Homomorphic Indistinguishability Obfuscation and its Applications

Kaartik Bhushan[1], Venkata Koppula[2], and Manoj Prabhakaran[1]

[1] IIT Bombay
[2] IIT Delhi

Abstract. In this work, we propose the notion of *homomorphic indistinguishability obfuscation* (HiO) and present a construction based on subexponentially-secure iO and one-way functions. An HiO scheme allows us to convert an obfuscation of circuit C to an obfuscation of $C' \circ C$, and this can be performed obliviously (that is, without knowing the circuit C). A naïve solution would be to obfuscate $C' \circ \mathsf{iO}(C)$. However, if we do this for k hops, then the size of the final obfuscation is exponential in k. HiO ensures that the size of the final obfuscation remains polynomial after repeated compositions. As an application, we show how to build function-hiding hierarchical multi-input functional encryption and homomorphic witness encryption using HiO.

1 Introduction

The goal of code obfuscation [5] is to compile programs such that the compiled version preserves functionality, but is "maximally unintelligible." Compared to traditional cryptographic notions like encryption which keep data locked away in a non-functional way, obfuscation draws its power from allowing the obfuscated code to be executable publicly. However, obfuscation does take away some functionality that unobfuscated code provides, namely, the ability to modify the code. In this work, we investigate the notion of homomorphic obfuscation that seeks to retain some of the functionality of modifying the code, while providing the protection that obfuscation provides.

A version of this question was first studied by Ananth et al. [3] and Garg and Pandey [18], who introduced the notions of patchable obfuscation and incremental obfuscation respectively (and these were further explored in [1]). In these works, one uses a secret key associated with the obfuscated program to modify it. While this is a remarkable feature, it is not comparable to the original feature of unobfuscated code whereby anyone can *publicly modify* the code.

On the other hand, allowing anyone to modify the program in any manner they wish runs contrary to the very notion of obfuscation. Indeed, by trying to change the (unobfuscated) code one bit at a time, one would often be able to recover the entire code. As such, the only reasonable notion of homomorphic obfuscation may appear to be to allow the use of a private key.

In this work, we take a different view of homomorphic obfuscation, that prioritizes the public nature of code modification. But to not contradict the spirit of obfuscation, which allows only black-box access to the obfuscated code, we require that the nature of modification should also be black-box. That is, the code modifications we seek will invoke the original code as a black-box. An immediate solution then, is to first construct a program that implements the modification by invoking the given obfuscated code rather than the original code (which is not available), and then obfuscating this new program (since the modification itself needs to be hidden). While this is perhaps reasonable for a single round of code modification, note that it involves nesting obfuscations, and the size of the code grows exponentially as multiple rounds of code modification are applied homomorphically to the original program.

This leaves us with the core technical challenge tackled in this work:

> *A homomorphic obfuscation scheme should allow one to iteratively apply black-box modifications to an obfuscated program, retaining the security of the resulting programs as well as their polynomial efficiency.*

The security property we shall focus on is *indistinguishability obfuscation* (iO) [5, 19], which is by far the most standard notion of obfuscation in the literature. Hence the security property we shall be interested in for our primitive – called Homomorphic iO (HiO) – is as follows: Consider two obfuscated programs $\mathcal{O}'_1, \mathcal{O}'_2$ that are obtained after several (but equal number of) homomorphic transformations applied to two (possibly different) obfuscated programs $\mathcal{O}_1, \mathcal{O}_2$; if $\mathcal{O}'_1$ and $\mathcal{O}'_2$ happen to be functionally equivalent, then they should be indistinguishable from each other. Note that the pair of original programs, the intermediate programs, or the transformations, need not be functionally equivalent.

A Motivating Application. Before proceeding further, we briefly discuss a motivating application of HiO. Suppose Alice receives an obfuscation (iO) of a program that signs its inputs using a built-in signing key, under a puncturable signature scheme.[3] Now suppose she would like to hand out this signing key to Bob after puncturing it at a few points. This new program can be implemented as a black-box transformation of the original (unobfuscated) program. If the obfuscation scheme is an HiO scheme, Alice can create an obfuscated version of the desired program by acting homomorphically on the obfuscated program that she received, and hand it over to Bob. And further, Bob can repeat the same with Carol, and so forth. At any point, someone receiving this obfuscated program cannot learn anything about the set of punctured points other than its size and what they can learn from oracle access.

In the sequel, we shall formalize and realize this primitive as a new primitive called *Puncture-Hiding Incrementally Puncturable Signatures* (PIPS).

Input-Based Output Transformations. For the ease of exposition, we shall first focus on a restricted form of black-box transformations: We may transform a function f into a function that maps x to $g(x, f(x))$, where g is an arbitrary function. Note that for the example above of PIPS, input-based output transformations are already sufficient. Later, in Section 7 we generalize this to a circuit structure, where each node of the circuit is a program.

1.1 Our Contributions

Our contributions are three-fold:

- We define the notion of *Homomorphic indistinguishability Obfuscation (*HiO*)* which extends iO with a feature to incrementally modify the obfuscated program publicly (i.e., without a secret key). Indistinguishability of two obfuscated programs holds as long as functional equivalence holds at the end of the two equally long chains of modifications, even if it does not at intermediate levels (Section 4).
- We present a construction for HiO assuming subexponentially-secure iO for all circuits and subexponentially-secure one-way functions (Section 5).
- Finally, we present several applications of HiO:
  - *Function-hiding Hierarchical-MiFE.* While both Hierarchical-MiFE [23] and function-hiding MiFE [2, 11] have been constructed in the literature, for the first time, by leveraging HiO, we give a single construction that offers both these properties for MiFE (Section 6.1).
  - *Circuit-hiding Homomorphic Witness Encryption.* Combining the features of Fully Homomorphic Encryption and Witness Encryption, we introduce the notion of Homomorphic Witness Encryption, and provide a construction using HiO (Section 6.2).
  - *Puncture-hiding Incrementally Puncturable Signatures.* We formalize the motivating example from the Introduction in the form of this primitive and provide a construction using HiO (Section 6.3).

Extensions. We also generalize HiO so that the blackbox transformations supported is not limited to a chain of circuits. In particular, we can support blackbox transformations in which a program can invoke more than one obfuscated program (which may in turn be the result of a similar transformation), thereby

[3] One scenario where iO of such a program is interesting is the following. One may want to delegate the ability to sign all strings of a certain length, except a few secret strings (e.g., certain sensitive keys). While these bad strings can be punctured out of the signing key, the punctured signing key will reveal them. On the other hand, this program is functionally equivalent to another program that only carries point obfuscations of the bad strings along with the the unpunctured signing key. By obfuscating this program then, one keeps the bad strings hidden, while also making sure that they cannot be signed.

yielding a tree or DAG composition structure. We show that our construction extends to this setting as well, in Section 7.

While we restrict ourselves to circuits in this paper, we note here that our techniques can be implemented in similar ways to the Turing Machine as well as RAM models of computation. For example, in order to build HiO for TMs, one would simply need to start with iO for TMs [9, 13, 15, 25] and use the remanining tools as is done in this paper.

1.2 Related Work

The notion of iO was introduced in [5] and has since been proven to be very powerful, with several applications in cryptography and complexity theory [7, 19, 23, 27]. [8] show that any iO scheme can be converted to one in which the size of the obfuscated circuit grows linearly with the size of the input circuit. This can be used in our construction in Section 5 to get linearly growing size of the obfuscated chain in terms of sum of sizes of each unobfuscated circuit in the chain. The notion of homomorphisms in cryptography has mainly been considered with encryption [21, 26] but also for other primitives like zero-knowledge [4], signatures [24] and secret-sharing schemes [10] among others. Non-compact function-private FHE schemes have been considered in [17, 22].

A variant of obfuscation called *patchable* iO was introduced in [3]. The major difference between our notion and their's is that in their notion, the original obfuscator needs to provide "patches", generated using the randomness used to compute the initial obfuscation, which can then be applied by anybody to the obfuscated circuit. As a result, the authors of that work call that notion as *semi-private homomorphic obfuscation* whereas we are interested in obtaining *public homomorphic obfuscation*. Same goes for the notion of *incremental* iO given in [18]. Both these works fall under the umbrella of *cryptography with updates* [1].

In this work, we give applications of HiO to add homomorphisms to the notions of function-hiding MiFE and witness encryption. The notion of Multi-input Functional Encryption (MiFE) was introduced in [23]. This paper also mentioned the notion of Hierarchical-MiFE in a paragraph. In [2], the authors showed a transformation from any secret-key MiFE scheme to a function-hiding MiFE scheme using ideas from [11]. The authors of [12] showed how to transform any public-key FE scheme to a hierarchical FE scheme, in the single-input regime. The notion of *witness encryption* was introduced in the work of [20].

2 Technical Overview

Before describing our idea for the main construction, we first note that existing ideas in the literature might not be enough to build HiO. For example, consider the FE to iO transformation of [2]. The way they achieve this is by building a technique for arity amplification for a secret-key MiFE scheme and then using the MiFE to iO transformation of [23] to achieve iO. The key ingredient for doing said arity amplification is the ability to convert any MiFE scheme to a function-hiding version. If we were to use similar ideas for building HiO starting from hierarchical FE, we would also need an analogous conversion from any hierarchical MiFE scheme to a function-hiding hierarchical MiFE. It is not clear how to do this and in fact we show that this could be seen as an application of HiO.

We also point out another approach, which directly relies on the following feature of MiFE. Consider a multi-input function $U(x_1, \ldots, x_n)$ (each input being a single bit); given a function key for U, a ciphertext for each position $i \leq s$ for $z_i \in \{0,1\}$, and two ciphertexts for each position $i > s$ for both 0 and 1, one can evaluate $U(z_1, \ldots, z_s, y_{s+1}, \ldots, y_n)$ for any choice of $y_{s+1}, \ldots, y_n$. Setting U to be a function which accepts circuits $C_1, \ldots, C_k$ and a string x and outputs $C_k \circ \cdots C_1(x)$, one can turn this into a HiO scheme (see Appendix A). But this construction, which could be seen as an extension of the iO construction from MiFE in [23], has two serious limitations: Firstly, there is an *a priori* limit k on the number of homomorphic transformations that can be applied that is set at the time of creating the first obfuscation. Secondly, the size of even the first obfuscation, which encodes only C_1, is as large as the final obfuscation (indeed, as the transformations are applied the size slightly decreases each time).

The above construction could be termed "levelled HiO." In the following we focus on the full-fledged "unlevelled" version of HiO.

2.1 Initial Idea

We start with a rather simplistic idea that avoids nested iO: to increment an obfuscation $\hat{C}_1$ of C_1 to that of $C_2 \circ C_1$, simply output $(\hat{C}_1, \hat{C}_2)$, where $\hat{C}_2$ is an independent obfuscation of C_2. This simplistic idea would retain indistinguishability between two chains of circuits $(C_1^0, \ldots, C_k^0)$ and $(C_1^1, \ldots, C_k^1)$ only if for each i, C_i^0 and C_i^1 are functionally equivalent. One reason why this obfuscation does not meet the security goals of HiO is that the intermediate results of a computation will be revealed. A natural approach to fix this issue would be to use encryption to hide intermediate values, as shown in Figure 1. Furthermore, we will need each obfuscated program (other than the first one) to identify and reject an input unless it has been generated by the previous one. If we try to use a randomized (authenticated) encryption scheme, we would need to use probabilistic iO [14] as the underlying circuit is now randomized; unfortunately, this approach fails due to the seemingly unavoidable technical limitations of the known constructions of probabilistic iO [14]. An alternative would be to use a deterministic encoding algorithm that offers hiding and authentication similar to an encryption scheme, in a manner that facilitates the requisite hybrid arguments in the security proof. As it turns out, the primitive Asymmetrically Constrainable Encryption (ACE) introduced by [15] fits our requirements.

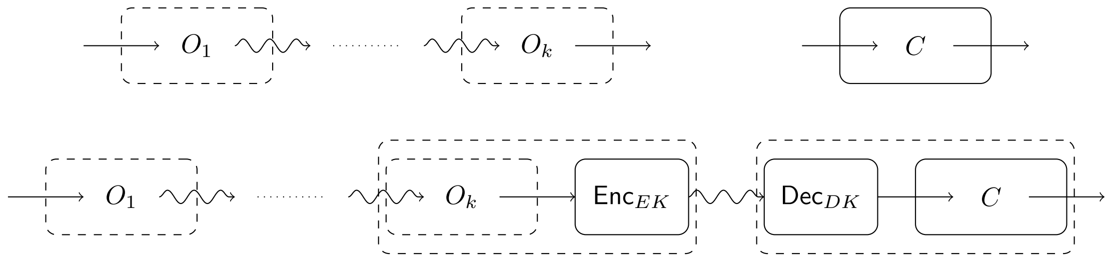

Fig. 1. Basic idea for extending a chain of obfuscated circuits $(O_1, \ldots, O_k)$ in our framework. Dashed boundary represents an obfuscated circuit while solid boundary is for standard circuits. Curved arrows denote ciphertexts while straight arrows denote plaintext values. Top figure represents the initial chain with C being the new circuit to be added. Bottom figure represents the final chain $(O_1, \ldots, O_{k-1}, \hat{O}_k, O_{k+1})$, obtained after sampling an encryption key-pair (EK, DK).

Asymmetrically Constrainable Encryption The notion of asymmetrically constrainable encryption (ACE) was proposed by Canetti et al [15] for a similar problem - succinct garbling of Turing machines. In this primitive, we have a setup algorithm outputting a master secret key, which can be used for generating encryption and decryption keys. Given a master secret key and a set S, we can generate a constrained encryption (resp. decryption) key $EK\{S\}$ (resp. $DK\{S\}$). The set S specifies the 'forbidden' region, where encryption/decryption does not work. The encryption algorithm is deterministic; it takes as input an encryption key and a message, and outputs a ciphertext. Similarly, the deterministic decryption algorithm takes as input a decryption key and a ciphertext, and outputs a message. For correctness, we require that for any two sets S, S', if a message $m \notin S \cup S'$, then encryption of m using $EK\{S\}$, when decrypted using $DK\{S'\}$, produces m. In addition to the encryption being determinisitc, the ciphertexts are also 'unique' — if a message m is encrypted using two different encryption keys $EK\{S\}$ and $EK\{S'\}$, then the resuting ciphertexts are identical (provided $m \notin S \cup S'$), and if two ciphertexts decrypt to the same value, then they must be equal. [4] For security, we require two properties. First, the punctured decryption keys should hide the constraint set. More formally, an adversary should not be able to distinguish between $DK\{S_0\}$ and $DK\{S_1\}$, even when it is given various ciphertexts and encryption keys (provided the ciphertexts are for messages $m \notin S_0 \Delta S_1$ and the encryption key is for a set U such that $S_0 \Delta S_1 \subseteq U$). Second, we require semantic security for the

[4] This primitive has a few other correctness properties, which are described formally in Section 3.1.

encryption — an adversary should not be able to distinguish between encryption of m_0 and m_1, even if it is given various ciphertexts and encryption/decryption keys (provided the constraint sets for the encryption and decryption keys contain both m_0 and m_1).

2.2 Warm-Up: A Weaker HiO

First we shall describe our construction and, as a warm-up, analyze it for a weaker security gurarantee, where indistinguishability is guaranteed only when functional equivalence holds at each level of the chain of compositions, instead of just at the end of the two chains. That is, in the security experiment the adversary is allowed to only send two sequences of circuits $\left(C_i^0\right)_{i\leq k}$, $\left(C_i^1\right)_{i\leq k}$ such that for all $x \in \{0,1\}^n$ and all $i \leq k$,

$$C_i^0\left(C_{i-1}^0\left(\dots C_1^0(x)\right)\dots\right) = C_i^1\left(C_{i-1}^1\left(\dots C_1^1(x)\right)\dots\right)$$

It receives the homomorphic obfuscation of either $\left(C_i^0\right)_{i\leq k}$ or $\left(C_i^1\right)_{i\leq k}$, and must guess which one was obfuscated.

Our construction is simple to describe: obfuscation of $C_k \circ \dots \circ C_1$ consists of iO obfuscations of circuits G_i which ACE-decrypt their input using one key, evaluate C_i on the result, and then ACE-encrypt the outcome using another key. The exceptions are the first and last circuits in the sequence, which omit the decryption and the encryption steps respectively, say, A_1 and B_k respectively. The construction in fact involves one level of nesting of iO: since G_i needs to be created without having direct access to the key used by G_{i-1}, it is in fact created from the obfuscation of B_i (see Figure 1).

