Breaking the Sub-Exponential Barrier in Obfustopia

Sanjam Garg[1], Omkant Pandey[2], Akshayaram Srinivasan[1], and Mark Zhandry[3]

[1] University of California, Berkeley
{sanjamg,akshayaram}@berkeley.edu
[2] Stony Brook University
omkant@gmail.com
[3] Princeton University
mzhandry@princeton.edu

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 $\mathcal{FE}$.

1 Introduction

Indistinguishability obfuscation ($i\mathcal{O}$) [5, 16] 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, [30, 12, 10, 4, 8]. Recently, $i\mathcal{O}$ also proved instrumental in proving the hardness of complexity class PPAD [7].

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[4]. Indeed, all known security proofs for $i\mathcal{O}$ relative to "simple" assumptions[5] iterate over all (exponentially-many) inputs anyway, and there are reasons to believe that this loss may be necessary [18] [6]. Indeed,

[4] Here, we do not define "simple." However, one can consider various notions of "simplicity" or "niceness" for assumptions, such as falsifiable assumptions [28] or complexity assumptions [22].

[5] Here, we exclude über assumptions such as semantically secure graded encodings [29], which encompass exponentially many distinct complexity assumptions.

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

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. [17], it has been shown in [27, 26] 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 [3, 9] incurs an exponential loss.* 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 [19] 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 [7] 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 [19]. 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 [8] 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 [17, 27, 26], 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 [23]. 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 [3, 9] 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 [23] 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. Our construction of TDP from FE is weaker in comparison to our construction from iO (and the construction of Bitansky et al. in [8]). 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 and NIKE for unbounded number of parties. Our techniques used for constructing trapdoor permutations are closely related to the techniques developed in proving PPAD-hardness of Garg et al. [19]. However, constructing trapdoor permutations poses additional challenges, namely the design of an efficient sampling algorithm that samples a domain element. Solving this problem requires development of new techniques and we elaborate them in the full version [20].

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 [11] 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 [15, 13], 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 [12] show how to use $i\mathcal{O}$ to remove the trusted setup. Later, Ananth et al. [1] shows how to remove the bound on the number of users by using the very strong *differing inputs* obfuscation. Khurana, Rao, and Sahai [25] further modify the Boneh-Zhandry protocol to get unbounded users with just $i\mathcal{O}$. In [12] and [25], $i\mathcal{O}$ 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 [18], 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 $\mathcal{FE}$.

Universal Samplers. Instead, we follow an alternate approach given by Hofheinz et al. [23] 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 [23] 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. [23]. 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, [1] and [25], 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 [6] 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. [23]. 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. [23] 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 [9, 19]. 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 [7]. These bit extension functions form a natural binary tree structure and "parsing" an input circuit C corresponds to traveling along the path from the root to the leaf labeled C. Each node x along the path from the root to C contains the key K prefix 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

[7] [19] 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.

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.

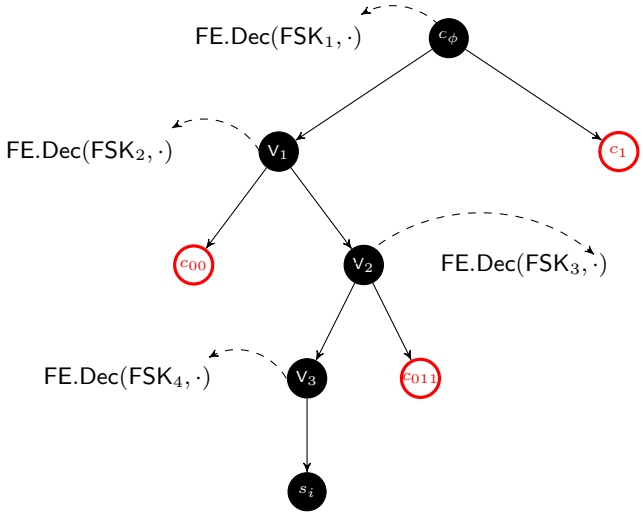

Fig. 1: Illustration of "tunneling" on $C_i = 010$ and $\kappa = 3$.

