*
- *If $\mathcal{A}$ makes a query f to functional key generation oracle such that $f(m_0) \neq f(m_1)$, output of the experiment is $\perp$. Otherwise, the output is b' which is the output of $\mathcal{A}$.*

Remark 9 *We say that the functional encryption scheme $\mathcal{FE}$ is **single-key, selectively secure** if the adversary $\mathcal{A}$ in $\mathsf{Expt}_{1^\kappa,b,\mathcal{A}}$ is allowed to query the functional key generation oracle $\mathsf{FE.KeyGen}(MSK, \cdot)$ on a single function f.*

Definition 10 (Compactness, [AJS15, BV15, AJ15]) *The functional encryption scheme $\mathcal{FE}$ is said to be compact if for all $\kappa \in \mathbb{N}$ and for all $m \in \{0,1\}^*$ the running time of the encryption algorithm $\mathsf{FE.Enc}$ is $\mathsf{poly}(\kappa, |m|)$.*

Bitansky et al. in [BV15] and Ananth et al. in [AJS15] show a generic transformation from any collusion-resistant $\mathcal{FE}$ for general circuits where the ciphertext size is independent of the number of collusions (but may depend arbitrarily on the circuit parameters) to a compact $\mathcal{FE}$ for general circuits. The property that the ciphertext size does not depend on the number of collusion is referred as *collusion-succinctness.*

Lemma 11 ([BV15, AJS15]) *Assuming the existence of selectively secure collusion-resistant functional encryption with collusion-succinct ciphertexts, there exists a selectively secure compact functional encryption scheme.*

Symmetric Key Encryption. A Symmetric-Key Encryption scheme $\mathcal{SKE}$ is a tuple of algorithms $(\mathsf{SK.KeyGen}, \mathsf{SK.Enc}, \mathsf{SK.Dec})$ with the following syntax:

- $\mathsf{SK.KeyGen}(1^\kappa)$: Takes as input an unary encoding of the security parameter κ and outputs a symmetric key SK.

- $\mathsf{SK.Enc}_{SK}(m)$: Takes as input a message $m \in \{0,1\}^*$ and outputs an encryption C of the message m under the symmetric key SK.
- $\mathsf{SK.Dec}_{SK}(C)$: Takes as input a ciphertext C and outputs a message m'.

We say that $\mathcal{SKE}$ is *correct* if for all κ and for all messages $m \in \{0,1\}^*$, $\Pr[\mathsf{SK.Dec}_{SK}(C) = m] = 1$ where $SK \leftarrow \mathsf{SK.KeyGen}(1^\kappa)$ and $C \leftarrow \mathsf{SK.Enc}_{SK}(m)$.

Definition 12 *For all κ and for all polysized adversaries $\mathcal{A}$,*

$$\left|\Pr[\mathsf{Expt}_{1^\kappa,0,\mathcal{A}} = 1] - \Pr[\mathsf{Expt}_{1^\kappa,1,\mathcal{A}} = 1]\right| \leq \mathsf{negl}(\kappa)$$

where $\mathsf{Expt}_{1^\kappa,b,\mathcal{A}}$ is defined below:

- ***Challenge Message Queries:*** *The adversary $\mathcal{A}$ outputs two messages m_0 and m_1 such that $|m_0| = |m_1|$ for all $i \in [n]$.*
- *The challenger samples $SK \leftarrow \mathsf{SK.KeyGen}(1^\kappa)$ and generates the challenge ciphertext C where $C \leftarrow \mathsf{SK.Enc}_{SK}(m_b)$. It then sends C to $\mathcal{A}$.*
- *Output is b' which is the output of $\mathcal{A}$.*

Public Key Encryption. A public-key Encryption scheme $\mathcal{PKE}$ is a tuple of algorithms ($\mathsf{PK.KeyGen}$, $\mathsf{PK.Enc}, \mathsf{PK.Dec}$) with the following syntax:

- $\mathsf{PK.KeyGen}(1^\kappa)$: Takes as input an unary encoding of the security parameter κ and outputs a public key, secret key pair (pk, sk).
- $\mathsf{PK.Enc}_{pk}(m)$: Takes as input a message $m \in \{0,1\}^*$ and outputs an encryption C of the message m under the public key pk.
- $\mathsf{PK.Dec}_{sk}(C)$: Takes as input a ciphertext C and outputs a message m'.

We say that $\mathcal{PKE}$ is *correct* if for all κ and for all messages $m \in \{0,1\}^*$, $\Pr[\mathsf{PK.Dec}_{sk}(C) = m] = 1$ where $(pk, sk) \leftarrow \mathsf{PK.KeyGen}(1^\kappa)$ and $C \leftarrow \mathsf{PK.Enc}_{pk}(m)$.

Definition 13 *For all κ and for all polysized adversaries $\mathcal{A}$ and for all messages $m_0, m_1 \in \{0,1\}^*$ such that $|m_0| = |m_1|$,*

$$|\Pr[\mathcal{A}(pk, \mathsf{PK.Enc}_{pk}(m_0)) = 1] - \Pr[\mathcal{A}(pk, \mathsf{PK.Enc}_{pk}(m_1)) = 1]| \leq \mathsf{negl}(\kappa)$$

where $(pk, sk) \leftarrow \mathsf{PK.KeyGen}(1^\kappa)$.

We require an additional property of the public key which is described below. We assume that $|pk| = \kappa$ where $(pk, sk) \leftarrow \mathsf{PK.KeyGen}(1^\kappa)$.

Definition 14 *A $\mathcal{PKE}$ is said to have random public keys if, $\{pk\}_\kappa \simeq \{U_\kappa\}_\kappa$ where $(pk, sk) \leftarrow \mathsf{PK.KeyGen}(1^\kappa)$ and U_κ is the uniform distribution over $\{0,1\}^\kappa$.*

The public key encryption due to El-Gamal [Gam85] based on DDH assumption satisfies the above property. We note that the public key encryption system from *Learning with Errors assumption* (LWE) due to Regev [Reg09] (having public keys that are computationally indistinguishable from random elements) is sufficient for our purposes.

Prefix Puncturable pseudorandom Functions. We now define the notion of prefix puncturable pseudorandom function PPRF which is satisfied by the construction of the pseudorandom function in [$GGM86$].

Definition 15 *A prefix puncturable pseudorandom function $\mathcal{PPRF}$ is a tuple of PPT algorithms* $(\mathsf{KeyGen}_{\mathcal{PPRF}}, \mathsf{PrefixPunc})$ *satisfying the following properties:*

- ***Functionality is preserved under* repeated *puncturing:*** *For all κ, for all $y \in \cup_{k=0}^{\mathsf{poly}(\kappa)}\{0,1\}^k$ and for all $x \in \{0,1\}^{\mathsf{poly}(\kappa)}$ such that there exists a $z \in \{0,1\}^*$ s.t. $x = y\|z$,*

$$\Pr[\mathsf{PrefixPunc}(\mathsf{PrefixPunc}(S, y), z) = \mathsf{PrefixPunc}(S, x)] = 1$$

  where $S \leftarrow \mathsf{KeyGen}_{\mathcal{PPRF}}$.

- ***Pseudorandomness at punctured prefix**: For all κ, for all $x \in \{0,1\}^{\mathsf{poly}(\kappa)}$, and for all poly sized adversaries $\mathcal{A}$*

$$|\Pr[\mathcal{A}(\mathsf{PrefixPunc}(S, x), \mathsf{Keys}) = 1] - \Pr[\mathcal{A}(U_\kappa, \mathsf{Keys}) = 1]| \leq \mathsf{negl}(\kappa)$$

  where $S \leftarrow \mathsf{KeyGen}_{\mathcal{PRF}}(1^\kappa)$ and $\mathsf{Keys} = \{\mathsf{PrefixPunc}(S, x_{[i-1]}\|(1-x_i))\}_{i\in[\mathsf{poly}(\kappa)]}$.

Notation. For brevity of notation, we will be denoting $\mathsf{PrefixPunc}(S, y)$ by S_y. For a key S_i (indexed by i), we will use $S_{i,y}$ to denote $\mathsf{PrefixPunc}(S_i, y)$.

3 TDP from IO in poly loss

We now give the definition of Trapdoor permutation with pseudorandom sampling which is a weakened notion than the traditional uniform sampling. This definition is equivalent to the one given in [BPW16].

Definition 16 ([GR13, BPW16]) *An efficiently computable family of functions:*

$$\mathcal{TDP} = \{\mathsf{TDP}_{PK} : D_{PK} \to D_{PK} \text{ and } PK \in \{0,1\}^{\mathsf{poly}(\kappa)}\}$$

over the domain D_{PK} associated with associated (probabilistic) $(\mathsf{KeyGen}, \mathsf{SampGen})$ *algorithms is a (standard) trapdoor permutation if it satisifies:*

1. ***Trapdoor Invertibility:** For any $(PK, SK) \leftarrow \mathsf{KeyGen}(1^\kappa)$, TDP_{PK} is a permutation over D_{PK}. For any $y \in D_{PK}$, $TDP^{-1}_{SK}(y)$ is efficiently computable given the trapdoor SK.*
2. ***Pseudorandom Sampling:** For any polysized distinguisher $\mathcal{A}$,*

$$\left|\Pr\left[\mathsf{Exp}_{\mathcal{A},0,\mathsf{PRS}} = 1\right] - \Pr\left[\mathsf{Exp}_{\mathcal{A},1,\mathsf{PRS}} = 1\right]\right| \leq \mathsf{negl}(\kappa)$$

   where $\mathsf{Exp}_{\mathcal{A},b,\mathsf{PRS}}$ is described in Figure 1.
3. ***One-wayness**: For all poly sized adversaries $\mathcal{A}$,*

$$\Pr\left[\mathcal{A}(PK, \mathsf{Samp}, TDP_{PK}(x)) = x : \begin{array}{c}(PK, SK) \leftarrow \mathsf{KeyGen}(1^\kappa)\\ \mathsf{Samp} \leftarrow \mathsf{SampGen}(SK)\\ x \leftarrow \mathsf{Samp}(1^\kappa)\end{array}\right] \leq \mathsf{negl}(\kappa)$$

(a) $r_1, r_2 \xleftarrow{\$} \{0,1\}^{\mathsf{poly}(\kappa)}$

(b) $(PK, SK) \leftarrow \mathsf{KeyGen}(1^\kappa; r_1)$.

(c) $\mathsf{Samp} \leftarrow \mathsf{SampGen}(SK; r_2)$

(d) **if**$(b = 0)$, $x \xleftarrow{\$} D_{PK}$.

(e) **else**, $x \leftarrow \mathsf{Samp}(1^\kappa)$.

(f) Output $\mathcal{A}(r_1, r_2, x)$

Figure 1: $\mathsf{Exp}_{\mathcal{A},b,\mathsf{PRS}}$

3.1 Construction of Trapdoor Permutations

In this section, we give a construction of Trapdoor Permutations and prove the one-wayness assuming the existence polynomially hard $i\mathcal{O}$, puncturable pseudorandom function $\mathcal{PRF}$ and injective $\mathcal{PRG}$ (used only in the proof).

Theorem 17 *Assuming the existence of one-way permutations and indistinguishablity obfuscation against polytime adversaries there exists a trapdoor permutation.*

We first give an informal description of our construction. The domain of our trapdoor permutation consists of tuples of the form $(x, \sigma_1, \cdots, \sigma_\kappa)$ where $x \in \{0,1\}^\kappa$ and $\sigma_i = \mathsf{PRF}_{S_i}(x_{[i]})$. That is, σ_i is a PRF computation (using key S_i) on the i-bit prefix of x. The permutation function F takes as input $(x, \sigma_1, \cdots, \sigma_{\kappa-1})$, checks if the input is in the domain and if yes outputs $(x+1, \sigma'_1, \cdots, \sigma'_\kappa)$ where $x+1$ is computed modulo 2^κ and σ'_i satisfies the above PRF relation for all $i \in [\kappa]$. The sampler Samp takes as input a random seed r (of length $\kappa/2$) as input, expands it using a length doubling PRG and computes the PRF on the prefixes of the output of the PRG.

The formal description of our construction of Trapdoor Permutation appears in Figure 2.

Proof of Theorem 17 It is easy to observe that the function computed by $F_{S_1,\cdots,S_\kappa}$ is a permutation over the points in the domain and the pseudorandomness property of the sampler follows from security of pseudorandom generator PRG (even when given the random coins used by KeyGen and $\mathsf{SampGen}$). We now prove the one-wayness of the above construction in the presence of the public key and the sampler. A high level overview of our proof is to indistinguishably change the public key to one that outputs $\perp$ on inputs of the form $(i-1, \cdot, \cdots, \cdot)$ where $(i, \mathsf{PRF}_{S_1}(i_{[1]}), \cdots, \mathsf{PRF}_{S_\kappa}(i_{[\kappa]}))$ is the inversion challenge. Clearly, the advantage of the adversary in inverting the challenge $(i, \mathsf{PRF}_{S_1}(i_{[1]}), \cdots, \mathsf{PRF}_{S_\kappa}(i_{[\kappa]}))$ in the final hybrid is 0.

Circuit F^* We denote by $F^*_{S_1,\cdots,S_\kappa,\alpha,\beta}$ the circuit which works exactly as $F_{S_1,\cdots,S_\kappa}$ on all inputs except on those inputs $(s, \cdot, \cdots, \cdot)$ where $\alpha \leq s \leq \beta$, it outputs $\perp$. This notation would be used in our hybrids.

Our Hybrids. We now describe our hybrids.

- $\mathsf{KeyGen}(1^\kappa)$:
  1. Sample $\{S_i\}_{i\in[\kappa]} \leftarrow \mathsf{KeyGen}_{\mathcal{PRF}}(1^\kappa)$. For all $i \in [\kappa]$, S_i is a seed for a PRF mapping i bits to κ bits. That is, $\mathsf{PRF}_{S_i} : \{0,1\}^i \to \{0,1\}^\kappa$.
  2. The public key is given by $i\mathcal{O}(F_{S_1,\cdots,S_\kappa})$ where $F_{S_1,\cdots,S_\kappa}$ is described in Figure 3 and the secret key is given by $S_1, \cdots, S_\kappa$.
- TDP_{PK} : Run the obfuscated circuit $i\mathcal{O}(F_{S_1,\cdots,S_\kappa})$ on the given input $(x, \sigma_1, \cdots, \sigma_\kappa)$.
- TDP^{-1}_{SK}: The Inverter $I_{S_1,\cdots,S_\kappa}$ is described in Figure 3.
- $\mathsf{SampGen}(SK)$: The sampler is given by $i\mathcal{O}(X_{S_1,\cdots,S_\kappa})$ where $X_{S_1,\cdots,S_\kappa}$ is described in Figure 3.
- Samp: Run the circuit $i\mathcal{O}(X_{S_1,\cdots,S_\kappa})$ on the given randomness r.

Figure 2: Construction of Trapdoor Permutation

- Hyb_0: Original experiment where the adversary is given a random challenge $((i, \sigma_1, \cdots \sigma_\kappa), i\mathcal{O}(X_{S_1,\cdots,S_\kappa}), i\mathcal{O}(F_{S_1,\cdots,S_\kappa}))$ where $r \xleftarrow{\$} \{0,1\}^{\kappa/2}$ and $i = \mathsf{PRG}(r)$.
- Hyb_1: Instead of setting $i = \mathsf{PRG}(r)$, we sample $i \xleftarrow{\$} \{0,1\}^\kappa$. The indistinguishability follows from the pseudorandomness property of PRG.
- Hyb_2: In this hybrid we change the public key of the permutation. The public key of the permutation is generated as $i\mathcal{O}(F^1_{S_1,\cdots,S_\kappa,i,v})$ instead of $i\mathcal{O}(F_{S_1,\cdots,S_\kappa})$ where $v \xleftarrow{\$} [2^{\kappa/2}]$. The function $F^1_{S_1,\cdots,S_\kappa,i,v}$ (padded to length $p(\kappa)$) is similar to that of $F_{S_1,\cdots,S_\kappa}$ except that on inputs $(x, \cdot, \cdots, \cdot)$ such that $i - 2^{\frac{\kappa}{4}} \le x \le i-1$ and $\mathsf{PRG}(i-x) = v$, it outputs $\perp$.

  Since v is chosen randomly from $[2^{\kappa/2}]$ it is not in the image of the PRG with overwhelming probability (actually with probability $1 - \frac{1}{2^{\frac{\kappa}{4}}}$). Hence, the two function have the same input-output behavior with overwhelming probability. The computational indistinguishability of Hyb_1 and Hyb_2 follows from the security of indistinguishability obfuscation.
- Hyb_3: In this hybrid, we change how v is computed. Instead of sampling v uniformly at random from $[2^{\kappa/2}]$, we generate v as $\mathsf{PRG}(u_0)$ where $u_0 \xleftarrow{\$} [2^{\kappa/4}]$. The public key of the permutation would now correspond to $i\mathcal{O}(F_{S_1,\cdots,S_\kappa,i,\mathsf{PRG}(u_0)})$.

  The computational indistinguishability of Hyb_2 and Hyb_3 follows from the pseudorandomness property of the PRG.

  Notation. We denote $\alpha_0 := i - u_0$ from now on.
- Hyb_4: In this hybrid we replace the public key for computing the permutation with $i\mathcal{O}(F^*_{S_1,\cdots,S_\kappa,\alpha_0,\alpha_0})$.

$F_{S_1,\cdots,S_\kappa}$

Input: $(i, \sigma_1, \cdots, \sigma_\kappa)$
Constants: $S_1, \cdots, S_\kappa$

1. For all $j \in [\kappa]$, check if $\sigma_j = \mathsf{PRF}_{S_j}(i_{[j]})$.
2. If any of the above checks fail, output $\perp$.
3. Else, for all $j \in [\kappa]$ compute $\sigma'_j = \mathsf{PRF}_{S_j}((i+1)_{[j]})$ where $i+1$ is computed modulo 2^κ.
4. Output $(i+1, \sigma'_1, \cdots, \sigma'_\kappa)$.

Padding: The circuit would be padded to size $p(\kappa)$ where $p(\cdot)$ is a polynomial that would be specified later.

$X_{S1,\cdots,S_\kappa}$

Input: $r \in \{0,1\}^{\kappa/2}$
Constants: $S_1, \cdots, S_\kappa$

1. Compute $i = \mathsf{PRG}(r)$.
2. For every $j \in [\kappa]$, compute $\sigma_j = \mathsf{PRF}_{S_j}(i_{[j]})$.
3. Output $(i, \sigma_1, \sigma_2, \cdots, \sigma_\kappa)$.