Now, to analyze this construction in the simplified setting, where we assume that functional equivalence holds at each level in the chain, the "encryption" aspect of ACE is not critical (since the intermediate results are identical in the two chains, and hence need not be hidden), but the authentication aspect is.

To argue security, we use a hybrid argument which goes over all input strings (thus leading to an exponential loss in security). Let the j^{th} hybrid be where each circuit D_i in the chain has two circuits C_i^0 and C_i^1 hardwired in it, and uses C_i^0 for all inputs $x \geq j$ and C_i^1 for all others. In order to move to the next hybrid, we need to make the switch for input $x = j$ from using C_i^0 to C_i^1, in every circuit in the chain. In order to do this, we will hardwire the output when $x = j$ in the first circuit D_0 as $(j, y_0^* = C_0^0(j) = C_0^1(j))$. In order to do the same for the second circuit D_1, we will need to puncture the decryption key DK_0 on the set $\{(j, \neq y_0^*)\}$ and then use the fact that such a key can never decrypt to a tuple belonging to this set. Furthermore, we would first need to puncture the corresponding encryption key EK_0 on some superset (say $\{(j,*)\}$) for the argument to go through. Once we have punctured DK_0, we could hardwire the corresponding ciphertext output in D_1 for $x = j$ as $(j, y_1^* = C_1^0 \circ C_0^0(j) = C_1^1 \circ C_0^1(j))$.

Proceeding similarly, we would have hardwired the outputs for $x = j$ in each circuit in the chain. Then starting from the last circuit, we can start switching to using the circuit C_i^1 for $x = j$. Such a switch would be possible due to functional equivalence at that level. This would have to be followed by unpuncturing the decryption key first and then doing the same for the corresponding encryption key. In this way, we would have made the desired switch in $O(k)$ hybrids.

2.3 Full-fledged HiO

Now we consider the actual definition when functional equivalence is only assumed at the end. While the construction remains the same as outlined above, we need a more careful proof of security. Thankfully, the ACE scheme provides us with all the desired properties for us to complete our reasoning for this case too. In particular, note that we never used ciphertext indistinguishability in the previous situation since there wasn't any need to hide the intermediate outputs, but we would need that property in this situation.

Proceeding similarly as before, we can start hardwiring the outputs $\alpha_i^0 = \mathsf{Enc}(EK_i, (j, y_i^0 = C_i^0 \circ \dots \circ C_0^0(j)))$ for $x = j$ inside each circuit D_i, for $i \in \{0, \dots, k-1\}$. In order to do this, we would also have punctured the encryption keys EK_i on the set $U = \{(j, \cdot)\}$ and the decryption keys DK_i on the set $S_i^0 = \{(j, \neq y_i^0)\}$. One could similarly hardwire the output (j, y_k^0) inside the circuit D_k for $x = j$. Note that $y_k^0 = y_k^1$ as functional equivalence holds at the very end. Our goal now is to switch to using C_k^1 for input $x = j$ inside the circuit D_k. Here we run into an issue. The decryption key DK_{k-1} is currently punctured on the set $S_{k-1}^0 = \{(j, \neq y_{k-1}^0)\}$. However, in order to make the switch to C_k^1, we would need to change the puncturing to the set $S_{k-1}^1 = \{(j, \neq y_{k-1}^1 = C_{k-1}^1 \circ \cdots \circ C_0^1(j))\}$, and then use safety of constrained decryption for functional equivalence. We cannot directly switch the puncturing from S_{k-1}^0 to S_{k-1}^1 since some message of the set difference $\{(j, y_{k-1}^0), (j, y_{k-1}^1)\}$ is available as a ciphertext in the system.

To solve this, we try to change the puncturing of DK_{k-1} from S_{k-1}^0 to U so that we could switch ciphertexts easily. These two decryption keys only differ on the tuple (j, y_{k-1}^0). We will have to handle this case outside the decryption process inside circuit D_k. This is possible due to the uniqueness of ciphertext property of the ACE scheme. In particular, only the ciphertext α_{k-1}^0 could decrypt to such a tuple[5]. We can hardwire this ciphertext inside D_k to give the same output as before and use decryption only for other ciphertexts. This way we can make the puncturing change to U without affecting functional equivalence. Now we can switch the hardwired ciphertexts inside D_{k-1} and D_k from α_{k-1}^0 to $\alpha_{k-1}^1 = \mathsf{Enc}(EK_{k-1}, y_{k-1}^1)$ using ciphertext indistinguishability. This is followed by changing puncturing of DK_{k-1} from U to S_{k-1}^1, removing the hardwired ciphertext α_{k-1}^1 from D_k and then switching to C_k^1 for $x = j$. The rest of the argument goes along the ideas presented earlier.

2.4 Application: Function-Hiding Hierarchical-MiFE

As an illustration of the power of HiO, we use it to give the first construction of a function-hiding Hierarchical-MiFE scheme. The notion of MiFE, introduced in [23], is a stronger functional encryption primitive allowing multiple parties to encrypt different messages which could be decrypted together using a function key to get the output. The authors showed that secret-key MiFE suffices to construct iO, which in turn is sufficient to construct even public-key MiFE, thus implying that these two notions are equivalent. Moreover, the authors mentioned a seemingly stronger notion of Hierarchical-MiFE which allows anyone with access to a function key sk_f to further *delegate* this with any function f' to obtain a new key $\mathsf{sk}_{f' \circ f}$ which could be used to compute the function $f' \circ f$. Furthermore, one could delegate any number of times.

While the authors did not give any construction of the primitive, similar notions for the weaker primitive of functional encryption have been considered previously. In particular, [12] showed that any FE scheme could be used to construct a hierarchical FE scheme. Similar ideas could be used to construct Hierarchical-MiFE from any MiFE scheme. We consider the even stronger notion of *function-hiding* Hierarchical-MiFE where the function key hides the function(s) that are being computed by that key. While function-hiding is not natural for functional encryption in the public-key setting, it is well-motivated in the secret-key setting. Indeed, in the non-hierarchical setting, [2] showed that any MiFE can be used to construct a function-hiding MiFE scheme. But their techniques do not extend to the hierarchical setting, and no construction has been provided for function-hiding Hierarchical-MiFE yet.

We show that any HiO scheme could be used to amplify a function-hiding MiFE scheme to a function-hiding Hierarchical-MiFE scheme. The construction is quite simple: instead of outputting a standard function-key, we output an HiO obfuscation of a circuit D which has sk_f hardwired inside it and decrypts the input ciphertexts $(\mathsf{ct}_1, \ldots, \mathsf{ct}_n)$ using this key to produce its output. For delegation, we use HiO composition to compose the current function key (which is an obfuscated circuit) with a new function f' (or its circuit representation $C_{f'}$) to get a new obfuscated circuit. Decryption would evaluate this obfuscated circuit on the ciphertexts and get the output.

To argue security of this construction, one starts with the real $\mathsf{H\text{-}MiFE}$experiment where the challenger chooses bit b as 0 and provides outputs to the adversary. In particular, for function query $\left((f_0^0, \ldots, f_k^0), (f_0^1, \ldots, f_k^1)\right)$,

[5] While uniqueness of ciphertexts is defined w.r.t. an unpunctured decryption key, we can prove that the statement still holds in our situation in presence of the punctured key $DK_{k-1}\{S_{k-1}^0\}$. This further uses the equivalence of constrained decryption and safety of constrained decryption properties. In particular, for messages in the punctured set, the punctured key always outputs $\perp$ while for other messages, it behaves identically to the unpunctured key and hence uniqueness of ciphertexts can be used.

the challenger computes $\mathsf{sk}_f^{(k)}$ where

$$\begin{aligned}\mathsf{sk}_f^{(0)} &\leftarrow \mathsf{HiO.Obfuscate}(1^\lambda, D_{\mathsf{sk}_{f_0^0}}),\\ \mathsf{sk}_f^{(j)} &\leftarrow \mathsf{HiO.Compose}(\mathsf{sk}_f^{(j-1)}, C_{f_j^0}) \ \text{ for } j \in \{1,\ldots,k\},\end{aligned}$$

where the circuits D and C have been described previously. Using HiO security, we could directly switch to

$$\begin{aligned}\mathsf{sk}_f^{(0)} &\leftarrow \mathsf{HiO.Obfuscate}(1^\lambda, C_{\mathsf{Id}}),\\ \mathsf{sk}_f^{(j)} &\leftarrow \mathsf{HiO.Compose}(\mathsf{sk}_f^{(j-1)}, C_{\mathsf{Id}}) \ \text{ for } j \in \{1,\ldots,k-1\},\\ \mathsf{sk}_f^{(k)} &\leftarrow \mathsf{HiO.Compose}(\mathsf{sk}_f^{(k-1)}, D_{\mathsf{sk}_{f_k^0\circ\cdots\circ f_0^0}}),\end{aligned}$$

where Id denotes the identity function, as the two chains are functionally equivalent. Now the only parameters of interest are the MiFE key $\mathsf{sk}_{f_k^0\circ\cdots\circ f_0^0}$ and the MiFE ciphertexts which could be switched from $b = 0$ to $b = 1$ directly while using the function-hiding property of MiFE. This concludes our overview.

3 Preliminaries

In this section we recall the definitions of cryptographic primitives employed in our constructions.

3.1 Asymmetrically Constrainable Encryption (ACE)

This primitive was defined and constructed by Canetti et al. [15].

Let $\mathcal{M}$ denote the message space. An asymmetrically constrainable encryption scheme over $\mathcal{M}$ consists of five polynomial-time algorithms Setup, GenEK, GenDK, Enc and Dec, described as follows. Setup, GenEK and GenDK are randomized algorithms, but Enc and Dec are deterministic.

– **Setup:** $\mathsf{Setup}(1^\lambda)$ is a randomized algorithm that takes as input a security parameter λ, and outputs a secret key SK.
– **(Constrained) Key Generation:** Let $S \subset \mathcal{M}$ be any set whose membership is decidable by a circuit C_S. That is, C_S maps $\mathcal{M} \to \{0,1\}$ and $C_S(m) = 1$ if and only if $m \in S$.
  - $\mathsf{GenEK}(SK, C_S)$ takes as input the secret key SK of the scheme and the description of circuit C_S for an admissible set S. It outputs an encryption key $EK\{S\}$. We write EK to denote $EK\{\emptyset\}$.
  - $\mathsf{GenDK}(SK, C_S)$ also takes as input the secret key SK of the scheme and the description of circuit C_S for an admissible set S. It outputs a decryption key $DK\{S\}$. We write DK to denote $DK\{\emptyset\}$.

  Unless mentioned otherwise, we will only consider admissible sets $S \subset \mathcal{M}$.
– **Encryption:** $\mathsf{Enc}(EK', m)$ is a deterministic algorithm that takes as input an encryption key EK' (that may be constrained) and a message $m \in \mathcal{M}$ and outputs a ciphertext c or reject symbol $\perp$.
– **Decryption:** $\mathsf{Dec}(DK', c)$ is a deterministic algorithm that takes as input a decryption key DK' (that may be constrained) and a ciphertext c and outputs a message $m \in \mathcal{M}$ or the reject symbol $\perp$.

Correctness. An ACE scheme is correct if the following properties hold:

1. *Correctness of Decryption*: For all n, all $m \in \mathcal{M}$, all sets S, S' such that $m \notin S \cup S'$,

$$\Pr\left[m' = m \;\middle|\; \begin{array}{c} SK \leftarrow \mathsf{Setup}(1^\lambda),\\ EK \leftarrow \mathsf{GenEK}(SK, C_{S'}),\\ DK \leftarrow \mathsf{GenDK}(SK, C_S),\\ c := \mathsf{Enc}(EK, m),\\ m' := \mathsf{Dec}(DK, c) \end{array} \right] = 1.$$

Informally, this says that $\mathsf{Dec} \circ \mathsf{Enc}$ is the identity on messages which are in neither of the punctured sets.

2. *Equivalence of Constrained Encryption*: For any message $m \in \mathcal{M}$ and any sets $S, S' \subset \mathcal{M}$ with m not in the symmetric difference $S \triangle S'$,

$$\Pr\left[c = c' \;\middle|\; \begin{array}{c} SK \leftarrow \mathsf{Setup}(1^\lambda), \\ EK \leftarrow \mathsf{GenEK}(SK, C_S), \\ EK' \leftarrow \mathsf{GenEK}(SK, C_{S'}), \\ c := \mathsf{Enc}(EK, m), \\ c' := \mathsf{Enc}(EK', m) \end{array} \right] = 1.$$

Informally, this says that punctured encryption keys are functionally the same except on the difference of the sets at which they are punctured.

3. *Unique Ciphertexts*: For all strings c and c',

$$\Pr\left[c = c' \;\middle|\; \begin{array}{c} SK \leftarrow \mathsf{Setup}(1^\lambda), \\ DK \leftarrow \mathsf{GenDK}(SK, \emptyset), \\ \mathsf{Dec}(DK, c) = \mathsf{Dec}(DK, c') \neq \bot \end{array} \right] = 1.$$

Informally, this says that two different ciphertexts cannot decrypt to the same message.

4. *Safety of Constrained Decryption*: For all strings c, all sets $S \subset \mathcal{M}$,

$$\Pr\left[\mathsf{Dec}(DK\{S\}, c) \in S \;\middle|\; \begin{array}{c} SK \leftarrow \mathsf{Setup}(1^\lambda), \\ DK\{S\} \leftarrow \mathsf{GenDK}(SK, C_S), \end{array} \right] = 0.$$

This says that a punctured $DK\{S\}$ will never decrypt to a message in S. Furthermore, for all messages $m \in S$,

$$\Pr\left[\mathsf{Dec}(DK\{S\}, c) = \bot \;\middle|\; \begin{array}{c} SK \leftarrow \mathsf{Setup}(1^\lambda), \\ EK \leftarrow \mathsf{GenEK}(SK, \emptyset), \\ DK\{S\} \leftarrow \mathsf{GenDK}(SK, C_S), \\ c := \mathsf{Enc}(EK, m) \end{array} \right] = 1.$$

This says that for ciphertexts encoding messages belonging to the punctured set, the punctured decryption key always outputs $\bot$.

5. *Equivalence of Constrained Decryption*: For any subsets S and S' of $\mathcal{M}$, if $\mathsf{Dec}(DK\{S\}, c) = m \neq \bot$ and $m \notin S'$, then $\mathsf{Dec}(DK\{S'\}, c) = m$. Informally, this says that punctured decryption keys differ in functionality only when necessary.

Security of Constrained Decryption. Intuitively, this property says that for any two sets S_0 and S_1, no adversary can distinguish between the constrained keys $DK\{S_0\}$ and $DK\{S_1\}$, even given additional auxilliary information in the form of a constrained encryption key EK' and ciphertexts $c_1, \ldots, c_t$. To rule out trivial attacks, EK' is constrained at least on $S_0 \triangle S_1$. Similarly, each c_i is an encryption of a message $m_i \notin S_0 \triangle S_1$.

Formally, we describe security of constrained decryption as a multi-stage game between an adversary $\mathcal{A}$ and a challenger.

– *Setup*: $\mathcal{A}$ choose sets S_0, S_1, U s.t. $S_0 \triangle S_1 \subseteq U \subseteq \mathcal{M}$ and sends their circuit descriptions (C_{S_0}, C_{S_1}, C_U) to the challenger. $\mathcal{A}$ also sends arbitrary polynomially many messages $m_1, \ldots, m_t$ such that $m_i \notin S_0 \triangle S_1$. The challenger chooses a bit $b \in \{0, 1\}$ and computes the following:
  1. $SK \leftarrow \mathsf{Setup}(1^\lambda)$
  2. $DK\{S_b\} \leftarrow \mathsf{GenDK}(SK, C_{S_b})$
  3. $EK \leftarrow \mathsf{GenEK}(SK, \emptyset)$

4. $c_i := \mathsf{Enc}(EK, m_i)$, for every $i \in [t]$
5. $EK\{U\} \leftarrow \mathsf{GenEK}(SK, C_U)$

Finally, it sends the tuple $(EK\{U\}, DK\{S_b\}, c_1, \ldots, c_t)$ to $\mathcal{A}$.

– *Guess*: $\mathcal{A}$ outputs a bit $b' \in \{0,1\}$.

