Breaking the Sub-Exponential Barrier in Obfustopia

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

February 6, 2017

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. A major 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 for 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 many current techniques.

A parallel research challenge is building obfuscation from simpler assumptions. Unfortunately, it appears that such a construction would likely incur an exponential loss in the security reduction. Thus, achieving any application of $i\mathcal{O}$ from simpler assumptions would also require a sub-exponential loss, *even if the $i\mathcal{O}$-to-application security proof incurred a polynomial loss.* Functional encryption ($\mathcal{FE}$) is known to be equivalent to $i\mathcal{O}$ up to a sub-exponential loss in the $\mathcal{FE}$-to-$i\mathcal{O}$ security reduction; yet, unlike $i\mathcal{O}$, $\mathcal{FE}$ *can* be achieved from simpler assumptions (namely, specific multilinear map assumptions) with only a polynomial loss.

In the interest of basing applications on weaker assumptions, we therefore argue for using $\mathcal{FE}$ as the starting point, rather than $i\mathcal{O}$, and restricting to reductions with only a polynomial loss. By significantly expanding on ideas developed by Garg, Pandey, and Srinivasan (CRYPTO 2016), we achieve the following early results in this line of study:

- We construct *universal samplers* based only on polynomially-secure public-key $\mathcal{FE}$. 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.
- We also construct trapdoor one-way permutations (OWP) based on polynomially-secure public-key $\mathcal{FE}$. This improves upon the recent result of Bitansky, Paneth, and Wichs (TCC 2016) which requires $i\mathcal{O}$ of *sub-exponential strength*. We proceed in two steps, first giving a construction requiring $i\mathcal{O}$ of *polynomial strength*, and then specializing the $\mathcal{FE}$-to-$i\mathcal{O}$ conversion to our specific application.

Many of the techniques that have been developed for using $i\mathcal{O}$, including many of those based on the "punctured programming" approach, become inapplicable when we insist on polynomial reductions to $\mathcal{FE}$. As such, our results above require many new ideas that will likely be useful for future works on basing security on $\mathcal{FE}$.

*University of California, Berkeley, sanjamg@berkeley.edu

†Stony Brook University, omkant@gmail.com

‡University of California, Berkeley, akshayaram@berkeley.edu

§Princeton University 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 a plethora of cryptographic primitives, many of them for the first time, [SW14, BZ14, BLR+15, AS16, BPW16]. Recently, $i\mathcal{O}$ also proved instrumental in proving the hardness of complexity class PPAD [BPR15].

A major 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."

Despite its enormous power, we only know of a limited set of techniques for working with $i\mathcal{O}$. In complex situations, these techniques often run into what we call the *sub-exponential barrier*. More specifically, the security proof of many $i\mathcal{O}$-based constructions end 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. There are often exponentially many such inputs, resulting in a sub-exponential loss in the security reduction.

To make matters worse, a sub-exponential loss seems inherent to achieving $i\mathcal{O}$ from "simple" assumptions, such as those based on multilinear maps[1]. Indeed, all known security proofs for $i\mathcal{O}$ relative to "simple" assumptions[2] iterate over all (exponentially-many) inputs anyway, and there are reasons to believe that this loss may be necessary [GGSW13] [3]. Indeed, any reduction from $i\mathcal{O}$ to a simple assumption would need to work for equivalent programs, but should fail for inequivalent programs (since inequivalent programs can be distinguished). Thus, such a reduction would seemingly need to decide if two programs compute equivalent functions; assuming $\mathcal{P} \neq \mathcal{NP}$, this in general cannot be done in polynomial time. This exponential loss would then carry over to any application of $i\mathcal{O}$, even if the $i\mathcal{O}$-to-application security reduction only incurred a polynomial loss. On the other hand, this exponential loss does *not* seem inherent to the vast majority of $i\mathcal{O}$ applications. This leaves us in an undesirable situation where the only way we know to instantiate an application from "simple" assumptions requires sub-exponential hardness assumptions, even though sub-exponential hardness is not inherent to the application.

One application for which an exponential loss does not appear inherent is Functional encryption ($\mathcal{FE}$), and indeed starting from the work of Garg et al. [GGHZ16], it has been shown in [LV16, Lin16] how to build $\mathcal{FE}$ from progressively simpler assumptions on multilinear maps with only a polynomial loss. Therefore, to bypass the difficulties above, we ask the following:

Can applications of $i\mathcal{O}$ be based instead on
$\mathcal{FE}$ with a polynomial security reduction?

There are two results that give us hope in this endeavor. First, it is known that $\mathcal{FE}$ is actually *equivalent* to $i\mathcal{O}$, except that *the $\mathcal{FE}$-to-$i\mathcal{O}$ reduction [AJ15, BV15] incurs an exponential loss.*

[1] Here, we do not define "simple." However, one can consider various notions of "simplicity" or "niceness" for assumptions, such as falsifiable assumptions [Nao03] or complexity assumptions [GK15].

[2] Here, we exclude über assumptions such as semantically secure graded encodings [PST14], which encompass exponentially many distinct complexity assumptions.

[3] We stress that this argument has not yet been formalized.

This hints at the possibility that, perhaps, specializing the $\mathcal{FE}$to-$i\mathcal{O}$-to-application reduction to particular applications can aleviate the need for sub-exponential hardness.

Second and very recently, Garg, Pandey, and Srinivasan [GPS16] 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). More importantly for us, they also presented a new reduction which bases the hardness of PPAD on standard polynomially-secure *functional encryption*, thus giving essentially the first non-trivial instance of using $\mathcal{FE}$ to build applications with only a polynomial loss.

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 [GPS16]. Our techniques are applicable, roughly, to any $i\mathcal{O}$ setting where the computation is changed on just a polynomial number of points; on all other points, the exact same circuit is used to compute the outputs. Notice that for such settings there exists an efficient procedure for checking functional equivalence. This enables us to argue indistinguishability based only on polynomial hardness assumptions. As it turns out, for many applications of $i\mathcal{O}$, the hybrid arguments involve circuits with the above specified structure. In this work, we focus on two such applications: *trapdoor permutations* and *universal samplers*.

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, LV16, Lin16], this gives us trapdoor permutations based on simple polynomial-hard assumptions on multilinear maps.

We then consider *universal samplers*, a notion put forward by Hofheinz, Jager, Khurana, Sahai, Waters, and Zhandry [HJK$^+$16]. It allows for a single trusted setup which can be used to sample common parameters for *any* protocol. 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.

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 non-trivial even given the work of Hofheinz et al. This is because the definitions presented in [HJK$^+$16] are not completely suitable to deal with an unbounded number of users. To support unbounded number of users, we devise 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.

Remark 1.1 *Our construction of TDP from FE is weaker in comparison to our construction from iO (and the construction of Bitansky et al. in [BPW16]). In particular, given the random coins*

used to sample the function and the trapdoor, the output of the sampler is no longer pseudorandom. This property is important for some applications of TDPs like the construction of OT.

An overview of our approach. In the following sections, we present a detailed overview of our approach of constructing Universal Samplers, NIKE for unbounded number of parties and Trapdoor Permutations.

1.1 Universal Samplers and Multiparty Non-interactive Key Exchange from $\mathcal{FE}$

Multiparty Non-Interactive Key Exchange (multiparty NIKE) was one of the early applications of multilinear maps and $i\mathcal{O}$. In multiparty NIKE, n parties simultaneously post a single message to a public bulletin board. Then they each read off the contents of the board, and are then able to derive a shared key K which is hidden to any adversary that does not engage in the protocol, but is able to see the contents of the public bulletin board.

Boneh and Silverberg [BS02] show that multilinear maps imply multiparty NIKE. However, (1) their protocol requires an a priori bound on the number of users n, and (2) due to limitations with current multilinear map candidates [GGH13a, CLT13], the protocol requires a trusted setup. The party that runs the trusted setup can also learn the shared key k, even if that party does not engage in the protocol.

Boneh and Zhandry [BZ14] show how to use $i\mathcal{O}$ to remove the trusted setup. Later, Ananth et al. [ABG$^+$13] shows how to remove the bound on the number of users by using the very strong *differing inputs* obfuscation. Khurana, Rao, and Sahai [KRS15] further modify the Boneh-Zhandry protocol to get unbounded users with just $i\mathcal{O}$. In [BZ14] and [KRS15], $i\mathcal{O}$ is invoked on programs for which are guaranteed to be equivalent; however it is computationally infeasible to actually verify this equivalence. Thus, following the arguments of [GGSW13], it would appear that any reduction to a few simple assumptions, no matter how specialized to the particular programs being obfuscated, would need to incur an exponential loss. Hence, these approaches do not seem suitable to achieving secure multiparty NIKE from polynomially secure $\mathcal{FE}$.

1.1.1 Universal Samplers

Instead, we follow an alternate approach given by Hofheinz et al. [HJK$^+$16] using universal samplers. A 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. 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.

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.

Fortunately for us, the $i\mathcal{O}$-based construction of [HJK+16] only invokes $i\mathcal{O}$ on programs for which it is trivial to verify equivalence. Thus, there seems hope that universal samplers can be based on simple assumptions without an exponential loss. In particular, there is hope to base universal samplers on the polynomial hardness of functional encryption.

Application to 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+16]. 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. Unfortunately, the only prior protocols for achieving an unbounded number of users, [ABG+13] and [KRS15], seems inherently tied to the Boneh-Zhandry approach, and it is not clear that their techniques can be adapted to universal samplers.

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 not 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, we 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 no treated circuits, 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 $\approx \log n$ pebbles by removing certain pebbles as we go. Interestingly, the pebbling scheme is exactly same as the one used in [Ben89] in the context of reversible computation. The pebbling scheme is also efficient: 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 $\approx \log n$.

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+16]. 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 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 of universal samplers 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+16] 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, GPS16]. 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 constrained PRF key K [4]. 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

[4] [GPS16] used the term prefix-punctured PRF to denote the same primitive. We use the term prefix constrained PRF as we feel that this name is more appropriate. This was also suggested by an anonymous Eurocrypt reviewer.

the path from the root to C contains the key K prefix constrained at x. The prefix constrained 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 prefix constrained 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. The simulator accomplishes this by "tunneling" the underlying binary tree along path C_i. To give a bit more details, the simulator "forces" the function keys at every level i to output a precomputed value say V_i (instead of the bit-extension) if the input to the function matches with a prefix of C_i. At the leaf level, if the input matches C_i then the function outputs s_i. Illustration of "tunneling" is given in Figure 1. We now explain how this "tunneling" is done.

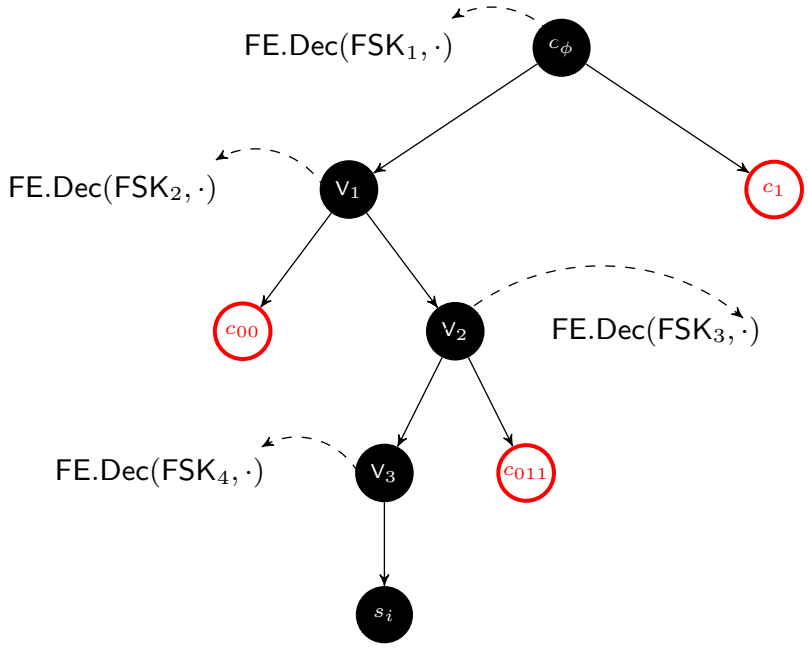

Figure 1: Illustration of "tunneling" on $C_i = 010$ and $\kappa = 3$.

At a high level, the "tunneling" is achieved by triggering a hidden "trapdoor" thread in the function keys using techniques illustrated in [ABSV15, GPS16]. This technique proceeds by first encrypting a set of precomputed values under a symmetric key sk and hardwires them 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 prefix constrained 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 show that we can indistinguishably "tunnel" 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 [GPS16] 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 "tunnel" 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 new path C_i^*. In order to get around this issue, we use multiple secret keys: one for each write query [5] which enables us to "tunnel" along a new path C_i^*.

1.2 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 $[u+1, i-1]$.

Our idea. Garg, Pandey and Srinivasan [GPS16] described a method to eliminate 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. Parts of this section are taken verbatim from [GPS16].

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

[5] 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.

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.

We prove the one-wayness of the above construction using the "multiple chains" approach of Garg et al. [GPS16]. Intuitively, this approach allows one to imagine signatures as "virtual chains" emanating out of every node. The first chain coming out of a point i is connected to its immediate successor $i+1$. The second chain is connected to a point two hops away namely, $i+2$ and so on. More generally, the j-th chain connects point i with point $i+2^j$. The number of chains coming out of node i is one more than the number of trailing ones in the binary representation of i. Equivalently, the number of chains is equal to the number of bits that are different in i and $i+1$. At a high level, puncturing the public key can be though of as cutting a chain of appropriate length. While the Bitansky et al. 's approach in [BPW16] allows to cut a chain of length 1 thus puncturing the public key at one point, "multiple chains" approach illustrated in [GPS16] allows us to puncture the public key at a large number of points (even exponential in number) in a "single-stroke." This allows us to base one-wayness of our construction only on polynomial hardness of indistinguishability obfuscation.

We now discuss a few additional technicalities that arise while formalizing the above idea. These issues do not arise while proving hardness of PPAD. The first technicality is that while attempting to invert a trapdoor challenge the adversary 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. Yet another issue is that, while iteratively puncturing the public key we cannot always cut chains of the longest length as in [GPS16]. This is 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.

1.3 Trapdoor Permutations from $\mathcal{FE}$

Garg, Pandey and Srinivasan in [GPS16] 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. Until this step, the construction is same as the one in [BV15] emulating the obfuscation of public-key of the TDP via functional encryption. The last function key implements a function that outputs encryptions of signatures on the next point $(x + 1)$ using the signatures at x as the secret key. Intuitively, an evaluator can obtain 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 perform this functionality, the bit extension functions compute a set of carefully constructed *prefix-constrained* PRF keys [GPS16] that are passed to the next level.

We note that our construction of public key of the permutation is almost identical [6] to the Garg et al.'s [GPS16] construction of successor circuit. We now move on to our construction of the sampler.

Challenge in Sampler Construction. Recall that the sampler in the $i\mathcal{O}$ based construction satisfied two properties:

1. **Pseudorandomness of Samples:** This property states that given a random input r, the first component of the output of the sampler is pseudorandom. To be more precise, if the output of the sampler is $(i, \cdot, \ldots, \cdot)$ then i should be computationally indistinguishable to random given the public key and the description of the sampler.

2. **Sparseness of Images:** Intuitively, this property states that the size of the image set of the sampler is much smaller than the size of the domain. This property is crucially used in the one-wayness proof to argue that (with overwhelming probability over the randomness of i) the sampler's image does not intersect with the range $[u, i - 1]$.

In the $i\mathcal{O}$ based construction, the sampler first expands the randomness using a length doubling pseudorandom generator PRG and then signs on all prefixes of the result. The first property namely, pseudorandomness of samples follows directly from the security of PRG. The sparseness of images follows from the property that PRG is length doubling.

If we try to directly implement the $i\mathcal{O}$ based sampler construction using the prefix constraining techniques in [GPS16], we run into the following problem. Recall that the sampler has to sign on all prefixes of $\mathsf{PRG}(r)$. This translates to constraining the keys $S_1, \cdots, S_\kappa$ along the prefixes of $\mathsf{PRG}(r)$. But $\mathsf{PRG}(r)$ can only be computed after r has been parsed in the encryption tree. This implies that along every path in the sub-tree where r is parsed, $S_1, \cdots, S_\kappa$ must be propagated. Thus, constraining any key S_i along a prefix of $\mathsf{PRG}(r)$, involves constraining it along every path in the sub-tree where r is parsed. This incurs an exponential loss in the security reduction!

In order to overcome this challenge, we modify our sampler construction so that instead of signing on $\mathsf{PRG}(r)$ it now signs on $r\|K_r$ ($r \in \{0,1\}^{\kappa/2}$ is the random input to the sampler) where K is a fresh prefix constrained key. Recall that K_r denotes the PRF key K constrained at prefix r. To give more details, 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

[6] 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 $(0^\kappa, \cdot, \cdots, \cdot)$.

are used to obtain an functional encryption of r. The bit extension functions also pass down a set of keys $S_1, \cdots, S_\kappa$ that are constrained 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 a new PRF key K constrained along the prefixes of r. The last function key (numbered $\kappa/2+1$) implements a function that takes in r, the set of keys $\{S_{1,r_{[1]}}, \cdots, S_{\kappa,r_{[\kappa/2]}}\}$ where $S_{i,r_{[j]}}$ denotes the key S_i prefix constrained on the first j-bit prefix of r and the new PRF key K that is also prefix constrained at r and outputs $(r\|K_r, S_{1,(r\|K_r)_{[1]}}, \cdots, S_{\kappa,(r\|K_r)_{[\kappa]}})$. Observe that the final function can compute this output given r and the set of prefix constrained keys. This sampler "conforms" well with the prefix puncturing techniques of Garg et al. [GPS16] and has the required sparseness of images guarantee. But the issue with this construction is in arguing the pseudorandomness of samples property. In particular, given the description of the sampler, we still can't use the prefix constraining techniques in [GPS16] to show that K_r can be replaced with a random string. This is because the sampler outputs K_r in the clear when run with input r.

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 $r\|K_r$ as well as K_r using r as the public key. To give more details, 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 $pk\|K_{pk}$ as well as encryption of K_{pk}. These encryptions are then decrypted using the secret key sk and $pk\|K_{pk}$ along with the signatures on its prefixes are output. To prove the pseudorandomness of samples, we rely on prefix constraining technique of Garg et al. [GPS16] and the semantic security of public key encryption (which enables us to replace the encryptions (including that of K_{pk}) that are output on input pk with junk values). [7]

Remark 1.2 *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 Preliminaries

κ denotes the security parameter. A function $\mu(\cdot) : \mathbb{N} \to \mathbb{R}^+$ is said to be negligible if for all polynomials $\mathsf{poly}(\cdot)$, $\mu(k) < \frac{1}{\mathsf{poly}(k)}$ for large enough k. 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. For a finite set S, we denote $x \xleftarrow{\$} S$ as the process of sampling x uniformly from the set S. We will use PPT to denote Probabilistic Polynomial Time algorithm. We denote $[k]$ 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. We denote the identity polynomial by $I(\cdot)$ i.e. $I(x) = x$. All adversarial functions are modeled as polynomial sized circuits. We assume that all cryptographic algorithms take the security parameter in unary as input and would not explicitly mention it in all cases. We assume without loss of generality that the length of the random tape used by all cryptographic algorithms is κ.

[7] 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.

A binary string $x \in \{0,1\}^k$ is represented as $x_1 \cdots x_k$. x_1 is the most significant (or the highest order bit) and x_k is the least significant (or the lowest order bit). The i-bit prefix $x_1 \cdots x_i$ of the binary string x is denoted by $x_{[i]}$. We denote $|x|$ to be the length of the binary string $x \in \{0,1\}^*$. 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 2.1 *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 need an additional property from an injective PRG. Let us consider a PRG where the expansion factor (or the output length) is given by $2\ell(\cdot)$. 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 2.2 *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 2.3 *Assuming the existence of one-way permutations, there exists a pseudorandom generator with expansion factor* $2\ell(\kappa)$ *where* $\ell(\kappa) > \kappa$ *and 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 2.4 *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 2.5 *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.
- $\mathsf{FE.Enc}_{\mathsf{PK}}(m)$: Takes as input a message $m \in \{0,1\}^*$ and outputs an encryption C of m under the public key PK.