At a high level, the "tunneling" is achieved by triggering a hidden "trapdoor" thread in the function keys using techniques illustrated in [2, 19]. 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 [19] 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 [8] which enables us to "tunnel" along a new path C_i^*.

2 Preliminaries

κ denotes the security parameter. A function $\mu(\cdot) : \mathbb{N} \to \mathbb{R}^+$ is said to be negligible if for all polynomials $\mathsf{poly}(\cdot)$, $\mu(k) < \frac{1}{\mathsf{poly}(k)}$ for large enough k. 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.

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 $x = y \| z$.

We assume the reader's familiarity with standard cryptographic primitives like injective pseudorandom generator, puncturable pseudorandom functions, indistinguishability obfuscation, functional encryption, symmetric and public key encryption. Below, we give the definition of Prefix Constrained Pseudorandom Function [19].

Prefix Constrained Pseudorandom Function. A PCPRF is a tuple of algorithms $(\mathsf{KeyGen}_{\mathcal{PCPRF}}, \mathsf{PrefixCons})$ with the following syntax. $\mathsf{KeyGen}_{\mathcal{PCPRF}}$ takes the security parameter (encoded in unary) and descriptions of two

[8] 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.

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 give the formal definition of PCPRF.

Definition 1. *A prefix constrained pseudorandom function $\mathcal{PCPRF}$ is a tuple of PPT algorithms* $(\mathsf{KeyGen}_{\mathcal{PCPRF}}, \mathsf{PrefixCons})$ *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 [21].

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

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

Fig. 2: Construction of Trapdoor Permutation

3 TDP from IO in poly loss

We consider trapdoor permutation with pseudorandom sampling which is a weakened notion than the traditional uniform sampling. We refer the reader to [8] for a formal definition.

3.1 Construction of Trapdoor Permutations

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

Theorem 1. *Assuming the existence of one-way permutations and indistinguishablity obfuscation against polytime adversaries there exists a trapdoor permutation with pseudorandom sampling.*

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 2.

Due to lack of space we give the proof of security in the full version of our paper [20].

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

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 $(i-1, \sigma'_1, \sigma'_2, \cdots, \sigma'_\kappa)$.

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

4 Trapdoor Permutation from FE

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

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

Fig. 4: $\mathsf{Exp}_{\mathcal{A},b,\mathsf{wPRS}}$

Definition 2. *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 4.
- ***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 2. The requirement of pseudorandom sampling considered in Bitanksy et al.'s work [8] 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 [14].

In this section, we construct trapdoor permutation satisfying the Definition 2 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 2. *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 [19] 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.

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

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

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

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

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

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

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

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 [19]) 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.

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 $\mathcal{PCPRF}$.
3. An injective length doubling pseudorandom generator $\mathsf{InjPRG} : \{0,1\}^{\kappa/8} \rightarrow \{0,1\}^{\kappa/4}$
4. A length doubling Left half injective pseudorandom generator $\mathsf{LeftInjPRG} : \{0,1\}^{\kappa} \rightarrow \{0,1\}^{2\kappa}$

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 5.

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.

Due to shortage of space, we defer the proof of Theorem 2 to the full version of the paper [20].

5 Universal Samplers

Intuitively, a universal sampler, defined by Hofheinz et al. [23] is a box that takes as input the description of a sampling procedure, and outputs a fresh-looking sample according to the sampling procedure. The difficulty is that we want the box to be public code, and that every user, when they run the sampler on a particular procedure, gets the same result. Moreover, we want the sample to appear as if it were a fresh random sample.