The advantage of $\mathcal{A}$ in this game is defined as $\mathsf{Adv}_{\mathcal{A}} = \Pr[b' = b] - \frac{1}{2}$. We require that $\mathsf{Adv}_{\mathcal{A}} \leq \mathsf{negl}(\lambda)$.

Selective Ciphertext Indistinguishability. Intuitively, this property says that no adversary can distinguish between encryptions of m_0 from encryptions of m_1, even given additional auxilliary information. The auxilliary information corresponds to constrained encryption and decryption keys EK', DK', as well as ciphertexts $c_1, \ldots, c_t$. In order to rule out trivial attacks, EK' and DK' should both be punctured on at least $\{m_0, m_1\}$, and none of $c_1, \ldots, c_t$ should be an encryption of m_0 or m_1.

Formally, we require that for all sets $S, U \subset \mathcal{M}$, for all $m_0^*, m_1^* \in S \cap U$, and for all $m_1, \ldots, m_t \in \mathcal{M} \setminus \{m_0^*, m_1^*\}$,

$$(EK\{S\}, DK\{U\}, c_0^*, c_1, \ldots, c_t) \approx (EK\{S\}, DK\{U\}, c_1^*, c_1, \ldots, c_t),$$

when we sample $SK \leftarrow \mathsf{Setup}(1^\lambda), EK \leftarrow \mathsf{GenEK}(SK, \emptyset), EK\{S\} \leftarrow \mathsf{GenEK}(SK, C_S), DK\{U\} \leftarrow \mathsf{GenDK}(SK, C_U), c_b^* \leftarrow \mathsf{Enc}(EK, m_b^*)$, and $c_i \leftarrow \mathsf{Enc}(EK, m_i)$.

The authors of [15] gave a construction of this primitive assuming iO and one-way functions, as mentioned in the following theorem.

Theorem 1 ([15]). *Assuming subexponentially-secure indistinguishability obfuscation for all circuits and subexponentially-secure one-way functions, there exists a secure ACE scheme.*

3.2 Multi-Input Functional Encryption (MiFE)

The notion of MiFE was introduced in [23]. A private-key MiFE scheme for n-ary functions in the space $\mathcal{F}$ consists of the following algorithms:

– $\mathsf{Setup}(1^\lambda, n)$ is a PPT algorithm that takes as input the security parameter λ and the function arity n, and outputs a master secret key msk.
– $\mathsf{KeyGen}(\mathsf{msk}, f)$ is a PPT algorithm that takes as input the master secret key msk and a function $f \in \mathcal{F}$, and outputs a functional key sk_f.
– $\mathsf{Enc}(\mathsf{msk}, m, i)$ is a PPT algorithm that takes as input the master secret key msk, a message m and an index $i \in [n]$, and outputs a ciphertext ct.
– $\mathsf{Dec}(\mathsf{sk}_f, \{\mathsf{ct}^{(i)}\}_{i \in [n]})$ is a deterministic algorithm that takes as input a functional key sk_f and n ciphertexts $\{\mathsf{ct}^{(i)}\}_{i \in [n]}$, and outputs a value y.

We need the following properties from the scheme:

– **Correctness**: There exists a negligible function[6] $\mathsf{negl}(\cdot)$ such that for all sufficiently large $\lambda \in \mathbb{N}$, every n-ary function $f \in \mathcal{F}$, and input tuple $(x_1, \ldots, x_n)$, it holds that

$$\Pr\left[\mathsf{Dec}(\mathsf{sk}_f, \{\mathsf{ct}^{(i)}\}_{i \in [n]}) \neq f(x_1, \ldots, x_n)\right] \leq \mathsf{negl}(\lambda)$$

where the probability is taken over the random choices of the following algorithms: $\mathsf{msk} \leftarrow \mathsf{Setup}(1^\lambda), \mathsf{sk}_f \leftarrow \mathsf{KeyGen}(\mathsf{msk}, f)$ and $\forall i \in [n] : \ \mathsf{ct}^{(i)} \leftarrow \mathsf{Enc}(\mathsf{msk}, x_i, i)$.

– **Security**: A secret-key MiFE scheme, for n-ary functions in $\mathcal{F}$, is $(1, q_{\mathsf{msg}})$-secure if for any PPT adversary $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all sufficiently large $\lambda \in \mathbb{N}$, the advantage of $\mathcal{A}$ is

$$\mathsf{Adv}_{\mathcal{A}}^{\mathrm{MiFE}} = \left|\Pr[\mathsf{Exp}_{\mathcal{A}}^{\mathrm{MiFE}}(1^\lambda, 0) = 1] - \Pr[\mathsf{Exp}_{\mathcal{A}}^{\mathrm{MiFE}}(1^\lambda, 1) = 1]\right| \leq \mathsf{negl}(\lambda),$$

where for each $b \in \{0,1\}$ and $\lambda \in \mathbb{N}$, the experiment $\mathsf{Exp}_{\mathcal{A}}^{\mathrm{MiFE}}(1^\lambda, b)$ is defined below:

[6] Alternately, we could also talk about a notion which satisfies perfect correctness, in which case negl would be the zero-function.

1. **Challenge message queries**: $\mathcal{A}$ submits q_{msg} queries, $\left\{\left((x^j_{1,0}, x^j_{1,1}), \ldots, (x^j_{n,0}, x^j_{n,1})\right)\right\}_{j\in[q_{\mathsf{msg}}]}$, to the challenger C.
2. C computes $\mathsf{msk} \leftarrow \mathsf{Setup}(1^\lambda)$. It then computes $\mathsf{ct}^j_i \leftarrow \mathsf{Enc}(\mathsf{msk}, x^j_{i,b})$ for all $i \in [n]$ and $j \in [q_{\mathsf{msg}}]$. The challenger then sends $\left\{\left(\mathsf{ct}^j_1, \ldots, \mathsf{ct}^j_n\right)\right\}_{j\in[q_{\mathsf{msg}}]}$ to the adversary $\mathcal{A}$.
3. **Function query**: $\mathcal{A}$ submits a function query f to C. If $f \notin \mathcal{F}$, the challenger aborts. Otherwise, it computes $\mathsf{sk}_f \leftarrow \mathsf{KeyGen}(\mathsf{msk}, f)$, and sends sk_f to $\mathcal{A}$.
4. If there exists a sequence[7] $(j_1, \ldots, j_n)_{j_i\in[q_{\mathsf{msg}}]}$ such that
$$f(x^{j_1}_{1,0}, \ldots, x^{j_n}_{n,0}) \neq f(x^{j_1}_{1,1}, \ldots, x^{j_n}_{n,1}),$$
then the output of the experiment is set to $\perp$. Otherwise, the output of the experiment is set to b', where b' is the output of $\mathcal{A}$.

The authors of [23] showed that this primitive is equivalent to any iO scheme.

Theorem 2 ([23]). *Assuming indistinguishability obfuscation for all circuits and one-way functions, there exists a secure MiFE scheme.*

3.3 Indistinguishability Obfuscation

An iO scheme consists of the following algorithms:

– $\mathsf{Obfuscate}(1^\lambda, C)$: The algorithm $\mathsf{Obfuscate}$ takes as input a security parameter λ and a circuit C, and outputs an obfuscated circuit $\hat{C}$.
– $\mathsf{Eval}(\hat{C}, x)$: The algorithm Eval takes as input an obfuscated ciruit $\hat{C}$ and an input string x, and outputs a string y.

The scheme must satisfy the following properties:

– **Functionality**: For any positive integer λ, circuit C, and input x,
$$\Pr\left[\hat{C} \leftarrow \mathsf{Obfuscate}(1^\lambda, C) \mid \mathsf{Eval}(\hat{C}, x) = C(x)\right] = 1.$$
– **Indistinguishability**: For any positive integer λ and circuits C^0, C^1 such that $|C^0| = |C^1|$ and $C^0 \equiv C^1$, then it holds that
$$\mathsf{Obfuscate}(1^\lambda, C^0) \approx \mathsf{Obfuscate}(1^\lambda, C^1).$$
– **Efficiency**: There exists a polynomial function poly such that for any positive integer λ and circuit C, if $\hat{C} \leftarrow \mathsf{Obfuscate}(1^\lambda, C)$ then it holds that $|\hat{C}| \leq \mathsf{poly}(|C|, \lambda)$.

4 Homomorphic iO

A scheme HiO is said to be a *homomorphic indistinguishability obfuscation* scheme if it consists of the following algorithms:

– $\mathsf{Obfuscate}(1^\lambda, C)$: The algorithm $\mathsf{Obfuscate}$ takes as input a security parameter λ and a circuit C, and outputs an obfuscated circuit $\hat{C}$.

[7] We need this complicated check in MiFE type primitives because the adversary could mix up ciphertexts from different queries, and decrypt them using the functional key, without any trouble.

– $\mathsf{Eval}(\hat{C}, x)$: The algorithm Eval takes as input an obfuscated ciruit $\hat{C}$ and an input string x, and outputs a string y.
– $\mathsf{Compose}(\hat{C}, C')$: The algorithm $\mathsf{Compose}$ takes as input an obfuscated circuit $\hat{C}$ and a circuit C', and outputs an obfuscated circuit $\hat{C}'$.

The scheme must satisfy the following properties:

– **Homomorphic Functionality**: For any positive integers λ, $k \geq 0$, circuits $C_0, \ldots, C_k$, and input x,

$$\Pr[\ \mathsf{Eval}(\hat{C}, x) = C_k \circ \cdots \circ C_0(x)\ ] = 1,$$

where the probability is taken over the randomness used in algorithms $\mathsf{Compose}$ and $\mathsf{Obfuscate}$ in the computation of

$$\hat{C} \leftarrow \mathsf{Compose}(\cdots \mathsf{Compose}(\mathsf{Obfuscate}(1^\lambda, C_0), C_1), \cdots), C_k).$$

– **Homomorphic Indistinguishability**: For any positive integers λ, $k \geq 0$, any circuits $C_0^0, \ldots, C_k^0, C_0^1, \ldots, C_k^1$, such that $|C_i^0| = |C_i^1|$ for all $i \in \{0, \ldots, k\}$ and

$$C_k^0 \circ \cdots \circ C_0^0 \equiv C_k^1 \circ \cdots \circ C_0^1,$$

then it holds that

$$\mathsf{Compose}(\cdots \mathsf{Compose}(\mathsf{Obfuscate}(1^\lambda, C_0^0), C_1^0), \cdots), C_k^0)$$

$$\approx$$

$$\mathsf{Compose}(\cdots \mathsf{Compose}(\mathsf{Obfuscate}(1^\lambda, C_0^1), C_1^1), \cdots, C_k^1).$$

– **Homomorphic Efficiency**: There exists a polynomial function poly such that for any positive integers λ, $k \geq 0$, and circuits $C_0, \ldots, C_k$ if

$$\hat{C} \leftarrow \mathsf{Compose}(\cdots \mathsf{Compose}(\mathsf{Obfuscate}(1^\lambda, C_0), C_1), \cdots), C_k),$$

then it holds that $|\hat{C}| \leq \mathsf{poly}(|C_0|, \ldots, |C_k|, \lambda)$.

Remark. We note that a stronger definition could have been stated where we demand that a homomorphically obfuscated circuit is indistinguishable from a fresh obfuscation of the underlying final circuit. In other words, for any $k \geq 0$, and for any circuits $C_0, \ldots, C_k$, it should hold that

$$\mathsf{Compose}(\cdots \mathsf{Compose}(\mathsf{Obfuscate}(1^\lambda, C_0), C_1), \cdots), C_k) \approx \mathsf{Obfuscate}(1^\lambda, C_k \circ \cdots \circ C_0).$$

This definition is stronger in the sense that it implies the original one stated above. We note that there is a trivial way to convert any HiO scheme satisfying the original definition to one satisfying this stronger definition. The way this could be achieved is to define the new Obfuscate algorithm to break its input circuit into all its individual parts, in case the input circuit could be written as a chain of multiple circuits, and then use the original obfuscation algorithm on the first circuit followed by composing obfuscations on the remaining circuits in order.

5 HiO from iO and $\mathcal{ACE}$

In this section, we show a construction of HiO from subexponentially-secure iO and subexponentially-secure one-way functions. The way we solve this is by interleaving encryption-decryption algorithms of an ACE scheme in between the two composed circuits and separately obfuscating both.

5.1 Our Construction

Our construction uses a standard iO scheme $\mathsf{iO} = (\mathsf{iO.Obfuscate}, \mathsf{iO.Eval})$ and an ACE scheme $\mathcal{ACE} = (\mathsf{Setup}, \mathsf{GenEK}, \mathsf{GenDK}, \mathsf{Enc}, \mathsf{Dec})$.

```
Obfuscate$(1^\lambda, C)$:
1. Compute $\hat{D} \leftarrow \mathsf{iO.Obfuscate}(1^\lambda, C)$ and output $\hat{D}$.
Eval$(\hat{D}, x)$:
1. Parse $\hat{D}$ as $(\hat{D}_0, \ldots, \hat{D}_k)$, for some $k \geq 0$.
2. Set $y_0 := x$. For $j = 0$ to $k$ : compute $y_{j+1} := \mathsf{iO.Eval}(\hat{D}_j, y_j)$.
3. If $k = 0$, output $y_1$. Otherwise, parse $y_{k+1}$ as $(x, y)$ and output $y$.
Compose$(\hat{D}, C')$:
1. Parse $\hat{D}$ as $(\hat{D}_0, \ldots, \hat{D}_k)$, for some $k \geq 0$.
2. Sample $SK \leftarrow \mathsf{Setup}(1^\lambda)$. Further, sample keys $EK \leftarrow \mathsf{GenEK}(SK, \emptyset)$ and $DK \leftarrow \mathsf{GenDK}(SK, \emptyset)$.
3. If $k = 0$, compute $\hat{D}'_k \leftarrow \mathsf{iO.Obfuscate}(1^\lambda, A[\hat{D}_k, EK])$ where $A$ has been described in Figure 2. Otherwise, compute $\hat{D}'_k \leftarrow \mathsf{iO.Obfuscate}(1^\lambda, G[\hat{D}_k, EK])$ where $G$ has been described in Figure 3.
4. Compute $\hat{D}_{k+1} \leftarrow \mathsf{iO.Obfuscate}(1^\lambda, B[C', DK])$, where $B$ has been described in Figure 4.
5. Output $(\hat{D}_0, \ldots, \hat{D}_{k-1}, \hat{D}'_k, \hat{D}_{k+1})$.
```

```
Hardcoded-values: $\hat{C}, EK$.
Input: $x$.
1. Compute $y := \mathsf{iO.Eval}(\hat{C}, x)$.
2. Compute $\alpha := \mathsf{Enc}(EK, (x, y))$.
3. Output $\alpha$.
```

Fig. 2. Circuit A

```
Hardcoded-values: $\hat{C}, EK$.
Input: $\alpha$.
1. If $\alpha = \perp$, then output $\perp$. Otherwise, proceed as follows.
2. Compute $t := \mathsf{iO.Eval}(\hat{C}, \alpha)$.
3. If $t = \perp$, output $\perp$. Otherwise, proceed as follows.
4. Compute $\alpha' := \mathsf{Enc}(EK, t)$.
5. Output $\alpha'$.
```

Fig. 3. Circuit G

Hardcoded-values: C, DK.

Input: α.

1. If $\alpha = \perp$, then output $\perp$. Otherwise, proceed as follows.
2. Compute $t := \mathsf{Dec}(DK, \alpha)$.
3. If $t = \perp$, then output $\perp$. Otherwise, parse t as (x, y) and proceed as follows.
4. Compute $y' := C(y)$.
5. Output (x, y').

Fig. 4. Circuit B

Correctness. Correctness of the above scheme follows in a straightforward manner from correctness of iO and that of $\mathcal{ACE}$.

Efficiency. Our scheme is efficient because we are not using k layers of iO to compose a chain of k circuits. In fact, we are using 2 layers of iO for the intermediate circuits and only a single layer for the end-points of an obfuscated chain. Moreover, using the iO scheme of [8], we can get the size of the composed circuit as $O(|C_0| + \ldots + |C_k|) + \mathsf{poly}(\lambda)$.