- $\mathsf{FE.KeyGen}(\mathsf{MSK}, f)$: 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 2.6 (Correctness) *The functional encryption scheme $\mathcal{FE}$ is correct if for all κ and for all messages $m \in \{0,1\}^*$,*

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

Definition 2.7 (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 $(\mathsf{PK}, \mathsf{MSK}) \leftarrow \mathsf{FE.Setup}(1^\kappa)$ and generates the challenge ciphertext $C \leftarrow \mathsf{FE.Enc}_{\mathsf{PK}}(m_b)$. 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 2.8 *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 2.9 (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 2.10 ([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 2.11 *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:

- *The challenger samples $SK \leftarrow \mathsf{SK.KeyGen}(1^\kappa)$.*
- **Left-Right Oracle Queries:** *The adversary submits two messages m_0, m_1 and the challenger responds with $\mathsf{SK.Enc}_{SK}(m_b)$. The adversary can make an arbitrary number of left-right queries.*
- *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 2.12 *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 2.13 *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 Constrained Pseudorandom Function. A PCPRF is a tuple of algorithms ($\mathsf{KeyGen}_{\mathcal{PCPRF}}$, PrefixCons) with the following syntax. $\mathsf{KeyGen}_{\mathcal{PCPRF}}$ takes the security parameter (encoded in unary) and descriptions of two polynomials p_{in} and p_{out} as input and outputs a PCPRF key $S \in \{0,1\}^\kappa$. PrefixCons is a deterministic algorithm and has two modes of operation:

1. **Normal Mode:** In the normal mode, PrefixCons takes a PCPRF key S and a string $y \in \cup_{k=0}^{p_{in}(\kappa)}\{0,1\}^k$ and outputs a prefix constrained key $S_y \in \{0,1\}^\kappa$ if $|y| < p_{in}(\kappa)$; else outputs $S_y \in \{0,1\}^{p_{out}(\kappa)}$. We assume that S_y contains implicit information about $|y|$.
2. **Repeated Constraining Mode:** In the repeated constraining mode, PrefixCons takes a prefix constrained key S_y and a string $z \in \cup_{k=0}^{p_{in}(\kappa)}\{0,1\}^k$ as input and works as follows. If $|y| + |z| > p_{in}(\kappa)$, it outputs $\perp$; else if $|y| + |z| < p_{in}(\kappa)$, it outputs the prefix constrained key $S_{y\|z} \in \{0,1\}^\kappa$; else it outputs $S_{y\|z} \in \{0,1\}^{p_{out}(\kappa)}$.

Henceforth, unless it is not directly evident from the context, we will not explicitly mention if PrefixCons is in the normal mode or in the repeated constraining mode. We note that there is no explicit evaluation procedure for PCPRF and the output of PCPRF on an input $x \in \{0,1\}^{p_{in}(\kappa)}$ is given by $\mathsf{PrefixCons}(S, x) \in \{0,1\}^{p_{out}(\kappa)}$.

We now describe the security properties that PCPRF must satisfy. The first property states that for any string $x \in \cup_{k\in[p_{in}(\kappa)]}\{0,1\}^k$, constraining a key S on x is equivalent to constraining the key first on *any* prefix of x and then constraining it on the suffix. Let T denote the set of all prefixes of x but with last bit flipped. For example, if $x = 101$, then $T = \{0, 11, 100\}$. The second property states that $\mathsf{PrefixCons}(S, x)$ is pseudorandom even given the set of keys constrained on every element in T.

Definition 2.14 *A prefix constrained pseudorandom function $\mathcal{PCPRF}$ is a tuple of PPT algorithms ($\mathsf{KeyGen}_{\mathcal{PCPRF}}$, PrefixCons) satisfying the following properties:*

- ***Functionality is preserved under* repeated *constraining:*** *For all κ, polynomials $p_{in}(\cdot), p_{out}(\cdot)$ and for all $x \in \cup_{k\in[p_{in}(\kappa)]}\{0,1\}^k$, $y, z \in \{0,1\}^*$ s.t. $x = y\|z$,*

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

  where $S \leftarrow \mathsf{KeyGen}_{\mathcal{PCPRF}}(1^\kappa, p_{in}(\cdot), p_{out}(\cdot))$.

- ***Pseudorandomness at constrained prefix**: For all κ, polynomials $p_{in}(\cdot)$, $p_{out}(\cdot)$, for all $x \in \cup_{k\in[p_{in}(\kappa)]}\{0,1\}^k$, and for all poly sized adversaries $\mathcal{A}$*

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

  where $S \leftarrow \mathsf{KeyGen}_{\mathcal{PCPRF}}(1^\kappa, p_{in}(\cdot), p_{out}(\cdot))$, $\ell = |\mathsf{PrefixCons}(S, x)|$ and $\mathsf{Keys} = \{\mathsf{PrefixCons}(S, x_{[i-1]}\|(1-x_i))\}_{i\in[|x|]}$.

The above properties are satisfied by the construction of the pseudorandom function in [GGM86].

Notation. For a key S_i (indexed by i), we will use $S_{i,y}$ to denote $\mathsf{PrefixCons}(S_i, y)$.

3 Universal Samplers

Intuitively, a universal sampler, defined by Hofheinz et al. [HJK+16] 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.

3.1 Definition

A *Universal Sampler* consists of an algorithm Setup that takes as input a security parameter κ (encoded in unary) and a size bound $\ell(\cdot)$, random tape size $r(\cdot)$ and an output size $t(\cdot)$. It outputs a program $\mathsf{Sampler}$. $\mathsf{Sampler}$ takes as input a circuit of size at most $\ell(\kappa)$, uses $r(\kappa)$ bits of randomness and outputs an $t(\kappa)$-bit string.

Intuitively, $\mathsf{Sampler}(C)$ will be a pseudorandom sample from C: $\mathsf{Sampler}(C) = C(s)$ for some s 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 output by $\mathsf{Sampler}(C)$ actually looks like a fresh random sample from C. Unfortunately, formalizing this requirement is tricky. Hofheinz et al. [HJK+16] 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 3.1 *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 *(encoded in unary) and three polynomials* $\ell(\cdot), r(\cdot), t(\cdot)$ *(for ease of notation, we denote* $\ell = \ell(\kappa)$*,* $r = r(\kappa)$ *and* $t = t(\kappa)$*), 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*, that uses* r *bits of randomness and has 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*, that uses* r *bits of randomness with output length* t*, as well as a sample* s *of length* t*.* 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.** *Roughly, this requirement says that in the absence of any* **Write** *queries, and honest and simulated sampler are indistinguishable. More precisely, 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* $\mathsf{Sampler} \leftarrow \mathsf{Setup}(1^\kappa, \ell, r, t)$*. If* $b = 1$*, the challenger initiates* $\mathsf{Sim}(1^\kappa, \ell, r, t)$*.*
  - A *is allowed to make* **Read** *queries on arbitrary circuits* C *of size at most* ℓ*, using* r *bits of randomness and 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.** *Roughly, this requirement says that, in the simulated sampler, if an additional* **Set** *query is performed on* (C, s) *where* s *is a fresh sample from* C*, then the simulated sampler is indistinguishable from the case where the* **Set** *query was not performed. More precisely, the advantage of any polynomial-time algorithm* B *is negligible in the following experiment:*

  - *The challenger flips a random bit* b*. It then initiates* $\mathsf{Sim}(1^\kappa, \ell, r, t)$*.*
  - B *first makes a* **Challenge** *query on circuit* C^* *of size at most* ℓ*, using* r *bits of randomness 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 allowed 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.*

In Appendix A we sketch a construction of Universal Samplers satisfying the above definition using iO. But as noted in the introduction, constructing iO — at least from "simple" assumptions — seems to inherently require an exponential loss in security. In the next section we give a construction of Universal Samplers from polynomially hard, compact Functional Encryption.

3.2 Construction from FE

In this section, we will construct Universal Samplers that satisfies Definition 3.1 from polynomially hard, compact Functional Encryption and Prefix Constrained Pseudorandom Function (which is implied by Functional Encryption).

Theorem 3.2 *Assuming the existence of selective secure, single key, compact public key functional encryption there exists an Universal Sampler scheme satisfying Definition 3.1.*

Our Construction. The formal description our construction appears in Figure 2.

3.3 Security

We give the description of a simulator that satisfies Definition 3.1 in Figure 4.

Correctness. We give a sketch of the proof of correctness. We need to argue the following properties of the sampler constructed by Sim:

1. For every read query C made by the adversary and response s given by Sim to that query, the output of the sampler on C should be equal to s.

2. For every set query (C, s) made by the adversary, the output of sampler on C should be equal to s.

Let $\mathcal{C}_q$ be the set of all circuits queried by the adversary in a set query. By definition, $q \leq n$. Let C be an arbitrary read query. Let x be the longest prefix of C that is shared with an element in $\mathcal{C}_q$. Since the adversary is not allowed to submit the same circuit in both read and set queries, we notice that $x \neq C$ or in other words x is a strict prefix of C. When we evaluate the sampler output by Sim on input C (refer Step 1 in Evaluating the Sampler given in Figure 2), we observe that for every prefix y of x (including x), we obtain $c_y = \mathsf{FE.Enc}_{\mathsf{PK}_{|y|+1}}(y, 0^\kappa, 0^\kappa, Z'_q, 1; r_y)$. This can be shown via a simple induction argument on the length of the prefix (by observing the elements in $\pi^q_{|y|-1}$) with the base case being the empty prefix. Again by an induction argument on the length of the prefix, we can show that for every prefix z of C of length $> |x|$, we obtain $c_z = \mathsf{FE.Enc}_{\mathsf{PK}_{|z|+1}}(z, S_z, K_z, Z'_q, 0; K'_z)$ (the base case is prefix of length $|x|+1$ and by choice of $\pi^q_{|x|}$ we obtain $c_{C_{[|x|+1]}} = \mathsf{FE.Enc}_{\mathsf{PK}_{|x|+2}}(C_{[|x|+1]}, S_{C_{[|x|+1]}}, K_{C_{[|x|+1]}}, Z'_q, 0; K'_{C_{[|x|+1]}})$). Hence, c_C obtained in the evaluation is equal to $\mathsf{FE.Enc}_{\mathsf{PK}_{\ell+1}}(C, S_C, K_C, Z'_q, 0; K'_C)$. Decrypting c_C with $\mathsf{FSK}_{\ell+1}$, gives $C(S_C)$ which is same as the response given by Sim. This shows the first part of the claim.

Setup

- **Input:** 1^κ and three polynomials $\ell(\cdot), r(\cdot), t(\cdot)$.

- **Sampled Ingredients**:

  1. Sample $S \leftarrow \mathsf{KeyGen}_{\mathcal{PCPRF}}(1^\kappa, \ell(\cdot), r(\cdot))$ and $K \leftarrow \mathsf{KeyGen}_{\mathcal{PCPRF}}(1^\kappa, \mathsf{rand}(\cdot), I(\cdot))$ where $\mathsf{rand}(\kappa) = 2\ell(\kappa)$ and $I(\kappa) = \kappa$. For ease of notation, we denote $\ell = \ell(\kappa)$ and $r = r(\kappa)$.
  2. For every $i \in [\ell + 1]$, sample $(\mathsf{PK}_i, \mathsf{MSK}_i) \leftarrow \mathsf{FE.Setup}(1^\kappa)$.
  3. For every $j \in [n]$, sample $sk_j \leftarrow \mathsf{SK.KeyGen}(1^\kappa)$. Let $|sk_j| = p(\kappa)$. For $i \in [\ell+1]$ and $j \in [n]$, let $\Pi_i^j \leftarrow \mathsf{SK.Enc}_{sk_j}(\pi_i^j)$ where $\pi_i^j = 0^{\mathsf{len}(\kappa)}$. Here $\mathsf{len}(\cdot)$ is an appropriate length function that would be specified later. For all $i \in [\ell + 1]$, let $\Pi_i = \{\Pi_i^j\}_{j\in[n]}$.

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

  1. For each $i \in [\ell]$, generate $\mathsf{FSK}_i \leftarrow \mathsf{FE.KeyGen}(\mathsf{MSK}_i, F_{i,PK_{i+1},\Pi_i})$ and $\mathsf{FSK}_{\ell+1} \leftarrow \mathsf{FE.KeyGen}(\mathsf{MSK}_{\ell+1}, G_{\Pi_{\ell+1}})$, where F_{i,PK_{i+1},Π_i} and $G_{\Pi_{\ell+1}}$ are circuits described in Figure 3.
  2. For every $j \in [n]$, $Z_j = (j, \perp)$. Let $Z := \{Z_j\}_{j\in[n]}$.
  3. Let $c_\phi = \mathsf{FE.Enc}_{\mathsf{PK}_1}(\phi, S, K, Z, 0)$.
  4. Output $(c_\phi, \{\mathsf{FSK}_i\}_{i\in[\ell+1]})$ as the sampler.

Evaluating the Sampler

- **Input:** Circuit C of size ℓ (padded with dummy symbols if its size is less than ℓ) using r bits of randomness and output length t and the sampler given by $(c_\phi, \{\mathsf{FSK}_i\}_{i\in[\ell+1]})$.

- **Evaluation**:

  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. Compute d_C as output of $\mathsf{FE.Dec}(\mathsf{FSK}_{\ell+1}, c_C)$.
  3. Output d_C.

Figure 2: Setup and Evaluating the Sampler

Let (C, s) be an arbitrary set query. When evaluating the sampler on input C, we observe that for every prefix y of C, we obtain $c_y = \mathsf{FE.Enc}_{\mathsf{PK}_{|y|+1}}(y, 0^\kappa, 0^\kappa, Z'_q, 1; r_y)$. Hence, c_C obtained in the evaluation procedure is equal to $\mathsf{FE.Enc}_{\mathsf{PK}_{\ell+1}}(C, 0^\kappa, 0^\kappa, Z'_q, 1; r_y)$. Decrypting c_C with $\mathsf{FSK}_{\ell+1}$ gives (C, s) as the output due to the choice of $\pi^q_{\ell+1}$.

F_{i,PK_{i+1},Π_i}

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

1. If ($\mathsf{mode} = 0$),
   (a) Output $\mathsf{FE.Enc}_{PK_{i+1}}(C\|0, S_{C\|0}, K_{C\|0}, Z, \mathsf{mode}; {K'}_{C\|0})$ and $\mathsf{FE.Enc}_{PK_{i+1}}(C\|1, S_{C\|1}, K_{C\|1}, Z, \mathsf{mode}; {K'}_{C\|1})$, where for $b \in \{0,1\}$, $K_{C\|b} = \mathsf{PrefixCons}(K_C, b\|0)$ and ${K'}_{C\|b} = \mathsf{PrefixCons}(K_C, b\|1)$ and $S_{C\|b} := \mathsf{PrefixCons}(S_C, b)$.
2. Else,
   (a) Let j^* be the minimum value of $j \in [n]$ such that $Z_{j+1} = (j+1, \perp)$.
   (b) Let $\pi_i^{j^*} \leftarrow \mathsf{SK.Dec}_{sk_{j^*}}(\Pi_i^{j^*})$ where $\pi_i^{j^*}$ is a collection of elements of the form $(C', \cdot, \cdot)$ for $C' \in \{0,1\}^{i-1}$. Recover $(C, (C\|b, c_{C\|b}), (C\|(1-b), c_{C\|(1-b)}))$ (if there are more than one value of $(C, \cdot, \cdot)$, select the lexicographically first such value) from $\pi_i^{j^*}$ 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, let j^* be the minimum value of $j \in [n]$ such that $Z_{j+1} = (j+1, \perp)$. Recover (C, d_C) from $\mathsf{SK.Dec}_{sk_{j^*}}(\Pi_i^{j^*})$ and output d_C.

Figure 3: Circuits for simulating Public Key.

Indistinguishability from honest generation. This property directly follows from the observation that when adversary makes only read queries, the distribution of the sampler generated by Sim is identically distributed to an honestly generated sampler.

Pseudorandomness of Samples. We show this through a hybrid argument.

- Hyb_0: This is $b = 0$ case in the pseudorandomness game. The **Challenge** query is answered by making a **Read** query to Sim, which responds with $s^* = C^*(S^1_{C^*})$.
- Hyb_1 : Informally speaking, in this hybrid we are going to "tunnel" through sampler circuit along all the paths given by the circuits in the set queries including the challenge circuit C^*.

- **Input:** 1^κ and three polynomials $\ell(\cdot), r(\cdot), t(\cdot)$.

- **Sampled Ingredients**:

  1. Sample $S \leftarrow \mathsf{KeyGen}_{\mathcal{PCPRF}}(1^\kappa, \ell(\cdot), r(\cdot))$ and $K \leftarrow \mathsf{KeyGen}_{\mathcal{PCPRF}}(1^\kappa, \mathsf{rand}(\cdot), I(\cdot))$
  2. For all $i \in [\ell+1]$, sample $(\mathsf{PK}_i, \mathsf{MSK}_i) \leftarrow \mathsf{FE.Setup}(1^\kappa)$.
  3. For all $j \in [n]$, sample $sk_j \leftarrow \mathsf{SK.KeyGen}(1^\kappa)$.

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

- **Set Queries:** For every **Set** query (C_i, s_i), the simulator records the query.

- **Setting π_i^j values**. Let q denote the number of **Set** queries made by the adversary. By definition, $q \leq n$. For each $j \in [q]$,

  1. Let $Z'_j = \{(1, sk_1), \cdots, (j, sk_j), (j+1, \perp), \cdots, (n, \perp)\}$.
  2. Let $\mathcal{C}_j = \{C_1, \cdots, C_j\}$ where C_k is the k-th **Set** query. For every non-empty prefix x of length at most $\ell - 1$ of some string C_k in $\mathcal{C}_j$,
     (a) Let $c_x \leftarrow \mathsf{FE.Enc}_{\mathsf{PK}_{|x|+1}}(x, 0^\kappa, 0^\kappa, Z'_j, 1; r_x)$ where r_x denotes uniformly chosen random string from $\{0,1\}^\kappa$.
     (b) Let y denote the string which is same as x except that the last bit of x is flipped. More formally, $y = x_{[|x|-1]} \| (1 - x_{|x|})$. Let $e_y = \mathsf{FE.Enc}_{\mathsf{PK}_{|y|+1}}(y, S_y, K_y, Z'_{j-1}, 0; K'_y)$.
     (c) Add the element $(x_{[|x|-1]}, (x, c_x), (y, e_y))$ to $\pi^j_{|x|}$.
  3. For each $i \in [\ell]$, pad π_i^j until its length is $0^{\mathsf{len}(\kappa)}$. Set $\pi^j_{\ell+1} := (C_1, s_1), \cdots, (C_j, s_j)$ and pad until its length is $0^{\mathsf{len}(\kappa)}$.
  4. For every $i \in [\ell+1]$, set $\Pi_i^j := \mathsf{SK.Enc}_{sk_j}(\pi_i^j)$.

  For $q+1 \leq j \leq n$ and for all $i \in [\ell+1]$, set $\Pi_i^j := \mathsf{SK.Enc}_{sk_j}(0^{\mathsf{len}(\kappa)})$. For each $i \in [\ell+1]$, let $\Pi_i = \{\Pi_i^j\}_{j \in [n]}$.

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

  1. For each $i \in [\ell]$, generate $\mathsf{FSK}_i \leftarrow \mathsf{FE.KeyGen}(\mathsf{MSK}_i, F_{i, PK_{i+1}, \Pi_i})$ and $\mathsf{FSK}_{\ell+1} \leftarrow \mathsf{FE.KeyGen}(\mathsf{MSK}_{\ell+1}, G_{\Pi_{\ell+1}})$ where F_{i, PK_{i+1}, Π_i} and $G_{\Pi_{\ell+1}}$ are circuits described in Figure 3.
  2. If $q > 1$ then set $c_\phi = \mathsf{FE.Enc}_{\mathsf{PK}_1}(\phi, 0^\kappa, 0^\kappa, Z'_q, 1)$; else set $c_\phi = \mathsf{FE.Enc}_{\mathsf{PK}_1}(\phi, S, K, Z'_0, 0)$ where $Z'_0 = (1, \perp), \cdots, (n, \perp)$.
  3. Output $(c_\phi, \{\mathsf{FSK}_i\}_{i \in [\ell+1]})$ as the sampler.