5.1 Definition

A *Universal Sampler* consists of an algorithm Setup that takes as input a security parameter κ (encoded in unary) and a size bound $\ell(\cdot)$, random

- $\mathsf{KeyGen}(1^\kappa)$:
  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 $\mathsf{V}_\phi^i := S_i$, $\mathsf{V}_\phi = \bigcup_{i\in[\tau+\kappa]} \mathsf{V}_\phi^i$ 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 $(\mathsf{PK}_i^1, \mathsf{MSK}_i^1) \leftarrow \mathsf{FE.Setup}(1^\kappa)$ 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 $\mathsf{FSK}_i^1 \leftarrow \mathsf{FE.KeyGen}(\mathsf{MSK}_i^1, F^1_{i,\mathsf{PK}_{i+1}^1,\Pi_1})$ and $\mathsf{FSK}_{\tau+\kappa+1}^1 \leftarrow \mathsf{FE.KeyGen}(\mathsf{MSK}_{\tau+\kappa+1}^1, G^1_{v,\Lambda_1,w})$, where $F^1_{i,\mathsf{PK}_{i+1}^1,\Pi_1}$ and $G^1_{v,\Lambda_1,w}$ are circuits described in Figure 6.
  7. Let $c_\phi^1 = \mathsf{FE.Enc}_{\mathsf{PK}_1}(\phi, \mathsf{V}_\phi, \mathsf{W}_\phi, \widetilde{K}_\phi, 0^{p(\kappa)}, 0)$.
  8. The Public Key PK is given by $(\{\mathsf{FSK}_i^1\}_{i\in[\tau+\kappa+1]}, c_\phi^1)$ 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 $c^1_{x_{[i-1]}\|0}, c^1_{x_{[i-1]}\|1} := \mathsf{FE.Dec}(\mathsf{FSK}_i^1, c^1_{x_{[i-1]}})$.
  2. Obtain $d_x = ((\psi_1, \ldots, \psi_{\tau+\kappa}), (\beta_j, \ldots, \beta_{\tau+\kappa}))$ as output of $\mathsf{FE.Dec}(\mathsf{FSK}^1_{\tau+\kappa+1}, c_x^1)$. Here, $j = f(x)$. Recall from Section **??** 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 $(x+1, \sigma_1', \cdots, \sigma_{\tau+\kappa}')$.

Fig. 5: Construction of TDP from $\mathcal{FE}$

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

- TDP_{SK}^{-1} : 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 $\mathsf{U}_\phi^i := S_i$ and $\mathsf{U}_\phi = \bigcup_{i\in[\tau+\kappa]} \mathsf{U}_\phi^i$.
  2. For every $i \in [\tau+1]$, choose $(\mathsf{PK}_i^2, \mathsf{MSK}_i^2) \leftarrow \mathsf{FE.Setup}(1^\kappa)$.
  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 $\mathsf{FSK}_i^2 \leftarrow \mathsf{FE.KeyGen}(\mathsf{MSK}_i^2, F^2_{i,\mathsf{PK}^2_{i+1},\Pi_2})$ and $\mathsf{FSK}_{\tau+1}^2 \leftarrow \mathsf{FE.KeyGen}(\mathsf{MSK}_{\tau+1}^2, G^2_{\Lambda_2})$ where $F^2_{i,\mathsf{PK}^2_{i+1},\Pi_2}, G^2_{\Lambda_2}$ are described in Figure 7.
  5. Let $c_\phi^2 \leftarrow \mathsf{FE.Enc}_{\mathsf{PK}_1^2}(\phi, U_\phi, K, \overline{K}, 0^{p(\kappa)}, 0)$.
  6. The sampler circuit has $\{\mathsf{FSK}_i^2\}_{i\in[\tau+1]}$ and c_ϕ^2 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 $c^2_{pk_{[i-1]}\|0}, c^2_{pk_{[i-1]}\|1} := \mathsf{FE.Dec}(\mathsf{FSK}_i^2, c^2_{pk_{[i-1]}})$.
  2. Obtain $(pk, h_{pk}) = (pk, (pk, \rho, \rho_1, \cdots, \rho_{\tau+\kappa}))$ as output of $\mathsf{FE.Dec}(\mathsf{FSK}_{\tau+1}^2, c_{pk}^2)$.
  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 $(pk \| K_{pk}, \sigma_1, \cdots, \sigma_{\tau+\kappa})$.

Fig. 5: Construction of TDP from $\mathcal{FE}$

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. [23] 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. *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.*

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

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 **??**).
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 **??** 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.