5.2 Proof of Security

Theorem 3 (HiO)**.** *Assuming a subexponentially-secure indistinguishability obfuscation scheme for all circuits and a subexponentially-secure ACE scheme, the scheme given in Section 5 is a secure* HiO *scheme supporting arbitrary number of hops.*

Proof: Let $\mathcal{X}$ denote the input space supported by the obfuscation scheme. The notation $\hat{D}$ is used to denote an obfuscated version of the circuit D. The proof follows by a hybrid argument. We will consider the argument for k hops. Hence, given circuits $C_0^0, \ldots, C_k^0, C_0^1, \ldots, C_k^1$, such that

$$C_k^0 \circ \cdots \circ C_0^0 \equiv C_k^1 \circ \cdots \circ C_0^1$$

then it should be the case that

$$\mathsf{Compose}(\cdots \mathsf{Compose}(\mathsf{Obfuscate}(1^\lambda, C_0^0), C_1^0), \cdots), C_k^0)$$

$$\approx$$

$$\mathsf{Compose}(\cdots \mathsf{Compose}(\mathsf{Obfuscate}(1^\lambda, C_0^1), C_1^1), \cdots), C_k^1)$$

► **Hybrid FIRST.** This is the first distribution in the above indistinguishability equation. First we define $k+1$ circuits $D_0, \ldots, D_k$[8] as follows:
   - Let $SK \leftarrow \mathsf{Setup}(1^\lambda)$. Further, sample $EK_i \leftarrow \mathsf{GenEK}(SK, \emptyset)$ and $DK_i \leftarrow \mathsf{GenDK}(SK, \emptyset)$ for $i \in [0, k]$. D_0 is $A[\hat{C}_0^0, EK_0]$, where $\hat{C}$ denotes an obfuscation of the circuit C.
   - For $i \in \{1, \ldots, k-1\}$, D_i is the same as $G[\hat{F}_i, EK_i]$ where F_i is $B[C_i^0, DK_{i-1}]$.
   - D_k is $B[C_k^0, DK_{k-1}]$.

   The hybrid output consists of $k+1$ obfuscated circuits $\hat{D}_0, \ldots, \hat{D}_k$ such that

$$\hat{D}_i \leftarrow \mathsf{iO.Obfuscate}(1^\lambda, D_i),$$

   for all $i \in \{0, \ldots, k\}$.

[8] All circuit descriptions are padded appropriately so that the corresponding circuit sizes are same across all hybrids.

▶ **Hybrid H_0'.** Let this be the hybrid where we change every circuit to a functionally equivalent but simpler form:

- D_0 is now $A_2[C_0^0, EK_0]$ where A_2 has been described in Figure 5.
- For $i \in \{1, \ldots, k-1\}$, D_i is now $G_2[C_i^0, DK_{i-1}, EK_i]$, where G_2 is described in Figure 6.
- D_k is same as $B[C_k^0, DK_{k-1}]$.

```
Hardcoded-values: $C, EK$.
Input: $x$.
1. Compute $y := C(x)$.
2. Compute $\alpha := \mathsf{Enc}(EK, (x, y))$.
3. Output $\alpha$.
```

Fig. 5. Circuit A_2

```
Hardcoded-values: $C, DK$, $EK'$.
Input: $\alpha$.
1. If $\alpha = \perp$, then output $\perp$. Otherwise, proceed as follows.
2. Compute $t := \mathsf{Dec}(DK, \alpha)$.
3. If $t = \perp$, then output $\perp$. Otherwise, parse $t$ as $(x, y)$ proceed as follows.
4. Compute $y' := C(y)$.
5. Compute $\alpha' := \mathsf{Enc}(EK', (x, y'))$.
6. Output $\alpha'$.
```

Fig. 6. Circuit G_2

Roughly, we have opened up the internal obfuscated circuits so that there are standard circuits inside every obfuscated circuit in the chain. Indistinguishability between FIRST and H_0' follows by using iO correctness and iO indistinguishability.

Lemma 1. *Assuming security and correctness of* iO*, hybrids* ***FIRST*** *and* H_0' *are computationally indistinguishable.*

Proof: We can prove indistinguishability via a sequence of sub-hybrids where in the i^{th} sub-hybrid, we change the circuit $\hat{D}_i$ in the chain from that in FIRST to that in H_0. Let us focus on the first sub-hybrid which only changes D_0 from $A[\hat{C}_0^0, EK_0]$ to $A_2[C_0^0, EK_0]$. Functional equivalence for these 2 circuits follows from iO correctness. Indistinguishability of the 2 sub-hybrids follows from iO security. One can similarly argue for the remaining sub-hybrids. □

Next, we present $|\mathcal{X}| + 1$ hybrids H_j for each $j \in \{0, \ldots, |\mathcal{X}|\}$.

▶ **Hybrid H_j.** For every $j \in \{0, \ldots, |\mathcal{X}|\}$, we will define the hybrid H_j. It consists of a chain of $k+1$ obfuscated circuits $\hat{D}_0, \ldots, \hat{D}_k$ defined as follows:

- D_0 is now $A_3[j, C_0^0, C_0^1, EK_0]$ where A_3 has been described in Figure 7.
- For $i \in \{1, \ldots, k-1\}$, D_i is now $G_3[j, C_i^0, C_i^1, DK_{i-1}, EK_i]$, where G_3 is described in Figure 8.
- D_k is same as $B_2[j, C_k^0, C_k^1, DK_{k-1}]$, where B_2 has been described in Figure 9.

```
Hardcoded-values: $j, C^0, C^1$, $EK$.
Input: $x$.
1. If $x \geq j$, compute $y := C^0(x)$. Else, compute $y := C^1(x)$.
2. Compute $\alpha := \mathsf{Enc}(EK, (x, y))$.
3. Output $\alpha$.
```

Fig. 7. Circuit A_3

```
Hardcoded-values: $j, C^0, C^1$, $DK, EK'$.
Input: $\alpha$.
1. If $\alpha = \perp$, output $\perp$. Otherwise, proceed as follows.
2. Compute $t := \mathsf{Dec}(DK, \alpha)$.
3. If $t = \perp$, output $\perp$. Otherwise, parse $t$ as $(x, y)$ and proceed as follows.
4. If $x \geq j$, compute $y' := C^0(y)$. Else, compute $y' := C^1(y)$.
5. Compute $\alpha' := \mathsf{Enc}(EK', (x, y'))$.
6. Output $\alpha'$.
```

Fig. 8. Circuit G_3

Lemma 2. *Assuming* iO *is a secure indistinguishability obfuscator, hybrids* H_0' *and* H_0 *are computationally indistinguishable.*

Proof: Note that the only change in these 2 hybrids is that each D_i has an extra circuit C_i^1 hardwired in it which is never being used. Hence, functional equivalence of the corresponding circuits would imply indistinguishability by iO security. □

Lemma 3. *For any* $j < |cX|$, *hybrids* H_j *and* H_{j+1} *are computationally indistinguishable.*

The proof of this lemma is included in Section 5.2.1.

▶ **SECOND.** This is the second distribution in the main indistinguishability equation. First we define $k+1$ circuits $D_0, \ldots, D_k$ as follows:

- Let $SK \leftarrow \mathsf{Setup}(1^\lambda)$. Further, sample $EK_i \leftarrow \mathsf{GenEK}(SK, \emptyset)$ and $DK_i \leftarrow \mathsf{GenDK}(SK, \emptyset)$ for $i \in [0, k]$. D_0 is $A[\hat{C}_0^1, EK_0]$.

```
Hardcoded-values: $j, C^0, C^1$, $DK$.
Input: $\alpha$.
1. If $\alpha = \perp$, output $\perp$. Otherwise, proceed as follows.
2. Compute $t := \mathsf{Dec}(DK, \alpha)$.
3. If $t = \perp$, output $\perp$. Otherwise, parse $t$ as $(x, y)$ and proceed as follows.
4. If $x \geq j$, compute $y' := C^0(y)$. Else, compute $y' := C^1(y)$.
5. Output $(x, y')$.
```

Fig. 9. Circuit B_2

- For $i \in \{1, \ldots, k-1\}$, D_i is the same as $G[\hat{F}_i, EK_i]$ where F_i is $B[C_i^1, DK_{i-1}]$.
- D_k is $B[C_k^1, DK_{k-1}]$.

The hybrid output consists of $k+1$ obfuscated circuits $\hat{D}_0, \ldots, \hat{D}_k$ such that

$$\hat{D}_i \leftarrow \mathsf{iO.Obfuscate}(1^\lambda, D_i),$$

for all $i \in \{0, \ldots, k\}$.

Lemma 4. *Assuming the correctness and security of* iO*, the hybrids $H_{|\mathcal{X}|}$ and SECOND are computationally indistinguishable.*

Proof: This proof is similar to the indistinguishability of hybrids FIRST and H_0'. □

This concludes the proof of our main theorem.

□

5.2.1 Proof of Lemma 3

Proof: We will prove this lemma via a sequence of hybrid experiments.

– $H_j^{0,0}$**.** This is the same as H_j.
– $H_j^{0,1}$**.** In this hybrid, we change the circuit D_0 so that it has a hardwired ciphertext for $x = j$. In other words, D_0 is now $A_4[j, C_0^0, C_0^1, EK_0, \alpha_0^0]$, where α_0^0 is the hardwired ciphertext output $\mathsf{Enc}(EK_0, (j, y_0^0 = C_0^0(j)))$, and A_4 has been described in Figure 10.

Claim 1. *Assuming* iO *is a secure indistinguishability obfuscator, hybrids $H_j^{0,1}$ and $H_j^{0,0}$ are computationally indistinguishable.*

– $H_j^{0,2}$**.** In this hybrid, we change the underlying encryption key EK_0 in D_0 to now be punctured over the set $U = \{(j, \cdot)\}$.

Claim 2. *Assuming* iO *is a secure indistinguishability obfuscator, and $\mathcal{ACE}$ satisfies* Equivalence of Constrained Encryption*, hybrids $H_j^{0,1}$ and $H_j^{0,2}$ are computationally indistinguishable.*

Proof: Indistinguishability follows by using iO security as the underlying circuits are functionally equivalent. This is because for $x = j$, we are not using the encryption key and are instead using the hardwired ciphertext. For all other $x \neq j$, the underlying message does not belong to the punctured set and hence the punctured key behaves identically to the unpunctured one. □

```
Hardcoded-values: $j, C^0, C^1, EK,$ $\alpha^*$
Input: $x$.
1. If $x = j$, output $\alpha^*$. Else,
   (a) If $x > j$, compute $y := C^0(x)$. For $x < j$, compute $y := C^1(x)$.
   (b) Compute $\alpha := \mathsf{Enc}(EK, (x, y))$.
   (c) Output $\alpha$.
```

Fig. 10. Circuit A_4

– $H_j^{0,3}$**.** In this hybrid, we change the decryption key DK_0 inside D_1 so that now it is punctured over the set $S_0^0 = \{(j, \neq y_0^0)\}$.

Claim 3. *Assuming $\mathcal{ACE}$ satisfies* Security of Constrained Decryption, *hybrids $H_j^{0,2}$ and $H_j^{0,3}$ are computationally indistinguishable.*

Proof: We can make this change because

- The set U over which the corresponding encryption key EK_0 is punctured is a superset of the set $S_0^0 \triangle \emptyset = S_0^0$.
- The only available ciphertext generated using EK_0 does not belong to the set $S_0^0 \triangle \emptyset = S_0^0$ i.e., it is actually of the form (j, y_0^0).

Indistinguishability follows from the security of constrained decryption in ACE scheme. □

– $H_j^{1,1}$**.** In this hybrid, we change the circuit D_1 so that now it has a hardwired ciphertext for the case when $x = j$. In other words, D_1 is now $G_4[j, C_1^0, C_1^1, DK_0\{S_0^0\}, EK_1, \alpha_1^0]$, where $\alpha_1^0 = \mathsf{Enc}(EK_1, (j, y_1^0 = C_1^0 \circ C_0^0(j)))$, and G_4 has been described in Figure 11.

```
Hardcoded-values: $j, C^0, C^1, DK, EK',$ $\alpha^*$
Input: $\alpha$.
1. If $\alpha = \perp$, output $\perp$. Otherwise, proceed as follows.
2. Compute $t := \mathsf{Dec}(DK, \alpha)$.
3. If $t = \perp$, output $\perp$. Otherwise, parse $t$ as $(x, y)$ and proceed as follows.
4. If $x = j$, then output $\alpha^*$. Else,
   (a) If $x > j$, compute $y' := C^0(y)$. Else, compute $y' := C^1(y)$.
   (b) Compute $\alpha' := \mathsf{Enc}(EK', (x, y'))$.
   (c) Output $\alpha'$.
```

Fig. 11. Circuit G_4

Claim 4. *Assuming* iO *is a secure indistinguishability obfuscator and $\mathcal{ACE}$ satisfies* Safety of Constrained Decryption *property, hybrids $H_j^{0,3}$ and $H_j^{1,1}$ are computationally indistinguishable.*

Proof: We will have to show functional equivalence between the two circuits to argue indistinguishability via iO security.

- Whenever the punctured decryption key decrypts to an input $x \neq j$, the two circuits behave identically.
- If the input ciphertext decrypted to a message of the form $(j, \cdot)$, note that the second argument must be y_0^0 because the punctured decryption key $DK_0\{S_0^0\}$ cannot decrypt to a message belonging to the set S_0^0, by safety of constrained decryption property. In the case when the input ciphertext decrypts to (j, y_0^0), the previous circuit would also output α_1^0.

□

– $H_j^{i,l}$**.** We define hybrids $\{H_j^{i,l}\}_{i \in \{1,\ldots,k-1\}, l \in \{1,2,3\}}$ in a similar fashion as before. $l = 1$ corresponds to hardwiring a ciphertext $\alpha_i^0 = \mathsf{Enc}(EK_i, y_i^0)$ for input $x = j$ inside circuit D_i, where $y_i^0 = C_i^0 \circ \cdots \circ C_0^0(j)$. $l = 2$ corresponds to puncturing the encryption key EK_i inside circuit D_i on the set $U = \{(j, \cdot)\}$. $l = 3$ corresponds to puncturing the corresponding decryption key DK_i in the circuit D_{i+1} on the set $S_i^0 = \{(j, \neq y_i^0)\}$.

– $H_j^{k,1}$**.** We define this hybrid similar to $H_j^{i,1}$ as before i.e., we hardwire the output $(j, y_k^0 = C_k^0 \circ \cdots \circ C_0^0(j))$ inside D_k for the input $x = j$. In other words, D_k is now $B_3[j, C_k^0, C_k^1, DK_{k-1}\{S_{k-1}^0\}, y_k^0]$, where B_3 has been described in Figure 12. Indistinguishability can be argued similarly as before using safety of constrained decryption. Furthermore, note that $y_k^0 = y_k^1 = C_k^1 \circ \cdots \circ C_k^1(j)$, as functional equivalence holds at the last level.

```
Hardcoded-values: $j, C^0, C^1, DK,$ $y^*$

Input: $\alpha$.
1. If $\alpha = \perp$, output $\perp$. Otherwise, proceed as follows.
2. Compute $t := \mathsf{Dec}(DK, \alpha)$.
3. If $t = \perp$, output $\perp$. Otherwise, parse $t$ as $(x, y)$ and proceed as follows.
4. If $x = j$, output $(x, y^*)$. Else
   (a) If $x > j$, compute $y' := C^0(y)$. Else, compute $y' := C^1(y)$.
   (b) Output $(x, y')$.
```

Fig. 12. Circuit B_3

– $H_j^{k+1,1}$ In this hybrid, we make multiple changes:
- change the hardwired ciphertext α_{k-1}^0 inside D_{k-1} to $\alpha_{k-1}^1 = \mathsf{Enc}(EK_{k-1}, (j, y_{k-1}^1))$, where $y_{k-1}^1 = C_{k-1}^1 \circ \cdots \circ C_0^1(j)$,
- unpuncture the decryption key $DK_{k-1}\{S_{k-1}^0\}$ inside D_k to DK_{k-1},
- unpuncture the encryption key $EK_{k-1}\{U\}$ inside D_{k-1} to EK_{k-1},
- change D_k so that now it uses the circuit C_k^1 for input $x = j$ i.e., D_k is now $B_2[j+1, C_k^0, C_k^1, DK_{k-1}]$.