- **Evaluating the Sampler**: Same as described in Figure 2.

Figure 4: Description of the simulator

Recall that "tunneling" through a path C means that at every prefix x of the string C, c_x obtained in Step 1 of the evaluation algorithm has the mode bit set to 1. Additionally, instead of encrypting the "secrets" S_x and K_x, c_x encrypts the junk values (say the all zeroes string) in those positions. We now provide the details of how to accomplish the tunneling below.

Let us assume that the adversary has made q ($\leq n-1$) set queries. For the sake of simplicity, we assume that $q > 0$. The case when $q = 0$ is analogous. Let $(C'_1, \cdot), (C'_2, \cdot), \cdots, (C'_q, \cdot)$ be the collection of all **set** queries in the same order as queried by the adversary. In the collection of all read and set queries, let the challenge index i^* be such that the i^*-th query is in between the set queries $(C'_{k^*}, \cdot)$ and $(C'_{k^*+1}, \cdot)$. We denote $(C_1, \cdots, C_{q+1}) = (C'_1, \cdots, C'_{k^*}, C^*, C'_{k^*+1}, \cdots, C'_q)$. Let $Z'_q = (1, sk_1), \cdots, (q, sk_q), (q+1, \perp), \cdots, (n, \perp)$ and let $Z'_{q+1} = (1, sk_1), \cdots, (q+1, sk_{q+1}), (q+2, \perp), \cdots, (n, \perp)$.

- $\mathsf{Hyb}_{0,0}$: In this hybrid, we are going to change how π_i^{q+1} is generated for every $i \in [\ell+1]$. For every prefix x of a string in $(C_1, \cdots, C_{q+1})$ of length at most $\ell - 1$,
  1. Let $e_x = \mathsf{FE.Enc}_{PK_{|x|+1}}(x, S_x, K_x, Z'_q, 0; K'_x)$. Notice that e_x is same as c_x obtained in Step 1 of evaluation algorithm when initialized with $c_\phi = \mathsf{FE.Enc}_{\mathsf{PK}_1}(\phi, 0^\kappa, 0^\kappa, Z'_q, 1)$.
  2. Let y denote the string which is same as x except that the last bit of x is flipped. Let e_y be defined analogously.
  3. We add $(x_{[|x|-1]}, (x, e_x), (y, e_y))$ to $\pi_{|x|}^{q+1}$.

  We finally set $\pi_{(\ell+1)}^{q+1} = (C_1, s_1), \cdots, (C_{q+1}, s_{q+1})$. For every $i \in [\ell+1]$, we pad π_i^{q+1} with dummy symbols so that its length $= \mathsf{len}(\kappa)$. For all $i \in [\ell+1]$, we set $\Pi_i^{q+1} = \mathsf{SK.Enc}_{sk_{q+1}}(\pi_i^{q+1})$.

  Notice that the only difference between Hyb_0 and $\mathsf{Hyb}_{0,0}$ is that in Hyb_0, Π_i^{q+1} is set to encryption of all zeroes string of length $\mathsf{len}(\kappa)$ whereas in Hyb_1, Π_i^{q+1} is generated as above. The probability that adversary outputs 1 in Hyb_0 and $\mathsf{Hyb}_{0,0}$ is negligible close from the semantic security of symmetric key encryption as shown in the claim below.

Lemma 3.3 *Assuming the semantic security of symmetric key encryption $\mathcal{SKE}$, the probability that adversary outputs 1 in Hyb_0 is negligible close to the probability it outputs 1 in $\mathsf{Hyb}_{0,0}$.*

Proof Assume for the sake of contradiction that absolute difference of the probabilities that adversary outputs 1 in Hyb_0 and $\mathsf{Hyb}_{0,0}$ is non-negligible. We construct an adversary $\mathcal{B}$ breaking the semantic security of symmetric key encryption.

$\mathcal{B}$ works exactly as Sim in Hyb_0 except that:

- It does not sample the symmetric key sk_{q+1} and samples all other secret keys by itself.
- For every $i \in [\ell+1]$, it sets π_i^{q+1} exactly as the simulator in $\mathsf{Hyb}_{0,0}$ and queries the external challenger with the challenge messages $\{\pi_i^{q+1}\}_{i\in[\ell+1]}$ and $(0^{\mathsf{len}(\kappa)})^{\ell+1}$. It receives the challenge ciphertexts $\{C_i\}_{i\in[\ell+1]}$ where $\{C_i\}$ is either an encryption of $\{\pi_i^{q+1}\}$ or $\{0^{\mathsf{len}(\kappa)}\}$. It sets $\Pi_i^{q+1} = C_i$ for every $i \in [\ell+1]$.
- It finally outputs whatever the adversary outputs.

Notice that if $\{C_i\}_{i\in[\ell+1]}$ is an encryption $\{\pi_i^{q+1}\}$ the view generated by $\mathcal{B}$ is identical to the view generated by the simulator in Hyb_0. Else, the view generated by $\mathcal{B}$ is identical to the view generated by Sim in $\mathsf{Hyb}_{0,0}$. Thus, $\mathcal{B}$ breaks the semantic security of symmetric encryption scheme.

■

– $\mathsf{Hyb}_{0,1}$: In this hybrid, for every prefix x of an element in $\{C_1, \cdots, C_{q+1}\}$, we change how e_x is generated in $\pi_{|x|}^{q+1}$.

We introduce a partial ordering of strings denoted by $\prec$ where $x \prec x'$ if $|x| < |x'|$. Let T be the partially ordered set of prefixes (including the empty prefix) of elements in $\{C_1, \cdots, C_{q+1}\}$. Let $\mathsf{Hyb}_{0,x}$ denote an hybrid where for all $x' \prec x$ in T, $e_{x'}$ is set to $\mathsf{FE.Enc}_{PK_{|x'|+1}}(x', 0^\kappa, 0^\kappa, Z'_{q+1}, 1; r_{x'})$ where $r_{x'}$ is chosen uniformly at random. Note that in the previously, $e_{x'}$ was set to $\mathsf{FE.Enc}_{PK_{|x'|+1}}(x', S_{x'}, K_{x'}, Z'_q, 0; K'_{x'})$. Let x^* be the last element in T. Notice that Hyb_{0,x^*} is distributed identically to Hyb_1. We first show the following lemma.

Lemma 3.4 *Assuming the single-key, selective security of functional encryption scheme the probability that adversary outputs 1 in* $\mathsf{Hyb}_{0,0}$ *and* $\mathsf{Hyb}_{0,\phi}$ *is negligibly close.*

Proof Assume for the sake of contradiction that the statement of lemma is not true. We construct an adversary $\mathcal{B}$ against the single-key, selective security of $\mathcal{FE}$ scheme.

$\mathcal{B}$ works exactly as the simulator in $\mathsf{Hyb}_{0,0}$ except that:

* It does not sample the public key, master secret key pair $(\mathsf{PK}_1, \mathsf{MSK}_1)$ but samples all other pairs by itself.
* It computes the function $F_{1,\mathsf{pk}_2,\Pi_1}$ exactly as the simulator in $\mathsf{Hyb}_{0,0}$. It constructs two messages $(\phi, S, K, Z'_q, 0)$ and $(\phi, 0^\kappa, 0^\kappa, Z'_{q+1}, 1)$ and gives these messages as the challenge messages to the external challenger. It also queries functional secret key for the function $F_{1,\mathsf{pk}_2,\Pi_1}$. It receives the challenge ciphertext c^* and sets $c_\phi = c^*$. It sets FSK_1 to be the same as FSK^* obtained from the challenger.
* It finally outputs whatever the adversary outputs.

Notice that the output of $F_{1,\mathsf{pk}_2,\Pi_1}$ is the same on both inputs $(\phi, S, K, Z'_q, 0)$ and $(\phi, 0^\kappa, 0^\kappa, Z'_{q+1}, 1)$. Additionally, the choice of these two messages does not depend on the public key PK_1. Thus, $\mathcal{B}$ represents a valid adversary against the selective security of $\mathcal{FE}$.

We observe that if c^* is an encryption of $(\phi, S, K, Z'_q, 0)$, the view generated by $\mathcal{B}$ is identical to the view generated by simulator in $\mathsf{Hyb}_{0,0}$. Else, the view generated by $\mathcal{B}$ is identical to the view generated by simulator in $\mathsf{Hyb}_{0,\phi}$. Thus, $\mathcal{B}$ breaks the single-key, selective security of $\mathcal{FE}$.

■

To complete the proof of indistinguishability between Hyb_0 and Hyb_1, we now show that the probability that the adversary outputs 1 in $\mathsf{Hyb}_{0,x}$ is negligibly close to the probability that it outputs 1 in $\mathsf{Hyb}_{0,x'}$ where $x \prec x'$ and x and x' are adjacent elements in the ordered set T.

* $\mathsf{Hyb}_{0,x,1}$: In this hybrid, we are going to change $K'^{1}_{x'}$ to uniformly chosen random value $r_{x'}$ at $\pi^{q+1}_{|x'|}$.

Lemma 3.5 *Assuming the pseudorandomness at constrained prefix property of* PCPRF*, the probability that adversary outputs 1 in* $\mathsf{Hyb}_{0,x}$ *and* $\mathsf{Hyb}_{0,x,1}$ *is negligibly close.*

Proof Assume for the sake of contradiction that the statement of the lemma is not true. We construct an adversary $\mathcal{B}$ against the pseudorandomness at constrained prefix property of PCPRF.

$\mathcal{B}$ works exactly as the simulator in Hyb_x except that:

1. It does not sample K.
2. It queries the external challenger with the prefix $s = x'_1\|0\|x'_2\|0\cdots x'_{|x'|-1}\|0\|x'_{|x'|}\|1$. In return, it receives a challenge string y which is either $\mathsf{PrefixCons}(K, s)$ or a random string as well as Keys which contains $\{K_{s_{[i-1]}\|(1-s_i)}\}_{i\in[|s|]}$.
3. It uses Keys to generate all encryptions in $\{\pi^{q+1}_i\}_{i\in[\ell+1]}$ except $e_{x'}$. Notice that in $\mathsf{Hyb}_{0,x}$ and $\mathsf{Hyb}_{0,x,1}$, Keys contain all the information needed to generate encryptions in $\{\pi^{q+1}_i\}_{i\in[\ell+1]}$ except $e_{x'}$.
4. It sets $e_{x'} = \mathsf{FE.Enc}_{\mathsf{PK}_{|x'|+1}}(x', S_{x'}, K_{x'}, 0; y)$.
5. It finally outputs whatever adversary outputs.

If $y = \mathsf{PrefixCons}(K, s)$ then the view generated by $\mathcal{B}$ is identical to view generated in $\mathsf{Hyb}_{0,x}$. Else, the view is identical to the view generated in $\mathsf{Hyb}_{0,x,1}$. Thus, $\mathcal{B}$ breaks the pseudorandomness at constrained prefix property of PCPRF.

■

* $\mathsf{Hyb}_{0,x,2}$: In this hybrid, we are going to change $c_{x'} = \mathsf{FE.Enc}_{PK_{|x'|+1}}(x', 0^\kappa, 0^\kappa, Z'_{q+1}, 1; r_{x'})$ at $\pi^{q+1}_{|x'|}$. 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 $\mathsf{Hyb}_{0,x'}$.

Lemma 3.6 *Assuming the single-key, selective security of functional encryption scheme the probability that adversary outputs 1 in* $\mathsf{Hyb}_{0,x,1}$ *and* $\mathsf{Hyb}_{0,x,2}$ *is negligibly close.*

Proof The proof of this lemma is very similar to proof of Lemma 6.8. ■

– Hyb_2 : In this hybrid, we are going to replace $(C^*, C^*(S_{C^*}))$ in $\pi^{q+1}_{\ell+1}$ with $(C^*, C^*(r^*))$ where r^* is chosen uniformly at random. This change is possible from the pseudorandomness at prefix punctured property of PCPRF. Observe that Hyb_2 is identically distributed to the case where the challenge query is a **Set** query of an uniform sample.

Lemma 3.7 *Assuming the pseudorandomness at constrained prefix property of* PCPRF*, the probability that adversary outputs 1 in* $\mathsf{Hyb}_{0,x}$ *and* $\mathsf{Hyb}_{0,x,1}$ *is negligibly close.*

Proof Assume for the sake of contradiction that the statement of the lemma is not true. We construct an adversary $\mathcal{B}$ against the pseudorandomness at constrained prefix property of PCPRF.

$\mathcal{B}$ works exactly as the simulator in Hyb_x except that:

1. It does not sample S.
2. It queries the external challenger with the prefix C^*. In return, it receives a challenge string y which is either S_{C^*} or a random string as well as Keys which contains $\{S_{C^*_{[i-1]}\|(1-C^*_i)}\}_{i\in[\ell]}$.
3. It uses Keys to generate all encryptions in $\{\pi_i^{q+1}\}_{i\in[\ell]}$. Notice that Keys contains all the information needed to generate encryptions in $\{\pi_i^{q+1}\}_{i\in[\ell]}$.
4. It sets $\pi_{\ell+1}^{q+1} = (C_1, s_1), \cdots, (C^*, y), \cdots, (C_{q+1}, s_{q+1})$.
5. It finally outputs whatever adversary outputs.

If $y = S_{C^*}$ then the view generated by $\mathcal{B}$ is identical to view generated in Hyb_1. Else, it is identical to the view generated in Hyb_2. Thus, $\mathcal{B}$ breaks the pseudorandomness at constrained prefix property of PCPRF.

■

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

4 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.

4.1 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.
- $\mathsf{KeyGen}(\{\mathsf{pv}_j\}_{j\in S}, \mathsf{sv}_i, i)$ 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 4.1 *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 without using complexity leveraging.

4.2 Construction

Our construction will use the universal samplers built in Section 3, 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 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 4.2 *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 4.3 *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.

4.3 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 first, 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 4.4 *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.*

We give the proof of this Theorem in Appendix B.

5 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 5.1 ([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} with the associated (probabilistic) $(\mathsf{KeyGen}, \mathsf{SampGen})$ algorithms is a (standard) trapdoor permutation if it satisfies:

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 5.

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

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

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

(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**, $x \leftarrow \mathsf{Samp}$.

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

Figure 5: $\mathsf{Exp}_{\mathcal{A},b,\mathsf{PRS}}$

5.1 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 5.2 *Assuming the existence of one-way permutations and indistinguishablity obfuscation against polytime adversaries there exists a trapdoor permutation satisfying Definition 5.1.*

Our Construction. Our construction uses the following primitives:

1. An indistinguishability Obfuscator $i\mathcal{O}$.
2. A puncturable pseudorandom function $\mathcal{PRF} = (\mathsf{KeyGen}_{\mathcal{PRF}}, \mathsf{PRF}, \mathsf{Punc})$.
3. A length doubling pseudorandom generator $\mathsf{PRG} : \{0,1\}^{\kappa/2} \to \{0,1\}^\kappa$.
4. Additionally, in the proof of security, we use a length doubling injective pseudorandom generator $\mathsf{InjPRG} : [2^{\kappa/4}] \to [2^{\kappa/2}]$.

The formal description of our construction appears in Figure 6.

5.2 Security

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.

- $\mathsf{KeyGen}(1^\kappa)$:
  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 7 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 7.
- $\mathsf{SampGen}(SK)$: 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 7.
- Samp: Run the circuit $i\mathcal{O}(X_{S_1,\cdots,S_\kappa})$ on the given randomness r.

Figure 6: Construction of Trapdoor Permutation

Circuit F^*. We denote by $F^*_{S_1,\cdots,S_\kappa,\alpha,\beta}$ (with $\beta \geq \alpha$) 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 $\perp$. The formal description of the circuit is given in Figure 8. This notation would be used in our hybrids.

Notation. In the following proof, we denote $\mathsf{Adv}(\mathsf{Hyb}_i)$ to be the probability that adversary inverts the challenge in Hyb_i.

Our Hybrids. We now describe our hybrids.

- 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 $i = \mathsf{PRG}(r)$ and $r \xleftarrow{\$} \{0,1\}^{\kappa/2}$.
- Hyb_1: Instead of setting $i = \mathsf{PRG}(r)$, we sample $i \xleftarrow{\$} \{0,1\}^\kappa$. The following claim follows directly from pseudorandomness property of PRG.

  Lemma 5.3 *Assuming the pseudorandomness property of* PRG*, we have* $|\mathsf{Adv}(\mathsf{Hyb}_0) - \mathsf{Adv}(\mathsf{Hyb}_1)| \leq \mathsf{negl}(\kappa)$.

  Proof Assume that the statement of the lemma is not true. We construct an adversary $\mathcal{B}$ that breaks the pseudorandomness property.

  $\mathcal{B}$ receives a challenge y from the external challenger and constructs $i\mathcal{O}(X_{S_1,\cdots,S_\kappa}), i\mathcal{O}(F_{S_1,\cdots,S_\kappa})$ by first sampling $S_1, \cdots, S_\kappa$ exactly as in Hyb_0. It computes $\sigma_i = \mathsf{PRF}_{S_i}(y)$ and runs the adversary with input $((y, \sigma_1, \cdots, \sigma_\kappa), i\mathcal{O}(X_{S_1,\cdots,S_\kappa}), i\mathcal{O}(F_{S_1,\cdots,S_\kappa}))$. If the adversary inverts the challenge then $\mathcal{B}$ outputs 1; else it outputs 0.

  If y is a random string then the distribution of adversary's input is identical to Hyb_1. Else, it is distributed identical to Hyb_0. Thus, $\mathcal{B}$ breaks the pseudorandomness property of PRG. ■

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

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.

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

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 7: Public Key, Sampler and the Inverter for the Trapdoor permutations

- Hyb_2: In this hybrid, we change the public key of the permutation. The public key of the

$F^*_{S_1,\cdots,S_\kappa,\alpha,\beta}$

Input: $(x, \sigma_1, \cdots, \sigma_\kappa)$
Constants: $S_1, \cdots, S_\kappa, \alpha, \beta$

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

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

Figure 8: $F^*_{S_1,\cdots,S_\kappa,\alpha,\beta}$

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 $i - 2^{\frac{\kappa}{4}} \leq x \leq i-1$ and $\mathsf{InjPRG}(i-x) = v$, it outputs $\perp$. The formal description of $F^1_{S_1,\cdots,S_\kappa,i,v}$ is described in Figure 9.

Lemma 5.4 *Assuming the security of $i\mathcal{O}$, we have* $|\mathsf{Adv}(\mathsf{Hyb}_1) - \mathsf{Adv}(\mathsf{Hyb}_2)| \leq \mathsf{negl}(\kappa)$.

Proof Assume that the statement of the lemma is not true. We construct an adversary $\mathcal{B}$ that breaks the security of $i\mathcal{O}$.

$\mathcal{B}$ samples $\mathcal{PRF}$ keys $S_1, \cdots, S_\kappa$. It computes $i\mathcal{O}(X_{S_1,\cdots,S_\kappa})$ exactly as in Hyb_1. It constructs two circuits $F^1_{S_1,\cdots,S_\kappa,i,v}$ and $F_{S_1,\cdots,S_\kappa}$ described in Figure 9 and Figure 7 respectively. It gives them to the external challenger as the challenge circuits and receives $i\mathcal{O}(C^*)$ where C^* is either $F^1_{S_1,\cdots,S_\kappa,i,v}$ or $F_{S_1,\cdots,S_\kappa}$. It samples $i \xleftarrow{\$} \{0,1\}^\kappa$ and constructs $\sigma_1, \cdots, \sigma_\kappa$ as in Hyb_1. It runs the adversary with $((i, \sigma_1, \cdots, \sigma_\kappa), i\mathcal{O}(C^*), i\mathcal{O}(X_{S_1,\cdots,S_\kappa}))$. If adversary inverts the challenge, $\mathcal{B}$ outputs 1; else outputs 0.

Since v is chosen randomly from $[2^{\kappa/2}]$ it is not in the image of the InjPRG with overwhelming probability (actually with probability $1 - \frac{1}{2^{\frac{\kappa}{4}}}$). Thus, the check in Step 3 does not pass for any $x \in [i - 2^{\kappa/4}, i-1]$ with overwhelming probability. Hence, with overwhelming probability $F^1_{S_1,\cdots,S_\kappa,i,v}$ and $F_{S_1,\cdots,S_\kappa}$ compute the exact same functionality. Thus, $\mathcal{B}$ represents a valid adversary against $i\mathcal{O}$ security.