Padding: The circuit would be padded to size $q(\kappa)$ where $q(\cdot)$ is a polynomial that would be specified later.

$I_{S_1,\cdots,S_\kappa}$

Input: $(i, \sigma_1, \cdots, \sigma_\kappa)$
Constants: $S_1, \cdots, S_\kappa$

1. Check whether for all $j \in [\kappa]$, $\sigma_j = \mathsf{PRF}_{S_j}(i_{[j]})$.
2. If any of the checks fail, output $\perp$.
3. Else, for all $j \in [\kappa]$ compute $\sigma'_j = \mathsf{PRF}_{S_j}((i-1)_{[j]})$ where $i-1$ is computed modulo 2^κ.
4. Output $(i-1, \sigma'_1, \sigma'_2, \cdots, \sigma'_\kappa)$.

Figure 3: Public Key, Sampler and the Inverter for the Trapdoor permutations

Observe that $F_{S_1,\cdots,S_\kappa,i,\mathsf{PRG}(u_0)}$ and $F^*_{S_1,\cdots,S_\kappa,\alpha_0,\alpha_0}$ (padded to length $p(\kappa)$) compute the exact same function due to the injectivity of the PRG. Hence, the computational indistinguishability of Hyb_3 and Hyb_4 follows from the security of indistinguishability obfuscation.

- $\mathsf{Hyb}_{5,j}$: In this hybrid, we replace the public key of our permutation with $i\mathcal{O}(F^*_{S_1,\cdots,S_\kappa,\alpha_0,\alpha_j})$ for $j \in \{0,\cdots,\delta(\alpha_0)+\mu(\alpha_{\delta(\alpha_0)})\}$ where $\delta(\cdot)$ and $\mu(\cdot)$ are defined below.

  Defining α_j values. For any string $\alpha \in \{0,1\}^\kappa$, let $f(\alpha)$ denote the index of the lowest order bit of α that is 0 (with the index of the highest order bit being 1). More formally, $f(\alpha)$ is the smallest j such that $u = \alpha_{[j]}||1^{\kappa-j}$. For example, if $\alpha = \overbrace{100}^{3}11$ then $f(\alpha) = 3$. Let $\ell(\beta,\gamma)$ denotes the smallest $j \in [\kappa]$ such that $\beta_{[j-1]} = \gamma_{[j-1]}$ and $\beta_j \neq \gamma_j$ if $\beta \neq \gamma$ and is a special symbol ζ otherwise. That is, it denotes the first index at which β and δ differ if $\beta \neq \delta$ and is equal to the special symbol ζ otherwise. Let $\delta(\alpha)$ denote the number of 0s in the positions $[\ell(\alpha,i-1)+1,\kappa]$ in the binary representation of α if $\ell(\alpha,i-1) \neq \zeta$ and is equal to 0 otherwise. Let $\rho(\alpha) = \ell(\alpha+1,i-1)$ if $\ell(\alpha+1,i-1) \neq \zeta$ and equal to κ otherwise. Let $\mu(\alpha)$ denote one more than the number of ones in the positions $[\ell(\alpha,i-1)+1,\kappa]$ in the binary representation of $i-1$ if $\ell(\alpha,i-1) \neq \zeta$ and is equal to 0 otherwise.

  Starting with a value $\alpha_0 \in \{0,1\}^\kappa$ we define for $j \in [0,\delta(\alpha_0)+\mu(\alpha_{\delta(\alpha_0)})-1]$,

$$\alpha_{j+1} = \begin{cases} \alpha_j + 2^{\kappa-f(\alpha_j)} & \text{if } j+1 \leq \delta(\alpha_0) \\ \alpha_j + 2^{\kappa-\rho(\alpha_j)} & \text{otherwise} \end{cases}$$

  Note that by this definition $\alpha_{\delta(\alpha_0)} = (i-1)_{[\ell(\alpha_0,i-1)-1]}||(\alpha_0)_i||1^{\kappa-\ell(\alpha_0,i-1)}$ and $\alpha_{\delta(\alpha_0)+\mu(\alpha_{\delta(\alpha_0)})} = i-1$. Illustrations of α_js are given in Figure 4,5.

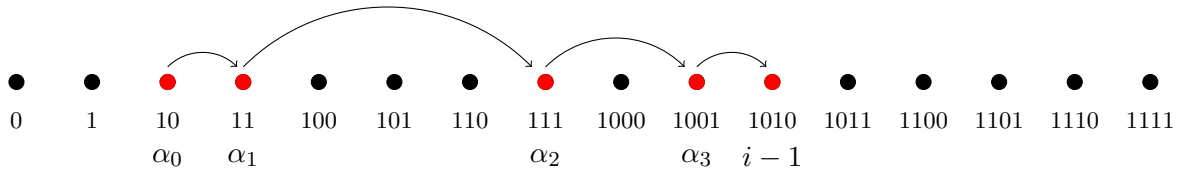

Figure 4: Illustration of the steps starting with $\alpha_0 = 0010$ and $i-1 = 1001$.

Indistinguishability Argument: Observe that $\mathsf{Hyb}_{5,0}$ is distributed identically to Hyb_4 and $\mathsf{Hyb}_{5,\delta(\alpha_0)+\mu(\alpha_{\delta(\alpha_0)})}$ is a hybrid where the public key of the permutation outputs $\perp$ on every input $(s,\cdot,\cdots,\cdot)$ where $\alpha_0 \leq s \leq i-1$. We now prove that $\mathsf{Hyb}_{5,j}$ is indistinguishable to $\mathsf{Hyb}_{5,j+1}$ for all $0 \leq j \leq \delta(\alpha_0)+\mu(\alpha_{\delta(\alpha_0)})-1$. We prove this through a sequence of hybrids. we let ν_j as the shorthand for $f(u_j)$ for $j < \delta(\alpha_0)$ and equal to $\rho(u_j)$ for $j \geq \delta(\alpha_0)$. Let $t_j = \alpha_{j_{[\nu_j]}} + 1$.

  - $\mathsf{Hyb}_{5,j,1}$: Let $S'_{\nu_j} \leftarrow \mathsf{Punc}(S_{\nu_j},t_j)$ and $\sigma^\star = \mathsf{PRF}_{S_{\nu_j}}(t_j)$. In this hybrid we replace the public key of the trapdoor permutation with $i\mathcal{O}(F^2_{S_1,\cdots,S'_{\nu_j},\cdots S_\kappa,\alpha_0,\alpha_j,\sigma^\star})$. The func-

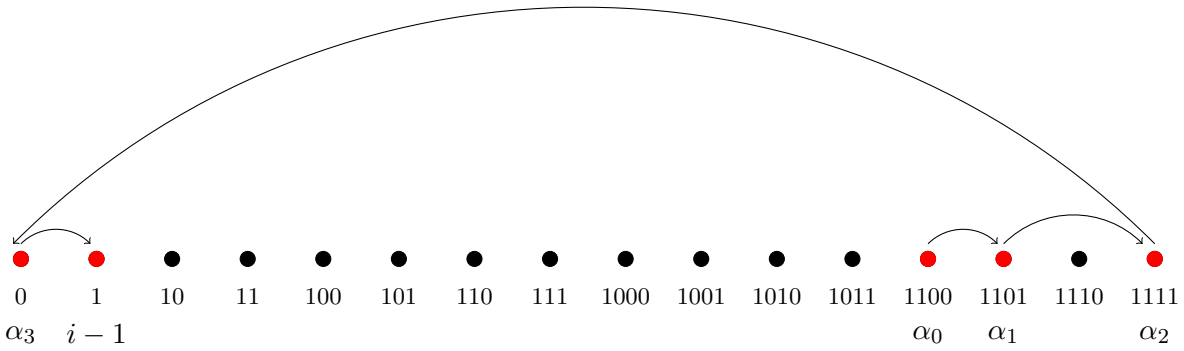

Figure 5: Illustration of the steps starting with $\alpha_0 = 1100$ and $i - 1 = 0001$.

tion $F^2_{S_1,\cdots,S'_{\nu_j},\cdots S_\kappa,\alpha_0,\alpha_j,\sigma^\star}$ (padded to length $p(\kappa)$) is identical to $F^*_{S_1,\cdots,S_{\nu_j},\cdots,S_\kappa,\alpha_0,\alpha_j}$ except that it has the punctured key S'_{ν_j} and uses $\sigma^\star$ to perform the computations using $\mathsf{PRF}_{S_{\nu_j}}(t_j)$. Since $\sigma^\star = \mathsf{PRF}_{S_{\nu_j}}(t_j)$, the two circuits compute the same functionality and hence the indistingusiahbility of $\mathsf{Hyb}_{5,j}$ and $\mathsf{Hyb}_{5,j,1}$ follows from the security of indistinguishability obfuscation.

Observation. The value $\sigma^\star$ is used only to check for the validity of the σ_{ν_j} in the interval $[\alpha_j + 1, \alpha_{j+1}]$. Note that if the input is in the interval $[\alpha_j + 1, \alpha_{j+1} - 1]$, then the next node on the path also has the ν_j^{th} associated signature to be same as $\sigma^\star$. So in that case, if the input is valid then the circuit $F^2_{S_1,\cdots,S'_{\nu_j},\cdots S_\kappa,\alpha_0,\alpha_j,\sigma^\star}$ outputs the input σ_{ν_j} in place of $\sigma^\star$.

– $\mathsf{Hyb}_{5,j,2}$: In this hybrid we replace the sampler $i\mathcal{O}(X_{S_1,\cdots,S_\kappa})$ with $i\mathcal{O}(X^2_{S_1,\cdots,S'_{\nu_j},\cdots,S_\kappa})$. $X^2_{S_1,\cdots,S'_{\nu_j},\cdots,S_\kappa}$ (padded to length $q(\kappa)$) is identical to $X_{S_1,\cdots,S_\kappa}$ except that if required to evaluate $\mathsf{PRF}_{S_{\nu_j}}$ on t_j, it outputs $\perp$. We now argue that $X^2_{S_1,\cdots,S'_{\nu_j},\cdots,S_\kappa}$ and $X_{S_1,\cdots,S_\kappa}$ compute the same functionality with overwhelming probability. Observe that $X^2_{S_1,\cdots,S'_{\nu_j},\cdots,S_\kappa}$ is required to compute $\mathsf{PRF}_{S_{\nu_j}}(t_j)$ if and only if $\mathsf{PRG}(r) \in [\alpha_j + 1, \alpha_{j+1}]$. We note that every $x \in [\alpha_j + 1, \alpha_{j+1}]$ is of the form $i - u_0 + c$ where c is at most u_0. In particular, c is independent of i and hence x is uniformly distributed in the interval $\{0,1\}^\kappa$ since i is randomly distributed in the interval $\{0,1\}^\kappa$. Hence, with overwhelming probability (equal to $1 - \frac{1}{2^{\kappa/2}}$), x is not in the image of PRG. By an union bound, no point in $[\alpha_j + 1, \alpha_{j+1}]$ is in the image of the PRG except with probability $\frac{1}{2^{\kappa/4}}$ (since the size of the interval is at most $2^{\kappa/4}$). Hence, the two circuits $X^2_{S_1,\cdots,S'_{\nu_j},\cdots,S_\kappa}$ and $X_{S_1,\cdots,S_\kappa}$ compute the same functionality with overwhelming probability and the indistinguishability of $\mathsf{Hyb}_{5,j,1}$ and $\mathsf{Hyb}_{5,j,2}$ follows from security of $i\mathcal{O}$.

– $\mathsf{Hyb}_{5,j,3}$: In this hybrid we change how $\sigma^\star$ that is hardwired in the public key $i\mathcal{O}(F^2_{S_1,\cdots,S'_{\nu_j},\cdots S_\kappa,\alpha_0,\alpha_j,\sigma^\star})$ is generated. In particular, we choose $\sigma^\star \xleftarrow{\$} \{0,1\}^\kappa$ instead of

setting $\sigma^\star = \mathsf{PRF}_{S_{\nu_j}}(t_j)$. The indistinguishability of $\mathsf{Hyb}_{5,j,2}$ and $\mathsf{Hyb}_{5,j,3}$ follows from the pseudorandomness at punctured point property of PRF.

- $\mathsf{Hyb}_{5,j,4}$: In this hybrid we replace the public key of the permutation with $i\mathcal{O}(F^3_{S_1,\cdots,S'_{\nu_j},\cdots S_\kappa,\alpha_0,\alpha_j,\mathsf{PRG}(\sigma^\star)})$. The circuit $F^3_{S_1,\cdots,S'_{\nu_j},\cdots S_\kappa,\alpha_0,\alpha_j,\mathsf{PRG}(\sigma^\star)}$ (padded to length $p(\kappa)$) is exactly similar to that of $F^2_{S_1,\cdots,S'_{\nu_j},\cdots S_\kappa,\alpha_0,\alpha_j,\sigma^\star}$ except that on input $(x,\sigma_1,\cdots,\sigma_\kappa)$ such that $x \in [\alpha_j+1,\alpha_{j+1}]$, it checks the validity of σ_{ν_j} by checking $\mathsf{PRG}(\sigma_{\nu_j}) = \mathsf{PRG}(\sigma^\star)$ instead of checking $\sigma_{\nu_j} = \sigma^\star$. Note that $F^3_{S_1,\cdots,S'_{\nu_j},\cdots S_\kappa,\alpha_0,\alpha_j,\mathsf{PRG}(\sigma^\star)}$ requires the value of $\sigma^\star$ only for checking the validity of the input. Both the circuits compute the exact same functionality since the PRG is injective. Hence, the indistinguishability of $\mathsf{Hyb}_{5,j,3}$ and $\mathsf{Hyb}_{5,j,4}$ follows from the security of $i\mathcal{O}$.
- $\mathsf{Hyb}_{5,j,5}$: In this hybrid the public key of the permutation corresponds to $i\mathcal{O}(F^3_{S_1,\cdots,S'_{\nu_j},\cdots S_\kappa,\alpha_0,\alpha_j,\tau^\star})$ where we set $\tau^\star$ to be a string chosen uniformly at random from $\{0,1\}^{2\kappa}$ instead of setting $\tau^\star = \mathsf{PRG}(\sigma^\star)$. The indistinguishability of hybrid $\mathsf{Hyb}_{5,j,4}$ and $\mathsf{Hyb}_{5,j,5}$ follows from the pseudorandomness property of PRG.
- $\mathsf{Hyb}_{5,j,6}$: In this hybrid the public key of the permutation corresponds to $i\mathcal{O}(F^*_{S_1,\cdots,S'_{\nu_j},\cdots S_\kappa,\alpha_0,\alpha_{j+1}})$. Observe that $F^*_{S_1,\cdots,S'_{\nu_j},\cdots S_\kappa,\alpha_0,\alpha_{j+1}}$ (padded to length $p(\kappa)$) and $F^3_{S_1,\cdots,S'_{\nu_j},\cdots S_\kappa,\alpha_0,\alpha_j,\tau^\star}$ compute the same functionality with overwhelming probability since $\tau^\star$ is chosen at random it is not in the image of the PRG with overwhelming probability. Hence, for no value of σ_{ν_j} (with overwhelming probability) the test $\mathsf{PRG}(\sigma_{\nu_j}) = \tau^*$ passes. Hence, the indistinguishability of $\mathsf{Hyb}_{5,j,5}$ and $\mathsf{Hyb}_{5,j,6}$ follows from the security of $i\mathcal{O}$.
- $\mathsf{Hyb}_{5,j,7}$: In this hybrid the public key of the permutation corresponds to $i\mathcal{O}(F^*_{S_1,\cdots,S_{\nu_j},\cdots S_\kappa,\alpha_0,\alpha_{j+1}})$. That is, it contains the unpunctured key S_{ν_j} instead of the punctured key S'_{ν_j}. Note that both $F^*_{S_1,\cdots,S_{\nu_j},\cdots S_\kappa,\alpha_0,\alpha_{j+1}}$ (padded to length $p(\kappa)$) and $F^*_{S_1,\cdots,S'_{\nu_j},\cdots S_\kappa,\alpha_0,\alpha_{j+1}}$ do not require the evaluation of $\mathsf{PRF}_{S_{\nu_j}}$ on t_j and hence the indistinguishability of hybrids $\mathsf{Hyb}_{5,j,6}$ and $\mathsf{Hyb}_{5,j,7}$ follows from security of $i\mathcal{O}$.
- $\mathsf{Hyb}_{5,j,8}$: In this hybrid, we replace the sampler with $i\mathcal{O}(X_{S_1,\cdots,S_\kappa})$ instead of $i\mathcal{O}(X^2_{S_1,\cdots,S'_{\nu_j},\cdots,S_\kappa})$. The indistinguishability of hybrids $\mathsf{Hyb}_{5,j,7}$ and $\mathsf{Hyb}_{5,j,8}$ follows the exact same argument as indistinguishability between $\mathsf{Hyb}_{5,j,1}$ and $\mathsf{Hyb}_{5,j,2}$.

Setting the parameters. Note that $\delta(\alpha_0) \leq \kappa-1$ and $\mu(\alpha_{\delta(\alpha_0)}) \leq \kappa$ and hence $\delta_{\alpha_0}+\mu(\alpha_{\delta(\alpha_0)}) \leq 2\kappa-1$. $p(\cdot)$ is the maximum size of the circuit computing the public key that appears in the construction and in the proof and $q(\cdot)$ is the maximum size of the circuit computing the sampler that appears in the construction and in the proof.

4 Trapdoor Permutation from FE

We start by defining a weaker (with respect to pseudorandom sampling) notion of trapdoor permutation.