Claim 5. *Assuming* iO *is a secure indistinguishability obfuscator and* $\mathcal{ACE}$ *is a secure asymmetrically constrainable encryption scheme, hybrids* $H_j^{k,1}$ *and* $H_j^{k+1,1}$ *are computationally indistinguishable.*

Proof: Proof of this claim is provided in Appendix B. □

– $H_j^{i,l}$**.** In hybrids $\{H_j^{i,l}\}_{i \in \{1,\ldots,k-1\}, l \in \{4,5,6,7,8,9,10,11\}}$, we do the following. For $i = k-1$ to 1, consider the sub-hybrids as follows. If $l = 4$, we hardwire the output α_i^1 inside circuit D_i when the input ciphertext is α_{i-1}^0. For $l = 5$, we change puncturing of the decryption key DK_{i-1} inside D_i from the set S_{i-1}^0 to the full set U. For $l = 6$, we change the hardwired ciphertext inside both D_{i-1} and D_i from α_{i-1}^0 to

$\alpha^1_{i-1} = \mathsf{Enc}(EK_{i-1}, (j, y^1_{i-1}))$. For $l = 7$, we change puncturing of the decryption key DK_{i-1} inside D_i from the set U to the set $S^1_{i-1} = \{(j, \neq y^1_{i-1})\}$.
For $l = 8$, we remove the hardwired ciphertext α^1_{i-1} inside D_i. For $l = 9$, we change D_i so that it uses the circuit C^1_i for input $x = j$. For $l = 10$, we unpuncture the decryption key DK_{i-1} inside D_i. For $l = 11$, we unpuncture the encryption key EK_{i-1} inside D_{i-1}. Indistinguishability between all these hybrids follows similarly to before.

– $H^{0,4}_j$**.** In this hybrid, we change the first circuit D_0 so that it uses the circuit C^1_0 for the input $x = j$. Note that this hybrid is the same as H_{j+1}.

□

6 Applications

In this section, we formally define function-hiding hierarchical MiFE, homomorphic witness encryption and puncture-hiding incrementally puncturable signatures, and present constructions for all of them from HiO.

6.1 Function-Hiding Hierarchical-MiFE from MiFE and HiO

6.1.1 Definition

The notion of Hierarchical-MiFE was briefly mentioned in [23] without giving a proper definition. Here, we define the notion along with the additional property of function-hiding. Let $\mathcal{F}_0$ be a function-space for n-ary functions. Let $\mathcal{F}_1, \mathcal{F}_2, \ldots$ denote function-spaces respecting composition i.e., the function $f = f_k \circ \cdots \circ f_0$ is well-defined for all positive integers $k \geq 0$ and functions $f_0 \in \mathcal{F}_0, \ldots, f_k \in \mathcal{F}_k$. A secret-key Hierarchical-MiFE scheme H-MiFE, for function-spaces $\mathcal{F}_0, \mathcal{F}_1, \ldots$, consists of 5 algorithms: Setup, KeyGen, Enc and Dec defined like in the case of standard MiFE, and Delegate defined as follows:

– $\mathsf{Delegate}(\mathsf{sk}_f, f')$ is a PPT algorithm that takes as input a function key sk_f, which could either be the output of KeyGen or a Delegate operation itself, and another function $f' \in \mathcal{F}_k$[9] and outputs a delegated function key $\mathsf{sk}_{f' \circ f}$.

We need the following properties from the scheme:

– **Correctness**: There exists a negligible function[10] $\mathsf{negl}(\cdot)$ such that for all sufficiently large $\lambda \in \mathbb{N}$, all $k \geq 0$, functions $f_0 \in \mathcal{F}_0, f_1 \in \mathcal{F}_1, \ldots, f_k \in \mathcal{F}_k$, and input tuples $(x_1, \ldots, x_n)$, it holds that

$$\Pr\left[\mathsf{Dec}(\mathsf{sk}^{(k)}_f, \{\mathsf{ct}^{(i)}\}_{i\in[n]}) \neq f_k \circ \cdots \circ f_0(x_1, \ldots, x_n)\right] \leq \mathsf{negl}(\lambda)$$

where the probability is taken over the random choices of the following algorithms: $\mathsf{msk} \leftarrow \mathsf{Setup}(1^\lambda), \mathsf{sk}^{(0)}_f \leftarrow \mathsf{KeyGen}(\mathsf{msk}, f_0), \forall j \in [k] :\ \mathsf{sk}^{(j)}_f \leftarrow \mathsf{Delegate}(\mathsf{sk}^{(j-1)}_f, f_j)$ and $\forall i \in [n] :\ \mathsf{ct}^{(i)} \leftarrow \mathsf{Enc}(\mathsf{msk}, x_i, i)$.

– **Hierarchical Efficiency**: There exists a polynomial function poly such that for any $\lambda \in \mathbb{N}$, $k \geq 0$, all functions $f_0 \in \mathcal{F}_0, f_1 \in \mathcal{F}_1, \ldots, f_k \in \mathcal{F}_k$, if $\mathsf{msk} \leftarrow \mathsf{Setup}(1^\lambda, n)$, then it holds that

$$|\mathsf{Delegate}(\mathsf{Delegate}(\cdots \mathsf{KeyGen}(\mathsf{msk}, f_0), f_1) \cdots, f_k)| \leq \mathsf{poly}(\lambda, |f_0|, \ldots, |f_k|, n).$$

– **Security**: H-MiFEis $(1, q_{\mathsf{msg}})$-secure if for any PPT adversary $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all sufficiently large $\lambda \in \mathbb{N}$, the advantage of $\mathcal{A}$ is

$$\mathsf{Adv}^{\mathsf{H}\text{-MiFE}}_{\mathcal{A}} = \left|\Pr[\mathsf{Exp}^{\mathsf{H}\text{-MiFE}}_{\mathcal{A}}(1^\lambda, 0) = 1] - \Pr[\mathsf{Exp}^{\mathsf{H}\text{-MiFE}}_{\mathcal{A}}(1^\lambda, 1) = 1]\right| \leq \mathsf{negl}(\lambda),$$

where for each $b \in \{0, 1\}$ and $\lambda \in \mathbb{N}$, the experiment $\mathsf{Exp}^{\mathsf{H}\text{-MiFE}}_{\mathcal{A}}(1^\lambda, b)$ is defined below:

[9] Here k denotes the number of delegations performed to obtain sk_f.

[10] Alternately, we could also talk about a notion which satisfies perfect correctness, in which case negl would be the zero-function.

1. **Challenge message queries**: $\mathcal{A}$ submits q_{msg} queries, $\left\{\left((x^j_{1,0}, x^j_{1,1}), \ldots, (x^j_{n,0}, x^j_{n,1})\right)\right\}_{j\in[q_{\mathsf{msg}}]}$, to the challenger C.
2. C computes $\mathsf{msk} \leftarrow \mathsf{Setup}(1^\lambda)$. It then computes $\mathsf{ct}^j_i \leftarrow \mathsf{Enc}(\mathsf{msk}, x^j_{i,b})$ for all $i \in [n]$ and $j \in [q_{\mathsf{msg}}]$. The challenger then sends $\left\{\left(\mathsf{ct}^j_1, \ldots, \mathsf{ct}^j_n\right)\right\}_{j\in[q_{\mathsf{msg}}]}$ to the adversary $\mathcal{A}$.
3. **Function query**: $\mathcal{A}$ submits a function query $\left((f^0_0, \ldots, f^0_k), (f^1_0, \ldots, f^1_k)\right)$ to C. If $f^b_i \notin \mathcal{F}_i$ for some $i \in \{0, \ldots, k\}, b \in \{0, 1\}$, then the challenger aborts. Otherwise, it computes $\mathsf{sk}^{(0)}_f \leftarrow \mathsf{KeyGen}(\mathsf{msk}, f^b_0)$, and repeats $\mathsf{sk}^{(j)}_f \leftarrow \mathsf{Delegate}(\mathsf{sk}^{(j-1)}_f, f^b_j)$ for $j \in [k]$. Finally, it sends $\mathsf{sk}^{(k)}_f$ to $\mathcal{A}$.
4. If there exists a sequence $(j_1, \ldots, j_n)_{j_i\in[q_{\mathsf{msg}}]}$ such that

$$f^0(x^{j_1}_{1,0}, \ldots, x^{j_n}_{n,0}) \neq f^1(x^{j_1}_{1,1}, \ldots, x^{j_n}_{n,1}),$$

where $f^b = f^b_k \circ \cdots \circ f^b_0$, then the output of the experiment is set to $\perp$. Otherwise, the output of the experiment is set to b', where b' is the output of $\mathcal{A}$.

6.1.2 Construction

In this section, we show that a function-hiding MiFE scheme can be coupled with a HiO scheme to get function-hiding Hierarchical-MiFE. Here we get two types of results:

– If our aim is to build the weaker primitive where the maximum number of delegations is known during the initial key generation, then we could use levelled-HiO to achieve this. Therefore, this notion is equivalent to that of MiFE.
– If the maximum number of delegations is not known in advance, then we could use full HiO for building this primitive. This shows that after suffering an exponential loss in security, one could transform any MiFE scheme to a function-hiding Hierarchical-MiFE scheme.

We give our construction when the number of delegations are not known in advance and prove its security. In the following description, the notation $[[C]]_s$ will be used to denote a circuit C padded to size s.

Theorem 4. *Let $\mathcal{F}_0, \mathcal{F}_1, \ldots$ denote function-spaces respecting composition i.e., the function $f = f_k \circ \cdots \circ f_0$ is well-defined for all positive integers $k \geq 0$ and functions $f_0 \in \mathcal{F}_0, \ldots, f_k \in \mathcal{F}_k$. Assuming a function-hiding MiFE scheme for function-space $\mathcal{F} = \{\{f_k \circ \cdots \circ f_0\}_{f_0\in\mathcal{F}_0,\ldots,f_k\in\mathcal{F}_k}\}_{k\geq 0}$ along with full HiO, there exists a function-hiding Hierarchical-MiFE scheme for function spaces $\mathcal{F}_0, \mathcal{F}_1, \ldots$.*

Proof: Consider the construction given in Figure 13. Note that for delegating function keys, we are using the property that given an obfuscated circuit $\hat{C}$, one can figure out the number of compositions that have been performed for obtaining this circuit in the HiO scheme. Our HiO scheme satisfies this property. Let $\mathcal{K}$ denote the key-space from which master secret keys are sampled in MiFE. Also, let $\mathcal{R}$ denote the randomness-space from which random strings are derived for key-generation in MiFE. Then the value s_k is defined as follows:

$$s_k = \max_{\mathsf{msk}\in\mathcal{K}, r\in\mathcal{R}, f_0\in\mathcal{F}_0,\ldots,f_k\in\mathcal{F}_k} \left| D_{\mathsf{sk}_{f^0_k\circ\cdots\circ f^0_0}} \right|,$$

where $\mathsf{sk}_{f^0_k\circ\cdots\circ f^0_0} := \text{MiFE}.\mathsf{KeyGen}(\mathsf{msk}, f^0_k \circ \cdots \circ f^0_0; r)$ and the circuit D has been described in the construction.

Correctness. This follows in a straightforward manner.

Security. We will argue security now. For this, consider the hybrids as follows:

– H_0 : This is the real experiment $\mathsf{Exp}^{\text{H-MiFE}}_{\mathcal{A}}(1^\lambda, 0)$ (as described in Section 6.1.1) where the challenger chooses the bit b to be 0.

Let $\mathsf{HiO} = (\mathsf{Obfuscate}, \mathsf{Eval}, \mathsf{Compose})$ be an HiO scheme, and $\mathrm{MiFE} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Delegate})$ be a function-hiding MiFE scheme.

$\underline{\mathsf{Setup}(1^\lambda, n)}$:

1. Run $\mathsf{msk} \leftarrow \mathrm{MiFE}.\mathsf{Setup}(1^\lambda, n)$.
2. Output msk.

$\underline{\mathsf{KeyGen}(\mathsf{msk}, f)}$:

1. Compute $\mathsf{sk}_f \leftarrow \mathrm{MiFE}.\mathsf{KeyGen}(\mathsf{msk}, f)$.
2. Consider the circuit D_{sk_f} which
   (a) takes as input $(\mathsf{ct}_1, \ldots, \mathsf{ct}_n)$,
   (b) outputs $\mathrm{MiFE}.\mathsf{Dec}(\mathsf{sk}_f, \mathsf{ct}_1, \ldots, \mathsf{ct}_n)$.
3. Compute $\hat{C} \leftarrow \mathsf{HiO}.\mathsf{Obfuscate}(1^\lambda, [[D_{\mathsf{sk}_f}]]_{s_0})$, where s_k has been defined in the description proceeding the construction.
4. Output $\mathsf{sk} := \hat{C}$.

$\underline{\mathsf{Delegate}(\mathsf{sk}, f')}$:

1. Parse sk as $\hat{C}$.
2. Compute k from $\hat{C}$ denoting the number of hops performed so far.
3. Compute $\hat{C}' \leftarrow \mathsf{HiO}.\mathsf{Compose}(\hat{C}, [[C_{f'}]]_{s_k})$.
4. Output $\hat{C}'$.

$\underline{\mathsf{Enc}(\mathsf{msk}, x, i)}$:

1. Compute $\mathsf{ct}_i \leftarrow \mathrm{MiFE}.\mathsf{Enc}(\mathsf{msk}, x, i)$.
2. Output ct_i.

$\underline{\mathsf{Dec}(\mathsf{sk}, \mathsf{ct}_1, \ldots, \mathsf{ct}_n)}$:

1. Parse sk as $\hat{C}$.
2. Compute $y := \mathsf{HiO}.\mathsf{Eval}(\hat{C}, \mathsf{ct}_1, \ldots, \mathsf{ct}_n)$.
3. Output y.

Fig. 13. Function-Hiding Hierarchical-MiFE from Function-Hiding MiFE and HiO

- H_1 : In this hybrid, we change the function query response from the challenger. On function query $\big((f_0^0,\ldots,f_k^0),(f_0^1,\ldots,f_k^1)\big)$, the challenger was previously computing $\mathsf{sk}_f^{(k)}$ where

$$\mathsf{sk}_f^{(0)} \leftarrow \mathsf{HiO.Obfuscate}(1^\lambda,[[D_{\mathsf{sk}_{f_0^0}}]]_{s_0}),$$
$$\mathsf{sk}_f^{(j)} \leftarrow \mathsf{HiO.Compose}(\mathsf{sk}_f^{(j-1)},[[C_{f_j^0}]]_{s_j}) \text{ for } j\in\{1,\ldots,k\},$$

where the circuits D and C have been described in Figure 13. In this hybrid, we change this response as follows:

$$\mathsf{sk}_f^{(0)} \leftarrow \mathsf{HiO.Obfuscate}(1^\lambda,[[C_{\mathsf{Id}}]]_{s_0}),$$
$$\mathsf{sk}_f^{(j)} \leftarrow \mathsf{HiO.Compose}(\mathsf{sk}_f^{(j-1)},[[C_{\mathsf{Id}}]]_{s_j}) \text{ for } j\in\{1,\ldots,k-1\},$$
$$\mathsf{sk}_f^{(k)} \leftarrow \mathsf{HiO.Compose}(\mathsf{sk}_f^{(k-1)},[[D_{\mathsf{sk}_{f_k^0\circ\cdots\circ f_0^0}}]]_{s_k}),$$

where Id denotes the identity function. In other words, we have put the entire computation of the chain of functions inside the MiFE function key $\mathsf{sk}_{f_k^0\circ\cdots f_0^0} \leftarrow$ MiFE.$\mathsf{KeyGen}(\mathsf{msk}, f_k^0\circ\cdots\circ f_0^0)$ as part of the very last circuit while the rest of the circuits inside the HiO chain are dummy identity functions. Indistinguishability between the 2 hybrids follows by HiO security as the underlying chains of circuits are functionally equivalent i.e.,

$$C_{f_k^0}\circ\cdots\circ D_{\mathsf{sk}_{f_0^0}} \equiv D_{\mathsf{sk}_{f_k^0\circ\cdots\circ f_0^0}}\circ\cdots\circ C_{\mathsf{Id}_0}.$$

- H_2 : In this hybrid, we can directly switch from $\mathsf{sk}_{f_k^0\circ\cdots\circ f_0^0}$ and $\mathsf{ct}_i^j \leftarrow \mathsf{Enc}(\mathsf{msk}, x_{i,0}^j)$ for all $i\in[n]$ and $j\in[q_{\mathsf{msg}}]$, to $\mathsf{sk}_{f_k^1\circ\cdots\circ f_0^1}$ and $\mathsf{ct}_i^j \leftarrow \mathsf{Enc}(\mathsf{msk}, x_{i,1}^j)$ for all $i\in[n]$ and $j\in[q_{\mathsf{msg}}]$. This follows from function-hiding security of MiFE as

$$f^0(x_{1,0}^{j_1},\ldots,x_{n,0}^{j_n}) = f^1(x_{1,1}^{j_1},\ldots,x_{n,1}^{j_n}),$$

for all sequences $(j_1,\ldots,j_n)_{j_i\in[q_{\mathsf{msg}}]}$, where $f^b = f_k^b\circ\cdots\circ f_0^b$.
- H_3 : In this hybrid, we change back from

$$\mathsf{sk}_f^{(0)} \leftarrow \mathsf{HiO.Obfuscate}(1^\lambda,[[C_{\mathsf{Id}}]]_{s_0}),$$
$$\mathsf{sk}_f^{(j)} \leftarrow \mathsf{HiO.Compose}(\mathsf{sk}_f^{(j-1)},[[C_{\mathsf{Id}}]]_{s_j}) \text{ for } j\in\{1,\ldots,k-1\},$$
$$\mathsf{sk}_f^{(k)} \leftarrow \mathsf{HiO.Compose}(\mathsf{sk}_f^{(k-1)},[[D_{\mathsf{sk}_{f_k^0\circ\cdots\circ f_0^0}}]]_{s_k}),$$

to

$$\mathsf{sk}_f^{(0)} \leftarrow \mathsf{HiO.Obfuscate}(1^\lambda,[[D_{\mathsf{sk}_{f_0^1}}]]_{s_0}),$$
$$\mathsf{sk}_f^{(j)} \leftarrow \mathsf{HiO.Compose}(\mathsf{sk}_f^{(j-1)},[[C_{f_j^1}]]_{s_j}) \text{ for } j\in\{1,\ldots,k\},$$

in the function query response $\mathsf{sk}_f^{(k)}$. Indistinguishability again relies on HiO security. Note that this hybrid is the same as the real experiment $\mathsf{Exp}_{\mathcal{A}}^{\mathsf{H}\text{-MiFE}}(1^\lambda,1)$. This concludes our proof.