Fig. 6: Circuits for simulating Public Key.

– 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* 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*

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

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

1. If (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 **??**).
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 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}.

Fig. 7: Circuits for simulating Sampler

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* 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.*

5.2 Construction from FE

In this section, we will construct Universal Samplers that satisfies Definition 3 from polynomially hard, compact Functional Encryption and Prefix Constrained Pseudorandom Function (which is implied by Functional Encryption).

Theorem 3. *Assuming the existence of selective secure, single key, compact public key functional encryption there exists an Universal Sampler scheme satisfying Definition 3.*

Our Construction. The formal description our construction appears in Figure 8.

Due to lack of space, we give the proof of security in the full version of the paper [20].

6 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 [15], our protocol has no trusted setup.

6.1 Definition

A multiparty key exchange protocol consists of:

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 9.
  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.

Fig. 8: Setup and Evaluating the Sampler

– $\mathsf{Publish}(\kappa)$ 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:

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.

Fig. 9: Circuits for simulating Public Key.

Definition 4. *A non-interactive multiparty key exchange protocol is statically secure if the following distributions are indistinguishable for any polynomial-sized set S:*

$\{\mathsf{pv}_j\}_{j\in S}, k$ *where* $(\mathsf{sv}_j, \mathsf{pv}_j) \leftarrow \mathsf{Publish}(\kappa) \forall j \in S, k \leftarrow \mathsf{KeyGen}(\{\mathsf{pv}_j\}_{j\in S}, s_1, 1)$ *and*
$\{\mathsf{pv}_j\}_{j\in S}, k$ *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 [11, 15, 13] require. Boneh and Zhandry [12] 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 [18]. 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 [1]. Recently Khurana, Rao and Sahai in [25] constructed a key exchange protocol supporting unbounded number of users based on indistinguishability obfuscation and a tool called as *somewhere statistically binding hash functions* [24]. Here, we get an unbounded protocol based on *functiona encryption* only, and without using complexity leveraging.

6.2 Construction

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

- $\mathsf{Publish}(\kappa)$. Run $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{PK.KeyGen}(\kappa)$. Also run the universal sampler setup algorithm $\mathsf{Sampler} \leftarrow \mathsf{Setup}(\kappa, \ell, t)$ where output size ℓ and circuit size bound t will be decided later. Output $\mathsf{pv} = (\mathsf{pk}, \mathsf{Sampler})$ as the public value and keep $\mathsf{sv} = \mathsf{sk}$ as the secret value.
- $\mathsf{KeyGen}(\{(\mathsf{pk}_j, \mathsf{Sampler}_j)\}_{j \in S}, \mathsf{sk}_i, i)$. Interpret S as the set $[1, n]$ for $n = |S|$, choosing some canonical ordering for the users in S (say, the lexicographic order of their public values). Define $\mathsf{Sampler} = \mathsf{Sampler}_1$. Define $C_{\mathsf{pk},\mathsf{pk}'}$ for two public keys $\mathsf{pk}, \mathsf{pk}'$ to be the circuit that samples a random $(\mathsf{sk}'', \mathsf{pk}'') \leftarrow \mathsf{PK.KeyGen}(\kappa)$, then encrypts sk'' under both pk and pk', obtaining encryptions c and c' respectively, and then outputs (pk'', c, c').
  Let $D_{\mathsf{pk},\mathsf{pk}'}$ be a similar circuit that samples a uniformly random string sk'' in the key space of $\mathcal{PKE}$, encrypts sk'' to get c, c' as before, and outputs $(0, c, c')$ where 0 is a string of zeros with the same length as a public key for $\mathcal{PKE}$. Let ℓ the the length of (pk'', c, c') and let t be the size of $C_{\mathsf{pk},\mathsf{pk}'}$ (which we will assume is at least as large as $D_{\mathsf{pk},\mathsf{pk}'}$).
  Next, define $\mathsf{pk}'_2 = \mathsf{pk}_1$, and recursively define $(\mathsf{pk}'_{j+1}, c_j, c'_j) = \mathsf{Sampler}(C_{\mathsf{pk}_j,\mathsf{pk}'_j})$ for $j = 2, \ldots, n-1$. Define sk'_{j+1} to be the secret key corresponding to pk'_{j+1}, which is also the secret key encrypted in c_j, c'_j. Finally, define $(0, c_n, c'_n) = \mathsf{Sampler}(D_{\mathsf{pk}_n,\mathsf{pk}'_n})$, and define sk'_{n+1} to be the secret key encrypted in c_n, c'_n.
  First, it is straightforward that given $\{\mathsf{pk}_j\}_{j \in [n]}$ and $\mathsf{Sampler}$, it is possible to compute $\mathsf{pk}'_j, c_j, c'_j$ for all $k \in [2, n]$. Thus anyone, including an eavesdropper, can compute these values.
  Next, we claim that if additionally given secret keys sk_j *or* sk'_j, it is possible to compute sk'_{j+1}. Indeed, sk'_{j+1} can be computed by decrypting c_j (using sk_j) or decrypting c'_j (using sk'_j). By iterating, it is