1. $(PK, SK) \leftarrow \mathsf{KeyGen}(1^\kappa)$.
2. $\mathsf{Samp} \leftarrow \mathsf{SampGen}(SK)$
3. **if**$(b = 0)$, $x \xleftarrow{\$} D_{PK}$.
4. **else**, $x \leftarrow \mathsf{Samp}(1^\kappa)$.
5. Output $\mathcal{A}(PK, \mathsf{Samp}, x)$

Figure 6: $\mathsf{Exp}_{\mathcal{A},b,\mathsf{PRS}}$

Definition 18 *An efficiently computable family of functions:*

$$\mathcal{TDP} = \{\mathsf{TDP}_{PK} : D_{PK} \to D_{PK} \text{ and } PK \in \{0,1\}^{\mathsf{poly}(\kappa)}\}$$

over the domain D_{PK} with associated (probabilistic) $(\mathsf{KeyGen}, \mathsf{SampGen})$ *algorithms is a (standard) trapdoor permutation if it satisifies:*

- ***Trapdoor Invertibility:*** *For any $(PK, SK) \leftarrow \mathsf{KeyGen}(1^\kappa)$, TDP_{PK} is a permutation over D_{PK}. For any $y \in D_{PK}$, $TDP^{-1}_{SK}(y)$ is efficiently computable given the trapdoor SK.*
- ***Pseudorandom Sampling:*** *For any $(PK, SK) \leftarrow \mathsf{KeyGen}(1^\kappa)$ and $\mathsf{Samp} \leftarrow \mathsf{SampGen}(SK)$, $\mathsf{Samp}(\cdot)$ samples pseudo random points in the domain D_{PK}. Formally, for any polysized distinguisher $\mathcal{A}$,*

$$\left|\Pr\left[\mathsf{Exp}_{\mathcal{A},0,\mathsf{PRS}} = 1\right] - \Pr\left[\mathsf{Exp}_{\mathcal{A},1,\mathsf{PRS}} = 1\right]\right| \leq \mathsf{negl}(\kappa)$$

  where $\mathsf{Exp}_{\mathcal{A},b,\mathsf{PRS}}$ is described in Figure 6.

- ***One-wayness****: For all poly sized adversaries $\mathcal{A}$,*

$$\Pr\left[\mathcal{A}(PK, \mathsf{Samp}, TDP_{PK}(x)) = x : \begin{array}{c}(PK, SK) \leftarrow \mathsf{KeyGen}(1^\kappa) \\ \mathsf{Samp} \leftarrow \mathsf{SampGen}(SK) \\ x \leftarrow \mathsf{Samp}(1^\kappa)\end{array}\right] \leq \mathsf{negl}(\kappa)$$

Remark 19 *The requirement of Pseudorandom sampling considered in Bitanksy et al.'s work [BPW16] is stronger than the one considered here in sense that they require the pseudorandomness property to hold even when given the random coins used by* KeyGen *and the* SampGen *algorithms. We do not achieve the stronger notion in this work. In particular, given the random coins used in* SampGen *the sampler's output is no longer pseudorandom. Therefore, our trapdoor permutations can be only used in applications where an honest party runs the* KeyGen *and* SampGen *algorithm. It cannot be used for example to achieve receiver privacy in EGL Oblivious Transfer protocol [EGL85].*

In this section, we show a construction of trapdoor permutation satisfying the above definition from polynomially hard public key functional encryption, prefix puncturable pseudorandom function, injective PRGs with left half injectivity property, strong randomness extractor and public key encryption with random public keys.

Theorem 20 *Assuming the existence of one-way permutations, public key functional encryption satisfying selective-indistinguishability security against polytime adversaries and public key encryption with (pseudo) random public keys, there exists a trapdoor permutation.*

We now recall the special key structure [GPS15] which would be used in our construction of trapdoor permutation.

Notation. We treat $1^i + 1$ as 0^i and $\phi + 1$ as ϕ.

Special Key Structure.

$$\mathsf{U}_x = \bigcup_{i\in[2\kappa]} \mathsf{U}^i_x \qquad \mathsf{U}^i_x = \begin{cases} \{S_{i,x_{[i]}}\} & \text{if } |x| > i \\ \{S_{i,x}\} & \text{otherwise} \end{cases}$$

$$\mathsf{V}_x = \bigcup_{i\in[2\kappa]} \mathsf{V}^i_x \qquad \mathsf{V}^i_x = \begin{cases} \{S_{i,x_{[i]}}, S_{i,x_{[i]}+1}\} & \text{if } |x| > i \text{ and } x = x_{[i]}\|1^{|x|-i} \\ \{S_{i,x}, S_{i,(x+1)\|0^{i-|x|}}\} & \text{if } |x| \le i \\ \emptyset & \text{otherwise} \end{cases}$$

$$\mathsf{W}_x = \bigcup_{i\in[2\kappa]} \mathsf{W}^i_x \qquad \mathsf{W}^i_x = \begin{cases} \{\mathsf{PRG}_0(S_{i,x_{[i]}})\} & \text{if } |x| \ge i \\ \emptyset & \text{otherwise} \end{cases}$$

For the empty string $x = \phi$, these sets can be initialized as follows.

$$\mathsf{U}_\phi = \bigcup_{i\in[2\kappa]} \mathsf{U}^i_\phi \qquad \mathsf{U}^i_\phi = \{S_i\}$$

$$\mathsf{V}_\phi = \bigcup_{i\in[2\kappa]} \mathsf{V}^i_\phi \qquad \mathsf{V}^i_\phi = \{S_i\}$$

$$\mathsf{W}_\phi = \bigcup_{i\in[2\kappa]} \mathsf{W}^i_\phi \qquad \mathsf{W}^i_\phi = \emptyset$$

Jumping ahead, the set of keys in U_x would be used by the sampler to generate the set of associated signatures on the sampled point. The set W_x (called as the vestigial set in [GPS15]) is used to check the validity of input i.e checking whether the input belongs to the domain. The set V_x is used to generate the associated signatures on the "next" point as defined by the permutation.

Properties of Special Key Structure.

Lemma 21 (Computability Lemma,[GPS15]) *There exists an* explicit *efficient procedure that given* $\mathsf{U}_x, \mathsf{V}_x, \mathsf{W}_x$ *computes* $\mathsf{U}_{x\|0}, \mathsf{V}_{x\|0}, \mathsf{W}_{x\|0}$ *and* $\mathsf{U}_{x\|1}, \mathsf{V}_{x\|1}, \mathsf{W}_{x\|1}$.

Proof We start by noting that it suffices to show that for each i, given $\mathsf{U}^i_x, \mathsf{V}^i_x, \mathsf{W}^i_x$ one can compute $\mathsf{U}^i_{x\|0}, \mathsf{V}^i_{x\|0}, \mathsf{W}^i_{x\|0}$ and $\mathsf{U}^i_{x\|1}, \mathsf{V}^i_{x\|1}, \mathsf{W}^i_{x\|1}$. We argue this next. Observe that two cases arise either $|x| < i$ or $|x| \ge i$. We deal with the two cases:

- $|x| < i$: Note that in this case, $\mathsf{U}^i_x = \{S_{i,x}\}$ and this can be used for computing $S_{i,x\|0}$ and $S_{i,x\|1}$ [4]. Similarly, V^i_x is $\{S_{i,x}, S_{i,(x+1)\|0^{i-|x|}}\}$ when $|x| < i$ and these values can be used to compute (by appropriate prefix puncturing) $S_{i,x\|0}$, $S_{i,x\|1}$, $S_{i,(x\|0)+1} = S_{i,x\|1}$ and $S_{i,((x\|1)+1)\|0^{i-|x|-1}} = S_{i,(x+1)\|0\|0^{i-|x|-1}} = S_{i,(x+1)\|0^{i-|x|}}$. Observe by case by case inspection that these values are sufficient for computing $\mathsf{U}^i_{x\|0}, \mathsf{V}^i_{x\|0}, \mathsf{W}^i_{x\|0}$ and $\mathsf{U}^i_{x\|1}, \mathsf{V}^i_{x\|1}, \mathsf{W}^i_{x\|1}$ in all cases.

- $|x| \geq i$: Observe that $U^i_x = \{S_{i,x_{[i]}}\}$ and $U^i_{x\|0} = U^i_{x\|1} = \{S_{i,x_{[i]}}\} = U^i_x$. Also, note that according to the constraints placed on x by the definition, if $\mathsf{V}^i_x = \emptyset$ then both $\mathsf{V}^i_{x\|0}$ and $\mathsf{V}^i_{x\|1}$ must be $\emptyset$ as well. On the other hand if $V^i_x \neq \emptyset$ then $\mathsf{V}^i_{x\|0}$ is still $\emptyset$ while $\mathsf{V}^i_{x\|1} = \mathsf{V}^i_x$. Additionally, $W^i_{x\|0} = W^i_{x\|1} = W^i_x$.

This concludes the proof. ■

Lemma 22 (Derivability Lemma,[GPS15]) *For every $i \in [\kappa], x \in \{0,1\}^i$ we have that, $S_{i,x+1}$ can be derived from keys in V^i_y if and only if y is a prefix of $x\|1^{\kappa-i}$ or $(x+1)\|1^{\kappa-i}$.*

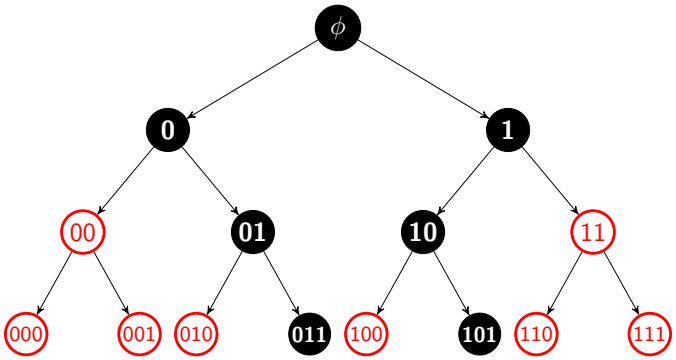

Figure 7: Black nodes represent the choices of $x \in \{0,1\}^{\leq 3}$ such that V^2_x can be used to derive $S_{2,10}$.

Proof We start by noting that for any $y \in \{0,1\}^{>i} \cap \{0,1\}^{\leq\kappa}$, by definition of V-sets we have that $\mathsf{V}^i_y = \mathsf{V}^i_{y_{[i]}}$ or $\mathsf{V}^i_y = \emptyset$. Hence it suffices to prove the above lemma for $y \in \{0,1\}^{\leq i}$.

We first prove that if y is a prefix of x or $(x+1)$ then we can derive $S_{i,x+1}$ from V^i_y. Two cases arise:

- Observe that if y is a prefix of x then we must have that either y is a prefix of $x+1$ or $x+1 = (y+1)\|0^{i-|y|}$. Next note that by definition of V-sets we have that $\mathsf{V}^i_y = \{S_{i,y}, S_{i,(y+1)\|0^{i-|y|}}\}$, and one of these values can be used to compute $S_{i,x+1}$.

- On the other hand if y is a prefix of $x+1$ then again by definition of V-sets we have that $\mathsf{V}^i_y = \{S_{i,y}, S_{i,(y+1)\|0^{i-|y|}}\}$, and $S_{i,y}$ can be used to compute $S_{i,x+1}$.

Next we show that no other $y \in \{0,1\}^{\leq i}$ allows for such a derivation. Note that by definition of V-sets we have that $V^i_y = \{S_{i,y}, S_{i,(y+1)\|0^{i-|y|}}\}$. We will argue that neither $S_{i,y}$ nor $S_{i,(y+1)\|0^{i-|y|}}$ can be used to derive $S_{i,x+1}$.

[4] Observe that since $|x| < i$, $S_{i,x\|b}$ for $b \in \{0,1\}$ is well-defined.

- We are given that y is not a prefix of $x+1$. This implies that $S_{i,y}$ cannot be used to derive $S_{i,x+1}$.
- Now we need to argue that $S_{i,(y+1)\|0^{i-|y|}}$ cannot be used to compute $S_{i,x+1}$. For this, it suffices to argue that $x+1 \neq (y+1)\|0^{i-|y|}$. If $x+1 = (y+1)\|0^{i-|y|}$ then y must be prefix of x. However, we are given that this is not the case. This proves our claim.

This concludes the proof. ■

Our Construction. At a very high level, TDP_{PK} algorithm checks whether the input is valid (i.e belongs to the domain) by checking if each of the associated signatures are valid. It performs this check in an "encrypted" form by checking if the left halves of the PRG evaluations on the input signatures are equal to the corresponding contents in the vestigial set. Once the input passes this test, the algorithm outputs the signatures on the next point in the domain. It performs this task by setting the associated signatures to be equal to the ones from the input if the corresponding prefixes are shared between the input and the next domain point. For those prefixes that change, it outputs the correct associated signatures from the V-set. The formal description our construction appears in Figure 8.

Setting $\mathsf{rand}(\cdot)$ We set $\mathsf{rand}(\kappa) = 4\kappa + r(\kappa)$ where $r(\kappa)$ is the maximum number of random bits needed to generate encryptions under $\gamma_1, \cdots, \gamma_\kappa$ as well as encryptions under the public keys pk.

Proof of Theorem 20 : We show that construction given in Figure 8 satisfies the three properties given in Definition 18.

Trapdoor Invertibility. Observe that the domain of the TDP consists of strings of the form $(x, \sigma_1, \cdots, \sigma_{2\kappa})$ where for all $i \in [2\kappa]$, $\sigma_i = S_{i,x_{[i]}}$. We first show that given $(x, \sigma_1, \cdots, \sigma_{2\kappa})$ the public key correctly computes $(x+1, \sigma'_1, \cdots, \sigma'_{2\kappa})$ where $x+1$ is computed modulo $2^{2\kappa}$ and $\sigma'_i = S_{i,(x+1)_{[i]}}$ for all $i \in [2\kappa]$.

We first observe that since $v \xleftarrow{\$} \{0,1\}^{\kappa/4}$, with probability $1 - \frac{1}{2^{\kappa/8}}$ it is not in the image of the PRG and hence Step 1 of $G^1_{v,\Lambda_1,w}$ is not triggered. Since mode is set to 0 in c^1_ϕ, alternate behavior (the "Hidden" mode) of $G^1_{v,\Lambda_1,w}$ and F^1_{i,PK^1_{i+1},Π_1} is not triggered during the honest execution of the permutation function.

Given that the above cases do not arise we have that $\psi_i = \mathsf{PRG}_0(S_{i,x_{[i]}})$ for every $i \in [2\kappa]$. This follows from the correctness of Functional Encryption scheme. Recall that in Step 3 the permutation function checks if $\mathsf{PRG}_0(\sigma_i) = \psi_i$. Now from the left half injectivity property of the PRG we have that these checks pass if and only if $\sigma_i = S_{i,x_{[i]}}$. Hence, if public key does not output $\perp$ then the input $(x, \sigma_1, \cdots, \sigma_{2\kappa})$ must be valid i.e belonging to the domain. Additionally, notice that for every i such that $x_{[i]} \neq (x+1)_{[i]}$, V^i_x contains $S_{i,(x+1)_{[i]}}$. This follows since for such an i, $x = x_{[i]}\|1^{2\kappa-i}$ and by definition V^i_x contains $S_{i,x_{[i]}+1} = S_{i,(x+1)_{[i]}}$ due to the special structure of x. It follows from the correctness of $\mathsf{SK.Dec}$, the public key of the trapdoor permutation correctly obtains $S_{i,(x+1)_{[i]}}$ for every i such that $x_{[i]} \neq (x+1)_{[i]}$. For the rest of the indexes, the associated signature is obtained directly from the input. Hence, the permutation function correctly computes the full set of associated signatures on $x+1$. Since on input $(x, \sigma_1, \cdots, \sigma_{2\kappa})$ the function outputs

- $\mathsf{KeyGen}(1^\kappa)$:

  1. Sample $\{S_i\}_{i\in[2\kappa]}$, K^1_ϕ from $\mathsf{KeyGen}_{\mathcal{PPRF}}(1^\kappa)$. Here S_i is a key that works for i bit inputs, namely $\mathsf{PPRF}_{S_i} : \{0,1\}^i \to \{0,1\}^\kappa$ for all $i \in [2\kappa]$. Similarly, K^1_ϕ works on inputs of length $\mathsf{rand}(\kappa)$ where $\mathsf{rand}(\cdot)$ is a polynomial that would be specified later. Initialize $\mathsf{V}^i_\phi := S_i$, $\mathsf{V}_\phi = \bigcup_{i\in[2\kappa]} \mathsf{V}^i_\phi$ and $\mathsf{W}_\phi = \emptyset$.
  2. Let $\mathsf{Ext}_w : \{0,1\}^{2\kappa} \to \{0,1\}^{\kappa/8}$ be a $(\kappa/4, \mathsf{negl}(\kappa))$ [5] strong randomness extractor with seed $w \in \{0,1\}^{q(\kappa)}$. Sample a seed $w \xleftarrow{\$} \{0,1\}^{q(\kappa)}$ for the extractor Ext.
  3. Sample $(PK^1_i, MSK^1_i) \leftarrow \mathsf{FE.Setup}(1^\kappa)$ for all $i \in [2\kappa+1]$.
  4. Sample $sk_1 \leftarrow \mathsf{SK.KeyGen}(1^\kappa)$ and let $\Pi_1 \leftarrow \mathsf{SK.Enc}_{sk_1}(\pi_1)$ and $\Lambda_1 \leftarrow \mathsf{SK.Enc}_{sk_1}(\lambda_1)$ where $\pi_1 = 0^{\ell_1(\kappa)}$ and $\lambda_1 = 0^{\ell'_1(\kappa)}$. Here $\ell_1(\cdot)$ and $\ell'_1(\cdot)$ are appropriate length functions specified later.
  5. Sample $v \xleftarrow{\$} \{0,1\}^{\kappa/4}$.
  6. For each $i \in [2\kappa]$ generate $\mathsf{FSK}^1_i \leftarrow \mathsf{FE.KeyGen}(MSK^1_i, F^1_{i,PK^1_{i+1},\Pi_1})$ and $\mathsf{FSK}^1_{2\kappa+1} \leftarrow \mathsf{FE.KeyGen}(MSK^1_{2\kappa+1}, G^1_{v,\Lambda_1,w})$, where F^1_{i,PK^1_{i+1},Π_1} and $G^1_{v,\Lambda_1,w}$ are circuits described in Figure 9.
  7. Let $c^1_\phi = \mathsf{FE.Enc}_{PK_1}(\phi, \mathsf{V}_\phi, \mathsf{W}_\phi, K^1_\phi, 0^\kappa, 0)$.
  8. The Public Key PK is given by $(\{\mathsf{FSK}^1_i\}_{i\in[2\kappa+1]}, c^1_\phi)$ and the secret key SK is given by $(S_1, \cdots, S_{2\kappa})$.