□

Combining Theorem 4 with our main result Theorem 3 and using the MiFE to iO transformation from [23], we get the following corollary.

Corollary 1. *Assuming a sub-exponentially secure MiFE scheme for all efficient functions, there exists a function-hiding Hierarchical-MiFE scheme for all efficient function-spaces respecting composition.*

6.2 Homomorphic Witness Encryption from HiO

6.2.1 Definition

The notion of witness encryption was introduced in [20] as an advanced encryption primitive where a plaintext m could be encrypted with respect to an NP statement x and decryption could be performed using a corresponding witness w for $x \in \mathcal{L}$. In this work, we consider the natural homomorphic extension of this concept which allows anyone holding a witness encryption ciphertext encrypting a plaintext m to be able to compute a ciphertext encrypting $C(m)$ for any C of their choice. Note that we are not considering the modification of the underlying NP statement x, which could also be an interesting notion in its own right.

We consider a function-hiding version for homomorphic security which guarantees that both the underlying plaintext and the circuits used for evaluation are hidden as long as the final output is the same. Moreover, this guarantee holds even when the underlying statement x belongs to the NP language $\mathcal{L}$ which is the first time that security for a witness encryption like primitive holds in this setting. In existing constructions, security has only been shown for the case when $x \notin \mathcal{L}$. Since this definition is enough for eliminating trivial constructions (like appending the circuits to standard witness encryption ciphertexts during homomorphic evaluation), we do not demand the property of efficiency which requires that the size of a homomporhically evaluated ciphertext should be independent to that of the circuit.

We define the notion of *homomorphic witness encryption (HWE)* here. An HWE scheme for an NP language $\mathcal{L}$ (with corresponding witness relation R) and message space $\mathcal{M}$, consists of 3 algorithms Enc, Dec and Eval defined as follows:

- $\mathsf{Enc}(1^\lambda, x, m)$ is a PPT algorithm that takes as input a security parameter 1^λ, an unbounded-length string x and a message $m \in \mathcal{M}$, and outputs a ciphertext ct.
- $\mathsf{Dec}(\mathsf{ct}, w)$ is a deterministic algorithm which takes as input a ciphertext ct and an unbounded length string w, and outputs a message m'.
- $\mathsf{Eval}(\mathsf{ct}, C)$ is a PPT algorithm that takes as input a ciphertext ct and a circuit C, and outputs another ciphertext ct'.

We need the following properties from the scheme:

- **Correctness**: For any security parameter λ, for any message $m \in \mathcal{M}$, and for any $x \in \mathcal{L}$ such that $R(x, w)$ holds, we have that
$$\Pr[\mathsf{Dec}(\mathsf{Enc}(1^\lambda, x, m), w) = m] = 1.$$
- **Soundness Security**: For any PPT adversary $\mathcal{A}$, and for any messages $m_0, m_1 \in \mathcal{M}$, there exists a negligible function negl such that for any $x \notin \mathcal{L}$, we have
$$\left|\Pr[\mathcal{A}(\mathsf{Enc}(1^\lambda, x, m_0)) = 1] - \Pr[\mathcal{A}(\mathsf{Enc}(1^\lambda, x, m_1)) = 1]\right| \leq \mathsf{negl}(\lambda).$$
- **Homomorphic Correctness**: For any security parameter λ, for any message $m \in \mathcal{M}$, for any $x \in \mathcal{L}$ such that $R(x, w)$ holds, for any $k \geq 1$, and for any circuits $C_1, \ldots, C_k$ whose input-output lengths are compatible for composition, we have that
$$\Pr[\mathsf{Dec}(\mathsf{Eval}(\cdots \mathsf{Eval}(\mathsf{Enc}(1^\lambda, x, m), C_1), \cdots, C_k), w) = C_k \circ \cdots \circ C_1(m)] = 1.$$
- **Homomorphic Security**: For any PPT adversary $\mathcal{A}$, for any messages $m_0, m_1 \in \mathcal{M}$, any $k \geq 1$, and for any circuits $C_1^0, \ldots, C_k^0, C_1^1, \ldots, C_k^1$ such that
$$C_k^0 \circ \cdots \circ C_1^0(m_0) = C_k^1 \circ \cdots \circ C_1^1(m_1),$$
there exists a negligible function negl such that for any x, we have
$$|\Pr[\mathcal{A}(\mathsf{ct}_0) = 1] - \Pr[\mathcal{A}(\mathsf{ct}_1) = 1]| \leq \mathsf{negl}(\lambda),$$
where
$$\mathsf{ct}_b \leftarrow \mathsf{Eval}(\cdots \mathsf{Eval}(\mathsf{Enc}(1^\lambda, x, m_b), C_1^b), \cdots, C_k^b),$$
for $b \in \{0, 1\}$.

Remark 1. This definition has a different flavor than the standard way in which traditional definitions of homomorphic encryption schemes handle the issue of eliminating trivial constructions. The standard way is to introduce the notion of *efficiency* which prevents appending the circuit to the ciphertext during homomorphic evaluation. Here, we instead rely on *homomorphic security* for disallowing such a trivial construction on top of any witness encryption, in line with how circuit-hiding FHE is defined.

Remark 2. Not only is this definition functionally stronger than the standard notion of witness encryption, but it also offers security guarantees in the case when statement x is in the language $\mathcal{L}$, a case which is not handled at all by the standard definition.

6.2.2 Construction

In this section, we show how to construct homomorphic witness encryption (Section 6.2.1) from HiO. This follows as a very straightforward extension of the construction of witness encryption from iO. Namely, a ciphertext corresponds to the obfuscation of a circuit which has x and m hardwired in it, takes a witness w as input, checks if it is valid for x via the NP relation R and outputs m if it succeeds, otherwise outputs $\perp$. Homomorphic evaluation just corresponds to HiO composition of this obfuscated circuit with the new circuit. Security follows directly from that of HiO, further showcasing its power.

Theorem 5. *Assuming an* HiO *scheme, there exists a homomorphic witness encryption scheme for any NP language $\mathcal{L}$.*

Proof: Consider the construction given in Figure 14. We will show below how this scheme satisfies all the desired properties.

```
Let HiO = (Obfuscate, Eval, Compose) be an HiO scheme.
Enc($1^\lambda, x, m$):
1. Consider the circuit $D_{x,m}$ which
    (a) takes as input a string $w$,
    (b) checks if $R(x, w) = 1$. If not, outputs $\perp$. Otherwise, outputs $m$.
2. Compute $\hat{D}_{x,m} \leftarrow \mathsf{HiO.Obfuscate}(1^\lambda, D_{x,m})$.
3. Output $\mathsf{ct} := \hat{D}_{x,m}$.
Dec(ct, $w$):
1. Parse ct as $\hat{D}$.
2. Compute $y := \mathsf{HiO.Eval}(\hat{D}, w)$.
3. Output $y$.
Eval(ct, $C$):
1. Parse ct as $\hat{D}$.
2. Consider the circuit $C'$ which
    (a) takes as input a string $y$,
    (b) checks if $y = \perp$. If yes, output $\perp$. Otherwise, output $C(y)$.
3. Compute $\hat{D}' \leftarrow \mathsf{HiO.Compose}(\hat{D}, C')$.
4. Output $\hat{D}'$.
```

Fig. 14. Homomorphic Witness Encryption from HiO

Correctness. Both the correctness and homomorphic correctness properties follow from the correctness of HiO.

Soundness Security. When $x \notin \mathcal{L}$, both programs D_{x,m_0} and D_{x,m_1} always output $\perp$ no matter what the input. Therefore, these are functionally equivalent and indistinguishability of the ciphertexts, which are obfuscated versions of these programs, follows from HiO security.

Homomorphic Security. Consider messages m_0, m_1 and circuits $C_1^0, \ldots, C_k^0, C_1^1, \ldots, C_k^1$ such that $C_k^0 \circ \cdots \circ C_1^0(m_0) = C_k^1 \circ \cdots \circ C_1^1(m_1)$. Let C_b denote the circuit $\bar{C}_k^b \circ \cdots \circ \bar{C}_1^b \circ D_{x,m_b}$, where $\bar{C}_i^b$ implements the circuit C_i^b after checking if its input is not $\perp$ as is defined in Figure 14, for $b \in \{0,1\}$. When $x \notin L$, both circuits C_0 and C_1 always output $\perp$. When $x \in \mathcal{L}$, there are 2 cases:

- For inputs w such that $R(x,w) = 1$, the circuit C_b outputs $C_k^b \circ \cdots \circ C_1^b(m_b)$ for $b \in \{0,1\}$. Since $C_k^0 \circ \cdots \circ C_1^0(m_0) = C_k^1 \circ \cdots \circ C_1^1(m_1)$, the outputs are equivalent.
- For inputs w such that $R(x,w) = 0$, both circuits C_0 and C_1 output $\perp$.

Therefore, the circuits C_0 and C_1 are always functionally equivalent when $C_k^0 \circ \cdots \circ C_1^0(m_0) = C_k^1 \circ \cdots \circ C_1^1(m_1)$. Ciphertext indistinguishability follows from HiO security.

$\square$

6.3 Puncture-Hiding Incrementally Puncturable Signatures from HiO

In this section, we consider an advanced puncturable signature scheme as motivated in the introduction and show it as an application of HiO. The notion of puncturable signature schemes has previously been considered in [6, 16].

6.3.1 Definition

A *Puncture-Hiding Incrementally Puncturable Signature* scheme for a message space $\mathcal{M}$ consists of 5 polynomial time algorithms KeyGen, Sign, Verify, Punc and $\mathsf{PuncSign}$, as follows:

- $\mathsf{KeyGen}(1^\lambda)$ is a probabilistic algorithm that takes as input the security parameter λ in unary, and outputs a key-pair $(\mathsf{vk}, \mathsf{sk})$ with vk denoting the verification key and sk denoting the signing key.
- $\mathsf{Sign}(\mathsf{sk}, m)$ is a deterministic algorithm that takes as input the signing key sk and a message $m \in \mathcal{M}$, and outputs a signature σ.
- $\mathsf{Verify}(\mathsf{vk}, \sigma, m)$ is a deterministic algorithm that takes as input the verification key vk, a signature σ and a message m, and outputs a bit b.
- $\mathsf{Punc}(\mathsf{sk}, m)$ is a probabilistic algorithm that takes as input a (possibly punctured) signing key sk and a message m, and outputs a new punctured signing key sk'.
- $\mathsf{PuncSign}(\mathsf{sk}', m')$ is a deterministic algorithm that takes as input a punctured signing key sk' and a message $m' \in \mathcal{M}$, and outputs a signature σ.

We require the following properties from this scheme:

- **Correctness of Verification**: For every security parameter λ and every message m, if $(\mathsf{vk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(1^\lambda)$ and $\sigma := \mathsf{Sign}(\mathsf{sk}, m)$ then
$$\mathsf{Verify}(\mathsf{vk}, \sigma, m) = 1.$$
- **Unforgeability**: For every adversary $\mathcal{A}$, there exists a negligible function negl such that
$$\Pr[\mathsf{Exp}_{\mathcal{A}}^{\mathrm{SIG-FORGE}}(\lambda) = 1] \leq \mathsf{negl}(\lambda),$$
where the experiment $\mathsf{Exp}_{\mathcal{A}}^{\mathrm{SIG-FORGE}}(\lambda)$ is defined as follows:
1. $(\mathsf{vk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(1^\lambda)$,

2. $(m, \sigma) \leftarrow \mathcal{A}^{\mathsf{Sign}(\mathsf{sk},\cdot)}(\mathsf{st}, \mathsf{vk})$,
3. Output 1 if $\mathcal{A}$ never queried $\mathsf{Sign}(\mathsf{sk}, \cdot)$ for the message m and $\mathsf{Verify}(\mathsf{vk}, \sigma, m) = 1$.

– **Punctured Correctness**: For all security parameters λ, all messages $m_1, \dots, m_t \in \mathcal{M}$, all messages $m \in \mathcal{M}$ s.t. $m \neq m_i$ for all $i \in [t]$, if $(\mathsf{vk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(1^\lambda)$, and

$$\mathsf{sk}^{(i)} \leftarrow \mathsf{Punc}(\mathsf{sk}^{(i-1)}, m_i)$$

for all $i \in [t]$, where $\mathsf{sk}^{(0)} := \mathsf{sk}$, then it holds that

$$\mathsf{Sign}(\mathsf{sk}, m) = \mathsf{PuncSign}(\mathsf{sk}^{(t)}, m).$$

– **Punctured Unforgeability**: For all adversaries $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$, there exists a negligible function negl such that

$$\Pr[\mathsf{Exp}_{\mathcal{A}}^{\mathrm{SIG-PFORGE}}(\lambda) = 1] \leq \mathsf{negl}(\lambda),$$

where the experiment $\mathsf{Exp}_{\mathcal{A}}^{\mathrm{SIG-PFORGE}}(\lambda)$ is defined as follows:
1. $(\mathsf{vk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(1^\lambda)$,
2. $(m_1, \dots, m_t, \mathsf{st}) \leftarrow \mathcal{A}_1^{\mathsf{Sign}(\mathsf{sk},\cdot)}(\mathsf{vk})$,
3. $\mathsf{sk}^{(0)} := \mathsf{sk}$, $\mathsf{sk}^{(i)} \leftarrow \mathsf{Punc}(\mathsf{sk}^{(i-1)}, m_i)$ for all $i \in [t]$,
4. $(m, \sigma) \leftarrow \mathcal{A}_2^{\mathsf{Sign}(\mathsf{sk},\cdot)}(\mathsf{st}, \mathsf{sk}^{(t)})$,
5. Output 1 if both $\mathcal{A}_1$ and $\mathcal{A}_2$ never queried $\mathsf{Sign}(\mathsf{sk}, \cdot)$ for the message m, if $m = m_i$ for some $i \in [t]$, and $\mathsf{Verify}(\mathsf{vk}, \sigma, m) = 1$.

– **Puncture-Hiding**: There exists a *size parameter* γ and a PPT simulator Sim such that for every security parameter λ, every integer $t > 0$, all messages $m_1, \dots, m_t$ and every sets of programs $P_1, \dots, P_t$ recognizing[11] the points $m_1, \dots, m_t$ respectively, with $|P_i| \leq \gamma$, the following two distributions are indistinguishable

$$\mathsf{sk}^{(t)} \approx \mathsf{Sim}(1^\lambda, \mathsf{sk}, \{P_1, \dots, P_t\}),$$

where $(\mathsf{vk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(1^\lambda), \mathsf{sk}^{(0)} := \mathsf{sk}$ and $\mathsf{sk}^{(i)} \leftarrow \mathsf{Punc}(\mathsf{sk}^{(i-1)}, m_i)$ for all $i \in [t]$.