Notice that if $C^* = F^1_{S_1,\cdots,S_\kappa,i,v}$, the inputs to the adversary is distributed identically to the inputs in Hyb_2; else it is distributed identically to the inputs in Hyb_1. Thus, $\mathcal{B}$ breaks the security of $i\mathcal{O}$. ■

$F^1_{S_1,\cdots,S_\kappa,i,v}$

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

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

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

Figure 9: $F^1_{S_1,\cdots,S_\kappa,i,v}$

- 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 $\mathsf{InjPRG}(u_0)$ where $u_0 \xleftarrow{\$} [2^{\kappa/4}]$. The public key of the permutation would now correspond to $i\mathcal{O}(F_{S_1,\cdots,S_\kappa,i,\mathsf{PRG}(u_0)})$.

  Lemma 5.5 *Assuming the pseudorandomness property of* InjPRG*, we have* $|\mathsf{Adv}(\mathsf{Hyb}_2) - \mathsf{Adv}(\mathsf{Hyb}_3)| \leq \mathsf{negl}(\kappa)$.

  Proof We construct an adversary $\mathcal{B}$ against the pseudorandomness property of InjPRG. $\mathcal{B}$ receives a challenge string y from the external challenger and sets $v = y$. It samples the rest of the inputs to the adversary as in Hyb_2. Notice that if y is a random string then the inputs to the adversary are distributed identically to inputs in Hyb_2; else it is distributed identically to inputs in Hyb_3. Thus, $\mathcal{B}$ breaks the pseudorandomness property of InjPRG. ■

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

  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 $i\mathcal{O}(F^*_{S_1,\cdots,S_\kappa,\alpha_0,\alpha_0})$. The only difference between Hyb_3 and Hyb_4 is that in Hyb_3 the public key is generated as $i\mathcal{O}(F^1_{S_1,\cdots,S_\kappa,i,\mathsf{InjPRG}(u_0)})$ whereas in Hyb_4 it is generated as $i\mathcal{O}(F^*_{S_1,\cdots,S_\kappa,\alpha_0,\alpha_0})$. We have the following lemma from the security of $i\mathcal{O}$.

  Lemma 5.6 *Assuming the security of* $i\mathcal{O}$*, we have* $|\mathsf{Adv}(\mathsf{Hyb}_3) - \mathsf{Adv}(\mathsf{Hyb}_4)| \leq \mathsf{negl}(\kappa)$.

  Proof We construct an adversary $\mathcal{B}$ against the security of $i\mathcal{O}$. $\mathcal{B}$ samples $S_1, \cdots, S_\kappa$ and constructs $i\mathcal{O}(X_{S_1,\cdots,S_\kappa})$ as in Hyb_3. It constructs two circuits $F^1_{S_1,\cdots,S_\kappa,i,\mathsf{InjPRG}(u_0)}$ and

$F^*_{S_1,\cdots,S_\kappa,\alpha_0,\alpha_0}$ (padded to length $p(\kappa)$) and gives it as challenge circuits to the external challenger. It receives $i\mathcal{O}(C^*)$ from the external challenger where C^* is either $F^1_{S_1,\cdots,S_\kappa,i,\mathsf{InjPRG}(u_0)}$ or $F^*_{S_1,\cdots,S_\kappa,\alpha_0,\alpha_0}$. It samples $i \xleftarrow{\$} \{0,1\}^\kappa$ and constructs the inversion challenge exactly as in Hyb_4. It then runs the adversary with the inversion challenge, $i\mathcal{O}(C^*)$ and $i\mathcal{O}(X_{S_1,\cdots,S_\kappa})$ as inputs. $\mathcal{B}$ outputs 1 if and only if the adversary inverts the challenge. Observe that $F^1_{S_1,\cdots,S_\kappa,i,\mathsf{InjPRG}(u_0)}$ 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 InjPRG. In particular, both these functions output $\perp$ on any input of the form $(\alpha_0,\cdot,\cdots,\cdot)$. Thus, $\mathcal{B}$ represents a valid adversary against $i\mathcal{O}$ security and breaks $i\mathcal{O}$ security.

If $C^* = F^*_{S_1,\cdots,S_\kappa,\alpha_0,\alpha_0}$ then the inputs to the adversary are distributed identically to inputs in Hyb_4; else it is distributed identically to inputs in Hyb_3. Thus, $\mathcal{B}$ breaks $i\mathcal{O}$ security. ■

- $\mathsf{Hyb}_{5,j}$: 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. We define the following functions. We assume that all functions take i as an implicit input.

  1. 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 $\alpha = \alpha_{[j]}||1^{\kappa-j}$. For example, if $\alpha = \overbrace{100}^{3}11$ then $f(\alpha) = 3$.
  2. Let $\ell(\beta,\gamma)$ denotes the unique $j \in [\kappa]$ 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. In other words, $\ell(\beta,\gamma)$ denotes the first index at which β and δ differ if $\beta \neq \delta$ and is equal to the special symbol ζ otherwise.
  3. 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. For example, if $\alpha = 0010$ and $i-1 = 1010$, $\ell(\alpha,i-1) = 1$ since the two strings differ in the first position. Then, $\delta(\alpha) = 2$ since there are two zeroes in positions $[2,4]$ in α.
  4. Let $\rho(\alpha) = \ell(\alpha+1,i-1)$ if $\ell(\alpha+1,i-1) \neq \zeta$ and equal to κ otherwise.
  5. 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 $j \in [0,\delta(\alpha_0)+\mu(\alpha_{\delta(\alpha_0)})-1]$,

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

  Intuition behind the definition of α_j's. Let us assume for the sake of simplicity that $\alpha_0 < i-1$. Let $i^* = \ell(\alpha_0,i-1)$. Recall that $\ell(\alpha_0,i-1)$ gives the first index where α_0 and $i-1$ differ. Since $\alpha_0 < i-1$ we have $(\alpha_0)_{i^*} = 0$ and $(i-1)_{i^*} = 1$. Recall the multiple chains techinque from [GPS16]. For all $j \leq \delta(\alpha_0)$, we cut the chain of maximum possible length. Note that by this definition $\alpha_{\delta(\alpha_0)} = (i-1)_{[\ell(\alpha_0,i-1)-1]}||(\alpha_0)_{i^*}||1^{\kappa-\ell(\alpha_0,i-1)}$ where $(\alpha_0)_{i^*} = 0$. Once

we reach $\alpha_{\delta(\alpha_0)}$, if we cut the chain of maximum possible length we overshoot $i-1$. Thus, we start cutting chains of smaller lengths until we reach $i-1$. Notice that $\alpha_{\delta(\alpha_0)+\mu(\alpha_{\delta(\alpha_0)})} = i-1$. Illustrations of α_js are given in Figure 10,11.

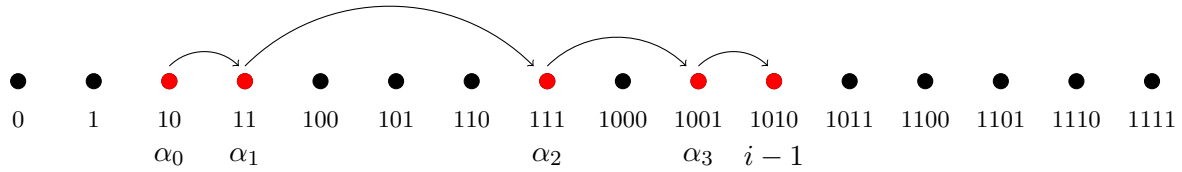

Figure 10: Illustration of the steps starting with $\alpha_0 = 0010$ and $i-1 = 1010$.

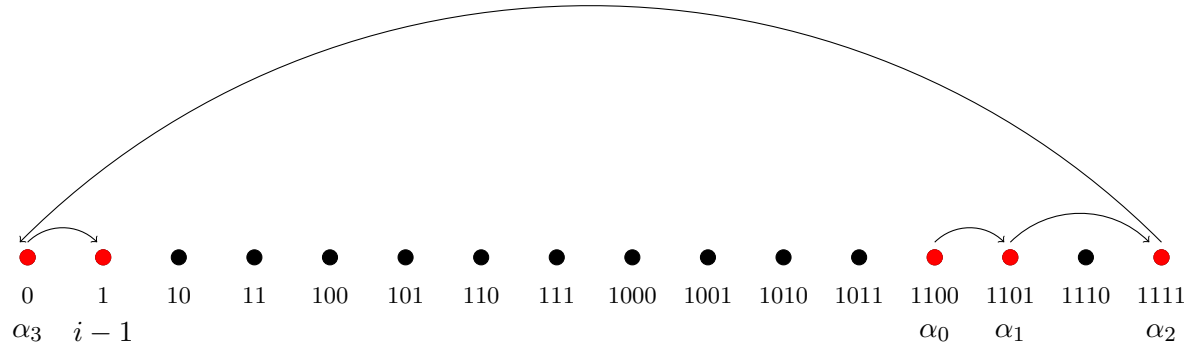

Figure 11: Illustration of the steps starting with $\alpha_0 = 1100$ and $i-1 = 0001$.

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 $\mathsf{Adv}(\mathsf{Hyb}_{5,j})$ is negligibly close to $\mathsf{Adv}(\mathsf{Hyb}_{5,j+1})$ 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 $f(\alpha_j)$ for $j < \delta(\alpha_0)$ and equal to $\rho(\alpha_j)$ for $j \geq \delta(\alpha_0)$. Let $t_j = (\alpha_j)_{[\nu_j]} + 1$. Note that by definition if for any $x \in \{0,1\}^\kappa$, $x_{[\nu_j]} = t_j$ then $\alpha_j + 1 \leq x \leq \alpha_{j+1}$.

- $\mathsf{Hyb}_{5,j,1}$: 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 $i\mathcal{O}(F^2_{S_1,\cdots,S'_{\nu_j},\cdots S_\kappa,\alpha_0,\alpha_j,\sigma^\star}, t_j)$. The function $F^2_{S_1,\cdots,S'_{\nu_j},\cdots S_\kappa,\alpha_0,\alpha_j,\sigma^\star,t_j}$ (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 $\mathsf{PRF}_{S_{\nu_j}}(t_j)$. We give the formal description of the circuit $F^2_{S_1,\cdots,S'_{\nu_j},\cdots S_\kappa,\alpha_0,\alpha_j,\sigma^\star}$ in Figure 12.

  Lemma 5.7 *Assuming the security of $i\mathcal{O}$, we have* $|\mathsf{Adv}(\mathsf{Hyb}_{5,j}) - \mathsf{Adv}(\mathsf{Hyb}_{5,j,1})| \leq \mathsf{negl}(\kappa)$.

$F^2_{S_1,\cdots,S'_{\nu_j},\cdots S_\kappa,\alpha_0,\alpha_j,\sigma^\star}$

Input: $(x, \sigma_1, \cdots, \sigma_\kappa)$

Constants: $S_1, \cdots, S'_{\nu_j}, \cdots, S_\kappa, \alpha_0, \alpha_j, \sigma^\star, t_j$

1. For all $k \in [\kappa] \setminus \{\nu_j\}$, check if $\sigma_k = \mathsf{PRF}_{S_k}(x_{[k]})$.
2. If $x_{[\nu_j]} = t_j$, check if $\sigma_{\nu_j} = \sigma^\star$. Else, check if $\sigma_{\nu_j} = \mathsf{PRF}_{S'_{\nu_j}}(x_{[\nu_j]})$
3. If any of the above checks fail, output $\perp$.
4. If $\alpha_0 \leq x \leq \alpha_j$, then output $\perp$.
5. Else, for all $k \in [\kappa] \setminus \{\nu_j\}$ compute $\sigma'_k = \mathsf{PRF}_{S_j}((x+1)_{[k]})$ where $x+1$ is computed modulo 2^κ.
6. If $(x+1)_{[\nu_j]} = t_j$, set $\sigma'_{\nu_j} = \sigma_{\nu_j}$. Else, set $\sigma'_{\nu_j} = \mathsf{PRF}_{S'_{\nu_j}}((x+1)_{[j]})$
7. Output $(x+1, \sigma'_1, \cdots, \sigma'_\kappa)$.

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

Figure 12: $F^2_{S_1,\cdots,S'_{\nu_j},\cdots S_\kappa,\alpha_0,\alpha_j,\sigma^\star,t_j}$

Proof We construct an adversary $\mathcal{B}$ against the security of $i\mathcal{O}$. $\mathcal{B}$ samples $S_1, \cdots, S_\kappa$ as in $\mathsf{Hyb}_{5,j}$. It samples $i \xleftarrow{\$} \{0,1\}^\kappa$. It punctures S_{ν_j} at the string t_j to obtain S'_{ν_j}. It constructs two circuits $F^2_{S_1,\cdots,S'_{\nu_j},\cdots S_\kappa,\alpha_0,\alpha_j,\sigma^\star,t_j}$ (padded to length $p(\kappa)$) and $F^*_{S_1,\cdots,S_{\nu_j},\cdots,S_\kappa,\alpha_0,\alpha_j}$ and provides them as the challenge circuits to the external challenger. In return, it obtains $i\mathcal{O}(C^*)$ where C^* is either one of the two circuits. $\mathcal{B}$ provides $i\mathcal{O}(C^*)$ as the public key to the adversary and generates other inputs identical to $\mathsf{Hyb}_{5,j}$. It outputs 1 if the adversary inverts the challenge; else it outputs 0.

We observe that the two circuits compute the exact same functionality which follows from setting $\sigma^\star = \mathsf{PRF}_{S_{\nu_j}}(t_j)$. Note that if the $x \in [\alpha_j + 1, \alpha_{j+1} - 1]$ (or in other words $(x+1)_{[\nu_j]} = t_j$), 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 $F^2_{S_1,\cdots,S'_{\nu_j},\cdots S_\kappa,\alpha_0,\alpha_j,\sigma^\star,t_j}$ outputs the input σ_{ν_j} in place of $\sigma^\star$. Hence, $\mathcal{B}$ represents a valid adversary against the security of $i\mathcal{O}$.

If $C^* = F^2_{S_1,\cdots,S'_{\nu_j},\cdots S_\kappa,\alpha_0,\alpha_j,\sigma^\star,t_j}$ then the distribution of inputs to $\mathcal{A}$ is identically distributed to the inputs in $\mathsf{Hyb}_{5,j,1}$; else the inputs are distributed identically to $\mathsf{Hyb}_{5,j}$. Thus, $\mathcal{B}$ breaks the security of $i\mathcal{O}$. ■

– $\mathsf{Hyb}_{5,j,2}$: In this hybrid we replace the sampler $i\mathcal{O}(X_{S_1,\cdots,S_\kappa})$ with $i\mathcal{O}(X^2_{S_1,\cdots,S'_{\nu_j},\cdots,S_\kappa,t_j})$. $X^2_{S_1,\cdots,S'_{\nu_j},\cdots,S_\kappa}$ (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 describe the sampler in Figure 13.

We now show the following claim.

$X^2_{S_1,\cdots,S'_{\nu_j},\cdots,S_\kappa}$

Input: $r \in \{0,1\}^{\kappa/2}$
Constants: $S_1, \cdots, S'_{\nu_j}, \cdots, S_{\kappa,t_j}$

1. Compute $x = \mathsf{PRG}(r)$.
2. If $x_{[\nu_j]} = t_j$, output $\perp$.
3. For every $j \in [\kappa] \setminus \{\nu_j\}$, compute $\sigma_j = \mathsf{PRF}_{S_j}(x_{[j]})$; compute $\sigma_{\nu_j} = \mathsf{PRF}_{S'_{\nu_j}}(x_{[\nu_j]})$.
4. Output $(x, \sigma_1, \sigma_2, \cdots, \sigma_\kappa)$.

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

Figure 13: $X^2_{S_1,\cdots,S'_{\nu_j},\cdots,S_\kappa,t_j}$

Lemma 5.8 *Assuming the security of $i\mathcal{O}$, we have* $|\mathsf{Adv}(\mathsf{Hyb}_{5,j,1}) - \mathsf{Adv}(\mathsf{Hyb}_{5,j,2})| \leq \mathsf{negl}(\kappa)$.

Proof We construct an adversary $\mathcal{B}$ against the security of $i\mathcal{O}$. $\mathcal{B}$ samples $S_1, \cdots, S_\kappa$ as in $\mathsf{Hyb}_{5,j}$. It samples $i \xleftarrow{\$} \{0,1\}^\kappa$. It punctures S_{ν_j} at the string t_j to obtain S'_{ν_j}. It constructs two circuits $(X^2_{S_1,\cdots,S'_{\nu_j},\cdots,S_\kappa})$ and $X^2_{S_1,\cdots,S'_{\nu_j},\cdots,S_\kappa}$ (padded to length $q(\kappa)$) and provides them as challenge circuits to the external challenger. In return, it obtains $i\mathcal{O}(C^*)$ where C^* is either one of the two circuits. $\mathcal{B}$ provides $i\mathcal{O}(C^*)$ as the sampler to the adversary and generates other inputs identical to $\mathsf{Hyb}_{5,j,1}$. It outputs 1 if the adversary inverts the challenge; else it outputs 0.
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 and u_0 is chosen independent of i. In particular, $c - u_0$ 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. Thus, $\mathcal{B}$ represents a valid adversary against $i\mathcal{O}$ security.
If $C^* = X^2_{S_1,\cdots,S'_{\nu_j},\cdots,S_\kappa}$ then the distribution of inputs to the adversary is identical to inputs in $\mathsf{Hyb}_{5,j,2}$; else the inputs are distributed identically to $\mathsf{Hyb}_{5,j,3}$. Thus, $\mathcal{B}$ breaks the $i\mathcal{O}$ security. ■

– $\mathsf{Hyb}_{5,j,3}$: 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 $\sigma^\star = \mathsf{PRF}_{S_{\nu_j}}(t_j)$.

$F^3_{S_1,\cdots,S'_{\nu_j},\cdots S_\kappa,\alpha_0,\alpha_j,\mathsf{InjPRG}(\sigma^\star),t_j}$

Input: $(x, \sigma_1, \cdots, \sigma_\kappa)$

Constants: $S_1, \cdots, S'_{\nu_j}, \cdots, S_\kappa, \alpha_0, \alpha_j, \mathsf{InjPRG}(\sigma^\star), t_j$

1. For all $k \in [\kappa] \setminus \{\nu_j\}$, check if $\sigma_k = \mathsf{PRF}_{S_k}(x_{[k]})$.
2. If $x_{[\nu_j]} = t_j$, check if $\mathsf{InjPRG}(\sigma_{\nu_j}) = \mathsf{InjPRG}(\sigma^\star)$. Else, check if $\sigma_{\nu_j} = \mathsf{PRF}_{S'_{\nu_j}}(x_{[\nu_j]})$
3. If any of the above checks fail, output $\perp$.
4. If $\alpha_0 \leq x \leq \alpha_j$, then output $\perp$.
5. Else, for all $k \in [\kappa] \setminus \{\nu_j\}$ compute $\sigma'_k = \mathsf{PRF}_{S_j}((x+1)_{[k]})$ where $x+1$ is computed modulo 2^κ.
6. If $(x+1)_{[\nu_j]} = t_j$, set $\sigma'_{\nu_j} = \sigma_{\nu_j}$. Else, set $\sigma'_{\nu_j} = \mathsf{PRF}_{S'_{\nu_j}}((x+1)_{[j]})$
7. 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.

Figure 14: $F^3_{S_1,\cdots,S'_{\nu_j},\cdots S_\kappa,\alpha_0,\alpha_j,\mathsf{InjPRG}(\sigma^\star),t_j}$

Lemma 5.9 *Assuming the pseudorandomness at punctured point property of $\mathcal{PRF}$, we have* $|\mathsf{Adv}(\mathsf{Hyb}_{5,j,2}) - \mathsf{Adv}(\mathsf{Hyb}_{5,j,3})| \leq \mathsf{negl}(\kappa)$.