- TDP_{PK}: The evaluation algorithm takes as input $(x, \sigma_1, \ldots, \sigma_{2\kappa})$ and outputs $(x+1, \sigma'_1, \ldots, \sigma'_{2\kappa})$ if the associated signatures $\sigma_1, \cdots, \sigma_{2\kappa}$ are valid. It proceeds as follows:

  1. For $i \in [2\kappa]$ compute $c^1_{x_{[i-1]}\|0}, c^1_{x_{[i-1]}\|1} := \mathsf{FE.Dec}(\mathsf{FSK}^1_i, c^1_{x_{[i-1]}})$.
  2. Obtain $d_x = ((\psi_1, \ldots, \psi_{2\kappa}), (\beta_j, \ldots, \beta_{2\kappa}))$ as output of $\mathsf{FE.Dec}(\mathsf{FSK}^1_{2\kappa+1}, c^1_x)$. Here $j = f(x)$. Recall from Section 3.1 that $f(x)$ is the smallest j such that $x = x_{[j]}\|1^{2\kappa-j}$.
  3. Output $\perp$ if $\mathsf{PRG}_0(\sigma_i) \neq \psi_i$ for any $i \in [2\kappa]$.
  4. For each $i \in [j-1]$ set $\sigma'_i = \sigma_i$.
  5. For each $i \in \{j, \ldots, 2\kappa\}$ set $\gamma_i = \mathsf{PRG}_1(\sigma_i)$ and σ'_i as $\mathsf{SK.Dec}_{\gamma_j,\cdots,\gamma_{2\kappa}}(\beta_i)$, decrypting β_i encrypted under $\gamma_j, \ldots \gamma_{2\kappa}$.
  6. Output $(x+1, \sigma'_1, \cdots, \sigma'_{2\kappa})$.

Figure 8: Construction of TDP from $\mathcal{FE}$

- TDP^{-1}_{SK} : The inversion algorithm on input $(x, \sigma_1, \cdots, \sigma_{2\kappa})$ checks for all $i \in [2\kappa]$ if $\sigma_i = S_{i,x_{[i]}}$ and if so it outputs $(x-1, \sigma'_1, \cdots, \sigma'_{2\kappa})$ where $x-1$ is computed modulo $2^{2\kappa}$ and for all $i \in [2\kappa]$ $\sigma'_i = S_{i,(x-1)_{[i]}}$.

- $\mathsf{SampGen}(SK)$:

  1. Choose K^2_ϕ and K from $\mathsf{KeyGen}_{\mathcal{PPRF}}(1^\kappa)$. K^2_ϕ works on inputs of length $\mathsf{rand}(\kappa)$ where $\mathsf{rand}(\cdot)$ is a polynomial that would be specified later and K works on inputs of length κ. Initialize $\mathsf{U}^i_\phi := S_i$ and $\mathsf{U}_\phi = \bigcup_{i\in[2\kappa]} \mathsf{U}^i_\phi$.
  2. Choose (PK^2_i, MSK^2_i) for all $i \in [\kappa]$.
  3. Sample $sk_2 \leftarrow \mathsf{SK.KeyGen}(1^\kappa)$ and set $\Pi_2 \leftarrow \mathsf{SK.Enc}_{sk_2}(\pi_2)$ and $\Lambda_2 \leftarrow \mathsf{SK.Enc}_{sk_2}(\lambda_2)$ where $\pi_2 = 0^{\ell_2(\kappa)}$ and $\lambda_2 = 0^{\ell'_2(\kappa)}$. Here $\ell_2(\cdot)$ and $\ell'_2(\cdot)$ are appropriate length functions specified later.
  4. For each $i \in [\kappa]$, generate $\mathsf{FSK}^2_i \leftarrow \mathsf{FE.KeyGen}(MSK^2_i, F^2_{i,PK^2_{i+1},\Pi_2})$ and $\mathsf{FSK}^2_{\kappa+1} \leftarrow \mathsf{FE.KeyGen}(MSK^2_{\kappa+1}, G^2_{\Lambda_2})$ where $F^2_{i,PK^2_{i+1},\Pi_2}, G^2_{\Lambda_2}$ are described in Figure 10.
  5. Let $c^2_\phi \leftarrow \mathsf{FE.Enc}_{PK^2_1}(\phi, U_\phi, K, K^2_\phi, 0^\kappa, 0)$.
  6. The sampler circuit has $\{\mathsf{FSK}^2_i\}_{i\in[\kappa]}$ and c^2_ϕ hardwired in its description and works as described below.

- Samp: The sampler takes pk where $(pk, sk) \leftarrow \mathsf{PK.KeyGen}(1^\kappa)$. It proceeds as follows:

  1. For $i \in [\kappa]$, compute $c^2_{pk_{[i-1]}\|0}, c^2_{pk_{[i-1]}\|1} := \mathsf{FE.Dec}(\mathsf{FSK}^2_i, c^2_{pk_{[i-1]}})$.
  2. Obtain $h_{pk} = (pk, \rho, \rho_1, \cdots, \rho_{2\kappa})$ as output of $\mathsf{FE.Dec}(\mathsf{FSK}^2_{\kappa+1}, c^2_{pk})$.
  3. Compute $K_{pk} := \mathsf{PK.Dec}_{sk}(\rho)$ and $\sigma_i := \mathsf{PK.Dec}_{sk}(\rho_i)$ for all $i \in [2\kappa]$
  4. Output $(pk\|K_{pk}, \sigma_1, \cdots, \sigma_{2\kappa})$.

Figure 8: Construction of TDP from $\mathcal{FE}$

F^1_{i,PK^1_{i+1},Π_1}

Hardcoded Values: i, PK^1_{i+1}, Π_1.
Input: ($x \in \{0,1\}^{i-1}$, V_x,W_x, K^1_x, sk_1, mode)

1. If ($\mathsf{mode} = 0$) then output $\mathsf{FE.Enc}_{PK^1_{i+1}}(x\|0, \mathsf{V}_{x\|0}, \mathsf{W}_{x\|0}, K^1_{x\|0}, sk, \mathsf{mode}; {K'}^1_{x\|0})$ and $\mathsf{FE.Enc}_{PK^1_{i+1}}(x\|1, \mathsf{V}_{x\|1}, \mathsf{W}_{x\|1}, K^1_{x\|1}, sk, \mathsf{mode}; {K'}^1_{x\|1})$, where for $b \in \{0,1\}$, $K^1_{x\|b} = \mathsf{PrefixPunc}(K^1_x, b\|0)$ and ${K^1}'_{x\|b} = \mathsf{PrefixPunc}(K^1_x, b\|1)$ and $(\mathsf{V}_{x\|0}, \mathsf{W}_{x\|0})$, $(\mathsf{V}_{x\|1}, \mathsf{W}_{x\|1})$ are computed using the efficient procedure from the Computability Lemma (Lemma 21).
2. Else recover $(x||0, c^1_{x\|0})$ and $(x\|1, c^1_{x\|1})$ from $\mathsf{SK.Dec}_{sk_1}(\Pi_1)$ and output $c^1_{x\|0}$ and $c^1_{x\|1}$.

$G^1_{v,\Lambda_1,w}$

Hardcoded Values: v, Λ_1, w
Input: $x \in \{0,1\}^{2\kappa}, \mathsf{V}_x, \mathsf{W}_x, K^1_x, sk_1, \mathsf{mode}$

1. If ($\mathsf{PRG}(\mathsf{Ext}_w(x)) = v$) then output $\perp$.
2. If $\mathsf{mode} = 0$, (Below $j = f(x)$. Recall from Section 3.1 that $f(x)$ is the smallest j such that $x = x_{[j]}\|1^{2\kappa-j}$.)
   (a) For each $i \in [2\kappa]$, set $\psi_i = \mathsf{PRG}_0(S_{i,x_{[i]}})$ (obtained from W^i_x for $i \le j$ and from V^i_x for $i > j$).
   (b) For each $i \in \{j, \ldots, 2\kappa\}$ set $\gamma_i = \mathsf{PRG}_1(S_{i,x_{[i]}})$ and $\beta_i = \mathsf{SK.Enc}_{\gamma_j,\cdots,\gamma_{2\kappa}}(S_{i,x_{[i]}+1})$, encrypting $S_{i,x_{[i]}+1}$ under $\gamma_j, \ldots \gamma_{2\kappa}$. (Using randomness obtained by expanding K^1_x sufficiently.). $S_{i,x_{[i]}}$ and $S_{i,(x+1)_{[i]}}$ are obtained from V^i_x for all $i \in [j, 2\kappa]$.
   (c) Output $((\psi_1, \ldots, \psi_{2\kappa}), (\beta_j, \ldots, \beta_{2\kappa}))$
3. Else recover (x, d_x) from $\mathsf{SK.Dec}_{sk_1}(\Lambda_1)$ and output d_x.

Figure 9: Circuits for simulating Public Key.

$(x+1, \sigma'_1, \cdots, \sigma'_{2\kappa})$ where $x+1$ is computed modulo $2^{2\kappa}$ we observe that the public key indeed computes a permutation on the domain.

The correctness of the trapdoor inversion follows directly from the definition of the domain.

Similarly, since mode is set to 0 in c^2_ϕ, the "hidden" mode in F^2_{i,PK^2_{i+1},Π_2} and $G^2_{\Lambda_2}$ is not triggered during an honest execution of the sampler. It follows from the correctness of functional encryption scheme that $h_{pk} = (pk, \rho, \rho_1, \cdots, \rho_{2\kappa})$ where $\rho = \mathsf{PK.Enc}_{pk}(K_{pk})$ and $\rho_i = \mathsf{PK.Enc}_{pk}(S_{i,(pk\|K_{pk})_{[i]}})$ for all $i \in [2\kappa]$. Hence, from the correctness of decryption of public key encryption, the output of

F^2_{i,PK^2_{i+1},Π_2}

Hardcoded Values: i, PK^2_{i+1}, Π_2.
Input: ($x \in \{0,1\}^{i-1}$, U_x, K_x, K^2_x, sk_2, mode)

1. If ($\mathsf{mode} = 0$) then output $\mathsf{FE.Enc}_{PK^2_{i+1}}(x\|0, \mathsf{U}_{x\|0}, K_{x\|0}, K^2_{x\|0}, sk, \mathsf{mode}; {K'}^2_{x\|0})$ and $\mathsf{FE.Enc}_{PK^2_{i+1}}(x\|1, \mathsf{U}_{x\|1}, K_{x\|1}, K^2_{x\|1}, sk, \mathsf{mode}; {K'}^2_{x\|1})$, where for $b \in \{0,1\}$, $K^2_{x\|b} = \mathsf{PrefixPunc}(K^2_x, b\|0)$ and ${K'}^2_{x\|b} = \mathsf{PrefixPunc}(K^2_x, b\|1)$ and $\mathsf{U}_{x\|0}$ and $\mathsf{U}_{x\|1}$ are computed as described in Computability Lemma (Lemma 21).
2. Else recover $(x||0, c^2_{x\|0})$ and $(x\|1, c^2_{x\|1})$ from $\mathsf{SK.Dec}_{sk_2}(\Pi_2)$ and output $c^2_{x\|0}$ and $c^2_{x\|1}$.

$G^2_{\Lambda_2}$

Hardcoded Values: Λ_2
Input: $pk \in \{0,1\}^\kappa, \mathsf{U}_{pk}, K_{pk}, K^2_{pk}, sk_2, \mathsf{mode}$

1. If $\mathsf{mode} = 0$,
   (a) For all $i \in [2\kappa]$, compute $\sigma_i := S_{i,(pk\|K_{pk})_{[i]}}$ from U_{pk}.
   (b) Compute $\rho \leftarrow \mathsf{PK.Enc}_{pk}(K_{pk})$ and $\rho_i \leftarrow \mathsf{PK.Enc}_{pk}(\sigma_i)$ for all $i \in [2\kappa]$ (Using randomness obtained by expanding K^2_{pk} sufficiently).
   (c) Output $(pk, \rho, \rho_1, \cdots, \rho_{2\kappa})$.
2. Else recover (pk, h_{pk}) from $\mathsf{SK.Dec}_{sk_2}(\Lambda_2)$ and output h_{pk}.

Figure 10: Circuits for simulating Sampler

the sampler is a set of correct associated signatures on the prefixes of $pk\|K_{pk}$.

Pseudorandom sampling: The pseudorandomness of the samples follows as a direct consequence of the following lemma.

Lemma 23 *($pk\|K_{pk}$) where $(pk, sk) \leftarrow \mathsf{PK.KeyGen}(1^\kappa)$ and $(pk\|K_{pk}, \cdot, \cdots, \cdot) \leftarrow \mathsf{Samp}(pk)$ is pseudorandom given $(\{\mathsf{FSK}^2_i\}_{i\in[\kappa+1]}, c^2_\phi)$ [6] and $(\{\mathsf{FSK}^1_i\}_{i\in[2\kappa+1]}, c^1_\phi)$ (which is equal to the public key).*

Proof We will show this through a hybrid argument.

- Hyb_0 : In this hybrid the adversary is given access to $\Big(pk\|K_{pk}, (\{\mathsf{FSK}^2_i\}_{i\in[\kappa+1]}, c^2_\phi), (\{\mathsf{FSK}^1_i\}_{i\in[2\kappa+1]}, c^1_\phi)\Big)$.

[6] Note that $(\{\mathsf{FSK}^2_i\}_{i\in[\kappa+1]}, c^2_\phi)$ defines the sampler.

New $\pi_2^\star, \lambda_2^\star$: We now change the plain text encrypted in Π_2, Λ_2 used in generating the sampler Samp as follows. Note that in Hyb_0, $\pi_2^\star$ was set to $0^{\ell_2(\kappa)}$ and $\lambda_2^\star$ was set to $0^{\ell_2'(\kappa)}$.

Let P_2 denote the set of all prefixes including the empty prefix of pk. Observe that $|P_2| = \kappa+1$. For every string $x \in \{0,1\}^{\leq\kappa}$, let c_x^2, h_x denote the output of Step 1,2 of the sampler when run with input x (Note that h_x is defined only for $x \in \{0,1\}^\kappa$). For every string $x \in P_2$, let y denote the string which is same as x except that the last bit of x is flipped. We define a new set Q_2 which is the set of all such y if $y \notin P_2$. Note that $|P_2 \cup Q_2| \leq 2\kappa + 2$. We define:

$$\pi_2^\star = ||_{x \in P_2 \cup Q_2} \ (x, c_x^2)$$
$$\lambda_2^\star = (pk, h_{pk})$$

We set $\ell_2(\kappa)$ and $\ell_2'(\kappa)$ to be the maximum size of $\pi_2^\star$ and $\lambda_2^\star$ respectively.

- Hyb_1 : In this hybrid we change Π_2, Λ_2 to $\Pi_2^\star, \Lambda_2^\star$ that encrypt $\pi_2^\star$ and $\lambda_2^\star$ (which are of length $\ell_2(\kappa)$ and $\ell_2'(\kappa)$) respectively instead of $0^{\ell_2(\kappa)}$ and $0^{\ell_2'(\kappa)}$. Notice that the knowledge of secret key sk_2 is not needed to simulate either Hyb_0 or Hyb_1 and hence the computational indistinguishability of Hyb_0 and Hyb_1 follows from the semantic security of symmetric key encryption (under the symmetric key sk_2).

- Hyb_2 : In this hybrid, we change how c_x^2 is generated for every $x \in P_2$. Note that for $x \in Q_2$ we do not change the ciphertext. In Hyb_1, $c_x^2 := \mathsf{FE.Enc}_{PK^2_{|x|+1}}(x, \mathsf{U}_x, K_x, K_x^2, 0^\kappa, 0; {K'}^2_x)$. In this hybrid, we change $c_x^2 := \mathsf{FE.Enc}_{PK^2_{|x|+1}}(x, 0^{2\kappa}, 0^\kappa, 0^\kappa, sk_2, 1; r_x^2)$ where r_x^2 is uniformly chosen random string. Observe that in Hyb_2, the "hidden" mode in $F^2_{i,PK^2_{|i+1|},\Pi_2^\star}$ and $G^2_{\Lambda_2^\star}$ is triggered along the path from the root to the leaf labeled pk.

  We are going to accomplish this change by a couple of intermediate hybrids. Before describing the intermediate hybrids, let us introduce an ordering of elements in P_2. For any two elements $x, y \in P_2$, $x < y$ if x is a prefix of y [7]. Observe that by this ordering ϕ is the smallest element in P_2. Let $\mathsf{Hyb}_{1,y}$ denote an hybrid where for all $x < y$, c_x^2 is set to $\mathsf{FE.Enc}_{PK_{|x|+1}}(x, 0^{2\kappa}, 0^\kappa, 0^\kappa, sk, 1; r_x^2)$. We first show that Hyb_1 is computationally indistinguishable to $\mathsf{Hyb}_{1,\phi}$ and then show that for adjacent unequal $x, x' \in P_2$ such that $x < x'$, $\mathsf{Hyb}_{1,x}$ is computationally indistinguishable to $\mathsf{Hyb}_{1,x'}$. Note that this is sufficient to show that Hyb_2 is computationally indistinguishable to Hyb_1 [8].

  - $\mathsf{Hyb}_{1,\phi}$: In this hybrid, we change c_ϕ^2 to $\mathsf{FE.Enc}_{PK_1^2}(\phi, 0^{2\kappa}, 0^\kappa, 0^\kappa, sk_2, 1)$. Computational indistinguishability of Hyb_1 and $\mathsf{Hyb}_{1,\phi}$ follows from the single key, selective security of functional encryption scheme. Notice that both $(\phi, 0^{2\kappa}, 0^\kappa, 0^\kappa, sk_2, 1)$ and $(\phi, U_\phi, K, K_\phi^2, 0^\kappa, 0)$ trigger the same output in $F^2_{1,PK_2,\Pi_2^\star}$. This is because of the values encrypted in $\Pi_2^\star$, $(\phi, 0^{2\kappa}, 0^\kappa, 0^\kappa, sk_2, 1)$ produces the exact same output as that of $(\phi, U_\phi, K, K_\phi^2, 0^\kappa, 0)$. Further, the choice of the two messages does not depend on the knowledge of PK_1^2.

[7] Note that by this ordering $y < y$.

[8] The loss in the reduction is only linear since $|P_2| = \kappa + 1$

– $\mathsf{Hyb}_{1,x,1}$: In this hybrid we change $c^2_{x'}$ (encrypted in $\Pi^\star_2$) to $\mathsf{FE.Enc}_{PK^2_{|x'|+1}}(x', \mathsf{U}_{x'}, K_{x'}, K^2_{x'}, 0^\kappa, 0; r^2_{x'})$ where $r^2_{x'}$ is chosen uniformly and independently at random. Note that for every prefix y of x' (that is not equal to x'), $y < x'$ and hence K^2_y is set to 0^κ in c^2_y. This observation along with the pseudorandomness at prefix punctured point property of the PPRF shows that $\mathsf{Hyb}_{1,x}$ is computationally indistinguishable to $\mathsf{Hyb}_{1,x,1}$.