Remark. While the definition for puncture-hiding does not directly imply that the punctured signing key hides the punctured points, but it can be ensured by providing the simulator Sim appropriate point-function obfuscations for the punctured messages. Since these hide the respective points, the same should hold even for the real punctured signing key as implied by its indistinguishability from the simulated output. Note that for this argument, the size parameter γ should be larger than the size of the point-function obfuscations employed. Our construction allows γ to be set to any polynomial size.

6.3.2 Construction

In this section, we construct puncture-hiding incrementally puncturable signatures, as defined in Section 6.3.1, assuming a puncturable signature scheme Sig and an HiO scheme.

Theorem 6. *Assuming a puncturable signature scheme for message space $\mathcal{M}$ with a deterministic signing algorithm and an HiO scheme, there exists an puncture-hiding puncturable signature scheme for message space $\mathcal{M}$ and any polynomial size parameter γ.*

Proof: Consider the construction given in Figure 15. We will show below how this scheme satisfies all the desired properties.

Correctness of Verification and Unforgeability. Both these properties follow directly from the corresponding properties of Sig.

[11] This means that each program P_i outputs 1 only on the input m_i and outputs 0 otherwise.

Let γ be a parameter (it should be larger than the size of a circuit that takes an input in $\mathcal{M}$ and compares it with a hardwired message)

Let $\mathsf{Sig} = (\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify}, \mathsf{Punc}, \mathsf{PuncSign})$ be a puncturable signature scheme for a message space $\mathcal{M}$.

Let $\mathsf{HiO} = (\mathsf{Obfuscate}, \mathsf{Eval}, \mathsf{Compose})$ be an HiO scheme.

```
$\underline{\mathsf{KeyGen}(1^\lambda)}$:
1. Run $(\mathsf{vk}, \mathsf{sk}) \leftarrow \mathsf{Sig.KeyGen}(1^\lambda)$ and output $(\mathsf{vk}, \mathsf{sk})$.

$\underline{\mathsf{Sign}(\mathsf{sk}, m)}$:
1. Compute $\sigma := \mathsf{Sig.Sign}(\mathsf{sk}, m)$ and output $\sigma$.

$\underline{\mathsf{Verify}(\mathsf{vk}, \sigma, m)}$:
1. Compute $b := \mathsf{Sig.Verify}(\mathsf{vk}, \sigma, m)$ and output $b$.

$\underline{\mathsf{Punc}(\mathsf{sk}, m)}$:
Let $Q_m^\gamma$ denote a circuit of size $\gamma$ such that $Q_m^\gamma(m') = 1$ iff $m = m'$.
1. If $\mathsf{sk}$ is a standard signing key of the scheme $\mathsf{Sig}$, then
    (a) Consider the circuit $C_{\mathsf{sk},m}$ which
            i. takes as input a message $m' \in \mathcal{M}$;
            ii. if $Q_m^\gamma(m') \neq 1$ computes $\sigma := \mathsf{Sig.Sign}(\mathsf{sk}, m')$ and outputs $(m', \sigma)$,
                and otherwise, outputs $(m', \perp)$.
    (b) Compute $\hat{C}_{\mathsf{sk},m} \leftarrow \mathsf{HiO.Obfuscate}(1^\lambda, C_{\mathsf{sk},m})$ and output $\hat{C}_{\mathsf{sk},m}$.
2. If $\mathsf{sk}$ is already an obfuscated circuit, then
    (a) Parse $\mathsf{sk}$ as $\hat{C}$.
    (b) Consider the circuit $C'_m$ which
            i. takes as input $(m', y)$, where $m' \in \mathcal{M}$ and $y$ is as long as a signature;
            ii. if $Q_m^\gamma(m') = 1$, outputs $(m', \perp)$, and otherwise, outputs $(m', y)$.
    (c) Compute $\hat{C'} \leftarrow \mathsf{HiO.Compose}(\hat{C}, C'_m)$ and output $\hat{C'}$.

$\underline{\mathsf{PuncSign}(\mathsf{sk}', m')}$:
1. Parse $\mathsf{sk}'$ as $\hat{C}$.
2. Compute $(m', y) := \mathsf{HiO.Eval}(\hat{C}, m')$ and output $y$.
```

Fig. 15. Puncture-Hiding Incrementally Puncturable Signature Sig' from puncturable signature Sig and HiO

Punctured Correctness. For all messages $m \neq m_i$, where $m_1, \ldots, m_t$ for some $t > 0$ are the messages on which sk has been punctured, it holds that the algorithm $\mathsf{PuncSign}(\mathsf{sk}^{(t)}, m)$ also computes $\mathsf{Sign}(\mathsf{sk}, m)$ and hence $\mathsf{Verify}(\mathsf{vk}, \sigma, m) = 1$. This follows from the correctness of Sig and that of HiO.

Punctured Unforgeability. Let $\mathsf{sk}^{(t)}$ be the punctured signing key as defined in the experiment $\mathsf{Exp}_{\mathcal{A}}^{\mathrm{SIG-PFORGE}}(\lambda)$ in the punctured unforgeability game, obtained after receiving the messages $m_1, \ldots, m_t$ from $\mathcal{A}_1$. In our construction, $\mathsf{sk}^{(t)}$ is an obfuscated circuit of our HiO scheme which ultimately checks if the input message m is equal to one of the messages $m_1, \ldots, m_t$. If yes, it outputs $\perp$. Otherwise, it outputs $\mathsf{Sig.Sign}(\mathsf{sk}, m)$. This final composed circuit is equivalent to the circuit which has the punctured key $\bar{\mathsf{sk}}^{(t)}$ hardwired inside it, where

$$\bar{\mathsf{sk}}^{(0)} := \mathsf{sk}, \ \bar{\mathsf{sk}}^{(i)} \leftarrow \mathsf{Sig.Punc}(\bar{\mathsf{sk}}^{(i-1)}, m_i)$$

for all $i \in [t]$. This alternative circuit does not directly check if the input message is one of the messages $m_1, \ldots, m_t$ or not; instead, it simply outputs the punctured signature $\mathsf{Sig.PuncSign}(\bar{\mathsf{sk}}^{(t)}, m)$. We need the property from the underlying scheme Sig that signatures using the punctured signing key on punctured messages are equal to $\perp$. This can be assumed without loss of generality. Then, we could use HiO security to switch to this circuit which just has $\bar{\mathsf{sk}}^{(t)}$ hardwired inside it, and then the punctured unforgeability property would follow from that of Sig.

Puncture-Hiding. Consider a security parameter λ, an integer $t > 0$, a sequence of messages $m_1, \ldots, m_t$ and a set of programs $P_1, \ldots, P_t$ recognizing the points $m_1, \ldots, m_t$ respectively. Let $(\mathsf{vk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(1^\lambda), \mathsf{sk}^{(0)} := \mathsf{sk}$ and $\mathsf{sk}^{(i)} \leftarrow \mathsf{Punc}(\mathsf{sk}^{(i-1)}, m_i)$ for all $i \in [t]$. Then the real punctured signing key can also be written as

$$\mathsf{sk}^{(t)} \leftarrow \mathsf{HiO.Compose}(\cdots \mathsf{HiO.Compose}(\mathsf{HiO.Obfuscate}(C_{\mathsf{sk},m_1}), C'_{m_2}) \cdots, C'_{m_t})$$

where the circuits $C_{\mathsf{sk},m}$ and C'_m have been described in Figure 15. We can describe the simulator Sim as follows:

1. takes as input a security parameter 1^λ, a signing key sk and point programs $P_1, \ldots, P_t$. W.l.o.g., $|P_i| = \gamma$ (by padding it, if necessary).
2. considers the circuit D_{sk,P_1} which
   (a) takes as input a message m,
   (b) checks if $P_1(m) = 1$. If yes, then outputs $(m, \perp)$. Otherwise outputs $(m, \mathsf{Sig.Sign}(\mathsf{sk}, m))$.
3. computes $\hat{D}_1 \leftarrow \mathsf{HiO.Obfuscate}(1^\lambda, D_{\mathsf{sk},P_1})$,
4. for $i = 2$ to t, does the following
   (a) considers the circuit D'_{P_i} which
      i. takes as input (m', y),
      ii. checks if $P_i(m') = 1$. If yes, outputs $(m', \perp)$. Otherwise outputs (m', y).
   (b) computes $\hat{D}_i \leftarrow \mathsf{HiO.Compose}(\hat{D}_{i-1}, D'_{P_i})$
5. outputs $\hat{D}_t$.

Therefore, the simulated output can be written as

$$\hat{D}_t \leftarrow \mathsf{HiO.Compose}(\cdots(\mathsf{HiO.Obfuscate}(D_{\mathsf{sk},P_1}), D'_{P_2}), \cdots, D'_{P_t}).$$

Observe that the corresponding two chains are functionally equivalent i.e.,

$$C'_{m_t} \circ \cdots \circ C'_{m_2} \circ C_{\mathsf{sk},m_1} \equiv D'_{P_t} \circ \cdots \circ D'_{P_2} \circ D_{\mathsf{sk},P_1}.$$

Further, by construction, $|C'_i| = |D'_i|$ for every i. Therefore, the desired indistinguishability follows from HiO security.

□

7 Extensions of HiO

In this section, we present some extensions of the notion of HiO that are supported by our ideas.

Composition in any Order. Consider the stronger notion of HiO which allows not just composing with new circuits at the end of the chain (on the output side), but also composing at the beginning of the chain (on the input side). Our scheme can be modified in a straightforward manner to support this extra functionality with a similar proof of security.

Merging Two Chains. Our construction can also be modified to support the property of merging two obfuscated chains. This is possible because the final circuit in a chain gives plaintext outputs while the first circuit in a chain also takes plaintext inputs, so this is well-defined. The proof of security will be *almost* identical to our original construction.

Composition DAG. Consider the notion of HiO where the compositional structure is represented by a DAG, unlike with a chain as in our original construction. To further elaborate, lets say we are given a DAG where each node is a circuit. First, the circuits representing the leaves are to be obfuscated. A parent node at a higher level needs to be combined with the obfuscations of all its children subgraphs and the whole thing needs to be obfuscated. The security requirement states that indistinguishability of the obfuscated DAGs holds as long as the two graphs in consideration are functionally equivalent and share the same topology. Note that this completely captures the spirit of obfuscation as mentioned in the introduction i.e., only allowing black-box modifications to obfuscated programs.

In our construction for such a primitive, each leaf obfuscated circuit would take the entire input string and propagate it forward, even if it only needs some portion of the string. Similarly, each internal node would also propagate the entire initial input. To compose a parent P with its children subgraphs $C_1, \ldots, C_l$, we will use similar ideas like in our original construction. We will first sample l $\mathcal{ACE}$ key-pairs $\{(EK_i, DK_i)\}_{i \in [l]}$, one for each child. We will put the root R_i of the child C_i in an obfuscated circuit which encrypts its output using EK_i. All the decryption keys will go into an obfuscated circuit which evaluates P after decrypting its input ciphertext ct_i using key DK_i. Refer to Figure 16 for a pictorial representation of this process. If all of the input ciphertexts do not agree on the initial input, then the parent node outputs $\perp$.

The security proof will go along similar lines as our original construction. We would go over as many hybrids as the number of strings in the input space. In hybrid H_j, each circuit in the obfuscated DAG would be using the corresponding circuit of the first DAG for all inputs $x \geq j$ and that of the second DAG for inputs $x < j$. To make the switch for $x = j$ in the entire graph, we would start by hardwiring the ciphertext outputs for $x = j$ from the leaves to the root. For each node, this would involve puncturing the corresponding encryption key. This would also involve puncturing each decryption key according to the output from its corresponding child.

The only non-trivial change from our original proof is in the part where we are switching to using the circuit from the second DAG in each node, in a top-down manner. This was the point in our original hybrids where we hardwired the output ciphertext for $x = j$ of the previous circuit in the next circuit in the chain. Analogously, we will hardwire the output ciphertexts of all the children in a parent node and handle these without using decryption. If all the input ciphertexts equal the hardwired values, then we use the hardwired output. If some non-zero number of input ciphertexts equal the hardwired values but not all of them, then we output $\perp$ as decryption would never agree on the initial input in such a case. Only if neither of the input ciphertexts equal the hardwired values, we use decryption like before. The rest of the details follow analogously like before.

References

[1] Prabhanjan Ananth, Aloni Cohen, and Abhishek Jain. “Cryptography with Updates”. In: *Advances in Cryptology – EUROCRYPT 2017*. 2017, pp. 445–472.

[2] Prabhanjan Ananth and Abhishek Jain. “Indistinguishability Obfuscation from Compact Functional Encryption”. In: *Advances in Cryptology – CRYPTO 2015*. 2015.

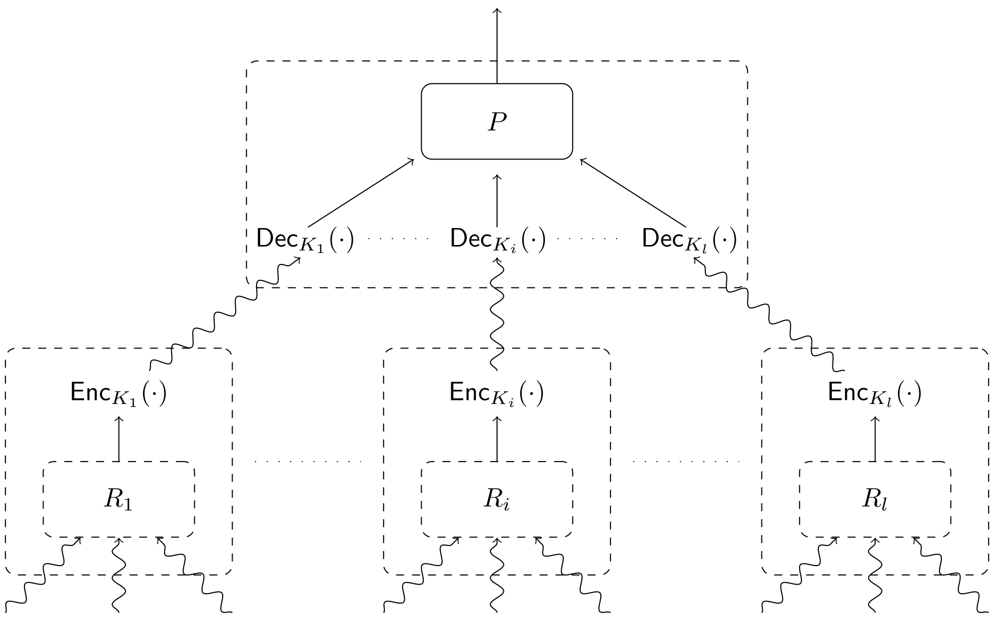
P
$\mathsf{Dec}_{K_1}(\cdot)$
$\mathsf{Dec}_{K_i}(\cdot)$
$\mathsf{Dec}_{K_l}(\cdot)$
$\mathsf{Enc}_{K_1}(\cdot)$
$\mathsf{Enc}_{K_i}(\cdot)$
$\mathsf{Enc}_{K_l}(\cdot)$
R_1
R_i
R_l

Fig. 16. Basic idea for composing a parent circuit P with its children subgraphs $C_1, \ldots, C_l$, whose roots are denoted as $R_1, \ldots, R_l$. Straight arrows represent plaintexts while curved arrows represent ciphertexts. Dashed boundaries denote obfuscated circuits while solid boundaries indicate standard circuits. If R_i's are leaves, then inputs to them should be denoted by straight arrows. The obfuscated circuit on top becomes the new root R.

[3] Prabhanjan Ananth, Abhishek Jain, and Amit Sahai. "Patchable Indistinguishability Obfuscation: iO for Evolving Software". In: *Advances in Cryptology – EUROCRYPT 2017*. 2017, pp. 127–155.

[4] Prabhanjan Ananth, Apoorvaa Deshpande, Yael Tauman Kalai, and Anna Lysyanskaya. "Fully Homomorphic NIZK and NIWI Proofs". In: *Theory of Cryptography - 17th International Conference, TCC 2019, Nuremberg, Germany, December 1-5, 2019, Proceedings, Part II*. Lecture Notes in Computer Science.