Proof We construct an $\mathcal{B}$ against pseudorandomness at punctured point property of $\mathcal{PRF}$. $\mathcal{B}$ chooses $i \xleftarrow{\$} \{0,1\}^\kappa$. It samples $S_1, \cdots, S_{\nu_j - 1}, S_{\nu_j+1}, \cdots, S_\kappa$. It queries with the external challenger on string t_j and obtains S'_{ν_j} which is a $\mathcal{PRF}$ key punctured at t_j and a challenge string y which is either $\mathsf{PRF}_{S_{\nu_j}}(t_j)$ or an uniformly chosen random string. It sets $\sigma^\star = y$ and constructs $i\mathcal{O}(X^2_{S_1,\cdots,S'_{\nu_j},\cdots,S_\kappa})$ and $i\mathcal{O}(F^2_{S_1,\cdots,S'_{\nu_j},\cdots S_\kappa,\alpha_0,\alpha_j,\sigma^\star,t_j})$. It runs the adversary with the above constructed public key and sampler along with the trapdoor challenge $(i, \sigma_1, \cdots, \sigma_\kappa)$ where $\sigma_k = \mathsf{PRF}_{S_k}(i_{[k]})$ for all $k \in [\kappa] \setminus \{\nu_j\}$ and $\sigma_{\nu_j} = \mathsf{PRF}_{S'_{\nu_j}}(i_{[\nu_j]})$. Notice that S'_{ν_j} can be uses to evaluate the $\mathcal{PRF}$ on $i_{[\nu_j]}$. If the adversary inverts the trapdoor challenge, $\mathcal{B}$ outputs 1; else outputs 0.

Notice that if the challenge string y is randomly chosen then the distribution of inputs is identically distributed to the inputs in $\mathsf{Hyb}_{5,j,3}$; else it is distributed identically to the inputs in $\mathsf{Hyb}_{5,j,2}$. Thus, $\mathcal{B}$ breaks pseudorandomness at punctured point property of $\mathcal{PRF}$. ■

– $\mathsf{Hyb}_{5,j,4}$: In this hybrid we replace the public key of the permutation with $i\mathcal{O}(F^3_{S_1,\cdots,S'_{\nu_j},\cdots S_\kappa,\alpha_0,\alpha_j,\mathsf{InjPRG}(\sigma^\star)})$. The circuit $F^3_{S_1,\cdots,S'_{\nu_j},\cdots S_\kappa,\alpha_0,\alpha_j,\mathsf{InjPRG}(\sigma^\star),t_j}$ (padded to length $p(\kappa)$) is similar to that of $F^2_{S_1,\cdots,S'_{\nu_j},\cdots S_\kappa,\alpha_0,\alpha_j,\sigma^\star,t_j}$ 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 $\mathsf{InjPRG}(\sigma_{\nu_j}) = \mathsf{InjPRG}(\sigma^\star)$ instead of checking $\sigma_{\nu_j} = \sigma^\star$. The formal description of $F^3_{S_1,\cdots,S'_{\nu_j},\cdots S_\kappa,\alpha_0,\alpha_j,\mathsf{InjPRG}(\sigma^\star),t_j}$ appears in Figure 14.

The circuits $F^3_{S_1,\cdots,S'_{\nu_j},\cdots S_\kappa,\alpha_0,\alpha_j,\mathsf{InjPRG}(\sigma^\star),t_j}$ and $F^2_{S_1,\cdots,S'_{\nu_j},\cdots S_\kappa,\alpha_0,\alpha_j,\sigma^\star,t_j}$ compute the exact

same functionality which follows from the injectivity of InjPRG. Hence, the following claim directly follows from $i\mathcal{O}$ security.

Lemma 5.10 *Assuming the security of $i\mathcal{O}$, we have* $|\mathsf{Adv}(\mathsf{Hyb}_{5,j,3}) - \mathsf{Adv}(\mathsf{Hyb}_{5,j,4})| \leq \mathsf{negl}(\kappa)$.

Proof We construct an adversary $\mathcal{B}$ against the security of $i\mathcal{O}$. $\mathcal{B}$ samples $S_1, \cdots, S_\kappa$ as in $\mathsf{Hyb}_{5,j}$. It samples $i \xleftarrow{\$} \{0,1\}^\kappa$. It punctures S_{ν_j} at the string t_j to obtain S'_{ν_j}. It constructs two circuits $F^3_{S_1,\cdots,S'_{\nu_j},\cdots S_\kappa,\alpha_0,\alpha_j,\mathsf{InjPRG}(\sigma^\star),t_j}$ and $F^2_{S_1,\cdots,S'_{\nu_j},\cdots S_\kappa,\alpha_0,\alpha_j,\sigma^\star,t_j}$ and provides them as challenge circuits to the external challenger. In return, it obtains $i\mathcal{O}(C^*)$ where C^* is either one of the two circuits. $\mathcal{B}$ provides $i\mathcal{O}(C^*)$ as the public to the adversary and generates other inputs identical to $\mathsf{Hyb}_{5,j,3}$. It outputs 1 if the adversary inverts the challenge; else it outputs 0.

As argued above the two challenge circuits are functionally equivalent from the injectivity of InjPRG. Thus, $\mathcal{B}$ represents a valid adversary against $i\mathcal{O}$ security.

If $C^* = F^3_{S_1,\cdots,S'_{\nu_j},\cdots S_\kappa,\alpha_0,\alpha_j,\mathsf{InjPRG}(\sigma^\star),t_j}$ then the distribution of inputs to the adversary is identical to inputs in $\mathsf{Hyb}_{5,j,4}$; else the inputs are distributed identically to $\mathsf{Hyb}_{5,j,3}$. Thus, $\mathcal{B}$ breaks the $i\mathcal{O}$ security. ■

– $\mathsf{Hyb}_{5,j,5}$: In this hybrid, the public key of the permutation corresponds to $i\mathcal{O}(F^3_{S_1,\cdots,S'_{\nu_j},\cdots S_\kappa,\alpha_0,\alpha_j,\tau^\star,t_j})$ where we set $\tau^\star$ to be a string chosen uniformly at random from $\{0,1\}^{2\kappa}$ instead of setting $\tau^\star = \mathsf{InjPRG}(\sigma^\star)$.

Lemma 5.11 *Assuming the pseudorandomness property of* InjPRG*, we have* $|\mathsf{Adv}(\mathsf{Hyb}_{5,j,5}) - \mathsf{Adv}(\mathsf{Hyb}_{5,j,4})| \leq \mathsf{negl}(\kappa)$.

Proof We construct an adversary $\mathcal{B}$ against pseudorandomness property of InjPRG. $\mathcal{B}$ receives a challenge string $\tau^\star$ from the external challenger. It samples $i \xleftarrow{\$} \{0,1\}^\kappa$ and samples PRF keys $S_1, \cdots, S_\kappa \leftarrow \mathsf{KeyGen}_{\mathcal{PRF}}$. It punctures S_{ν_j} at the string t_j to obtain the punctured key S'_{ν_j}. It computes $i\mathcal{O}(F^3_{S_1,\cdots,S'_{\nu_j},\cdots S_\kappa,\alpha_0,\alpha_j,\tau^\star,t_j})$ and sets it as the public key of the permutation. It computes the sampler and the inversion challenge as in $\mathsf{Hyb}_{5,j,4}$ and runs the adversary with the challenge, sampler and the public key as input. If the adversary inverts the challenge, then it outputs 1. Else, it outputs 0.

If $\tau^\star$ is randomly chosen then the distribution of the inputs to the adversary is identical to the inputs in $\mathsf{Hyb}_{5,j,5}$; else the it is distributed identically to the inputs in $\mathsf{Hyb}_{5,j,4}$. Thus, $\mathcal{B}$ breaks the pseudorandomness property of InjPRG. ■

– $\mathsf{Hyb}_{5,j,6}$: In this hybrid the public key of the permutation corresponds to $i\mathcal{O}(F^4_{S_1,\cdots,S'_{\nu_j},\cdots S_\kappa,\alpha_0,\alpha_j,\alpha_{j+1}})$ where $F^4_{S_1,\cdots,S'_{\nu_j},\cdots S_\kappa,\alpha_0,\alpha_j,\alpha_{j+1}}$ is described in Figure 15.

Lemma 5.12 *Assuming the security of $i\mathcal{O}$, we have* $|\mathsf{Adv}(\mathsf{Hyb}_{5,j,5}) - \mathsf{Adv}(\mathsf{Hyb}_{5,j,6})| \leq \mathsf{negl}(\kappa)$.

Proof We construct an adversary $\mathcal{B}$ against the security of $i\mathcal{O}$. $\mathcal{B}$ samples $S_1, \cdots, S_\kappa$ as in $\mathsf{Hyb}_{5,j}$. It samples $i \xleftarrow{\$} \{0,1\}^\kappa$ and $\tau^* \xleftarrow{\$} \{0,1\}^{2\kappa}$. It punctures S_{ν_j} at the string t_j to

$F^4_{S_1,\cdots,S'_{\nu_j},\cdots S_\kappa,\alpha_0,\alpha_j,\alpha_{j+1}}$

Input: $(x, \sigma_1, \cdots, \sigma_\kappa)$
Constants: $S_1, \cdots, S'_{\nu_j}, \cdots, S_\kappa, \alpha_0, \alpha_j, \alpha_{j+1}$

1. For all $k \in [\kappa] \setminus \{\nu_j\}$, check if $\sigma_k = \mathsf{PRF}_{S_k}(x_{[k]})$.
2. If $\alpha_j + 1 \leq x \leq \alpha_{j+1}$, output $\perp$. Else, check if $\sigma_{\nu_j} = \mathsf{PRF}_{S'_{\nu_j}}(x_{[\nu_j]})$
3. If any of the above checks fail, output $\perp$.
4. If $\alpha_0 \leq x \leq \alpha_j$, then output $\perp$.
5. Else, for all $k \in [\kappa] \setminus \{\nu_j\}$ compute $\sigma'_k = \mathsf{PRF}_{S_j}((x+1)_{[k]})$ where $x+1$ is computed modulo 2^κ.
6. If $(x+1)_{[\nu_j]} = t_j$, set $\sigma'_{\nu_j} = \sigma_{\nu_j}$. Else, set $\sigma'_{\nu_j} = \mathsf{PRF}_{S'_{\nu_j}}((x+1)_{[j]})$
7. 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.

Figure 15: $F^4_{S_1,\cdots,S'_{\nu_j},\cdots S_\kappa,\alpha_0,\alpha_j,\alpha_{j+1}}$

obtain S'_{ν_j}. It constructs two circuits $F^3_{S_1,\cdots,S'_{\nu_j},\cdots S_\kappa,\alpha_0,\alpha_j,\tau^*),t_j}$ and $F^4_{S_1,\cdots,S'_{\nu_j},\cdots S_\kappa,\alpha_0,\alpha_j,\alpha_{j+1}}$ and provides them as challenge circuits to the external challenger. In return, it obtains $i\mathcal{O}(C^*)$ where C^* is either one of the two circuits. $\mathcal{B}$ provides $i\mathcal{O}(C^*)$ as the public to the adversary and generates other inputs identical to $\mathsf{Hyb}_{5,j,5}$. It outputs 1 if the adversary inverts the challenge; else it outputs 0.

Observe that since τ^* is chosen uniformly at random from $\{0,1\}^{2\kappa}$, with overwhelming probability it is not in the image of InjPRG. Hence, for no value of σ_{ν_j} (with overwhelming probability) the test $\mathsf{PRG}(\sigma_{\nu_j}) = \tau^*$ passes. Thus the two challenge circuits compute the same functionality with overwhelming probability. Thus, $\mathcal{B}$ represents a valid adversary against $i\mathcal{O}$ security.

If $C^* = F^3_{S_1,\cdots,S'_{\nu_j},\cdots S_\kappa,\alpha_0,\alpha_j,\tau^*),t_j}$ then the distribution of inputs to the adversary is identical to the distribution in $\mathsf{Hyb}_{5,j,5}$; else the distribution is identical to inputs in $\mathsf{Hyb}_{5,j,6}$. Thus, $\mathcal{B}$ breaks the $i\mathcal{O}$ security. ■

– $\mathsf{Hyb}_{5,j,7}$: 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}.

Lemma 5.13 *Assuming the security of $i\mathcal{O}$, we have* $|\mathsf{Adv}(\mathsf{Hyb}_{5,j,6}) - \mathsf{Adv}(\mathsf{Hyb}_{5,j,7})| \leq \mathsf{negl}(\kappa)$.

Proof We construct an adversary $\mathcal{B}$ against the security of $i\mathcal{O}$. $\mathcal{B}$ samples $S_1, \cdots, S_\kappa$ as in $\mathsf{Hyb}_{5,j}$. It samples $i \xleftarrow{\$} \{0,1\}^\kappa$. It punctures S_{ν_j} at the string t_j to obtain S'_{ν_j}. It constructs two circuits $F^*_{S_1,\cdots,S_{\nu_j},\cdots S_\kappa,\alpha_0,\alpha_{j+1}}$ and $F^4_{S_1,\cdots,S'_{\nu_j},\cdots S_\kappa,\alpha_0,\alpha_j,\alpha_{j+1}}$ and provides them as challenge circuits to the external challenger. In return, it obtains $i\mathcal{O}(C^*)$ where

C^* is either one of the two circuits. $\mathcal{B}$ provides $i\mathcal{O}(C^*)$ as the public to the adversary and generates other inputs identical to $\mathsf{Hyb}_{5,j,5}$. It outputs 1 if the adversary inverts the challenge; else it outputs 0.

Note that both $F^*_{S_1,\cdots,S_{\nu_j},\cdots S_\kappa,\alpha_0,\alpha_{j+1}}$ (padded to length $p(\kappa)$) and $F^4_{S_1,\cdots,S'_{\nu_j},\cdots S_\kappa,\alpha_0,\alpha_j,\alpha_{j+1}}$ do not require the evaluation of $\mathsf{PRF}_{S_{\nu_j}}$ on t_j. Hence they are functionally equivalent from the functionality is preserved under puncturing property of $\mathcal{PRF}$. Thus, $\mathcal{B}$ represents a valid adversary against $i\mathcal{O}$ security.

If $C^* = F^4_{S_1,\cdots,S'_{\nu_j},\cdots S_\kappa,\alpha_0,\alpha_j,\alpha_{j+1}}$ then the distribution of inputs to the adversary is identical to the distribution in $\mathsf{Hyb}_{5,j,6}$; else the distribution is identical to inputs in $\mathsf{Hyb}_{5,j,7}$. Thus, $\mathcal{B}$ breaks the $i\mathcal{O}$ security. ■

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 $i\mathcal{O}(X^2_{S_1,\cdots,S'_{\nu_j},\cdots,S_\kappa})$.

Lemma 5.14 *Assuming the security of $i\mathcal{O}$, we have $|\mathsf{Adv}(\mathsf{Hyb}_{5,j,7}) - \mathsf{Adv}(\mathsf{Hyb}_{5,j,8})| \leq \mathsf{negl}(\kappa)$.*

Proof The proof of this Lemma is identical to Lemma 5.8. ■

Note that the distribution of inputs in $\mathsf{Hyb}_{5,j,8}$ is identical to the distribution in $\mathsf{Hyb}_{5,j+1}$.

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.

6 Trapdoor Permutation from FE

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

Definition 6.1 *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 weakly samplable trapdoor permutation if it satisfies:

- ***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.*
- ***Weak 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 $\mathcal{A}$,*

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

where $\mathsf{Exp}_{\mathcal{A},b,\mathsf{wPRS}}$ is described in Figure 16.

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**, $x \leftarrow \mathsf{Samp}$.
5. Output $\mathcal{A}(PK, \mathsf{Samp}, x)$

Figure 16: $\mathsf{Exp}_{\mathcal{A},b,\mathsf{wPRS}}$

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

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

Remark 6.2 *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 construct trapdoor permutation satisfying the Definition 6.1 from polynomially hard public key functional encryption, prefix puncturable pseudorandom function, left half injective pseudorandom generator, strong randomness extractor and public key encryption with random public keys.

Theorem 6.3 *Assuming the existence of one-way permutations, single-key, selective secure, public key functional encryption and public key encryption with (pseudo) random public keys, there exists a weakly samplable trapdoor permutation.*

We now recall the special key structure [GPS16] which forms a crucial part of our construction of trapdoor permutation.

Notation. We treat $1^i{+}1$ as 0^i and $\phi{+}1$ as ϕ. Let $\mathsf{LeftInjPRG}$ be a left half injective pseudorandom generator. Let τ be the size of public key output by $\mathsf{PK.KeyGen}(1^\kappa)$. Below, for every $i \in [\kappa + \tau]$, $S_i \leftarrow \mathsf{KeyGen}_{\mathcal{PCPRF}}(1^\kappa.C_i(\cdot), I(\cdot))$ where $C_i(\kappa) = i$ and $I(\kappa) = \kappa$. Recall $S_{i,x}$ denotes a prefix constrained PRF key S_i constrained at a prefix x.

Special Key Structure.

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

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

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

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

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

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

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

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 [GPS16]) 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.

We state and prove some properties of Special Key Structure below.

Lemma 6.4 (Computability Lemma,[GPS16]) *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 $|x| \geq i$. 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 $S_{i,x\|0} := \mathsf{PrefixCons}(S_{i,x}, 0)$ and $S_{i,x\|1} := \mathsf{PrefixCons}(S_{i,x}, 1)$ [8]. 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 constraining) $S_{i,x\|0}$, $S_{i,x\|1}$, $S_{i,(x\|0)+1} = S_{i,x\|1}$ 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 6.5 (Derivability Lemma,[GPS16]) *For every* $i \in [\kappa+\tau], x \in \{0,1\}^i$ *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* $x\|1^{\kappa+\tau-i}$ *or* $(x+1)\|1^{\kappa+\tau-i}$.

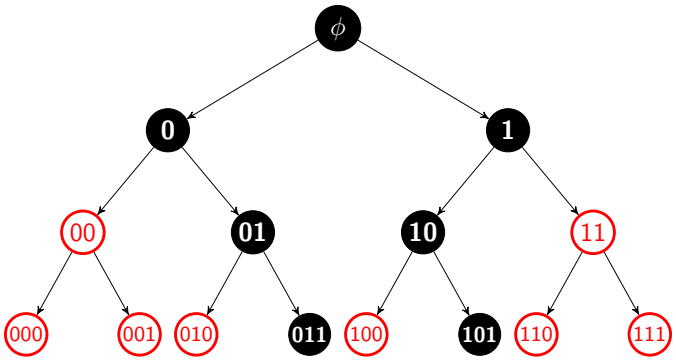

Figure 17: Black nodes represent the choices of $x \in \{0,1\}^{\leq 3}$ such that V_x^2 can be used to derive $S_{2,10}$.

Proof We start by noting that for any $y \in \{0,1\}^{>i} \cap \{0,1\}^{\leq \kappa+\tau}$, by definition of V-sets we have that $\mathsf{V}_y^i = \mathsf{V}_{y_{[i]}}^i$ (when $|y| > i$ and $y = y_{[i]} \| 1^{|y|-i}$) or $\mathsf{V}_y^i = \emptyset$ (when $|y| > i$ and $y \neq y_{[i]} \| 1^{|y|-i}$). 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_y^i. 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 $\mathsf{V}_y^i = \{S_{i,y}, S_{i,(y+1)\|0^{i-|y|}}\}$, 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 $\mathsf{V}_y^i = \{S_{i,y}, S_{i,(y+1)\|0^{i-|y|}}\}$, 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 $V_y^i = \{S_{i,y}, S_{i,(y+1)\|0^{i-|y|}}\}$. We will argue that neither $S_{i,y}$ nor $S_{i,(y+1)\|0^{i-|y|}}$ can be used to derive $S_{i,x+1}$.

- 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. The construction of weakly samplable trapdoor permutation uses the following primitives:

1. A single-key, selective secure public key functional encryption scheme $\mathcal{FE}$.
2. A prefix constrained pseudorandom function $\mathcal{PCPRF}$.

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

3. An injective length doubling pseudorandom generator $\mathsf{InjPRG} : \{0,1\}^{\kappa/8} \to \{0,1\}^{\kappa/4}$

4. A length doubling Left half injective pseudorandom generator $\mathsf{LeftInjPRG} : \{0,1\}^{\kappa} \to \{0,1\}^{2\kappa}$

In the construction, we denote $\mathsf{SK.Enc}_{sk_1,\cdots,sk_n}(m)$ to be $\mathsf{SK.Enc}_{sk_n}(\mathsf{SK.Enc}_{sk_{n-1}}(\cdots \mathsf{SK.Enc}_{sk_1}(m)))$. The formal description our construction appears in Figure 18.