– $\mathsf{Hyb}_{1,x,2}$: In this hybrid we change $c^2_{x'}$ (encrypted in $\Pi^\star_2$) to $\mathsf{FE.Enc}_{PK^2_{|x'|+1}}(x', 0^{2\kappa}, 0^\kappa, 0^\kappa, sk_2, 1; r^2_{x'})$. Computational indistinguishability of $\mathsf{Hyb}_{1,x,1}$ and $\mathsf{Hyb}_{1,x,2}$ follows from single key, selective security of functional encryption. Note that $F^2_{|x'|+1, PK^2_{|x'|+2}, \Pi^\star_2}$ (or $G^2_{\Lambda^\star_2}$) has the same output for both $(x', 0^{2\kappa}, 0^\kappa, 0^\kappa, sk, 1)$ and $(x', \mathsf{U}_{x'}, K_{x'}, K^2_{x'}, 0^\kappa, 0)$ because of the values encrypted in $\Pi^\star_2$ (or $\Lambda^\star_2$). Further, the choice of the two messages does not depend on the knowledge of $PK^2_{|x'|+1}$.
Observe that $\mathsf{Hyb}_{1,x,2}$ is identically distributed to $\mathsf{Hyb}_{1,x'}$

Note that in Hyb_2 on input pk, the $G^2_{\Lambda^*_2}$ uses h_{pk} encrypted in $\Lambda^\star_2$ to generate the output. Note that h_{pk} is given by $(pk, \rho, \rho_1, \cdots, \rho_{2\kappa})$ where ρ is an encryption of K_{pk} and ρ_i is an encryption of $S_{i,(pk\|K_{pk})_{[i]}}$ under pk. Note that the encryptions are generated by expanding K^2_{pk} sufficiently.

- Hyb_3 : In this hybrid, we are going to generate the encryptions $\rho, \{\rho_i\}_{i\in[2\kappa]}$ in h_{pk} (encrypted in $\Lambda^\star_2$) using true random strings instead of using stings generated by expanding K^2_{pk}. Note that in Hyb_2, K^2_z where z is a prefix of pk does not appear anywhere (They are all set to 0^κ in c^2_z). Hence, any string derived from K^2_{pk} can be changed to truly random strings by relying on the pseudorandomness under prefix puncturing property of PPRF.

- Hyb_4 : In this hybrid, we are going to change $\rho, \{\rho_i\}_{i\in 2[\kappa]}$ in h_{pk} (encrypted in $\Lambda^\star_2$) to encryptions of 0^κ. Observe that pk is sampled using the $\mathsf{PK.KeyGen}$ algorithm. Also, knowledge of the secret key sk associated with pk is not needed to simulate Hyb_3 or Hyb_4 [9]. Thus, we can make this change by relying on the semantic security of public key encryption.

- Hyb_5 : In this hybrid, we are going to change K_{pk} in the challenge given to the adversary to a random string z. Observe that in Hyb_4, K_y where y is a prefix of pk does not appear anywhere (They are all set to 0^κ in c^2_y.) and $S_{j,(pk\|K_{pk})_{[j]}}$ for $j > \kappa$ is also set to 0^κ in h_{pk}. Thus, K_y where y is a prefix of pk and K_{pk} are not needed to simulate the sampler in Hyb_4 and Hyb_5. Hence, indistinguishability between Hyb_4 and Hyb_5 follows from the pseudorandomness under prefix puncturing property of PPRF.

- Hyb_6 to Hyb_{10}: We essentially reverse the modifications we had done to the function keys and the initial ciphertext used in the computation of the sampler.

Note that in Hyb_{10}, an adversary is given $(pk\|z)$, the public key $(\{\mathsf{FSK}^1_i\}_{i\in[2\kappa+1]}, c^1_\phi)$ and the sampler $(\{\mathsf{FSK}^2_i\}_{i\in[\kappa+1]}, c^2_\phi)$. It follows from the random public key property of the public key encryption pk is randomly distributed and z is randomly distributed. Also, we know that pk, z do not occur anywhere (in an information theoretic sense) in the modified function keys and the initial ciphertext used in the sampler computation. Hence, $pk\|z$ is randomly distributed. ■

[9] Note that knowledge of pk is needed to generate the two sequence of messages.

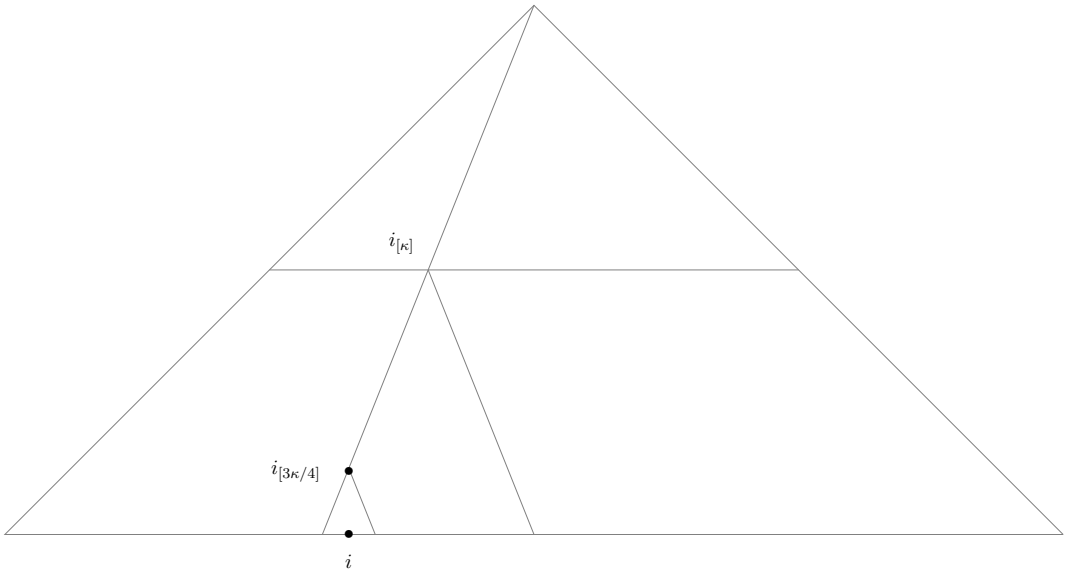

Figure 11: Condition for Aborting

One-Wayness: We now show that the trapdoor permutation is hard to invert given the public key and the sampler.

- Hyb_0 : This hybrid is same as the inversion experiment where the adversary is given $((pk\|K_{pk})+1, \sigma_1, \cdots, \sigma_{2\kappa})$ where $(pk, sk) \leftarrow \mathsf{PK.KeyGen}(1^\kappa)$ and the sampler is run with random input as pk to obtain $(pk\|K_{pk}, \cdot, \cdots, \cdot)$ and then we compute TDP_{PK} on this. For brevity of notation we will denote $(pk\|K_{pk}) + 1$ as i.

- Hyb_1 : We modify the function keys and the initial ciphertext used in the computation of the sampler such that it is distributed identically to Hyb_5 in the proof of Lemma 23. Hyb_0 and Hyb_1 are computationally indistinguishable as a direct consequence of Lemma 23. Let i denote the challenge to the adversary $pk\|z$ as per the proof of Lemma 23.

  Let $i_1 \cdots i_{2\kappa}$ denote the binary representation of i. If $i_{\kappa+1} \cdots i_{7\kappa/4}$ is equal to $0^{3\kappa/4}$, we abort. Figure 11 illustrates the condition for aborting. Notice that since $z = i_{\kappa+1} \cdots i_{2\kappa}$ is randomly distributed the probability that we abort is at most $\frac{1}{2^{3\kappa/4}}$ which is negligible. Hence, we subsequently assume that we have not aborted [10]. Observe that if we have not aborted then as per the proof of Lemma 23, in the final hybrid, the description of the sampler (the function keys and the initial ciphertext) cannot be used to derive (in an information theoretic sense) $S_{j,z_{[j]}}$ where $z_{[\kappa]} = i_{[\kappa]}$ and $j \geq \kappa$ (They are all set to 0^κ).

- Hyb_2 : In this hybrid, we change how the value v that is hardwired in $G_{v,\Lambda_1,w}$ is generated. Let us introduce a lexicographic ordering of the binary strings of length $3\kappa/4$. Let $i^\star$ be the string just before $i_{[3\kappa/4]}$ by this ordering. We set $v := \mathsf{PRG}(\mathsf{Ext}_w(i^\star\|u_0))$ where $u_0 \in \{0,1\}^{\kappa/4}$. The indistinguishability of Hyb_1 and Hyb_2 can be observed from the following argument:

[10] We introduce abort in our analysis to simplify the proof. We note that it is possible to prove the one-wayness property with a tighter security reduction without using abort.

– $\mathsf{Hyb}_{1,1}$: In this hybrid, instead of choosing v uniformly at random from $[2^{\kappa/4}]$, we set $v := \mathsf{PRG}(v')$ where $v' \xleftarrow{\$} \{0,1\}^{\kappa/8}$. The indistinguishability of Hyb_1 and $\mathsf{Hyb}_{1,1}$ follows from the pseudorandomness property of PRG.

– $\mathsf{Hyb}_{1,2}$: In this hybrid, we set $v := \mathsf{PRG}(\mathsf{Ext}_w(i^\star \| u_0))$ where $u_0 \xleftarrow{\$} \{0,1\}^{\kappa/4}$. Note that $i^\star \| u_0$ has min entropy $\kappa/4$ (because u_0 is chosen uniformly at random from $\{0,1\}^{\kappa/4}$) and hence the statistical closeness of $\mathsf{Hyb}_{1,1}$ and $\mathsf{Hyb}_{1,2}$ follows from the extractor guarantee. Note that $\mathsf{Hyb}_{1,2}$ is identically distributed to Hyb_2.

Note that in this hybrid the public key outputs $\perp$ on all inputs of the form $(i - u_0, \cdot, \cdots, \cdot)$. For brevity of notation we denote $\alpha_0 := i - u_0$.

In the subsequent hybrids, we are going to puncture the public key of the permutation such that it outputs $\perp$ on all inputs in the range $[\alpha_0, i-1]$. Once we have done that no adversary has non-zero advantage in inverting the permutation at $(i, \cdot, \cdots, \cdot)$. Observe that since we have not aborted, for all $z \in [\alpha_0, i-1]$ the sampler does not contain $S_{j,z_{[j]}}$ for all $j \geq \kappa$ [11]. This observation will be crucial in allowing us to puncture the public key.

Recalling notation for α_j. We denote $\alpha_0 := i - u_0$. Recall from Section 3.1, for any string $\alpha \in \{0,1\}^{2\kappa}$, let $f(\alpha)$ denote the index of the lowest order bit of α that is 0 (with the index of the highest order bit being 1). More formally, $f(\alpha)$ is the smallest j such that $\alpha = \alpha_{[j]} || 1^{2\kappa - j}$. For example, if $\alpha = \overbrace{100}^{3}111$ then $f(\alpha) = 3$. Recall $\ell(\beta, \gamma)$ denotes the smallest $j \in [2\kappa]$ such that $\beta_{[j-1]} = \gamma_{[j-1]}$ and $\beta_j \neq \gamma_j$ if $\beta \neq \gamma$ and is a special symbol ζ otherwise. Recall $\rho(\alpha_k) = \ell(\alpha_k + 1, i-1)$ if $\ell(\alpha_k + 1, i-1) \neq \zeta$ and equal to 2κ otherwise. Let $\delta(\alpha)$ denote the number of 0s in the positions $[\ell(\alpha, i-1) + 1, 2\kappa]$ in the binary representation of α if $\ell(\alpha, i-1) \neq \zeta$ and is equal to 0 otherwise. Let $\mu(\alpha)$ denote one more than the number of ones in the positions $[\ell(\alpha, i-1) + 1, 2\kappa]$ in the binary representation of $i-1$ if $\ell(\alpha, i-1) \neq \zeta$ and is equal to 0 otherwise.

Starting with a value $\alpha_0 \in \{0,1\}^{2\kappa}$ we define for $j \in [0, \delta(\alpha_0) + \mu(\alpha_{\delta(\alpha_0)}) - 1]$,

$$\alpha_{j+1} = \begin{cases} \alpha_j + 2^{2\kappa - f(\alpha_j)} & \text{if } j+1 \leq \delta(\alpha_0) \\ \alpha_j + 2^{2\kappa - \rho(\alpha_j)} & \text{otherwise} \end{cases}$$

Note that by this definition $\alpha_{\delta(\alpha_0) + \mu(\alpha_{\delta(\alpha_0)})} = i - 1$. Also, $\delta(\alpha_0) + \mu(\alpha_{\delta(\alpha_0)}) \leq 4\kappa - 1$.

New π_1, λ_1 values. As in Section 3.1 we process the hybrids according to α_j values. For any $x \in \{0,1\}^{\leq 2\kappa}$, let c_x^1 denote the ciphertext and d_x the clear output in execution of Steps 1 and 2 of TDP_{PK} in Hyb_2 on input x. We let P_1 be the set of all prefixes of $\alpha_0, \alpha_1, \ldots \alpha_{\delta(\alpha_0) + \mu(\alpha_{\delta(\alpha_0)})}$ including the empty string ϕ. Note that $|P_1| \leq 4\kappa(2\kappa + 1)$. Additionally we define Q_1 as follows. For every $x \in P_1$, let y be the value with the last bit of x flipped. We add y to Q_1 if $y \notin P_1$. We set:

$$\pi_1^\star = ||_{x \in P_1 \cup Q_1} \ (x, c_x^1)$$

[11] In fact it does not contain $S_{j,z_{[j]}}$ such that $z_{[j]} = i_{[j]}$ for all $j \geq \kappa$. Since we have not aborted, for every $y \in [\alpha_0, i-1]$, we have $y_{[\kappa]} = i_{[\kappa]}$

$$\lambda_1^\star = ||_{x \in P_1 \cap \{0,1\}^{2\kappa}} (x, d_x)$$

We set $\ell_1(\kappa)$ and $\ell_1'(\kappa)$ to be the polynomials that describe an upper bound on the lengths of $\pi_1^\star$ and $\lambda_1^\star$ over all choices of $\alpha_0 \in \{0,1\}^{2\kappa}$.

- Hyb_3 : In this hybrid we change how the hardcoded values Π_1 and Λ_1 are generated. Unlike hybrids Hyb_1 and Hyb_2 where these values were generated as encryptions of $0^{\ell_1(\kappa)}$ and $0^{\ell_1'(\kappa)}$, in this hybrid we generate them as encryptions $\pi_1^\star$ and $\lambda_1^\star$ describe above, respectively. Let us denote the new hardcoded values to be $\Pi_1^\star$ and $\Lambda_1^\star$.

  Notice that knowledge of secret key sk_2 is not needed in simulating the function keys and the initial ciphertext used in the function computation. Also, the length of the underlying messages is same in Hyb_2 and Hyb_3. Thus, computational indistinguishability between Hyb_2 and Hyb_3 follows from the semantic security of the symmetric key encryption scheme (under sk_1).

- Hyb_4: In this hybrid, for $x \in P_1$ we change the the c_x^1 values embedded in $\pi_1^\star$. Recall that in hybrid Hyb_2 for each x, c_x^1 is generated as $\mathsf{FE.Enc}_{PK^1_{|x|+1}}(x, \mathsf{V}_x, \mathsf{W}_x, K_x^1, 0^\kappa, 0; K'^1_x)$. We change the c_x^1 to be now generated as $\mathsf{FE.Enc}_{PK^1_{|x|+1}}(x, 0^{2\kappa}, 0^{2\kappa}, 0^\kappa, sk, 1; \omega_x)$ using fresh randomness ω_x.[12]

  Computational indistinguishability between Hyb_3 and Hyb_4 follows by a sequence of sub-hybrids. We define an ordering on elements of P_1 as follows. For $x, y \in P$ we say that $x < y$ if either $|x| < |y|$, or $|x| = |y|$ and $x < y$ [13]. Next we define the hybrid $\mathsf{Hyb}_{3,y}$ to be a modification of Hyb_3 where for all $x \in P$ such that $x \leq y$ we have that c_x^1 is generated as $\mathsf{FE.Enc}_{PK^1_{|x|+1}}(x, 0^{2\kappa}, 0^{2\kappa}, 0^\kappa, sk, 1; \omega_x)$ using fresh randomness ω_x.

  We first argue that Hyb_3 is computationally indistinguishable to $\mathsf{Hyb}_{3,\phi}$ and then argue that $\mathsf{Hyb}_{3,x'}$ and $\mathsf{Hyb}_{3,x}$ are indistinguishable for any two adjacent values x' and x in P such that $x' < x$. This is sufficient to show that Hyb_3 and Hyb_4 are indistinguishable with a polynomial loss in the security reduction. We argue this via a three step hybrid argument.

  1. $\mathsf{Hyb}_{3,\phi}$: In this hybrid, we change c_ϕ^1 to $\mathsf{FE.Enc}_{PK_1^1}(\phi, 0^{2\kappa}, 0^{2\kappa}, 0^\kappa, sk, 1)$. Notice that both $(\phi, 0^{2\kappa}, 0^{2\kappa}, 0^\kappa, sk, 1)$ and $(\phi, \mathsf{V}_\phi, \mathsf{W}_\phi, K_\phi^1, 0^\kappa, 0)$ give the same output on $F^1_{1,PK_2^1,\Pi_2^\star}$ because of the value encrypted in $\Pi_2^\star$. Further more the choice of two messages does not depend on the value of PK_1^1. Hence, computational indistinguishability of Hyb_3 and $\mathsf{Hyb}_{3,1}$ follows from the single key selective security of Functional encryption with public key PK_1^1.
  2. $\mathsf{Hyb}_{3,x,1}$: In this hybrid we change c_x^1 to $\mathsf{FE.Enc}_{PK^1_{|x|+1}}(x, \mathsf{V}_x, \mathsf{W}_x, K_x^1, 0^\kappa, 0; \omega_x)$ using fresh randomness ω_x.

     Note that for all prefixes x'' of x we have that $x'' < x$. Therefore for all such x'' we have that $c_{x''} = \mathsf{FE.Enc}_{PK^1_{|x''|+1}}(x'', 0^{2\kappa}, 0^{2\kappa}, 0^\kappa, sk, 1; \omega_{x''})$. This fact along with the pseudorandom at punctured point property implies computational indistinguishability from the previous hybrid, namely $\mathsf{Hyb}_{3,x}$.

[12] Note that we do not change ciphertexts corresponding to $x \in Q$.

[13] Note that ϕ is the smallest value in P by this ordering.