possible to compute sk'_k for every $k > j$. This 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. *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.*

Due to lack of space, we give the proof of Theorem 4 in the full version of the paper [20].

Acknowledgments. Research supported in part from DARPA/ARL SAFEWARE Award W911NF15C0210, DARPA/ARO Award W911NF15C0226, AFOSR Award FA9550-15-1-0274, NSF CRII Award 1464397, AFOSR YIP Award and research grants by the Okawa Foundation and Visa Inc. The views expressed are those of the author and do not reflect the official policy or position of the funding agencies.

References

1. 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`.
2. 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.
3. Prabhanjan Ananth and Abhishek Jain. Indistinguishability obfuscation from compact functional encryption. In *CRYPTO*, pages 308–326, 2015.
4. 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.
5. 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.
6. Charles H. Bennett. Time/space trade-offs for reversible computation. *SIAM J. Comput.*, 18(4):766–776, 1989.
7. Nir Bitansky, Omer Paneth, and Alon Rosen. On the cryptographic hardness of finding a nash equilibrium. In *FOCS*, 2015.
8. Nir Bitansky, Omer Paneth, and Daniel Wichs. Perfect structure on the edge of chaos. *TCC*, 2016.

9. Nir Bitansky and Vinod Vaikuntanathan. Indistinguishability obfuscation from functional encryption. In *FOCS*, 2015.
10. 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.
11. Dan Boneh and Alice Silverberg. Applications of multilinear forms to cryptography. Cryptology ePrint Archive, Report 2002/080, 2002. `http://eprint.iacr.org/2002/080`.
12. 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.
13. 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.
14. Shimon Even, Oded Goldreich, and Abraham Lempel. A randomized protocol for signing contracts. *Commun. ACM*, 28(6):637–647, 1985.
15. 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.
16. 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.
17. Sanjam Garg, Craig Gentry, Shai Halevi, and Mark Zhandry. Fully secure functional encryption from multilinear maps. In *TCC*, 2016.
18. 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.
19. 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.
20. Sanjam Garg, Omkant Pandey, Akshayaram Srinivasan, and Mark Zhandry. Breaking the sub-exponential barrier in obfustopia. *IACR Cryptology ePrint Archive*, 2016:102, 2016.
21. Oded Goldreich, Shafi Goldwasser, and Silvio Micali. How to construct random functions. *J. ACM*, 33(4):792–807, 1986.
22. Shafi Goldwasser and Yael Tauman Kalai. Cryptographic assumptions: A position paper. Cryptology ePrint Archive, Report 2015/907, 2015. `http://eprint.iacr.org/2015/907`.
23. 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`.

24. 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.
25. 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.
26. Huijia Lin. Indistinguishability obfuscation from DDH on 5-linear maps and locality-5 prgs. *IACR Cryptology ePrint Archive*, 2016:1096, 2016.
27. 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.
28. 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.
29. 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.
30. 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.