Setting $\mathsf{rand}(\cdot)$ We set $\mathsf{rand}(\kappa)$ to be the maximum number of random bits needed to generate $\tau+\kappa$ encryptions under $\gamma_1, \cdots, \gamma_\kappa$ as well as $\tau+\kappa+1$ encryptions under the public keys pk.

6.1 Proof of Theorem 6.3

We show that our construction satisfies the three properties given in Definition 6.1.

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

We first observe that since $v \xleftarrow{\$} \{0,1\}^{\kappa/4}$, with probability $1 - \frac{1}{2^{\kappa/8}}$, v is not in the image of the InjPRG 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 infer that $\psi_i = \mathsf{LeftInjPRG}_0(S_{i,x_{[i]}})$ for every $i \in [\tau+\kappa]$. This follows from the correctness of $\mathcal{FE}$ scheme. Recall that in Step 3, evaluation function (i.e. TDP_{PK}) checks if $\mathsf{LeftInjPRG}_0(\sigma_i) = \psi_i$. It follows from the left half injectivity property of the $\mathsf{LeftInjPRG}$ 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 $(x, \sigma_1, \cdots, \sigma_{\tau+\kappa})$ 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, $x = x_{[i]} \| 1^{\tau+\kappa-i}$ 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 $\mathsf{SK.Dec}$, TDP_{PK} correctly obtains $S_{i,(x+1)_{[i]}}$ in Step 5 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, TDP_{PK} correctly computes the full set of associated signatures on $x+1$. Since on input $(x, \sigma_1, \cdots, \sigma_{\tau+\kappa})$ the function outputs $(x+1, \sigma'_1, \cdots, \sigma'_{\tau+\kappa})$ where $x+1$ is computed modulo $2^{\tau+\kappa}$, 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 $h_{pk} = (pk, \rho, \rho_1, \cdots, \rho_{\tau+\kappa})$ where $\rho = \mathsf{PK.Enc}_{pk}(K_{pk})$ and $\rho_i = \mathsf{PK.Enc}_{pk}(S_{i,(pk\|K_{pk})_{[i]}})$ for all $i \in [\tau+\kappa]$. Hence, from the correctness of decryption of public key encryption, the output of the sampler is a set of correct associated signatures on the prefixes of $pk\|K_{pk}$.

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

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

  1. For each $i \in [\tau+\kappa]$, sample $S_i \leftarrow \mathsf{KeyGen}_{\mathcal{PCPRF}}(1^\kappa, C_i(\cdot), I(\cdot))$ where $C_i(\kappa) = i$ and $I(\kappa) = \kappa$. Sample $\widetilde{K} \leftarrow \mathsf{KeyGen}_{\mathcal{PCPRF}}(1^\kappa, \mathsf{quad}(\cdot), \mathsf{rand}(\cdot))$ where $\mathsf{quad}(\kappa) = 2(\kappa+\tau)+1$. For every $i \in [\tau+\kappa]$, initialize $\mathsf{V}^i_\phi := S_i$, $\mathsf{V}_\phi = \bigcup_{i\in[\tau+\kappa]} \mathsf{V}^i_\phi$ and $\mathsf{W}_\phi = \emptyset$.
  2. Let $\mathsf{Ext}_w : \{0,1\}^{\tau+\kappa} \to \{0,1\}^{\kappa/8}$ be a $(\kappa/4, \mathsf{negl}(\kappa))$ strong randomness extractor with seed length $q(\kappa)$. Sample a seed $w \xleftarrow{\$} \{0,1\}^{q(\kappa)}$ for the extractor Ext.
  3. Sample $(\mathsf{PK}^1_i, \mathsf{MSK}^1_i) \leftarrow \mathsf{FE.Setup}(1^\kappa)$ for all $i \in [\tau+\kappa+1]$.
  4. Sample $sk_1 \leftarrow \mathsf{SK.KeyGen}(1^\kappa)$ where $|sk_1| = p(\kappa)$ 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 $i \in [\tau+\kappa]$, generate $\mathsf{FSK}^1_i \leftarrow \mathsf{FE.KeyGen}(\mathsf{MSK}^1_i, F^1_{i,\mathsf{PK}^1_{i+1},\Pi_1})$ and $\mathsf{FSK}^1_{\tau+\kappa+1} \leftarrow \mathsf{FE.KeyGen}(\mathsf{MSK}^1_{\tau+\kappa+1}, G^1_{v,\Lambda_1,w})$, where $F^1_{i,\mathsf{PK}^1_{i+1},\Pi_1}$ and $G^1_{v,\Lambda_1,w}$ are circuits described in Figure 19.
  7. Let $c^1_\phi = \mathsf{FE.Enc}_{\mathsf{PK}_1}(\phi, \mathsf{V}_\phi, \mathsf{W}_\phi, \widetilde{K}_\phi, 0^{p(\kappa)}, 0)$.
  8. The Public Key PK is given by $(\{\mathsf{FSK}^1_i\}_{i\in[\tau+\kappa+1]}, c^1_\phi)$ and the secret key SK is given by $(S_1, \cdots, S_{\tau+\kappa})$.

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

  1. For $i \in [\tau+\kappa]$, 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 $d_x = ((\psi_1, \ldots, \psi_{\tau+\kappa}), (\beta_j, \ldots, \beta_{\tau+\kappa}))$ as output of $\mathsf{FE.Dec}(\mathsf{FSK}^1_{\tau+\kappa+1}, c^1_x)$. Here, $j = f(x)$. Recall from Section 5.2 that $f(x)$ is the smallest k such that $x = x_{[k]}\|1^{\tau+\kappa-k}$.
  3. Output $\perp$ if $\mathsf{LeftInjPRG}_0(\sigma_i) \neq \psi_i$ for any $i \in [\tau+\kappa]$.
  4. For each $i \in [j-1]$, set $\sigma'_i = \sigma_i$.
  5. For each $i \in \{j, \ldots, \tau+\kappa\}$, set $\gamma_i = \mathsf{LeftInjPRG}_1(\sigma_i)$ and σ'_i as $\mathsf{SK.Dec}_{\gamma_j,\ldots,\gamma_{\tau+\kappa}}(\beta_i)$, iteratively decrypting β_i encrypted under $\gamma_j, \ldots, \gamma_{\tau+\kappa}$.
  6. Output $(x+1, \sigma'_1, \cdots, \sigma'_{\tau+\kappa})$.

Figure 18: Construction of TDP from $\mathcal{FE}$

Lemma 6.6 *$(pk\|K_{pk})$ where $(pk, sk) \leftarrow \mathsf{PK.KeyGen}(1^\kappa)$ and $(pk\|K_{pk}, \cdot, \cdots, \cdot) \leftarrow \mathsf{Samp}(pk)$ is pseudorandom given $(\{\mathsf{FSK}^2_i\}_{i\in[\kappa+1]}, c^2_\phi)$ [9] and $(\{\mathsf{FSK}^1_i\}_{i\in[\tau+\kappa+1]}, c^1_\phi)$ (which is equal to the public key).*

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

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

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

  1. Choose $\overline{K} \leftarrow \mathsf{KeyGen}_{\mathcal{PCPRF}}(1^\kappa, 2\upsilon(\cdot)+1, \mathsf{rand}(\cdot))$ and $K \leftarrow \mathsf{KeyGen}_{\mathcal{PCPRF}}(1^\kappa, \upsilon(\cdot), I(\cdot))$ where $\upsilon(\kappa) = \tau$. Initialize $\mathsf{U}^i_\phi := S_i$ and $\mathsf{U}_\phi = \bigcup_{i\in[\tau+\kappa]} \mathsf{U}^i_\phi$.
  2. For every $i \in [\tau+1]$, choose $(\mathsf{PK}^2_i, \mathsf{MSK}^2_i) \leftarrow \mathsf{FE.Setup}(1^\kappa)$.
  3. Sample $sk_2 \leftarrow \mathsf{SK.KeyGen}(1^\kappa)$ where $|sk_2| = p(\kappa)$ 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 $i \in [\tau]$, generate $\mathsf{FSK}^2_i \leftarrow \mathsf{FE.KeyGen}(\mathsf{MSK}^2_i, F^2_{i,\mathsf{PK}^2_{i+1},\Pi_2})$ and $\mathsf{FSK}^2_{\tau+1} \leftarrow \mathsf{FE.KeyGen}(\mathsf{MSK}^2_{\tau+1}, G^2_{\Lambda_2})$ where $F^2_{i,\mathsf{PK}^2_{i+1},\Pi_2}, G^2_{\Lambda_2}$ are described in Figure 20.
  5. Let $c^2_\phi \leftarrow \mathsf{FE.Enc}_{\mathsf{PK}^2_1}(\phi, U_\phi, K, \overline{K}, 0^{p(\kappa)}, 0)$.
  6. The sampler circuit has $\{\mathsf{FSK}^2_i\}_{i\in[\tau+1]}$ 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 $i \in [\tau]$, 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 $(pk, h_{pk}) = (pk, (pk, \rho, \rho_1, \cdots, \rho_{\tau+\kappa}))$ as output of $\mathsf{FE.Dec}(\mathsf{FSK}^2_{\tau+1}, c^2_{pk})$.
  3. Compute $K_{pk} := \mathsf{PK.Dec}_{sk}(\rho)$ and $\sigma_i := \mathsf{PK.Dec}_{sk}(\rho_i)$ for all $i \in [\tau+\kappa]$
  4. Output $(pk\|K_{pk}, \sigma_1, \cdots, \sigma_{\tau+\kappa})$.

Figure 18: Construction of TDP from $\mathcal{FE}$

Proof We will show this through a hybrid argument.

Notation. Let $\mathsf{Adv}(\mathsf{Hyb}_i)$ be the probability that adversary outputs 1 when given inputs generated as in Hyb_i.

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

  New $\pi^\star_2, \lambda^\star_2$: 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, $\pi^\star_2$ was set to $0^{\ell_2(\kappa)}$ and $\lambda^\star_2$ was set to $0^{\ell'_2(\kappa)}$.

  Let P_2 denote the set of all prefixes including the empty prefix of pk. Observe that $|P_2| = \tau+1$.

$F^1_{i,\mathsf{PK}^1_{i+1},\Pi_1}$

Hardcoded Values: $i, \mathsf{PK}^1_{i+1}, \Pi_1$.
Input: ($x \in \{0,1\}^{i-1}$, V_x,W_x, $\widetilde{K}_x$, sk, mode)

1. If ($\mathsf{mode} = 0$)
   (a) Output $\mathsf{FE.Enc}_{\mathsf{PK}^1_{i+1}}(x\|0, \mathsf{V}_{x\|0}, \mathsf{W}_{x\|0}, \widetilde{K}_{x\|0}, sk, \mathsf{mode}; \widetilde{K}'_{x\|0})$ and $\mathsf{FE.Enc}_{\mathsf{PK}^1_{i+1}}(x\|1, \mathsf{V}_{x\|1}, \mathsf{W}_{x\|1}, \widetilde{K}_{x\|1}, sk, \mathsf{mode}; \widetilde{K}'_{x\|1})$, where for $b \in \{0,1\}$, $\widetilde{K}_{x\|b} = \mathsf{PrefixCons}(\widetilde{K}_x, b\|0)$ and $\widetilde{K}'_{x\|b} = \mathsf{PrefixCons}(\widetilde{K}_x, b\|1)$ 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 6.4).
2. Else, compute $\pi_1 \leftarrow \mathsf{SK.Dec}_{sk_1}(\Pi_1)$ and parse π_1 as a set of tuples of the form (z, c^1_z). Recover $(x||0, c^1_{x\|0})$ and $(x\|1, c^1_{x\|1})$ from π_1. Output $c^1_{x\|0}$ and $c^1_{x\|1}$.

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

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

1. If ($\mathsf{InjPRG}(\mathsf{Ext}_w(x)) = v$) then output $\perp$.
2. If $\mathsf{mode} = 0$, (Below $j = f(x)$. Recall from Section 5.2 that $f(x)$ is the smallest j such that $x = x_{[j]}\|1^{\tau+\kappa-j}$.)
   (a) For each $i \in [\tau+\kappa]$, set $\psi_i = \mathsf{LeftInjPRG}_0(S_{i,x_{[i]}})$ (obtained from W^i_x for $i \le j$ and from V^i_x for $i > j$).
   (b) For each $i \in \{j, \ldots, \tau+\kappa\}$ set $\gamma_i = \mathsf{LeftInjPRG}_1(S_{i,x_{[i]}})$ and $\beta_i = \mathsf{SK.Enc}_{\gamma_j,\cdots,\gamma_{\tau+\kappa}}(S_{i,x_{[i]}+1})$, encrypting $S_{i,x_{[i]}+1}$ under $\gamma_j, \ldots \gamma_{\tau+\kappa}$ using $\mathsf{PrefixCons}(\widetilde{K}_x, 0)$ as the random tape. In the above, $S_{i,x_{[i]}}$ and $S_{i,x_{[i]}+1}$ are obtained from V^i_x for all $i \in [j, \tau+\kappa]$.
   (c) Output $((\psi_1, \ldots, \psi_{\tau+\kappa}), (\beta_j, \ldots, \beta_{\tau+\kappa}))$
3. Else, recover (x, d_x) from $\mathsf{SK.Dec}_{sk_1}(\Lambda_1)$ and output d_x.

Figure 19: Circuits for simulating Public Key.

For every string $x \in \{0,1\}^{\le\tau}$, let c^2_x, 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 $x \in \{0,1\}^\tau$). For every string $x \in P_2$, let y

$F^2_{i,\mathsf{PK}^2_{i+1},\Pi_2}$

Hardcoded Values: $i, \mathsf{PK}^2_{i+1}, \Pi_2$.
Input: $(x \in \{0,1\}^{i-1}, \mathsf{U}_x, K_x, \overline{K}_x, sk_2, \mathsf{mode})$

1. If $(\mathsf{mode} = 0)$,
   (a) Output $\mathsf{FE.Enc}_{\mathsf{PK}^2_{i+1}}(x\|0, \mathsf{U}_{x\|0}, K_{x\|0}, \overline{K}_{x\|0}, sk, \mathsf{mode}; K'_{x\|0})$ and $\mathsf{FE.Enc}_{\mathsf{PK}^2_{i+1}}(x\|1, \mathsf{U}_{x\|1}, K_{x\|1}, \overline{K}_{x\|1}, sk, \mathsf{mode}; K'_{x\|1})$, where for $b \in \{0,1\}$, $\overline{K}_{x\|b} = \mathsf{PrefixCons}(\overline{K}_x, b\|0)$ and $K'_{x\|b} = \mathsf{PrefixCons}(\overline{K}_x, b\|1)$ and $\mathsf{U}_{x\|0}$ and $\mathsf{U}_{x\|1}$ are computed as described in Computability Lemma (Lemma 6.4).
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: $pk \in \{0,1\}^\kappa, \mathsf{U}_{pk}, K_{pk}, \overline{K}_{pk}, sk_2, \mathsf{mode}$

1. If $\mathsf{mode} = 0$,
   (a) For all $i \in [\tau + \kappa]$, 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 $i \in [\tau + \kappa]$ using $\mathsf{PrefixCons}(\overline{K}_{pk}, 0)$ as the random tape.
   (c) Output $(pk, (pk, \rho, \rho_1, \cdots, \rho_{\tau+\kappa}))$.
2. Else, recover (pk, h_{pk}) from $\mathsf{SK.Dec}_{sk_2}(\Lambda_2)$ and output h_{pk}.

Figure 20: Circuits for simulating Sampler

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 $|P_2 \cup Q_2| = 2\tau + 1$. We define:

$$\pi^\star_2 = ||_{x \in P_2 \cup Q_2} \ (x, c^2_x)$$
$$\lambda^\star_2 = (pk, h_{pk})$$

We set $\ell_2(\kappa)$ and $\ell'_2(\kappa)$ to be the maximum size of $\pi^\star_2$ and $\lambda^\star_2$ respectively. Intuitively, defining P_2, Q_2 and $\pi^\star_2, \lambda^\star_2$ as above allows us to "tunnel" through the sampler along the path given by pk.

- 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 $0^{\ell_2'(\kappa)}$.

  Lemma 6.7 *Assuming the semantic security of $\mathcal{SKE}$, $|\mathsf{Adv}(\mathsf{Hyb}_0) - \mathsf{Adv}(\mathsf{Hyb}_1)| \leq \mathsf{negl}(\kappa)$.*

  Proof We construct an adversary against the semantic security of symmetric encryption. $\mathcal{B}$ samples $(pk, sk) \leftarrow \mathsf{PK.KeyGen}(1^\kappa)$. It generates the public key of the trapdoor permutation exactly as in Hyb_0. It generates two sets of messages, $(0^{\ell_2(\kappa)}, 0^{\ell_2'(\kappa)})$ and $(\pi_2^\star, \lambda_2^\star)$ (defined above). It queries these messages as the challenge messages to the external challenger and obtains Π_2, Λ_2 as the challenge ciphertext. It constructs the sampler exactly as in Hyb_0 except that:

  1. It does not sample sk_2.
  2. It uses the challenge ciphertext Π_2, Λ_2 instead of generating them as encryptions of all zeroes string.

  $\mathcal{B}$ runs the adversary with the challenge, public key and the constructed sampler and outputs whatever it outputs.

  Note that if Π_2, Λ_2 correspond to encryptions of $(0^{\ell_2(\kappa)}, 0^{\ell_2'(\kappa)})$ the distribution of inputs to the adversary is identical to Hyb_0; else it is identical to Hyb_1. Thus, $\mathcal{B}$ breaks the semantic security of $\mathcal{SKE}$. ■

- 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, $c_x^2 := \mathsf{FE.Enc}_{PK^2_{|x|+1}}(x, \mathsf{U}_x, K_x, \overline{K}_x, 0^{p(\kappa)}, 0; K'^2_x)$. In this hybrid, we change $c_x^2 := \mathsf{FE.Enc}_{PK^2_{|x|+1}}(x, 0^{|\mathsf{U}_x|}, 0^\kappa, 0^\kappa, sk_2, 1; r_x^2)$ 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 $x, y \in P_2$, $x \prec y$ if x is a prefix of y [10]. Observe that by this ordering ϕ is the smallest element in P_2. Let $\mathsf{Hyb}_{1,y}$ denote an hybrid where for all $x \prec y$, c_x^2 is set to $\mathsf{FE.Enc}_{PK_{|x|+1}}(x, 0^{|\mathsf{U}_x|}, 0^\kappa, 0^\kappa, sk_2, 1; r_x^2)$. 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 $x \prec x'$, $\mathsf{Hyb}_{1,x}$ is computationally indistinguishable to $\mathsf{Hyb}_{1,x'}$. Note that this is sufficient to show that Hyb_2 is computationally indistinguishable to Hyb_1 [11].

  – $\mathsf{Hyb}_{1,\phi}$: In this hybrid, we change c_ϕ^2 to $\mathsf{FE.Enc}_{PK_1^2}(\phi, 0^{|\mathsf{U}_x|}, 0^\kappa, 0^\kappa, sk_2, 1)$.

  Lemma 6.8 *Assuming the single-key, selective security of $\mathcal{FE}$, we have $|\mathsf{Adv}(\mathsf{Hyb}_1) - \mathsf{Adv}(\mathsf{Hyb}_{1,\phi})| \leq \mathsf{negl}(\kappa)$.*

[10] Note that by this ordering $y \prec y$.

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

Proof We construct an adversary $\mathcal{B}$ against the single-key, selective security of $\mathcal{FE}$. $\mathcal{B}$ samples $(pk, sk) \leftarrow \mathsf{PK.KeyGen}(1^\kappa)$. It constructs the public key of the trapdoor permutation exactly as in Hyb_1. The sampler constructed by $\mathcal{B}$ is same as that in Hyb_1 except that:

1. It does not sample $(\mathsf{PK}_1^2, \mathsf{MSK}_1^2)$ but samples all other public keys and master secret keys.
2. It constructs two messages $(\phi, 0^{|\mathsf{U}_x|}, 0^\kappa, 0^\kappa, sk_2, 1)$ and $(\phi, \mathsf{U}_\phi, K, \overline{K}, 0^{p(\kappa)}, 0)$ and provides them as challenge messages to the external challenger. In return, it gets a challenge ciphertext c^* which encrypts one of the above two messages. It sets $c_\phi^2 = c^*$.
3. It queries the external challenger with the function $F^2_{1,PK_2,\Pi_2^\star}$ and obtains the functional secret key FSK corresponding to the above function. It sets $\mathsf{FSK}_1^2 = \mathsf{FSK}$.