3. $\mathsf{Hyb}_{3,x,2}$: In this hybrid we change c^1_x to $\mathsf{FE.Enc}_{PK^1_{|x|+1}}(x, 0^{2\kappa}, 0^{2\kappa}, 0^\kappa, sk, 1; \omega_x)$ using fresh randomness ω_x.

   The computational indistinguishability of this hybrid from $\mathsf{Hyb}_{3,x,2}$ relies on the selective security of the functional encryption scheme with public-key $PK^1_{|x|+1}$. Note that we can invoke security of functional encryption as the change in the messages being encrypted does not change the output of the decryption using key $\mathsf{FSK}_{|x|+1}$. Also, the choice of the two messages does not depend on the value of $PK^1_{|x|+1}$.

- $\mathsf{Hyb}_{5,j+1}$: In hybrid $\mathsf{Hyb}_{5,j+1}$ for $j \in \{0, \ldots, \delta(\alpha_0) + \mu(\alpha_{\delta(\alpha_0)}) - 1\}$, we make two changes with respect to $\mathsf{Hyb}_{5,j}$. We define $\mathsf{Hyb}_{5,0}$ to be same as Hyb_4. Just like in Section 3.1 we let ν_j as the shorthand for $f(\alpha_j)$ for $j < \delta(\alpha_0)$ and equal to $\rho(\alpha_j)$ for $j \geq \delta(\alpha_0)$. Let $t_j = \alpha_{j_{[\nu_j]}} + 1$.

  – We change the set $\mathsf{W}^{\nu_j}_{t_j}$ to be a uniformly random string $z \leftarrow \{0,1\}^{2\kappa}$ rather than containing the value $\mathsf{PRG}_0(S_{\nu_j,t_j})$. Note that this change needs to be made at two places. Namely we set $\mathsf{W}^{\nu_j}_{t_j} = \{z\}$ in c^1_z where z is a sibling path of α_{j+1} and from there on this value will be percolated to all its descendents as well. Additionally we set ψ_{ν_j} included in $d_{\alpha_{j+1}}$ to be z.

  – We now generate encryptions $\beta_{\nu_j}, \ldots, \beta_\kappa$ included in $d_{\alpha_{j+1}}$ with encryption of 0^κ.

  Note that as a consequence of this change the public key now starts to output $\perp$ additionally on all inputs in $\{\alpha_j+1, \ldots, \alpha_{j+1}\}$. This is because for every input σ_{ν_j} we have that $z \neq \mathsf{PRG}_0(\sigma_{\nu_j})$ with overwhelming probability. Since in hybrid $\mathsf{Hyb}_{5,j}$ the successor was already outputting $\perp$ on inputs $\{\alpha_0, \ldots, \alpha_j\}$ we have that the successor outputs $\perp$ on all inputs in $\{\alpha_0, \ldots, \alpha_{j+1}\}$.

  Now we argue computational indistinguishability between $\mathsf{Hyb}_{5,j}$ and $\mathsf{Hyb}_{5,j+1}$.

  – $\mathsf{Hyb}_{5,j,1}$: In this hybrid instead we replace the key S_{ν_j,t_j} with a random string $S' \leftarrow \{0,1\}^\kappa$. Now S' (instead of S_{ν_j,t_j}) is used in $W^{\nu_j}_{t_j}$, in generating γ_{ν_j} used in $d_{\alpha_{j+1}}$ and in ψ_{ν_j} in $d_{\alpha_{j+1}}$.

    Computational indistinguishability follows from the pseudorandomness at prefix punctured point property of PPRF. This argument relies on that facts that the punctured keys in the sampler cannot be used to derive S_{ν_j,t_j} and that no V set that hasn't been removed can be used to obtain S_{ν_j,t_j}. More formally,

    - Recall a prior observation that the sampler cannot be used to derive (in an information theoretic sense) $S_{j,z_{[j]}}$ for all $z \in [\alpha_0, i-1]$ and for all $j \geq \kappa$. In particular, since $\nu_j > \kappa$ (this is because α_0 and $i-1$ are at distance at most $2^{\kappa/4+1}$ and thus $|\alpha_{j+1} - \alpha_j| \leq 2^{\kappa/4} + 1$) we can infer that the sampler does not contain S_{ν_j,t_j}.
    - $\mathsf{V}^{\nu_j}_y$ values have been removed whenever y is a prefix of α_j or α_{j+1}. Note that it follows from the Derivability Lemma (Lemma 22) that these were the only V-sets that could be used to derive S_{ν_j,t_j}.
    - Additionally $\sigma_{\nu_j} = S_{\nu_j,t_j}$ is encrypted in β_{ν_j} and this value is included in d_{α_j}. But this has already been replaced with an encryption of 0^κ except d_{α_0} which is set to $\perp$.

  – $\mathsf{Hyb}_{5,j,2}$: In this hybrid instead we replace the $\beta_{\nu_j}, \ldots, \beta_\kappa$ in $d_{\alpha_{j+1}}$ to be generated using fresh randomness.

Observe that K_y^1 where y is a prefix of α_{j+1} does not occur anywhere in the simulation of $\mathsf{Hyb}_{5,j,2}$ and in $\mathsf{Hyb}_{5,j,1}$. Thus, computational indistinguishability follows from the pseudorandomness at prefix punctured point property.

- $\mathsf{Hyb}_{5,j,3}$: In this hybrid, we change $\mathsf{PRG}_0(S')$ and $\mathsf{PRG}_1(S')$ to be random strings z, z'. Change of $\mathsf{PRG}_0(S')$ to z implies that the set $\mathsf{W}_{t_j}^{\nu_j}$ is $\{z\}$ and ψ_{ν_j} in $d_{\alpha_{j+1}}$ is also set to z. Similarly γ_{ν_j} will be z'.

  Indistinguishability between $\mathsf{Hyb}_{5,j,2}$ and $\mathsf{Hyb}_{5,j,3}$ follows from the pseudorandomness property of the PRG as the plaintext value S' is not needed anywhere in the simulation of $\mathsf{Hyb}_{5,j,2}$ or in $\mathsf{Hyb}_{5,j,3}$.
- $\mathsf{Hyb}_{5,j,4}$: In this hybrid we set encryption of all $\beta_{\nu_j}, \ldots, \beta_\kappa$ in $d_{\alpha_{j+1}}$ with encryption of 0^κ. By semantic security we have this hybrid is computationally indistinguishable from the previous. Here we rely on the fact that one of the keys γ_{ν_j} has been replaced with random.

  Note that hybrid $\mathsf{Hyb}_{5,j,4}$ is same as hybrid $\mathsf{Hyb}_{5,j+1}$.

Concluding the proof. Observe that the hybrid $\mathsf{Hyb}_{5,0}$ is defined to be identical to hybrid Hyb_4 and $\mathsf{Hyb}_{5,\delta(\alpha_0)+\mu(\alpha_{\delta(\alpha_0)})}$ is such that the public key outputs $\perp$ on all inputs of the form $(i-1, \cdot, \ldots, \cdot)$. Consequently, no adversary can invert the challenge $(i, \sigma_1, \cdots, \sigma_{2\kappa})$ in this final hybrid with probability better than 0.

5 Universal Samplers

Intuitively, a universal sampler, defined by Hofheinz et al. [HJK+14] is a box that takes as input the description of a sampling procedure, and outputs a fresh-looking sample according to the sampling procedure. The difficulty is that we want the box to be public code, and that every user, when they run the sampler on a particular procedure, gets the same result. Moreover, we want the sample to appear as if it were a fresh random sample.

5.1 Definition

A *Universal Sampler* consists of an algorithm Setup that takes as input a security parameter κ and a size bound ℓ (encoded in unary), and an output size t. It outputs a program $\mathsf{Sampler}$. $\mathsf{Sampler}$ takes as input a circuit of size at most ℓ with output length t, and outputs an t-bit string.

Intuitively, $\mathsf{Sampler}(C)$ will be a pseudorandom sample from C: $\mathsf{Sampler}(C) = C(r)$ for some r pseudorandomly chosen based on C. We will actually not formalize a standalone correctness requirement, but instead correctness will follow from our security notion.

For security, we ask that the sample outputted by $\mathsf{Sampler}(C)$ actually looks like a fresh random sample from C. Unfortunately, formalizing this requirement is tricky. Hofheinz et al. [HJK+14] defined two notions: the first is a "static" and "bounded" security notion, while the second stronger notion is "adaptive" and "unbounded". The latter definition requires random oracles, so it is unfortunately uninstantiable in the standard model. We will provide a third definition which strikes some middle ground between the two, and is still instantiable in the standard model.

Definition 24 *A Universal Sampler given by* Setup *is* n*-time* statically secure with interactive simulation *if there exists an efficient randomized simulator* Sim *such that the following hold.*

- Sim *takes as input* κ, ℓ, t*, and ultimately will output a simulated sampler* $\mathsf{Sampler}$*. However, before doing so,* Sim *provides the following interface for additional input:*
  - **Read** *queries: here the user submits an input circuit* C *of size at most* ℓ *with output length* t*.* Sim *will respond with a sample* s *that will ultimately be the output of the simulated sampler on* C*.* Sim *supports an unbounded number of* **Read** *queries.*
  - **Set** *queries: here the user submits in input circuit* C *of size at most* ℓ *with output length* t*, as well as a sample* s*.* Sim *will record* C, s*, and set the output of the simulated sampler on* C *to be* s*.* Sim *supports up to* n **Set** *queries. We require that there is no overlap between circuits* C *in* **Read** *and* **Set** *queries, and that all* **Set** *queries are for distinct circuits.*
  - **Finish** *query: here, the user submits nothing, and* Sim *closes its interfaces, terminates, and outputs a sampler* $\mathsf{Sampler}$*.*

  Sim *must be capable of taking the queries above in any order.*
- **Correctness.** $\mathsf{Sampler}$ *is consistent with any queries made. That is, if a* **Read** *query was made on* C *and the response was* s*, then* $\mathsf{Sampler}(C) = s$*. Similarly, if a* **Set** *query was made on* C, s*, then* $\mathsf{Sampler}(C) = s$*.*
- **Indistinguishability from honest generation.** *The advantage of any polynomial-time algorithm* A *is negligible in the following experiment:*
  - *The challenger flips a random bit* b*. If* $b = 0$*, the challenger runs* $\mathsf{Sampler} \leftarrow \mathsf{Setup}(1^\kappa, 1^\ell, 1^t)$*. If* $b = 1$*, the challenger initiates* $\mathsf{Sim}(1^\kappa, 1^\ell, 1^t)$*.*
  - A *is allowed to make* **Read** *queries on arbitrary circuits* C *of size at most* ℓ *with output length* t*. If* $b = 0$*, the challenger runs* $s \leftarrow \mathsf{Sampler}(C)$ *and responds with* s*. If* $b = 1$*, the challenger forwards* C *to* Sim *as a* **Read** *query, and when* Sim *responds with* s*, the challenger forwards* s *to* A*.*
  - *Finally,* A *sends a* **Finish** *query. If* $b = 0$*, the challenger then sends* $\mathsf{Sampler}$ *to* A*. If* $b = 1$*, the challenger sends a* **Finish** *query to* Sim*, gets* $\mathsf{Sampler}$ *from* Sim*, and forwards* $\mathsf{Sampler}$ *to* A*.*
  - A *then tries to guess* b*. The advantage of* A *is the advantage* A *has in guessing* b*.*
- **Pseudorandomness of samples.** *The advantage of any polynomial-time algorithm* B *is negligible in the following experiment:*
  - *The challenger flips a random bit* b*. It then initiates* $\mathsf{Sim}(1^\kappa, 1^\ell, 1^t)$*.*
  - B *first makes a* **Challenge** *query on circuit* C^* *of size at most* ℓ *and output length* t*, as well as an integer* i^**.*
  - B *is allowed to make arbitrary* **Read** *and* **Set** *queries, as long as the number of* **Set** *queries is at most* $n - 1$*, and the queries are all on distinct circuits that are different from* C^**. The* **Read** *and* **Set** *queries can occur in any order; the only restriction is that the* **Challenge** *query comes before all* **Read** *and* **Set** *queries.*
  - *After* $i^* - 1$ **Read** *and* **Set** *queries, the challenger does the following:*

* *If $b = 0$, the challenger makes a* **Read** *query to* Sim*, and forwards the response s^* to B.*
* *If $b = 1$, the challenger computes a fresh random sample $s^* \leftarrow C^*(r)$, and makes a* **Set** *query to* Sim *on C^*, s^*. Then it gives s^* to B.*

Thus the i^*th query made to Sim *is on circuit C^*, and the only difference between $b = 0$ and $b = 1$ is whether the output of the simulated sampler will be a pseudorandom sample or a fresh random sample from C^*.*

– *B is allows to continue making arbitrary* **Read** *and* **Set** *queries, as long as the number of* **Set** *queries is at most $n-1$ and the queries are all on distinct circuits that are different from C^*.*

– *Finally B makes a* **Finish** *query, at which point the challenger makes a* **Finish** *query to* Sim*. It obtained a simulated sampler* $\mathsf{Sampler}$*, which it then gives to B.*

– *B then tries to guess b. The advantage of B is the advantage B has in guessing b.*

5.2 Approach Using Obufscation

Using obfuscation, obtaining a universal sampler meeting the above definition is straightforward. Setup chooses a random seed S for a puncturable PRF $\mathcal{PRF}$. Then it obfuscations the program $P(C) = C(\ \mathcal{PRF}_S(C)\ )$.

The simulator Sim works as follows. Upon initialization, it chooses a random seen S. To answer a **Read** query on C, it simply outputs $C(\ \mathcal{PRF}_S(C)\ )$. Upon receiving a **Write** query on C, s, it just records the pair C, s. Finally, upon receiving the **Finish** query, it obfuscates the program $P_{(C_1,s_1),\ldots,(C_n,s_n)}$ where (C_i, s_i) are the **Write** queries received, and $P_{(C_1,s_1),\ldots,(C_n,s_n)}$ is the program that outputs s_i on input C_i for **Write** queries (C_i, s_i), and otherwise outputs $P(C) = C(\ \mathcal{PRF}_S(C)\ )$.

Correctness of simulation is trivial, as is the indistinguishability from honest generation. Pseudorandomness of samples follows from a straightforward application of the punctured programming technique. It is proved through the following sequence of hybrids:

- **Hybrid 0.** This is the $b = 0$ case in the pseudorandomness game. The **Challenge** query is answered by making a **Read** query to Sim, which responds with $s^* = C^*(\ \mathcal{PRF}_S(C^*)\ )$. The final program produced by Sim is $P_{(C_1,s_1),\ldots,(C_n,s_n)}$ where $n \leq k-1$ and (C_i, s_i) correspond to the write queries made by the adversary.

- **Hybrid 1.** In this hybrid, we puncture S at C^*, and hardcore s^* into the program as the output on C^*. We did not change the functionality of the program, so indistinguishability from **Hybrid 0** follows from iO.

- **Hybrid 2.** Now we replace $\mathcal{PRF}_S(C^*)$ with a truly random r^* in the generation of s^*. That is $s^* = C^*(r^*)$. Indistinguishability from **Hybrid 1** follows from punctured PRF security.

- **Hybrid 3.** Now we un-puncture S, but keep s^* as the hardcoded output of the program on input C^*. This does not change the functionality of the program, so indistinguishability from **Hybrid 2** follows from iO.

  Notice that the program is now identical to $P_{(C^*,s^*),(C_1,s_1),\ldots,(C_n,S_n)}$. Therefore, this exactly simulates the case $b = 1$. Thus the $b = 0$ and $b = 1$ cases are indistinguishability, and security therefore follows.

- **Sampled Ingredients**:

  1. Sample S and K_ϕ from $\mathsf{KeyGen}_{\mathcal{PPRF}}(1^\kappa)$. Here S is a key that works for ℓ bit inputs and produces a r bit outputs, namely $\mathsf{PPRF}_S : \{0,1\}^\ell \to \{0,1\}^r$ where ℓ and r are polynomial functions of κ. Similarly, K_ϕ work on inputs of length $\mathsf{rand}(\kappa)$ where $\mathsf{rand}(\cdot)$ is a polynomial that would be specified later.
  2. Sample $(PK_i, MSK_i) \leftarrow \mathsf{FE.Setup}(1^\kappa)$ for all $i \in [\ell+1]$.
  3. Sample $sk_1, \cdots, sk_n \leftarrow \mathsf{SK.KeyGen}(1^\kappa)$ and let $\Pi_i^j \leftarrow \mathsf{SK.Enc}_{sk}(\pi_i^j)$ where $\pi_i^j = 0^{\mathsf{len}(\kappa)}$ for $i \in [\ell+1]$ and $j \in [n]$. Here $\mathsf{len}(\cdot)$ is an appropriate length function that would be specified later. Let $\Pi_i = \{\Pi_i^j\}_{j\in[n]}$.

- **Functional encryption ciphertext and keys to simulate obfuscation of Public Key**:

  1. For each $i \in [\ell]$ generate $\mathsf{FSK}_i \leftarrow \mathsf{FE.KeyGen}(MSK_i, F_{i,PK_{i+1},\Pi_i})$ and $\mathsf{FSK}_{\ell+1} \leftarrow \mathsf{FE.KeyGen}(MSK_{\ell+1}, G_{\Pi_{\ell+1}})$, where F_{i,PK_{i+1},Π_i} and $G_{\Pi_{\ell+1}}$ are circuits described in Figure 13.
  2. Let $Z := \{Z_i\}_{i\in[n]}$ where for every $i \in [n]$, $Z_i = (0^\ell, 0^\kappa)$
  3. Let $c_\phi = \mathsf{FE.Enc}_{PK_1}(\phi, S, K_\phi, Z, 0)$.

- **Sampler**: The sampler takes as input a description of a circuit C and outputs $C(S_C)$. It proceeds as follows:

  1. For $i \in [\ell]$ compute $c_{C_{[i-1]}\|0}, c_{C_{[i-1]}\|1} := \mathsf{FE.Dec}(\mathsf{FSK}_i, c_{C_{[i-1]}})$.
  2. Obtain d_C as output of $\mathsf{FE.Dec}(\mathsf{FSK}_{\ell+1}, c_C)$.
  3. Output d_C.

Figure 12: Setup

6 Construction from FE

In this section, we will construct Universal Samplers that satisfies Definition 24 from polynomially hard Functional Encryption and Prefix Puncturable pseudorandom functions.

Theorem 25 *Assuming the existence of one-way functions and selective secure single key public key functional encryption there exists an Universal Sampler scheme satisfying Definition 24.*

Our Construction. The formal description our construction appears in Figure 12.