[5] Boaz Barak, Oded Goldreich, Rusell Impagliazzo, Steven Rudich, Amit Sahai, Salil Vadhan, and Ke Yang. "On the (Im)possibility of Obfuscating Programs". In: *Advances in Cryptology — CRYPTO 2001*. 2012.

[6] Mihir Bellare, Igors Stepanovs, and Brent Waters. "New Negative Results on Differing-Inputs Obfuscation". In: *Advances in Cryptology – EUROCRYPT 2016*. 2016.

[7] Nir Bitansky, Omer Paneth, and Alon Rosen. "On the Cryptographic Hardness of Finding a Nash Equilibrium". In: *2015 IEEE 56th Annual Symposium on Foundations of Computer Science*. 2015.

[8] Nir Bitansky and Vinod Vaikuntanathan. "Indistinguishability Obfuscation from Functional Encryption". In: *2015 IEEE 56th Annual Symposium on Foundations of Computer Science*. 2015, pp. 171–190.

[9] Nir Bitansky, Sanjam Garg, Huijia Lin, Rafael Pass, and Sidharth Telang. "Succinct Randomized Encodings and Their Applications". In: *Proceedings of the Forty-Seventh Annual ACM Symposium on Theory of Computing*. STOC '15. 2015.

[10] Elette Boyle, Niv Gilboa, Yuval Ishai, Huijia Lin, and Stefano Tessaro. "Foundations of Homomorphic Secret Sharing". In: *9th Innovations in Theoretical Computer Science Conference, ITCS 2018, January 11-14, 2018, Cambridge, MA, USA*. LIPIcs.

[11] Zvika Brakerski and Gil Segev. "Function-Private Functional Encryption in the Private-Key Setting". In: *Theory of Cryptography*. 2015.

[12] Zvika Brakerski and Gil Segev. *Hierarchical Functional Encryption*. Cryptology ePrint Archive, Paper 2015/1011. 2015.

[13] Ran Canetti and Justin Holmgren. "Fully Succinct Garbled RAM". In: *Proceedings of the 2016 ACM Conference on Innovations in Theoretical Computer Science*. ITCS '16. 2016.

[14] Ran Canetti, Huijia Lin, Stefano Tessaro, and Vinod Vaikuntanathan. "Obfuscation of Probabilistic Circuits and Applications". In: *Theory of Cryptography - 12th Theory of Cryptography Conference, TCC 2015, Warsaw, Poland, March 23-25, 2015, Proceedings, Part II*. Ed. by Yevgeniy Dodis and Jesper Buus Nielsen. Vol. 9015. Lecture Notes in Computer Science. Springer, 2015, pp. 468–497.

[15] Ran Canetti, Justin Holmgren, Abhishek Jain, and Vinod Vaikuntanathan. "Succinct Garbling and Indistinguishability Obfuscation for RAM Programs". In: *Proceedings of the Forty-Seventh Annual ACM Symposium on Theory of Computing*. STOC '15. 2015.

[16] Suvradip Chakraborty, Manoj Prabhakaran, and Daniel Wichs. "Witness Maps and Applications". In: *Public-Key Cryptography – PKC 2020*. 2020.

[17] Wutichai Chongchitmate and Rafail Ostrovsky. "Circuit-Private Multi-key FHE". In: *Public-Key Cryptography - PKC 2017 - 20th IACR International Conference on Practice and Theory in Public-Key Cryptography, Amsterdam, The Netherlands, March 28-31, 2017, Proceedings, Part II*. Lecture Notes in Computer Science.

[18] Sanjam Garg and Omkant Pandey. "Incremental Program Obfuscation". In: *Advances in Cryptology – CRYPTO 2017*. 2017, pp. 193–223.

[19] Sanjam Garg, Craig Gentry, Shai Halevi, Mariana Raykova, Amit Sahai, and Brent Waters. "Candidate Indistinguishability Obfuscation and Functional Encryption for all Circuits". In: *2013 IEEE 54th Annual Symposium on Foundations of Computer Science*. 2013.

[20] Sanjam Garg, Craig Gentry, Amit Sahai, and Brent Waters. "Witness Encryption and Its Applications". In: STOC '13. 2013, 467–476.

[21] Craig Gentry. "Fully Homomorphic Encryption Using Ideal Lattices". In: *Proceedings of the Forty-First Annual ACM Symposium on Theory of Computing*. STOC '09. 2009, 169–178.

[22] Craig Gentry, Shai Halevi, and Vinod Vaikuntanathan. "i-Hop Homomorphic Encryption and Rerandomizable Yao Circuits". In: *Advances in Cryptology – CRYPTO 2010*. 2010.

[23] Shafi Goldwasser, S. Dov Gordon, Vipul Goyal, Abhishek Jain, Jonathan Katz, Feng-Hao Liu, Amit Sahai, Elaine Shi, and Hong-Sheng Zhou. "Multi-input Functional Encryption". In: *Advances in Cryptology – EUROCRYPT 2014*. 2014.

[24] Sergey Gorbunov, Vinod Vaikuntanathan, and Daniel Wichs. "Leveled Fully Homomorphic Signatures from Standard Lattices". In: *Proceedings of the Forty-Seventh Annual ACM Symposium on Theory of Computing*. STOC '15. 2015.

[25] Venkata Koppula, Allison Bishop Lewko, and Brent Waters. "Indistinguishability Obfuscation for Turing Machines with Unbounded Memory". In: *Proceedings of the Forty-Seventh Annual ACM Symposium on Theory of Computing*. STOC '15. 2015, 419–428.

[26] R L Rivest, L Adleman, and M L Dertouzos. "On Data Banks and Privacy Homomorphisms". In: *Foundations of Secure Computation, Academia Press* (1978), pp. 169–179.

[27] Amit Sahai and Brent Waters. "How to Use Indistinguishability Obfuscation: Deniable Encryption, and More". In: *Proceedings of the Forty-Sixth Annual ACM Symposium on Theory of Computing*. STOC '14. 2014, 475–484.

Appendix

A Levelled HiO from MiFE

By levelled HiO, we mean the weaker primitive where the maximum number of hops supported by the scheme is fixed in advance. In other words, the Obfuscate algorithm itself takes this number k as input which denotes the maximum number of hops supported by the scheme. In this section, we realise this primitive using MiFE for arbitrary polynomial k.

Theorem 7 (Levelled HiO). *Assuming indistinguishability obfuscation for all circuits and one-way functions, there exists a levelled* HiO *scheme supporting arbitrary number of hops k.*

Proof: Consider the scheme given in Figure 17. This is a straightforward modification of the MiFE to iO transformation presented in [23] to achieve levelled HiO. We avoid providing further details here for brevity.

Let n denote the input length of the first circuit and s denote the circuit size.

Let MiFE $=$ (Setup, KeyGen, Enc, Dec) be an MiFE scheme.

$\underline{\mathsf{Obfuscate}(1^\lambda, C, k)}$:

1. Choose the function arity $n' := sk + n$.
2. Sample $\mathsf{msk} \leftarrow \text{MiFE}.\mathsf{Setup}(1^\lambda, n')$.
3. Consider the n'-bit input function U which does the following
   - It parses its first sk bits as k circuits $C_1, \dots, C_k$ and parses the last n bits as an input x.
   - It then computes and outputs $C_k \circ \dots \circ C_1(x)$.

   Compute $\mathsf{sk}_U \leftarrow \text{MiFE}.\mathsf{KeyGen}(\mathsf{msk}, U)$.
4. Let $C[i]$ be the i^{th} bit in the bit-representation of C. For $i \in \{1, \dots, s\}$, compute $\mathsf{ct}_i \leftarrow \text{MiFE}.\mathsf{Enc}(\mathsf{msk}, C[i], i)$.
5. For $i \in \{s+1, \dots, sk+n\}$ and $b \in \{0,1\}$, compute $\mathsf{ct}_{i,b} \leftarrow \text{MiFE}.\mathsf{Enc}(\mathsf{msk}, b, i)$.
6. Output $\hat{C} = (\{\mathsf{ct}_i\}_{i\in\{1,\dots,s\}}, \{\mathsf{ct}_{i,b}\}_{i\in\{s+1,\dots,sk+n\}, b\in\{0,1\}}, \mathsf{sk}_U)$.

$\underline{\mathsf{Eval}(\hat{C}, x)}$:

1. Parse $\hat{C}$ as $(\{\mathsf{ct}_i\}_{i\in\{1,\dots,sj\}}, \{\mathsf{ct}_{i,b}\}_{i\in\{sj+1,\dots,sk+n\}, b\in\{0,1\}}, \mathsf{sk}_U)$, for some $1 \le j \le k$.
2. Consider the appropriate identity function C_{id} represented as a circuit of size s. For $l = j$ to $k-1$, $i \in \{1, \dots, s\}$, set $\mathsf{ct}_{sl+i} := \mathsf{ct}_{sl+i, C_{id}[i]}$.
3. For $i \in \{1, \dots, n\}$, set $\mathsf{ct}_{sk+i} := \mathsf{ct}_{sk+i, x[i]}$.
4. Compute $y := \text{MiFE}.\mathsf{Dec}(\mathsf{sk}_U, \{\mathsf{ct}_i\}_{i\in\{1,\dots,n'\}})$. Output y.

$\underline{\mathsf{Compose}(\hat{C}, C')}$:

1. Parse $\hat{C}$ as $(\{\mathsf{ct}_i\}_{i\in\{1,\dots,sj\}}, \{\mathsf{ct}_{i,b}\}_{i\in\{sj+1,\dots,sk+n\}, b\in\{0,1\}}, \mathsf{sk}_U)$, for some $1 \le j \le k$.
2. If $j = k$, output $\hat{C}$.
3. Else, for $i \in \{1, \dots, s\}$, set $\mathsf{ct}_{sj+i} := \mathsf{ct}_{sj+i, C'[i]}$.
4. Output $(\{\mathsf{ct}_i\}_{i\in\{1,\dots,s(j+1)\}}, \{\mathsf{ct}_{i,b}\}_{i\in\{s(j+1)+1,\dots,sk+n\}, b\in\{0,1\}}, \mathsf{sk}_U)$.

Fig. 17. Levelled HiO from MiFE

□

Via this transformation, we have shown that we can get levelled HiO from iO. Furthermore, an iO scheme can be realized from any levelled HiO scheme since it already satisfies the basic iO properties. Therefore, these two primitives are equivalent in some sense.

B Proof of Claim 5

Proof: We show this via a sequence of sub-hybrids starting from $H_j^{k,1}$:

– $H_j^{k,2}$**.** In this hybrid, we change the final circuit D_k to now hardwire the output (j, y_k^0) when the input ciphertext is $\alpha_{k-1}^0 = \mathsf{Enc}(EK_{k-1}, y_{k-1}^0)$. In other words, D_k is now $B_4[j, C_k^0, C_k^1, DK_{k-1}\{S_{k-1}^0\}, \alpha_{k-1}^0, y_k^0]$, where B_4 has been described in Figure 18.

```
Hardcoded-values: $j, C^0, C^1, DK, \alpha^*, y^*$.
Input: $\alpha$.
1. If $\alpha = \perp$, output $\perp$. Otherwise, proceed as follows.
2. If $\alpha = \alpha^*$, output $(j, y^*)$. Else
   (a) Compute $t := \mathsf{Dec}(DK, \alpha)$.
   (b) If $t = \perp$, output $\perp$. Otherwise, parse $t$ as $(x, y)$ and proceed as follows.
   (c) If $x = j$, output $(x, y^*)$. Else
       i. If $x > j$, compute $y' := C^0(y)$. Else, compute $y' := C^1(y)$.
       ii. Output $(x, y')$.
```

Fig. 18. Circuit B_4

Claim 6. *Assuming* iO *is a secure indistinguishability obfuscator and* $\mathcal{ACE}$ *satisfies* Equivalence of Constrained Decryption *property, hybrids* $H_j^{k,1}$ *and* $H_j^{k,2}$ *are computationally indistinguishable.*

Proof: To argue indistinguishability, we show functional equivalence between the two circuits as follows:

- For all ciphertexts not equal to α^0_{k-1}, the two circuits behave identically.
- If the input ciphertext were α^0_{k-1}, then the first circuit would decrypt using $DK_{k-1}\{S^0_{k-1}\}$ correctly to (j, y^0_{k-1}) and hence would eventually output (j, y^0_k). Correctness of decryption follows from equivalence of constrained decryption.

□

– $H_j^{k,3}$**.** In this hybrid, we change the decryption key DK_{k-1} inside D_k to now be punctured on the full set $U = (j, \cdot)$.

Claim 7. *Assuming* iO *is a secure indistinguishability obfuscator and* $\mathcal{ACE}$ *satisfies* Uniqueness of Ciphertexts, Safety of Constrained Decryption *and* Equivalence of Constrained Decryption *properties, hybrids* $H_j^{k,2}$ *and* $H_j^{k,3}$ *are computationally indistinguishable.*

Proof: Note that the only difference between the two decryption keys is the input (j, y^0_{k-1}). The previous decryption key $DK_{k-1}\{S^0_{k-1}\}$ could have decrypted to the tuple (j, y^0_{k-1}) but the current decryption key $DK_{k-1}\{U\}$ cannot. The only possible ciphertext that could decrypt, when using the unpunctured decryption key DK_{k-1}, to the tuple (j, y^0_{k-1}) is α^0_{k-1} because of uniqueness of ciphertexts. We show that this is also the case for the punctured key $DK_{k-1}\{S^0_{k-1}\}$ as

- for ciphertexts encrypting messages in the set S^0_{k-1}, the key outputs $\perp$ due to safety of constrained decryption,
- for other ciphertexts it behaves identically to the unpunctured key DK_{k-1} due to equivalence of constrained decryption.

For input ciphertext α^0_{k-1}, both circuits use the hardwired output and hence they never use the underlying decryption keys. Therefore, indistinguishability follows by functional equivalence. □

– $H_j^{k,4}$**.** In this hybrid, we change the hardwired ciphertext α^0_{k-1} inside both D_{k-1} and D_k to $\alpha^1_{k-1} = \mathsf{Enc}(EK_{k-1}, (j, y^1_{k-1}))$, where $y^1_{k-1} = C^1_{k-1} \circ \cdots \circ C^1_0(j)$.

Claim 8. *Assuming* $\mathcal{ACE}$ *satisfies* Selective Ciphertext Indistinguishability *property, hybrids* $H_j^{k,3}$ *and* $H_j^{k,4}$ *are computationally indistinguishable.*

Proof: Note that the two messages $m^*_0 = (j, y^0_{k-1})$ and $m^*_1 = (j, y^1_{k-1})$ belong to the set U on which both EK_{k-1} and DK_{k-1} are punctured. Therefore, indistinguishability follows from selective ciphertext indistinguishability of the ACE scheme. □

- $H_j^{k,5}$**.** In this hybrid, we change puncturing of the decryption key DK_{k-1} inside D_k from the set U to the set $S_{k-1}^1 = \{(j, \neq y_{k-1}^1)\}$. Indistinguishability follows similarly to that of hybrids $H_j^{k,2}$ and $H_j^{k,3}$.
- $H_j^{k,6}$**.** In this hybrid, we change the final circuit D_k back to $B_3[j, C_k^0, C_k^1, DK_{k-1}\{S_{k-1}^1\}, y_k^0 = y_k^1]$ i.e., we remove the hardwired ciphertext from the circuit. Indistinguishability follows similarly to that of hybrids $H_j^{k,1}$ and $H_j^{k,2}$.
- $H_j^{k,7}$**.** In this hybrid, we change D_k so that now it uses the circuit C_k^1 for input $x = j$ i.e., D_k is now $B_2[j+1, C_k^0, C_k^1, DK_{k-1}\{S_{k-1}^1\}]$. Indistinguishability follows similarly to that of hybrids $H_j^{0,3}$ and $H_j^{1,1}$.
- $H_j^{k,8}$**.** In this hybrid, we unpuncture the decryption key $DK_{k-1}\{S_{k-1}^1\}$ inside D_k to now just be DK_{k-1}. Indistinguishability follows similarly to that of hybrids $H_j^{0,2}$ and $H_j^{0,3}$.
- $H_j^{k,9}$**.** In this hybrid, we unpuncture the encryption key $EK_{k-1}\{U\}$ inside D_{k-1} to just be EK_{k-1}. Indistinguishability follows similarly to that of hybrids $H_j^{0,1}$ and $H_j^{0,2}$.

Note that hybrid $H_j^{k,9}$ is the same as $H_j^{k+1,1}$.

□