$\mathcal{B}$ runs the adversary with the challenge, sampler and public key as constructed above and outputs whatever the adversary outputs.

Notice that both $(\phi, 0^{\tau+\kappa}, 0^\kappa, 0^\kappa, sk_2, 1)$ and $(\phi, U_\phi, K, K_\phi^2, 0^\kappa, 0)$ trigger the same output in $F^2_{1,PK_2,\Pi_2^\star}$ because of the values encrypted in $\Pi_2^\star$. Further, the choice of the two messages does not depend on the knowledge of PK_1^2. Thus, $\mathcal{B}$ represents a valid adversary against $\mathcal{FE}$ scheme.

If c^* is an encryption of $(\phi, 0^{|\mathsf{U}_x|}, 0^\kappa, 0^\kappa, sk_2, 1)$ then the inputs to the adversary is distributed identically to $\mathsf{Hyb}_{1,\phi}$; else the inputs are distributed identically to Hyb_1. Thus, $\mathcal{B}$ breaks the single-key, selective security of $\mathcal{FE}$. ■

– $\mathsf{Hyb}_{1,x,1}$: In this hybrid we change $c^2_{x'}$ (encrypted in $\Pi_2^\star$) to $\mathsf{FE.Enc}_{\mathsf{PK}^2_{|x'|+1}}(x', \mathsf{U}_{x'}, K_{x'}, \overline{K}_{x'}, 0^\kappa, 0; r^2_{x'})$ where $r^2_{x'}$ is chosen uniformly and independently at random.

Lemma 6.9 *Assuming the pseudorandomness at constrained prefix property of $\mathcal{PCPRF}$, we have* $|\mathsf{Adv}(\mathsf{Hyb}_{1,x,1}) - \mathsf{Adv}(\mathsf{Hyb}_{1,x})| \leq \mathsf{negl}(\kappa)$

Proof We construct an adversary $\mathcal{B}$ against the pseudorandomness at constrained prefix property of $\mathcal{PCPRF}$. $\mathcal{B}$ samples $(pk, sk) \leftarrow \mathsf{PK.KeyGen}(1^\kappa)$. It constructs the public key of the trapdoor permutation exactly as in $\mathsf{Hyb}_{1,x}$. The sampler constructed by $\mathcal{B}$ is same as that in $\mathsf{Hyb}_{1,x}$ except that:

1. It does not sample the $\mathcal{PCPRF}$ key $\overline{K}$.
2. It queries the external challenger with the prefix $s = x'_1\|0\|x'_2\|0\cdots x'_{|x'|-1}\|0\|x'_{|x'|}\|1$. In return, it receives a challenge string y which is either $\mathsf{PrefixCons}(K, s)$ or a random string as well as Keys which contains $\{\overline{K}_{s_{[i-1]}\|(1-s_i)}\}_{i\in[|s|]}$.
3. It uses Keys to generate all encryptions in $\pi_2^\star$ except $c^2_{x'}$. Notice that in $\mathsf{Hyb}_{1,x}$ and $\mathsf{Hyb}_{1,x,1}$, Keys contain all the information needed to generate encryptions in $\pi_2^\star$ except $c^2_{x'}$ as for every prefix y of s, $\overline{K}_y$ is set to 0^κ in the $\mathcal{FE}$ ciphertexts in $\pi_2^\star$.
4. It sets $c^2_{x'}$ encrypted in $\Pi_2^\star$ as $\mathsf{FE.Enc}_{\mathsf{PK}^2_{|x'|+1}}(x', \mathsf{U}_{x'}, K_{x'}, \overline{K}_{x'}, 0^\kappa, 0; y)$.

$\mathcal{B}$ runs the adversary with the challenge, sampler and public key as constructed above and outputs whatever the adversary outputs.

Note that if y is a random string then the inputs to the adversary is distributed identically to $\mathsf{Hyb}_{1,x,1}$; else the inputs are distributed identically to $\mathsf{Hyb}_{1,x}$. Thus, $\mathcal{B}$ against the pseudorandomness at constrained prefix property of $\mathcal{PCPRF}$. ■

– $\mathsf{Hyb}_{1,x,2}$: In this hybrid we change $c^2_{x'}$ (encrypted in $\Pi^\star_2$) to $\mathsf{FE.Enc}_{PK^2_{|x'|+1}}(x', 0^{\tau+\kappa}, 0^\kappa, 0^\kappa, sk_2, 1; r^2_{x'})$. 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 $(x', 0^{\tau+\kappa}, 0^\kappa, 0^\kappa, sk, 1)$ 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 $PK^2_{|x'|+1}$. The proof of the following lemma is similar to proof of Lemma 6.8 and thus we omit the proof.

Lemma 6.10 *Assuming the single-key, selective security of $\mathcal{FE}$, we have* $|\mathsf{Adv}(\mathsf{Hyb}_{1,x,2}) - \mathsf{Adv}(\mathsf{Hyb}_{1,x,1})| \leq \mathsf{negl}(\kappa)$.

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 $(pk, \rho, \rho_1, \cdots, \rho_{\tau+\kappa})$ 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 using $\mathsf{PrefixCons}(\overline{K}_{pk}, 0)$ as the random tape.

- Hyb_3 : In this hybrid, we are going to generate the encryptions $\rho, \{\rho_i\}_{i\in[\tau+\kappa]}$ in h_{pk} (encrypted in $\Lambda^\star_2$) using true random strings instead of using stings generated by $\mathsf{PrefixCons}(\overline{K}_{pk}, 0)$.

Lemma 6.11 *Assuming the pseudorandomness at constrained prefix property of $\mathcal{PCPRF}$, we have* $|\mathsf{Adv}(\mathsf{Hyb}_2) - \mathsf{Adv}(\mathsf{Hyb}_3)| \leq \mathsf{negl}(\kappa)$.

Proof We construct an adversary $\mathcal{B}$ against the pseudorandomness at constrained prefix property of $\mathcal{PCPRF}$. $\mathcal{B}$ samples $(pk, sk) \leftarrow \mathsf{PK.KeyGen}(1^\kappa)$. It constructs the public key of the trapdoor permutation exactly as in Hyb_2. The sampler constructed by $\mathcal{B}$ is same as that in Hyb_2 except that:

1. It does not sample the $\mathcal{PCPRF}$ key $\overline{K}$.
2. It queries the external challenger with the prefix $s = pk_1\|0\|pk_2\|0\cdots pk_{\tau-1}\|0\|pk_\tau\|1$. In return, it receives a challenge string y which is either $\mathsf{PrefixCons}(K, s)$ or a random string as well as Keys which contains $\{\overline{K}_{s_{[i-1]}\|(1-s_i)}\}_{i\in[|s|]}$.
3. It uses Keys to generate all encryptions in $\pi^\star_2$. Notice that in Hyb_2 and Hyb_3, Keys contain all the information needed to generate encryptions in $\pi^\star_2$. This is because for every prefix y of s, $\overline{K}_y$ is set to 0^κ in the $\mathcal{FE}$ ciphertexts in $\pi^\star_2$.
4. It uses the string y as the random tape to generate encryptions $\rho, \rho_1, \cdots, \rho_{\tau+\kappa}$. It sets $h_{pk} = (pk, \rho, \rho_1, \cdots, \rho_{\tau+\kappa})$

$\mathcal{B}$ runs the adversary with the challenge, sampler and public key as constructed above and outputs whatever the adversary outputs.

Note that if y is a random string then the inputs to the adversary is distributed identically to Hyb_3; else the inputs are distributed identically to Hyb_2. Thus, $\mathcal{B}$ against the pseudorandomness at constrained prefix property of $\mathcal{PCPRF}$. ■

- Hyb_4 : In this hybrid, we are going to change $\rho, \{\rho_i\}_{i\in 2[\kappa]}$ in h_{pk} (encrypted in $\Lambda_2^\star$) to encryptions of 0^κ.

  Lemma 6.12 *Assuming the semantic security of $\mathcal{PKE}$, we have $|\mathsf{Adv}(\mathsf{Hyb}_4) - \mathsf{Adv}(\mathsf{Hyb}_3)| \leq \mathsf{negl}(\kappa)$.*

  Proof We construct an adversary against the semantic security of $\mathcal{PKE}$. $\mathcal{B}$ interacts with the external challenger and obtains the public key pk. It constructs the public key of the trapdoor permutation exactly as in Hyb_3. The sampler constructed by $\mathcal{B}$ is same as that in Hyb_3 except that:

  1. It constructs two sets of messages $(K_{pk}, \{S_{i,(pk\|K_{pk})_{[i]}}\}_{i\in[\tau+\kappa]})$ and $(0^\kappa, 0^\kappa, \cdots, 0^\kappa)$ and gives them as challenge messages to the external challenger. In return, it receives $(c, c_1, \cdots, c_{\tau+\kappa})$ as the challenge ciphertext that encrypts one of the above two sets of messages. It sets $\rho = c$ and $\rho_i = c_i$ for all $i \in [\tau + \kappa]$ and sets $h_{pk} = (pk, \rho, \rho_1, \cdots, \rho_{\tau+\kappa})$.

  $\mathcal{B}$ runs the adversary with the challenge, sampler and public key as constructed above and outputs whatever the adversary outputs.

  Note that if $c, c_1, \cdots, c_{\tau+\kappa}$ is an encryption of $(K_{pk}, \{S_{i,(pk\|K_{pk})_{[i]}}\}_{i\in[\tau+\kappa]})$ then the inputs to the adversary is distributed identically to Hyb_4; else the inputs are distributed identically to Hyb_3. Thus, $\mathcal{B}$ against the semantic security of $\mathcal{PKE}$. ■

- Hyb_5 : In this hybrid, we are going to change K_{pk} in the challenge given to the adversary to a random string z.

  Lemma 6.13 *Assuming the pseudorandomness at constrained prefix property of $\mathcal{PCPRF}$, we have $|\mathsf{Adv}(\mathsf{Hyb}_4) - \mathsf{Adv}(\mathsf{Hyb}_5)| \leq \mathsf{negl}(\kappa)$.*

  Proof We construct an adversary $\mathcal{B}$ against the pseudorandomness at constrained prefix property of $\mathcal{PCPRF}$. $\mathcal{B}$ samples $(pk, sk) \leftarrow \mathsf{PK.KeyGen}(1^\kappa)$. It constructs the public key of the trapdoor permutation exactly as in Hyb_4. The sampler constructed by $\mathcal{B}$ is same as that in Hyb_4 except that:

  1. It does not sample the $\mathcal{PCPRF}$ key K.
  2. It queries the external challenger with the prefix pk. In return, it receives a challenge string y which is either $\mathsf{PrefixCons}(K, pk)$ or a random string as well as Keys which contains $\{K_{pk_{[i-1]}\|(1-pk_i)}\}_{i\in[\tau]}$.
  3. It uses Keys to generate all encryptions in $\pi_2^\star$ and $\lambda_2^\star$. Notice that in Hyb_4 and Hyb_5, Keys contain all the information needed to generate encryptions in $\pi_2^\star$ and $\lambda_2^\star$. This is because for every prefix y of pk, K_y is set to 0^κ in the $\mathcal{FE}$ ciphertexts in $\pi_2^\star$ and $S_{j,(pk\|K_{pk})_{[j]}}$ for $j > \kappa$ is also set to 0^κ in h_{pk}.
  4. It sets the challenge $(pk\|y, \sigma_1, \cdots, \sigma_{\tau+\kappa})$ where $\sigma_i = S_{i,(pk\|y)_{[i]}}$ for all $i \in [\tau + \kappa]$.

$\mathcal{B}$ runs the adversary with the challenge, sampler and public key as constructed above and outputs whatever the adversary outputs.

Note that if y is a random string then the inputs to the adversary is distributed identically to Hyb_5; else the inputs are distributed identically to Hyb_4. Thus, $\mathcal{B}$ against the pseudorandomness at constrained prefix property of $\mathcal{PCPRF}$. ■

- Hyb_6: Same as Hyb_3 except that the challenge is generated as $((pk\|z), \sigma_1, \cdots, \sigma_{\kappa+\tau})$ where z is chosen uniformly at random from $\{0,1\}^\kappa$ and $\sigma_i = S_{i,(pk\|z)_{[i]}}$. We have via an identical argument to Lemma 6.12 that

  Lemma 6.14 *Assuming the semantic security of $\mathcal{PKE}$, we have* $|\mathsf{Adv}(\mathsf{Hyb}_5) - \mathsf{Adv}(\mathsf{Hyb}_6)| \leq \mathsf{negl}(\kappa)$.

- Hyb_7 : Same as Hyb_2 except that the challenge is generated as $((pk\|z), \sigma_1, \cdots, \sigma_{\kappa+\tau})$ where z is chosen uniformly at random from $\{0,1\}^\kappa$ and $\sigma_i = S_{i,(pk\|z)_{[i]}}$. The following lemma follows via an identical argument to Lemma 6.11.

  Lemma 6.15 *Assuming the pseudorandomness at constrained prefix property of $\mathcal{PCPRF}$, we have* $|\mathsf{Adv}(\mathsf{Hyb}_6) - \mathsf{Adv}(\mathsf{Hyb}_7)| \leq \mathsf{negl}(\kappa)$.

- Hyb_8 : Same as Hyb_1 except that the challenge is generated as $((pk\|z), \sigma_1, \cdots, \sigma_{\kappa+\tau})$ where z is chosen uniformly at random from $\{0,1\}^\kappa$ and $\sigma_i = S_{i,(pk\|z)_{[i]}}$. We have the following Lemma via an identical arguments to Lemma 6.8, Lemma 6.9 and Lemma 6.10.

  Lemma 6.16 *Assuming the pseudorandomness at constrained prefix property of $\mathcal{PCPRF}$ and the single-key selective security of $\mathcal{FE}$, we have* $|\mathsf{Adv}(\mathsf{Hyb}_7) - \mathsf{Adv}(\mathsf{Hyb}_8)| \leq \mathsf{negl}(\kappa)$.

- Hyb_9 : Same as Hyb_0 except that the challenge is generated as $((pk\|z), \sigma_1, \cdots, \sigma_{\kappa+\tau})$ where z is chosen uniformly at random from $\{0,1\}^\kappa$ and $\sigma_i = S_{i,(pk\|z)_{[i]}}$. The following Lemma follows from an identical argument to Lemma 6.7.

  Lemma 6.17 *Assuming the semantic security of $\mathcal{SKE}$,* $|\mathsf{Adv}(\mathsf{Hyb}_8) - \mathsf{Adv}(\mathsf{Hyb}_9)| \leq \mathsf{negl}(\kappa)$.

Note that in Hyb_9, an adversary is given $(pk\|z)$, the public key $(\{\mathsf{FSK}^1_i\}_{i\in[\tau+\kappa+1]}, c^1_\phi)$ and the sampler $(\{\mathsf{FSK}^2_i\}_{i\in[\tau+1]}, c^2_\phi)$. 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. ■

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

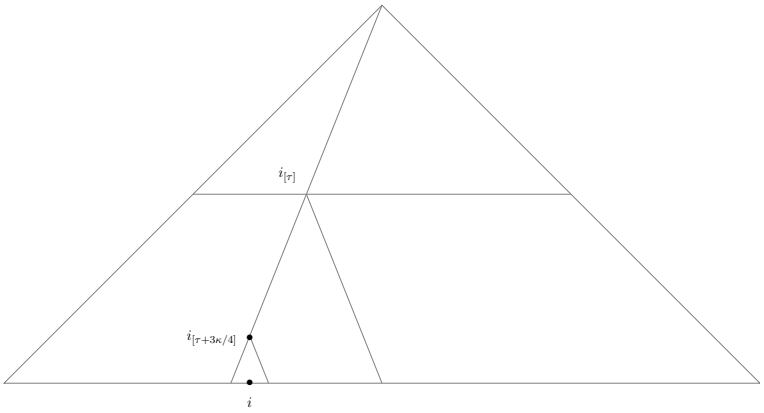

Figure 21: Condition for Aborting

Notation. Let $\mathsf{Adv}(\mathsf{Hyb}_i)$ denote the probability that adversary inverts the trapdoor challenge in Hyb_i.

- Hyb_0 : This hybrid is same as the inversion experiment where the adversary is given $((pk\|K_{pk}), \sigma_1, \cdots, \sigma_{\tau+\kappa})$ as the trapdoor challenge where $(pk, sk) \leftarrow \mathsf{PK.KeyGen}(1^\kappa)$ and the sampler is run with input pk to obtain $(pk\|K_{pk}, \cdot, \cdots, \cdot)$. The adversary is additionally provided with $(\{\mathsf{FSK}_i^2\}_{i\in[\tau+1]}, c_\phi^2)$ and $(\{\mathsf{FSK}_i^1\}_{i\in[\tau+\kappa+1]}, c_\phi^1$ as the sampler and public key respectively.

- 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 6.6. We infer that $|\mathsf{Adv}(\mathsf{Hyb}_0) - \mathsf{Adv}(\mathsf{Hyb}_1)| \leq \mathsf{negl}(\kappa)$ as a direct consequence of Lemma 6.6.

  Let i denote the first component of inversion challenge namely, $pk\|z$ in Hyb_1. Recall that z is sampled uniformly at random from $\{0,1\}^\kappa$.

  Let $i_1 \cdots i_{\tau+\kappa}$ denote the binary representation of i. If $i_{\tau+1} \cdots i_{\tau+3\kappa/4}$ is equal to $0^{3\kappa/4}$, we abort. Figure 21 illustrates the condition for aborting. Notice that since $z = i_{\tau+1} \cdots i_{\tau+\kappa}$ is randomly distributed, the probability that we abort is at most $\frac{1}{2^{3\kappa/4}}$ which is negligible. Hence, from now on we assume that we have not aborted [12].

- Hyb_2 : In this hybrid, we change how the value v that is hardwired in $G^1_{v,\Lambda_1,w}$ is generated. Let $i^\star = i_{[3\kappa/4]} - 1$. We set $v := \mathsf{InjPRG}(\mathsf{Ext}_w(i^\star\|u_0))$ where $u_0 \xleftarrow{\$} \{0,1\}^{\kappa/4}$. We accomplish this change via a couple of intermediate steps:

  - $\mathsf{Hyb}_{1,1}$: In this hybrid, instead of choosing v uniformly at random from $\{0,1\}^{\kappa/4}$, we set $v := \mathsf{InjPRG}(v')$ where $v' \xleftarrow{\$} \{0,1\}^{\kappa/8}$.

    Lemma 6.18 *Assuming the pseudorandomness property of* InjPRG*, we have* $|\mathsf{Adv}(\mathsf{Hyb}_1) - \mathsf{Adv}(\mathsf{Hyb}_{1,1})| \leq \mathsf{negl}(\kappa)$.

[12] 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.

Proof We construct an adversary $\mathcal{B}$ against the pseudorandomness property of InjPRG. $\mathcal{A}$ samples $(pk, sk) \leftarrow \mathsf{PK.KeyGen}(1^\kappa)$ and $z \xleftarrow{\$} \{0,1\}^\kappa$ and set $i = pk \| z$. It constructs the inversion challenge $(i, \cdot, \ldots, \cdot)$ and the sampler exactly as in Hyb_1. It constructs the public key of the permutation as in Hyb_1 except that:

1. In constructing $G^1_{v,\lambda_1,w}$, it does not sample $v \xleftarrow{\$} \{0,1\}^{\kappa/4}$.
2. It interacts with the external challenger and receives a challenge string y which is either the output of InjPRG on a random seed or a uniformly chosen random string. It sets $v = y$ and constructs the rest of the components in $G^1_{v,\lambda_1,w}$ as in Hyb_1.

$\mathcal{B}$ runs the adversary on the inversion challenge, sampler and the constructed public key. It outputs 1 if the adversary inverts the challenge; else it outputs 0.