Setting $\mathsf{rand}(\cdot)$. We set $\mathsf{rand}(\kappa) = 2\kappa$ where κ is the security parameter.

Security We will now show a construction of a simulator that satisfies Definition 24.

$F_{i,PK_{i+1},\Pi}$

Hardcoded Values: i, PK_{i+1}, Π_i.
Input: ($C \in \{0,1\}^{i-1}$, S_C, K_C, Z, mode)

1. If ($\mathsf{mode} = 0$) then output $\mathsf{FE.Enc}_{PK_{i+1}}(C\|0, S_{x\|0}, K_{C\|0}, sk, \mathsf{mode}; {K'}_{C\|0})$ and $\mathsf{FE.Enc}_{PK_{i+1}}(C\|1, S_{C\|1}, K_{C\|1}, sk, \mathsf{mode}; {K'}_{C\|1})$, where for $b \in \{0,1\}$, $K_{C\|b} = \mathsf{PrefixPunc}(K_C, b\|0)$ and $K'_{C\|b} = \mathsf{PrefixPunc}(K_C, b\|1)$ and $S_{C\|b} := \mathsf{PrefixPunc}(S_C, b)$.
2. Else recover (lexicographically the first Z_j) such that $Z_j = (C_j, sk_j)$ where $C_{j_{[i-1]}} = C$. Compute $c_{C\|0}, c_{C\|1} \leftarrow \mathsf{SK.Dec}_{sk_j}(\Pi_i^j)$ and output $(c_{C\|0}, c_{C\|1})$.

$G_{\Pi_{\ell+1}}$

Hardcoded Values: $\Pi_{\ell+1}$
Input: $C \in \{0,1\}^\ell, S_C, K_C, sk, \mathsf{mode}$

1. If $\mathsf{mode} = 0$, output $C(S_C)$.
2. Else recover (lexicographically the first Z_j) such that $Z_j = (C_j, sk_j)$ where $C_j = C$. Compute $d_C \leftarrow \mathsf{SK.Dec}_{sk_j}(\Pi_i^j)$ and output d_C.

Figure 13: Circuits for simulating Public Key.

The correctness of the simulator follows from our description of F_{i,PK_{i+1},Π_i} and $G_{\Pi_{\ell+1}}$ as well as π_i^j for $j \leq q$ and $i \in [\ell+1]$. In particular, setting π_i^j as described in Figure 14 ensures that for every (C_i, s_i) that the adversary queries for the **Set** query the sampler outputs s_i.

Indistinguishability from honest generation is easy to observe.

We now show the pseudorandomness of samples. We show this through a standard hybrid argument.

- Hyb_0: This is the $b = 0$ case in the pseudorandomness game. The **Challenge** query is answered by making a **Read** query to Sim, which responds with $s^* = C^*(S^1_{C^*})$. The simulator adds C^* to the list L_C and adds s^* to the list L_s.
- Hyb_1 : In this hybrid, we are going to "puncture" the sampler circuit along the path C^*. We are going to accomplish this through the following sub-hybrids. Let us assume that the adversary has made $q \leq n-1$ **Set** queries
  - $\mathsf{Hyb}_{0,0}$: In this hybrid, we are going to change how π_i^{q+1} is generated for $i \in [\ell+1]$. For every prefix, x of $C^\star$ of length at most $\ell - 1$, let e_x denote the output of Step 1 of the sampler. For every string x that is a prefix of C^*, let y denote the string which is

- **Sampled Ingredients**:

  1. Sample S^1 and K^1_ϕ from $\mathsf{KeyGen}_{\mathcal{PPRF}}(1^\kappa)$. Here S^1 is a key that works for ℓ bit inputs and produces a r bit outputs, namely $\mathsf{PPRF}_{S^1} : \{0,1\}^\ell \to \{0,1\}^r$ where ℓ and r are polynomial functions of κ. Similarly, K^1_ϕ work on inputs of length $\mathsf{rand}(\kappa)$ where $\mathsf{rand}(\cdot)$.
  2. Sample $(PK^1_i, MSK^1_i) \leftarrow \mathsf{FE.Setup}(1^\kappa)$ for all $i \in [\ell+1]$.
  3. Sample $sk_1, \cdots, sk_n \leftarrow \mathsf{SK.KeyGen}(1^\kappa)$.

- **Read Queries:** For every **Read** query on C that the adversary makes, simulator answers with $C(S^1_C)$.

- **Set Queries:** For every **Set** query (C_i, s_i), the simulator adds C_i to a list L_C and adds s_i to a list L_s.

- **Setting π^j_i values**. Let q denote the number of set queries made by the adversary. By definition, $q \leq n$. Let C_i be the i^{th} set query made by the adversary. We set $Z_i = (C_i, sk_i)$ for all $i \in [q]$ and set $Z_i = (0^\ell, 0^\kappa)$ for all $q+1 \leq i \leq n$ and denote $Z' = \{Z_i\}_{i\in[n]}$. For every prefix x (including the empty prefix) of length at most $\ell - 1$ of C_i, let $c_x \leftarrow \mathsf{FE.Enc}_{PK_{|x|+1}}(x, 0^\kappa, 0^\kappa, Z', 1; r_x)$ where r_x denotes uniformly chosen random string. For every string x that is a prefix of C_i, let y denote the string which is same as x except that the last bit of x is flipped. For every $y \in \{0,1\}^{\leq \ell-1}$, e_y denotes the output of Step 1 of the sampler. We set $\pi^i_{|x|} = (x, c_x), (y, e_y)$ and $\Pi^i_{|x|} = \mathsf{SK.Enc}_{sk_i}((x, c_x), (y, e_y))$. We set $\pi^i_{\ell+1} = (C_i, s_i)$. For other unset values, we let $\pi^j_i = 0^{\mathsf{len}(\kappa)}$.

- **Functional encryption ciphertext and keys to simulate obfuscation of Public Key**:

  1. For each $i \in [\ell]$ generate $\mathsf{FSK}_i \leftarrow \mathsf{FE.KeyGen}(MSK_i, F_{i,PK_{i+1},\Pi_i})$ and $\mathsf{FSK}_{\ell+1} \leftarrow \mathsf{FE.KeyGen}(MSK_{\ell+1}, G_{\Pi_{\ell+1}})$, where F_{i,PK_{i+1},Π^i} and $G_{\Pi_{\ell+1}}$ are circuits described in Figure 13.
  2. Let $c_\phi = \mathsf{FE.Enc}_{PK_1}(\phi, 0^\kappa, 0^\kappa, Z', 1)$.

- **Sampler**: The sampler takes as input a description of a circuit C and outputs $C(S_C)$. It works exactly as described in Figure 12.

Figure 14: Description of the simulator

same as x except that the last bit of x is flipped. Similarly, e_y denotes the output of Step 1 of the sampler on input y. We set $\pi^{q+1}_{|x|} = (x, e_x), (y, e_y)$ and $\pi^{q+1}_{(\ell+1)} = (C^*, s^*)$ [14]. Computational indistinguishability of Hyb_0 from $\mathsf{Hyb}_{0,0}$ follows from the semantic security of symmetric key encryption.

- $\mathsf{Hyb}_{0,1}$: In this hybrid, we change the encryptions output along the path C^*. We denote $x < x'$ if $|x| < |x'|$. Let $Z_{q+1} = (C^*, sk_{q+1})$. Let Z'' be equal to Z' except that it has Z_{q+1} in the $q + 1^{th}$ location. For every prefix of x (including the empty prefix) of C^*, let $\mathsf{Hyb}_{0,x}$ denote that for all $x' < x$, $c_{x'} = \mathsf{FE.Enc}_{PK_{|x'|+1}}(x', 0^\kappa, 0^\kappa, Z'', 1; r_{x'})$ where $r_{x'}$ is chosen uniformly at random. Note that in the previous hybrid, $c_{x'} = \mathsf{FE.Enc}_{PK_{|x'|+1}}(x', S^1_{x'}, K^1_{x'}, Z', 0; {K'}^1_{x'})$ [15]. We now show that $\mathsf{Hyb}_{0,x}$ is computationally indistinguishable from $\mathsf{Hyb}_{0,x'}$ where $x < x'$ and they are adjacent according to the ordering defined.
  - $\mathsf{Hyb}_{0,x,1}$: In this hybrid, we are going to change ${K'}^1_{x'}$ to uniformly chosen random value $r_{x'}$ at $\pi^j_{|x'|}$ for all $j \in [q+1]$ such that $\pi^j_{|x'|}$ contains x'. This change is possible from the pseudorandomness at punctured point property of PPRF. In particular, K^1_z for all $z < x'$ has already been set to 0^κ.
  - $\mathsf{Hyb}_{0,x,2}$: In this hybrid, we are going to change $c_{x'} = \mathsf{FE.Enc}_{PK_{|x'|+1}}(x', 0^\kappa, 0^\kappa, Z'', 1; r_{x'})$ at $\pi^j_{|x'|}$ for all $j \in [q+1]$ such that $\pi^j_{|x'|}$ contains x'. This change is possible from the selective security of functional encryption. Observe that $\mathsf{FSK}_{|x'|+1}$ decrypts both the ciphertexts to the same value. Note that $\mathsf{Hyb}_{0,x,2}$ is identical to $\mathsf{Hyb}_{0,x'}$
- Hyb_2 : In this hybrid, we are going to change $\pi^{q+1}_{\ell+1}$ to $C^*, C^*(r^*)$ where r^* is chosen uniformly at random. This change is possible from the pseudorandomness at prefix punctured property of PPRF as S_x for all prefixes x of C^* has already been set to 0^κ. Observe that Hyb_2 is identically distributed to the case where the challenge query is a **Set** query of an uniform sample.

Setting the parameters. We set $\mathsf{len}(\kappa)$ to be the maximum size of π^j_i for all $i \in [\ell + 1]$ and for all $j \in [n]$ used in the construction of the simulator and in security proof.

7 Multiparty Non-interactive Key Exchange

In this section, we build multiparty non-interactive key exchange for an unbounded number of users. Moreover, in constrast to the original multilinear map protocols [GGH13a], our protocol has no trusted setup.

7.1 Definition

A multiparty key exchange protocol consists of:

- $\mathsf{Publish}(\kappa)$ takes as input the security parameter and outputs a user secret sv and public value pv. pv is posted to the bulletin board.

[14] We pad $\pi^{q+1}_{|x|}$ and $\pi^{q+1}_{(\ell+1)}$ with some dummy symbols until their length is $\mathsf{len}(\kappa)$

[15] It might be the case that $c_{x'}$ is already of the required form. This happens when C^* shares the same prefix x' with another **Set** query. In that case, we don't make any changes.

- $\mathsf{KeyGen}(\{\mathsf{pv}_j\}_{j\in S}, \mathsf{sv}_i, i)$ takes as input the public values of a set S of users, plus one of the user's secrets sv_i. It outputs a group key $k \in \mathcal{K}$.

For correctness, we require that all users generate the same key:

$$\mathsf{KeyGen}(\{\mathsf{pv}_j\}_{j\in S}, \mathsf{sv}_i, i) = \mathsf{KeyGen}(\{\mathsf{pv}_j\}_{j\in S}, \mathsf{sv}_{i'}, i')$$

for all $(\mathsf{sv}_j, \mathsf{pv}_j) \leftarrow \mathsf{Publish}(\kappa)$ and $i, i' \in S$. For security, we have the following:

Definition 26 *A non-interactive multiparty key exchange protocol is statically secure if the following distributions are indistinguishable for any polynomial-sized set S:*

$$\{\mathsf{pv}_j\}_{j\in S}, k \text{ where } (\mathsf{sv}_j, \mathsf{pv}_j) \leftarrow \mathsf{Publish}(\kappa) \forall j \in S, k \leftarrow \mathsf{KeyGen}(\{\mathsf{pv}_j\}_{j\in S}, s_1, 1) \text{ and}$$
$$\{\mathsf{pv}_j\}_{j\in S}, k \text{ where } (\mathsf{sv}_j, \mathsf{pv}_j) \leftarrow \mathsf{Publish}(\kappa) \forall j \in G, k \leftarrow \mathcal{K}$$

Notice that our syntax does not allow a trusted setup, as the original constructions based on multilinear maps [BS02, GGH13a, CLT13] require. Boneh and Zhandry [BZ14] give the first multiparty key exchange protocol without trusted setup, based on obfuscation. A construction of obfuscation from a finite set of assumptions with polynomial security appears implausible due to an argument of [GGSW13]. Notice as well that our syntax does not allow the key generation to depend on the number of users who wish to share a group key. To date, prior key exchange protocols satisfying this property relied on strong knowledge variants of obfuscation [ABG+13]. Recently Khurana, Rao and Sahai in [KRS15] constructed a key exchange protocol supporting unbounded number of users based on indistinguishability obfuscation and a tool called as *somewhere statistically binding hash functions* [HW15]. Here, we get an unbounded protocol based on *functiona encryption* only, and which does not require complexity leveraging.

7.2 Construction

Our construction will use the universal samplers built in Section 5, as well as any public key encryption scheme.

- $\mathsf{Publish}(\kappa)$. Run $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{PK.KeyGen}(\kappa)$. Also run the universal sampler setup algorithm $\mathsf{Sampler} \leftarrow \mathsf{Setup}(\kappa, \ell, t)$ where output size ℓ and circuit size bound t will be decided later. Output $\mathsf{pv} = (\mathsf{pk}, \mathsf{Sampler})$ as the public value and keep $\mathsf{sv} = \mathsf{sk}$ as the secret value.

- $\mathsf{KeyGen}(\{(\mathsf{pk}_j, \mathsf{Sampler}_j)\}_{j\in S}, \mathsf{sk}_i, i)$. Interpret S as the set $[1, n]$ for $n = |S|$, choosing some canonical ordering for the users in S (say, the lexicographic order of their public values). Define $\mathsf{Sampler} = \mathsf{Sampler}_1$.

  Define $C_{\mathsf{pk},\mathsf{pk}'}$ for two public keys $\mathsf{pk}, \mathsf{pk}'$ to be the circuit that samples a random $(\mathsf{sk}'', \mathsf{pk}'') \leftarrow \mathsf{PK.KeyGen}(\kappa)$, then encrypts sk'' under both pk and pk', obtaining encryptions c and c' respectively, and then outputs (pk'', c, c').

  Let $D_{\mathsf{pk},\mathsf{pk}'}$ be a similar circuit that samples a uniformly random string sk'' in the key space of $\mathcal{PKE}$, encrypts sk'' to get c, c' as before, and outputs $(0, c, c')$ where 0 is a string of zeros with the same length as a public key for $\mathcal{PKE}$. Let ℓ the the length of (pk'', c, c') and let t be the size of $C_{\mathsf{pk},\mathsf{pk}'}$ (which we will assume is at least as large as $D_{\mathsf{pk},\mathsf{pk}'}$).

Next, define $\mathsf{pk}'_2 = \mathsf{pk}_1$, and recursively define $(\mathsf{pk}'_{j+1}, c_j, c'_j) = \mathsf{Sampler}(C_{\mathsf{pk}_j, \mathsf{pk}'_j})$ for $j = 2, \ldots, n-1$. Define sk'_{j+1} to be the secret key corresponding to pk'_{j+1}, which is also the secret key encrypted in c_j, c'_j. Finally, define $(0, c_n, c'_n) = \mathsf{Sampler}(D_{\mathsf{pk}_n, \mathsf{pk}'_n})$, and define sk'_{n+1} to be the secret key encrypted in c_n, c'_n.

First, it is straightforward that given $\{\mathsf{pk}_j\}_{j\in[n]}$ and $\mathsf{Sampler}$, it is possible to compute $\mathsf{pk}'_j, c_j, c'_j$ for all $k \in [2, n]$. Thus anyone, including an eavesdropper, can compute these values.

Next, we claim that if additionally given secret keys sk_j *or* sk'_j, it is possible to compute sk'_{j+1}. Indeed, sk'_{j+1} can be computed by decrypting c_j (using sk_j) or decrypting c'_j (using sk'_j). By iterating, it is possible to compute sk'_k for every $k > j$. This in particular implies that all users in $[n]$ can compute sk_{n+1}.

Security. We now argue that any eavesdropper cannot learn any information about sk. Our theorem is the following:

Theorem 27 *If $\mathcal{PKE}$ is a secure public key encryption scheme and* Setup *is a m-time statically secure universal sampler with interactive simulation, the the construction above is a statically secure NIKE for up to 2^m users. In particular, by setting $m = \kappa$, the scheme is secure for an* unbounded *number of users.*

We prove this theorem by introducing a collection of hybrids. For a subset $T \subseteq [3, n+1]$ of size at most m, define the hybrid $\mathbf{Hybrid}_T$ as follows. $(\mathsf{sk}_j, \mathsf{pk}_j)$ for $j \in [n]$ are generated randomly from $\mathsf{PK.KeyGen}$. Similarly, $(\mathsf{sk}'_i, \mathsf{pk}'_i)$ for $i \in T$ are generated randomly from $\mathsf{PK.KeyGen}$. For each $j > 1$, a random $\mathsf{Sampler}_j$ is generated from Setup. Define $(\mathsf{sk}'_2, \mathsf{pk}'_2) = (\mathsf{sk}_1, \mathsf{pk}_1)$. Finally, $\mathsf{Sampler} = \mathsf{Sampler}_1$ is simulated using Sim as follows.

For $j = 3, \ldots, n$, do the following:

- If $j \notin T, j \leq n$, make a **Read** query on $C_{\mathsf{pk}_{j-1}, \mathsf{pk}'_{j-1}}$, obtaining $\mathsf{pk}'_j, c_{j-1}, c'_{j-1}$.
- If $j = n+1 \notin T$, make a **Read** query on $D_{\mathsf{pk}_n, \mathsf{pk}'_n}$, obtaining c_n, c'_n.
- If $j \in T, j \leq n$, let $c_{j-1} = \mathsf{PK.Enc}(\mathsf{pk}_{j-1}, 0)$ and $c'_{j-1} = \mathsf{PK.Enc}(\mathsf{pk}'_{j-1}, 0)$. Then make a **Set** query on $C_{\mathsf{pk}_{j-1}, \mathsf{pk}'_{j-1}}, (\mathsf{pk}'_j, c_{j-1}, c'_{j-1})$.
- If $j = n+1 \in T$, let $c_n = \mathsf{PK.Enc}(\mathsf{pk}_n, 0)$ and $c'_n = \mathsf{PK.Enc}(\mathsf{pk}'_{j-1}, 0)$. Then make a **Set** query on $D_{\mathsf{pk}_n, \mathsf{pk}'_n}, (0, c_n, c'_n)$.

Then make a **Finish** query, and output the resulting $\mathsf{Sampler}$ as $\mathsf{Sampler}_1$. In short, we simulate $\mathsf{Sampler}$ so that the ciphertexts c_{j-1}, c'_{j-1} for all $j \in T$ encrypt 0 instead of the secret key sk'_j.

First, we observe that if $T = \emptyset$, then there are no **Set** queries at all, and thus $\mathsf{Sampler}$ is indistinguishable from a correctly generated sampler. Next, we note that if $n+1 \in T$, then sk'_{n+1} is information-theoretically independent of the adversary's view. Thus, in this case, security holds. Our goal then is to move from $T = \emptyset$ to some T that contains $n+1$. We first make the following claim:

Claim 28 *Let $T \subset [3, n+1]$ be a set of size at most m, let $i^* \in T$ such that either $i^* - 1 \in T$ or $i^* = 3$, and let $T' = T \setminus \{i^*\}$. Then $\mathbf{Hybrid}_{T'}$ and $\mathbf{Hybrid}_T$ are indistinguishable.*

Proof First, we describe an intermediate hybrid which is identical to $\mathbf{Hybrid}_{T,i^*}$, except that in the **Set** query on $j = i^*$, we now generate $c_{i^*-1} = \mathsf{PK.Enc}(\mathsf{pk}_{i^*-1}, \mathsf{sk}_{i^*})$ and $c'_{i^*-1} = \mathsf{PK.Enc}(\mathsf{pk}'_{i^*-1}, \mathsf{sk}_{i^*})$. Notice that simulating $\mathbf{Hybrid}_T$ and $\mathbf{Hybrid}_{T,i^*}$ do not rely on the knowledge of sk_{i^*-1} or sk'_{i^*-1}. Therefore, the ciphertexts c_{i^*-1}, c'_{i^*-1} are secure. Thus $\mathbf{Hybrid}_T$ and $\mathbf{Hybrid}_{T,i^*}$ are indistinguishable by the security of $\mathcal{PKE}$.

Now we show that $\mathbf{Hybrid}_{T,i^*}$ and $\mathbf{Hybrid}_{T'}$ are indistinguishable. Notice that, for $i^* \leq n$ in $\mathbf{Hybrid}_{T,i^*}$, $(\mathsf{pk}_{i^*}, c_{i^*-1}, c'_{i^*-1})$ is a fresh sample from $C_{\mathsf{pk}_{i^*-1},\mathsf{pk}'_{i^*-1}}$. The analagous statement holds for $i^* = n+1$. Thus the only difference between the two hybrids is that this sample is pseudorandom in $\mathbf{Hybrid}_{T'}$, and freshly random in $\mathbf{Hybrid}_{T,i^*}$. Moreover, we know the sampler circuit $C_{\mathsf{pk}_{i^*-1},\mathsf{pk}'_{i^*-1}}$ before initiating the simulator. Thus, by the pseudorandomness of samples property of the simulator, these two hybrids are actually indistinguishable. ■

Now it remains to show that there is a sequence of hybrids for sets $T_0, \ldots, T_t$ such that $T_0 = \emptyset$, $n+1 \in T_t$, $|T_r| \leq m$ for all $r \in [0,t]$, and finally T_r and T_{r+1} only differ on a single point j_r, and $j-1 \in (T_r \cap T_{r+1}) \cup \{2\}$. We also require that t is polynomial in n. Once this algorithmic problem is solved, we have a complete security proof. In the following, we abstract out this algorithmic problem, and show how to solve it.

7.3 An Algorithmic Problem

We now describe the pebbling strategy of Bennet in [Ben89].

Consider the positive integer line $1, 2, \ldots$. Suppose we are given k pebbles. At the start, all k pebbles are in our hand. We make a sequence of moves where we place a pebble on the line or remove a pebble back into our hand, subject to the following restrictions:

- The total number of pebbles on the line can never exceed k.
- In any move, we can only place or remove a pebble at integer $i > 1$ if there is currently a pebble at integer $i-1$. This restriction does not apply to $i = 1$: we can always place or remove a pebble at position 1, as long as we do not exceed k pebbles.

Our goal is to place a pebble at the highest possible integer, and get there using as few moves as possible.

Theorem 29 *For any integer $n < 2^k$, it is possible to make $O(n^{\log_2 3}) \approx O(n^{1.585})$ moves and get a pebble at position n. For any $n \geq 2^k$, it is impossible to get a pebble at position n.*

Proof First we observe to get a pebble placed at n, for each $i \in [1, n-1]$ there must have been at some point a pebble placed at location i.

Next, we observe that it suffices to show we can get a pebble at position $n = 2^k - 1$ for every k using $O(3^k) = O(n^{\log_2 3})$ steps. Indeed, for more general n, we run the protocol for $n' = 2^k - 1$ where $k = \lceil \log_2(n-1) \rceil$, but stop the first time we get a pebble at position n. Since $n'/n \leq 3$, the running time is at most $O(n^{\log_2 3})$.

Now for the algorithm. The sequence of steps will create a fractal pattern, and we describe the steps recursively.

We assume an algorithm A_{k-1} using $k-1$ pebbled that can get a pebble at position $2^{k-1} - 1$. The steps are as follows:

- Run A_{k-1}. There is now a pebble at position $2^{k-1}-1$ on the line.
- Place the remaining pebble at position 2^{k-1}, which is allowed since there is a pebble at position $2^{k-1}-1$.
- Run A_{k-1} in reverse, recovering all of the $k-1$ pebbles used by A. The result is that there is a single pebble on the line at position 2^{k-1}.
- Now associate the portion of the number line starting at $2^{k-1}+1$ with a new number line. That is, associate $2^{k-1}+a$ on the original number line with a on the new number line. To distinguish the old from the new number line, we will denote position a on the new number line as $\hat{a}$, so that $2^{k-1}+a=\hat{a}$. We now have $k-1$ pebbles, and on this new number line, all of the same rules apply. In particular, we can always add or remove a pebble from the first position $\hat{1}=2^{k-1}+1$ since we have left a pebble at 2^{k-1}. Therefore, we can run A_{k+1} once more on the new number line starting at $\hat{1}$. The end result is a pebble at position $\widehat{2^{k-1}-1}=2^{k-1}+(2^{k-1}-1)=2^k-1$.

It remains to analyze the running time. The algorithm makes 3 recursive calls to A_{k-1}, so by induction the overall running time is $O(3^k)$, as desired.

We now explain why the n obtained is optimal. It suffices to show that it is not possible to get a pebble at position 2^k. We do not know if the running time obtained by our algorithm is optimal, though we believe it asyptotically optimal for $n=2^{k-1}$.

We make the following stronger claim, which in particular shows that $n=2^{k-1}$ is impossible. In any configuration reachable starting from an empty number line given k pebbles, the jth pebble must be no higher than position $2^{k-j}(2^j-1)$. In particular the kth pebble must be at position $2^0(2^k-1)=2^k-1$ or lower.

Suppose for some k,j, it was possible to have the jth pebble at position $2^{k-j}(2^j-1)+r$ for some $r>0$. Clearly, for $k=0$, this is impossible (since there can never be a pebble anywhere). Therefore, there is a minimal k for which this is possible, and let j be the smallest j for this k that contradicts the claim. By the minimality of j, as long as there are *any* pebbles at positions greater than $2^{k-(j-1)}(2^{j-1}-1)=2^{k-j}(2^j-1)-2^{k-j}$, there must be $j-1$ pebbles at or below this position. In particular, if there is a pebble at position r, there can never be more than $k-j$ pebbles in the interval $I=[2^{k-j}(2^j-1)-2^{k-j}+1, 2^{k-j}(2^j-1)+r-1]$. Let A be the algorithm that gets the jth pebble to position $2^{k-j}(2^j-1)+r$. We now claim that we can derive from A an algorithm B that uses $k-j<k$ pebbles and gets a pebble at position $2^{k-j}+r-1\geq 2^{k-j}$, which violates the minimality of k.

We now describe B. B simulates A in reverse, with the following modification. First, A is simulated on the shifted number line starting at position $-(2^{k-j}(2^j-1)-2^{k-j})$. B will only place pebbles in the interval $[1, 2^{k-j}+r-1]$, which corresponds to the interval I from A's perspective. For all other pebbles used by A, B will place a "virtual" at that location. Second, B will stop the first time A removes the (virtual) pebble at position $2^{k-j}(2^j-1)+r$, which corresponds to B's position $2^{j-k}+r$. Since A is removing a pebble at this location, there must be a pebble at position $2^{k-j}+r-1$, which will be a real pebble. Thus B gets a real pebble to $2^{k-j}+r-1$. The entire reverse excecution of A has a pebble at position $2^{k-j}(2^j-1)+r$ (from A's perspective), so by the above observation there are at most $k-j$ pebbles in the interval I. Thus B only ever uses $k-j$ pebbles. Lastly, B follows all the rules of the game since A does. Thus B uses $k-j<k$ pebbles to get a pebble at position at or higher than 2^{k-j}, which violates the minimality of k. ■

References

[ABG+13] Prabhanjan Ananth, Dan Boneh, Sanjam Garg, Amit Sahai, and Mark Zhandry. Differing-inputs obfuscation and applications. Cryptology ePrint Archive, Report 2013/689, 2013. http://eprint.iacr.org/2013/689.

[ABSV15] Prabhanjan Ananth, Zvika Brakerski, Gil Segev, and Vinod Vaikuntanathan. From selective to adaptive security in functional encryption. In *Advances in Cryptology - CRYPTO 2015 - 35th Annual Cryptology Conference, Santa Barbara, CA, USA, August 16-20, 2015, Proceedings, Part II*, pages 657–677, 2015.

[AJ15] Prabhanjan Ananth and Abhishek Jain. Indistinguishability obfuscation from compact functional encryption. In *CRYPTO*, pages 308–326, 2015.

[AJS15] Prabhanjan Ananth, Abhishek Jain, and Amit Sahai. Achieving compactness generically: Indistinguishability obfuscation from non-compact functional encryption. *IACR Cryptology ePrint Archive*, 2015:730, 2015.

[AS16] Prabhanjan Vijendra Ananth and Amit Sahai. Functional encryption for turing machines. In *Theory of Cryptography - 13th International Conference, TCC 2016-A, Tel Aviv, Israel, January 10-13, 2016, Proceedings, Part I*, pages 125–153, 2016.

[Ben89] Charles H. Bennett. Time/space trade-offs for reversible computation. *SIAM J. Comput.*, 18(4):766–776, 1989.

[BGI+12] Boaz Barak, Oded Goldreich, Russell Impagliazzo, Steven Rudich, Amit Sahai, Salil P. Vadhan, and Ke Yang. On the (im)possibility of obfuscating programs. *J. ACM*, 59(2):6, 2012.

[BGI14] Elette Boyle, Shafi Goldwasser, and Ioana Ivan. 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*, pages 501–519, 2014.

[BLR+15] Dan Boneh, Kevin Lewi, Mariana Raykova, Amit Sahai, Mark Zhandry, and Joe Zimmerman. Semantically secure order-revealing encryption: Multi-input functional encryption without obfuscation. In Elisabeth Oswald and Marc Fischlin, editors, *EUROCRYPT 2015, Part II*, volume 9057 of *LNCS*, pages 563–594, Sofia, Bulgaria, April 26–30, 2015. Springer, Heidelberg, Germany.

[BPR15] Nir Bitansky, Omer Paneth, and Alon Rosen. On the cryptographic hardness of finding a nash equilibrium. In *FOCS*, 2015.

[BPW16] Nir Bitansky, Omer Paneth, and Daniel Wichs. Perfect structure on the edge of chaos. *TCC*, 2016.

[BS02] Dan Boneh and Alice Silverberg. Applications of multilinear forms to cryptography. Cryptology ePrint Archive, Report 2002/080, 2002. http://eprint.iacr.org/2002/080.

[BSW11] Dan Boneh, Amit Sahai, and Brent Waters. Functional encryption: Definitions and challenges. In *Theory of Cryptography - 8th Theory of Cryptography Conference, TCC 2011, Providence, RI, USA, March 28-30, 2011. Proceedings*, pages 253–273, 2011.

[BV15] Nir Bitansky and Vinod Vaikuntanathan. Indistinguishability obfuscation from functional encryption. In *FOCS*, 2015.

[BW13] Dan Boneh and Brent Waters. Constrained pseudorandom functions and their applications. In *Advances in Cryptology - ASIACRYPT 2013 - 19th International Conference on the Theory and Application of Cryptology and Information Security, Bengaluru, India, December 1-5, 2013, Proceedings, Part II*, pages 280–300, 2013.

[BZ14] Dan Boneh and Mark Zhandry. Multiparty key exchange, efficient traitor tracing, and more from indistinguishability obfuscation. In Juan A. Garay and Rosario Gennaro, editors, *CRYPTO 2014, Part I*, volume 8616 of *LNCS*, pages 480–499, Santa Barbara, CA, USA, August 17–21, 2014. Springer, Heidelberg, Germany.

[CLP15] Kai-Min Chung, Huijia Lin, and Rafael Pass. Constant-round concurrent zero-knowledge from indistinguishability obfuscation. In Rosario Gennaro and Matthew J. B. Robshaw, editors, *CRYPTO 2015, Part I*, volume 9215 of *LNCS*, pages 287–307, Santa Barbara, CA, USA, August 16–20, 2015. Springer, Heidelberg, Germany.

[CLT13] Jean-Sébastien Coron, Tancrède Lepoint, and Mehdi Tibouchi. Practical multilinear maps over the integers. In Ran Canetti and Juan A. Garay, editors, *CRYPTO 2013, Part I*, volume 8042 of *LNCS*, pages 476–493, Santa Barbara, CA, USA, August 18–22, 2013. Springer, Heidelberg, Germany.

[EGL85] Shimon Even, Oded Goldreich, and Abraham Lempel. A randomized protocol for signing contracts. *Commun. ACM*, 28(6):637–647, 1985.

[Gam85] Taher El Gamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Transactions on Information Theory*, 31(4):469–472, 1985.

[GGH13a] Sanjam Garg, Craig Gentry, and Shai Halevi. Candidate multilinear maps from ideal lattices. In Thomas Johansson and Phong Q. Nguyen, editors, *EUROCRYPT 2013*, volume 7881 of *LNCS*, pages 1–17, Athens, Greece, May 26–30, 2013. Springer, Heidelberg, Germany.

[GGH+13b] Sanjam Garg, Craig Gentry, Shai Halevi, Mariana Raykova, Amit Sahai, and Brent Waters. Candidate indistinguishability obfuscation and functional encryption for all circuits. In *54th FOCS*, pages 40–49, Berkeley, CA, USA, October 26–29, 2013. IEEE Computer Society Press.

[GGHZ16] Sanjam Garg, Craig Gentry, Shai Halevi, and Mark Zhandry. Fully secure functional encryption from multilinear maps. In *TCC*, 2016.

[GGM86] Oded Goldreich, Shafi Goldwasser, and Silvio Micali. How to construct random functions. *J. ACM*, 33(4):792–807, 1986.

[GGSW13] Sanjam Garg, Craig Gentry, Amit Sahai, and Brent Waters. Witness encryption and its applications. In Dan Boneh, Tim Roughgarden, and Joan Feigenbaum, editors, *45th ACM STOC*, pages 467–476, Palo Alto, CA, USA, June 1–4, 2013. ACM Press.

[GL89] Oded Goldreich and Leonid A. Levin. A hard-core predicate for all one-way functions. In *Proceedings of the 21st Annual ACM Symposium on Theory of Computing, May 14-17, 1989, Seattle, Washigton, USA*, pages 25–32, 1989.

[GPS15] Sanjam Garg, Omkant Pandey, and Akshayaram Srinivasan. On the exact cryptographic hardness of finding a nash equilibrium. *IACR Cryptology ePrint Archive*, 2015:1078, 2015.

[GR13] Oded Goldreich and Ron D. Rothblum. Enhancements of trapdoor permutations. *J. Cryptology*, 26(3):484–512, 2013.

[HJK+14] Dennis Hofheinz, Tibor Jager, Dakshita Khurana, Amit Sahai, Brent Waters, and Mark Zhandry. How to generate and use universal samplers. Cryptology ePrint Archive, Report 2014/507, 2014. `http://eprint.iacr.org/2014/507`.

[HSW14] Susan Hohenberger, Amit Sahai, and Brent Waters. Replacing a random oracle: Full domain hash from indistinguishability obfuscation. In Phong Q. Nguyen and Elisabeth Oswald, editors, *EUROCRYPT 2014*, volume 8441 of *LNCS*, pages 201–220, Copenhagen, Denmark, May 11–15, 2014. Springer, Heidelberg, Germany.

[HW15] Pavel Hubacek and Daniel Wichs. On the communication complexity of secure function evaluation with long output. In Tim Roughgarden, editor, *ITCS 2015*, pages 163–172, Rehovot, Israel, January 11–13, 2015. ACM.

[KPTZ13] Aggelos Kiayias, Stavros Papadopoulos, Nikos Triandopoulos, and Thomas Zacharias. Delegatable pseudorandom functions and applications. In *2013 ACM SIGSAC Conference on Computer and Communications Security, CCS'13, Berlin, Germany, November 4-8, 2013*, pages 669–684, 2013.

[KRS15] Dakshita Khurana, Vanishree Rao, and Amit Sahai. Multi-party key exchange for unbounded parties from indistinguishability obfuscation. In *Advances in Cryptology - ASIACRYPT 2015 - 21st International Conference on the Theory and Application of Cryptology and Information Security, Auckland, New Zealand, November 29 - December 3, 2015, Proceedings, Part I*, pages 52–75, 2015.

[O'N10] Adam O'Neill. Definitional issues in functional encryption. *IACR Cryptology ePrint Archive*, 2010:556, 2010.

[PPS15] Omkant Pandey, Manoj Prabhakaran, and Amit Sahai. Obfuscation-based non-black-box simulation and four message concurrent zero knowledge for NP. In Yevgeniy Dodis and Jesper Buus Nielsen, editors, *TCC 2015, Part II*, volume 9015 of *LNCS*, pages 638–667, Warsaw, Poland, March 23–25, 2015. Springer, Heidelberg, Germany.

[Reg09] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. *J. ACM*, 56(6), 2009.

[SW14] Amit Sahai and Brent Waters. How to use indistinguishability obfuscation: deniable encryption, and more. In *Symposium on Theory of Computing, STOC 2014, New York, NY, USA, May 31 - June 03, 2014*, pages 475–484, 2014.