Notice that if y is a random string then inputs to the adversary are distributed identically to Hyb_1; else it is identically distributed to $\mathsf{Hyb}_{1,1}$. Thus, $\mathcal{B}$ breaks the pseudorandomness property of InjPRG. ■

– $\mathsf{Hyb}_{1,2}$: In this hybrid, we set $v := \mathsf{InjPRG}(\mathsf{Ext}_w(i^\star \| u_0))$ where $u_0 \xleftarrow{\$} \{0,1\}^{\kappa/4}$.

Lemma 6.19 *Assuming the statistical closeness of the output of* Ext*, we have* $|\mathsf{Adv}(\mathsf{Hyb}_{1,1}) - \mathsf{Adv}(\mathsf{Hyb}_{1,2})| \leq \mathsf{negl}(\kappa)$.

Sketch of Proof (Informal) We set the X source to be of the form $i^\star \| R$ where R is a random variable that is uniformly distributed on $\{0,1\}^{\kappa/4}$. Note that the min-entropy of the source X is at least $\kappa/4$. We set $v = \mathsf{InjPRG}(y)$ where y is the challenge string (which is either $\mathsf{Ext}_w(X)$ or is uniformly chosen) and use the seed w to construct $G^1_{v,\Lambda_1,w}$. If y is a random string then we generate distribution identical to $\mathsf{Hyb}_{1,1}$; else we generate distribution identical to $\mathsf{Hyb}_{1,2}$. Thus, depending on the adversary inverts the challenge we can distinguish between whether y is a random string or output of the extractor thus breaking the statistical closeness of the output of Ext. ■

Note that in this hybrid the public key outputs $\perp$ on all inputs of the form $(i^* \| u_0, \cdot, \ldots, \cdot)$. 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 $y \in [\alpha_0, i-1]$ the components in the sampler cannot be used to derive (in an information theoretic sense) $S_{j,y_{[j]}}$ for all $j \geq \tau$ [13]. 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 5.2, for any string $\alpha \in \{0,1\}^{\tau+\kappa}$, 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 $\alpha = \alpha_{[j]} || 1^{\tau+\kappa-j}$.

For example, if $\alpha = \overbrace{100}^{3} 111$ then $f(\alpha) = 3$. Recall $\ell(\beta, \gamma)$ denotes the unique $j \in [\tau + \kappa]$

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

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 $\tau + \kappa$ otherwise. Let $\delta(\alpha)$ denote the number of 0s in the positions $[\ell(\alpha, i-1) + 1, \tau + \kappa]$ 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 $[\ell(\alpha, i-1) + 1, \tau + \kappa]$ in the binary representation of $i-1$ if $\ell(\alpha, i-1) \neq \zeta$ and is equal to 0 otherwise.

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

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

Note that by this definition $\alpha_{\delta(\alpha_0)+\mu(\alpha_{\delta(\alpha_0)})} = i-1$. Also, $\delta(\alpha_0) + \mu(\alpha_{\delta(\alpha_0)}) \leq 8\kappa - 1$.

New π_1, λ_1 values. As in Section 5.2 we process the hybrids according to α_j values. For any $x \in \{0,1\}^{\leq \tau+\kappa}$, let c_x^1 denote the ciphertext and d_x denote the final output in the execution of Steps 1 and 2 of TDP_{PK} in Hyb_2 on input x. Note that d_x is defined only for $x \in \{0,1\}^{\tau+\kappa}$. 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 $|P_1| \leq 8\kappa(\tau + \kappa + 1)$. 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:

$$\pi_1^\star = ||_{x \in P_1 \cup Q_1} \ (x, c_x^1)$$
$$\lambda_1^\star = ||_{x \in P_1 \cap \{0,1\}^{\tau+\kappa}} \ (x, d_x)$$

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 $\alpha_0 \in \{0,1\}^{\tau+\kappa}$.

- 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_1 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) by an identical argument to the proof of Lemma 6.7.

  Lemma 6.20 *Assuming the semantic security of $\mathcal{SKE}$, we have* $|\mathsf{Adv}(\mathsf{Hyb}_2) - \mathsf{Adv}(\mathsf{Hyb}_3)| \leq \mathsf{negl}(\kappa)$.

- 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 $\mathsf{FE.Enc}_{PK_{|x|+1}^1}(x, \mathsf{V}_x, \mathsf{W}_x, \widetilde{K}_x, 0^\kappa, 0; {K'}_x^1)$. We change the c_x^1 to be now generated as $\mathsf{FE.Enc}_{PK_{|x|+1}^1}(x, 0^{|\mathsf{V}_x|}, 0^{|\mathsf{W}_x|}, 0^\kappa, sk_1, 1; \omega_x)$ using fresh randomness ω_x.[14]

[14] Note that we do not change ciphertexts corresponding to $x \in Q_1$.

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 $x, y \in P_1$ we say that $x \prec y$ if either $|x| < |y|$, or $|x| = |y|$. [15]. Next we define the hybrid $\mathsf{Hyb}_{3,y}$ to be a modification of Hyb_3 where for all $x \in P_1$ such that $x \prec y$ we have that c^1_x is generated as $\mathsf{FE.Enc}_{PK^1_{|x|+1}}(x, 0^{|\mathsf{V}_x|}, 0^{|\mathsf{W}_x|}, 0^\kappa, sk_1, 1; \omega_x)$ 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^1_ϕ to $\mathsf{FE.Enc}_{PK^1_1}(\phi, 0^{|\mathsf{V}_x|}, 0^{|\mathsf{W}_x|}, 0^\kappa, sk_1, 1)$. Notice that both $(\phi, 0^{|\mathsf{V}_x|}, 0^{|\mathsf{W}_x|}, 0^\kappa, sk_1, 1)$ and $(\phi, \mathsf{V}_\phi, \mathsf{W}_\phi, \widetilde{K}, 0^\kappa, 0)$ give the same output on $F^1_{1,PK^1_2,\Pi^\star_2}$ because of the value encrypted in $\Pi^\star_2$. Further more the choice of two messages does not depend on the value of PK^1_1. Hence, following lemma follows from an identical argument to Lemma 6.8.

   Lemma 6.21 *Assuming the single-key, selective security of $\mathcal{FE}$ scheme, we have* $|\mathsf{Adv}(\mathsf{Hyb}_3) - \mathsf{Adv}(\mathsf{Hyb}_{3,\phi})| \leq \mathsf{negl}(\kappa)$.

2. $\mathsf{Hyb}_{3,x,1}$: In this hybrid we change c^1_x to $\mathsf{FE.Enc}_{PK^1_{|x|+1}}(x, \mathsf{V}_x, \mathsf{W}_x, \widetilde{K}_x, 0^\kappa, 0; \omega_x)$ 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 $c_{x''} = \mathsf{FE.Enc}_{PK^1_{|x''|+1}}(x'', 0^{|\mathsf{V}_x|}, 0^{|\mathsf{W}_x|}, 0^\kappa, sk_1, 1; \omega_{x''})$. Thus, we lemma stated below follows via an identical argument to the proof of Lemma 6.9.

   Lemma 6.22 *Assuming the pseudorandomness at the constrained prefix property of $\mathcal{PCPRF}$, we have* $|\mathsf{Adv}(\mathsf{Hyb}_{3,x}) - \mathsf{Adv}(\mathsf{Hyb}_{3,x,1})| \leq \mathsf{negl}(\kappa)$.

3. $\mathsf{Hyb}_{3,x,2}$: In this hybrid we change c^1_x to $\mathsf{FE.Enc}_{PK^1_{|x|+1}}(x, 0^{|\mathsf{V}_x|}, 0^{|\mathsf{W}_x|}, 0^\kappa, sk_1, 1; \omega_x)$ using fresh randomness ω_x.
   The claim that the probability that adversary inverts the challenge in this hybrid is negligibly close to $\mathsf{Adv}(\mathsf{Hyb}_{3,x,1})$ 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 $PK^1_{|x|+1}$. We have the following lemma through an identical argument to Lemma 6.10.

   Lemma 6.23 *Assuming the single-key, selective security of $\mathcal{FE}$ scheme, we have* $|\mathsf{Adv}(\mathsf{Hyb}_{3,x,2}) - \mathsf{Adv}(\mathsf{Hyb}_{3,x,1})| \leq \mathsf{negl}(\kappa)$.

- $\mathsf{Hyb}_{5,j+1}$: 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 5.2, 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 $t_j = \alpha_{j[\nu_j]} + 1$.

[15] Note that ϕ is the smallest value in P_1 by this ordering.

– We change the set $\mathsf{W}_{t_j}^{\nu_j}$ to be a uniformly random string $y \leftarrow \{0,1\}^\kappa$ rather than containing the value $\mathsf{LeftInjPRG}_0(S_{\nu_j,t_j})$. Note that this change needs to be made at two places. Namely, we set $\mathsf{W}_{t_j}^{\nu_j}$ in c_s^1 where s 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 $\beta_{f(\alpha_{j+1})}, \ldots, \beta_{\tau+\kappa}$ 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 $y \neq \mathsf{LeftInjPRG}_0(\sigma_{\nu_j})$ 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 $W_{t_j}^{\nu_j}$, in generating γ_{ν_j} used in $d_{\alpha_{j+1}}$ and in ψ_{ν_j} in $d_{\alpha_{j+1}}$.

Lemma 6.24 *Assuming the pseudorandomness at constrained prefix property of* PCPRF*, we have* $|\mathsf{Adv}(\mathsf{Hyb}_{5,j}) - \mathsf{Adv}(\mathsf{Hyb}_{5,j,1})| \leq \mathsf{negl}(\kappa)$.

Proof We construct an adversary $\mathcal{B}$ against the pseudorandomness at prefix constrained property of PCPRF. $\mathcal{B}$ samples $(pk, sk) \leftarrow \mathsf{PK.KeyGen}(1^\kappa)$ and samples $z \xleftarrow{\$} \{0,1\}^\kappa$. It sets $i = pk\|z$ as the first component of inversion challenge. For each $k \in [\tau + \kappa] \setminus \{\nu_j\}$, it samples $\mathcal{PCPRF}$ key S_k as in $\mathsf{Hyb}_{5,j}$. It uses the sampled S_k to generate $\sigma_k = S_{k,pk\|z_{[k]}}$. Now,

1. $\mathcal{B}$ interacts with the external challenger and provides t_j as prefix on which it wants to constrain the $\mathcal{PCPRF}$ key. It receives a challenge string y and Keys which contains $\{S_{\nu_j,t_{j[k-1]}}\|1-(t_j)_k\}_{k\in[|t_j|]}$.
2. It sets $\mathsf{W}_{t_j}^{\nu_j} = \mathsf{LeftInjPRG}_0(y)$, sets ψ_{ν_j} included in $d_{\alpha_{j+1}}$ to be $\mathsf{LeftInjPRG}_0(y)$ and γ_{ν_j} used to generate the encryption $\beta_{f(\alpha_{j+1})}, \ldots, \beta_{\tau+\kappa}$ to be $\mathsf{LeftInjPRG}_1(y)$.
3. $\mathcal{B}$ uses the keys to generate the sampler, the ν_j-th signature on the inversion challenge and the rest of the components in public key exactly as in $\mathsf{Hyb}_{5,j}$.

$\mathcal{B}$ runs the adversary on the inversion challenge, sampler and the constructed public key. It outputs 1 if the adversary inverts the challenge; else it outputs 0.

We now argue that Keys contain all the information needed to generate the sampler, the ν_j-th signature on the inversion challenge and the rest of the components in public key.

- Recall a prior observation that the sampler cannot be used to derive (in an information theoretic sense) $S_{j,s_{[j]}}$ for all $s \in [\alpha_0, i-1]$ 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}.
- $\mathsf{V}_y^{\nu_j}$ 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 6.5) 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 $\perp$.
- From the choice of ν_j and t_j, we infer that Keys cannot be used to derive the ν_j-th component of the signature only on string p where $\alpha_j + 1 \leq p \leq \alpha_{j+1}$. Since $\alpha_{j+1} \leq i - 1$, we infer that Keys can be used to derive the nu_j-th component of the challenge signature.

Notice that if y is chosen uniformly at random then distribution of inputs to the adversary is identical to $\mathsf{Hyb}_{5,j,1}$; else the distribution is identical to $\mathsf{Hyb}_{5,j}$. Thus, $\mathcal{B}$ breaks the pseudorandomness at constrained prefix property of PCPRF. ■

– $\mathsf{Hyb}_{5,j,2}$: 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 $\widetilde{K}_y$ 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, we have the following lemma from the pseudorandomness at constrained prefix property through an identical argument in Lemma 6.11.

Lemma 6.25 *Assuming the pseudorandomness at constrained prefix property,* $|\mathsf{Adv}(\mathsf{Hyb}_{5,j,1}) - \mathsf{Adv}(\mathsf{Hyb}_{5,j,2})| \leq \mathsf{negl}(\kappa)$.

– $\mathsf{Hyb}_{5,j,3}$: In this hybrid, we change $\mathsf{LeftInjPRG}_0(S')$ and $\mathsf{LeftInjPRG}_1(S')$ to be random strings y_0, y_1. Change of $\mathsf{LeftInjPRG}_0(S')$ to y_0 implies that the set $\mathsf{W}_{t_j}^{\nu_j}$ is $\{y_0\}$ and ψ_{ν_j} in $d_{\alpha_{j+1}}$ is also set to y_0. Similarly γ_{ν_j} will be y_1.

Lemma 6.26 *Assuming the pseudorandomness property of* $\mathsf{LeftInjPRG}$*, we have* $|\mathsf{Adv}(\mathsf{Hyb}_{5,j,2}) - \mathsf{Adv}(\mathsf{Hyb}_{5,j,3})| \leq \mathsf{negl}(\kappa)$.

Proof We construct an adversary $\mathcal{B}$ against the pseudorandomness property of $\mathsf{LeftInjPRG}$.

$\mathcal{B}$ samples $(pk, sk) \leftarrow \mathsf{PK.KeyGen}(1^\kappa)$ and samples $z \xleftarrow{\$} \{0,1\}^\kappa$. It sets $i = pk \| z$ as the first component of inversion challenge. It generates the sampler and the inversion challenge exactly as in $\mathsf{Hyb}_{5,j,2}$. It generates the public key as in $\mathsf{Hyb}_{5,j,2}$ except that:

1. It interacts with an external challenger and obtains y as the challenge string where y is the output $\mathsf{LeftInjPRG}$ on a random seed or chosen uniformly at random. Let y_0 be the first half of the string y and let y_1 be the second half of the string y.
2. It sets $\mathsf{W}_{t_j}^{\nu_j} = y_0$, sets ψ_{ν_j} included in $d_{\alpha_{j+1}}$ to be y_0 and γ_{ν_j} used to generate the encryptions $\beta_{f(\alpha_{j+1})}, \ldots, \beta_{\tau+\kappa}$ to be y_1.

$\mathcal{B}$ runs the adversary on the inversion challenge, sampler and the constructed public key. It outputs 1 if the adversary inverts the challenge; else it outputs 0.

Notice that if y is uniformly chosen random string then distribution of the inputs to the adversary is identical to $\mathsf{Hyb}_{5,j,3}$; else the distribution of inputs to the adversary is identical to $\mathsf{Hyb}_{5,j,2}$. Thus, $\mathcal{B}$ breaks the pseudorandomness property of $\mathsf{LeftInjPRG}$. ■

– $\mathsf{Hyb}_{5,j,4}$: In this hybrid we set encryptions in $d_{\alpha_{j+1}}$ i.e. $\beta_{f(\alpha_{j+1})}, \cdots, \beta_{\tau+\kappa}$ with encryption of 0^κ.

Lemma 6.27 *Assuming the semantic security of* $\mathcal{SKE}$*, we have* $|\mathsf{Adv}(\mathsf{Hyb}_{5,j,3}) - \mathsf{Adv}(\mathsf{Hyb}_{5,j,4})| \leq \mathsf{negl}(\kappa)$.

Proof We construct an adversary $\mathcal{B}$ against the semantic security of $\mathcal{SKE}$. $\mathcal{B}$ generates the sampler and the inversion challenge exactly as in $\mathsf{Hyb}_{5,j,3}$. In generating the public key of the permutation, it works similar to $\mathsf{Hyb}_{5,j,3}$ except that:

1. It samples γ_k for all $k \in [f(\alpha_{j+1}), \tau+\kappa] \setminus \{\nu_j\}$ as in $\mathsf{Hyb}_{5,j,3}$
2. It queries the external challenger with two sets of messages $\{\mathsf{SK.Enc}_{\gamma_{f(\alpha_j)},\cdots,\nu_j-1}(0^\kappa)\}_{k\in[f(\alpha_{j+1}),\tau+\kappa]}$ and $\{\mathsf{SK.Enc}_{\gamma_{f(\alpha_j)},\cdots,\nu_j-1}(S_{k,\alpha_{j+1}[k]})\}_{k\in[f(\alpha_{j+1}),\tau+\kappa]}$. It receives $\{c_k^*\}_{k\in[f(\alpha_{j+1}),\tau+\kappa]}$ as the set of challenge ciphertexts.
3. It generates $\beta_k = \mathsf{SK.Enc}_{\gamma_{\nu_j+1},\cdots,\tau+\kappa}(c_k^*)$ for every $k \in [f(\alpha_{j+1}), \tau+\kappa]$ and uses $\{\beta_k\}$ to generate $d_{\alpha_{j+1}}$.

$\mathcal{B}$ runs the adversary on the inversion challenge, sampler and the constructed public key. It outputs 1 if the adversary inverts the challenge; else it outputs 0.

Notice that if $\{c_k^*\}$ are encryptions of 0^κ then the then distribution of the inputs to the adversary is identical to $\mathsf{Hyb}_{5,j,4}$; else the distribution of inputs to the adversary is identical to $\mathsf{Hyb}_{5,j,3}$. Thus, $\mathcal{B}$ breaks the semantic security of $\mathcal{SKE}$. ■

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 $(i,\sigma_1,\cdots,\sigma_{\tau+\kappa})$ in this final hybrid with probability better than 0.

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.

[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.

[GK15] Shafi Goldwasser and Yael Tauman Kalai. Cryptographic assumptions: A position paper. Cryptology ePrint Archive, Report 2015/907, 2015. `http://eprint.iacr.org/2015/907`.

[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.

[GPS16] Sanjam Garg, Omkant Pandey, and Akshayaram Srinivasan. Revisiting the cryptographic hardness of finding a nash equilibrium. In *Advances in Cryptology - CRYPTO 2016 - 36th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2016, Proceedings, Part II*, pages 579–604, 2016.

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

[HJK+16] Dennis Hofheinz, Tibor Jager, Dakshita Khurana, Amit Sahai, Brent Waters, and Mark Zhandry. How to generate and use universal samplers. ASIACRYPT 2016, 2016. `http://eprint.iacr.org/2014/507`.

[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 Tetsu Iwata and Jung Hee Cheon, editors, *ASIACRYPT 2015, Part I*, volume 9452 of *LNCS*, pages 52–75, Auckland, New Zealand, November 30 – December 3, 2015. Springer, Heidelberg, Germany.

[Lin16] Huijia Lin. Indistinguishability obfuscation from DDH on 5-linear maps and locality-5 prgs. *IACR Cryptology ePrint Archive*, 2016:1096, 2016.

[LV16] Huijia Lin and Vinod Vaikuntanathan. Indistinguishability obfuscation from ddh-like assumptions on constant-degree graded encodings. In *IEEE 57th Annual Symposium on Foundations of Computer Science, FOCS 2016, 9-11 October 2016, Hyatt Regency, New Brunswick, New Jersey, USA*, pages 11–20, 2016.

[Nao03] Moni Naor. On cryptographic assumptions and challenges (invited talk). In Dan Boneh, editor, *CRYPTO 2003*, volume 2729 of *LNCS*, pages 96–109, Santa Barbara, CA, USA, August 17–21, 2003. Springer, Heidelberg, Germany.

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

[PST14] Rafael Pass, Karn Seth, and Sidharth Telang. Indistinguishability obfuscation from semantically-secure multilinear encodings. In Juan A. Garay and Rosario Gennaro, editors, *CRYPTO 2014, Part I*, volume 8616 of *LNCS*, pages 500–517, Santa Barbara, CA, USA, August 17–21, 2014. 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.

A Constructing Universal Samplers 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 seed 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.

B Optimality of n in Theorem B.1

Theorem B.1 *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 execution 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. ■