Homomorphic Indistinguishability Obfuscation and Its Applications

Kaartik Bhushan
IIT Bombay, India

Venkata Koppula
IIT Delhi, India

Manoj Prabhakaran
IIT Bombay, India

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.

2012 ACM Subject Classification Theory of computation → Computational complexity and cryptography

Keywords and phrases Program Obfuscation, Homomorphisms

Digital Object Identifier 10.4230/LIPIcs.ITCS.2024.14

Related Version *ePrint Version*: https://ia.cr/2023/925

Funding *Kaartik Bhushan*: Funded by Prime Minister Research Fellowship, Government of India.

1 Introduction

The goal of code obfuscation [6] 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. [5] and Garg and Pandey [19], 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) [6, 17], 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.[1] 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

[1] 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.

an arbitrary function. Note that for the example above of PIPS, input-based output transformations are already sufficient. In the full version, 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 [22] and function-hiding MiFE [3, 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 in the full version.
  - *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 in the full version.
  - *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 in the full version.

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

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 [25, 7, 14, 13] and use the remanining tools as is done in this paper.

1.2 Related Work

The notion of iO was introduced in [6] and has since been proven to be very powerful, with several applications in cryptography and complexity theory [17, 22, 27, 8]. [9, 4, 24] gave constructions of constant-rate iO schemes in which the size of the obfuscated circuit grows linearly with the size of the input circuit. Either of these 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 [26, 20] but also for other primitives like zero-knowledge [2], signatures [23] and secret-sharing schemes [10] among others. Non-compact function-private FHE schemes have been considered in [16, 21].

A variant of obfuscation called *patchable* iO was introduced in [5]. 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 [19]. 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 [22]. This paper also mentioned the notion of Hierarchical-MiFE in a paragraph. In [3], 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 [18].

2 Technical Overview

2.1 Difficulties with Existing Ideas

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 [3]. 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 [22] 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 the full version). But this construction, which could be seen as an extension of the iO construction from MiFE in [22], 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.

2.2 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 [15] 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 [15]. 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 [14] fits our requirements.

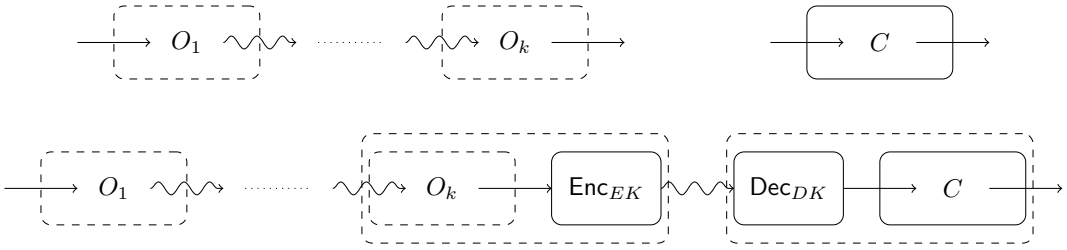

Figure 1 Basic idea for extending a chain of obfuscated circuits $(O_1, \ldots, O_k)$ in our framework. Dashed boundary represents an obfuscated circuit while solid boundary is for standard circuits. Curved arrows denote ciphertexts while straight arrows denote plaintext values. Top figure represents 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). $\hat{O}_k$ internally encrypts the output of O_k and O_{k+1} first decrypts this ciphertext before applying the circuit C to the result and then outputs it in the clear.

Asymmetrically Constrainable Encryption

The notion of asymmetrically constrainable encryption (ACE) was proposed by Canetti et al [14] 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. [2] For security, we require two properties. First, the punctured decryption keys should hide the constraint set. More

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

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 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).

2.3 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 $i \leq k$,

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

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 $C_k \circ \cdots \circ C_1$ 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[3] 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. Here, the notation A, B and G is used to denote a distinction between the kinds of circuits that are present in the final chain. A is a circuit that takes plaintext inputs but gives out encrypted outputs, G takes encrypted inputs and gives out encrypted outputs while B only takes encrypted inputs but gives out plaintext outputs. Hence, the first circuit in the chain is of type A, the last circuit is of type B while all intermediate circuits are of type G. 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 $(j, y_1^* = C_1^0 \circ C_0^0(j) = C_1^1 \circ C_0^1(j))$.

[3] We will additionally be propagating the entire initial input throughout the chain for our security proof.

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.

2.4 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 $\alpha_i^0 = \mathsf{Enc}(EK_i, (j, y_i^0 = C_i^0 \circ \cdots \circ C_0^0(j)))$ for $x = j$ inside each circuit D_i, for $i \in \{0, \ldots, k-1\}$. 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[4]. 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.

2.5 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 [22], is a stronger functional encryption primitive allowing multiple parties to encrypt different messages which

[4] 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.

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, [3] 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 H-MiFEexperiment where the challenger chooses bit b as 0 and provides outputs to the adversary. In particular, for function query $\left((f_0^0, \ldots, f_k^0), (f_0^1, \ldots, f_k^1)\right)$, the challenger computes $\mathsf{sk}_f^{(k)}$ where

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

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

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

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. [14].

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',

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

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

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

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

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

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

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

5. *Equivalence of Constrained Decryption*: For any subsets S and S' of $\mathcal{M}$, if $\mathsf{Dec}(DK\{S\}, c) = m \neq \perp$ 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 [14] gave a construction of this primitive assuming iO and one-way functions, as mentioned in the following theorem.

▶ **Theorem 1** ([14]). *Assuming subexponentially-secure indistinguishability obfuscation for all circuits and subexponentially-secure one-way functions, there exists a secure ACE scheme.*

The remaining preliminaries are included in the full version.

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 $\hat{C}$.
- $\mathsf{Eval}(\hat{C}, x)$: The algorithm Eval takes as input an obfuscated circuit $\hat{C}$ 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 $\mathsf{Obfuscate}$ 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. For this, the algorithm $\mathsf{Obfuscate}$ must take the value of k as input and hence, this would only be possible with levelled-HiO.

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 $\mathcal{ACE} = (\mathsf{Setup}, \mathsf{GenEK}, \mathsf{GenDK}, \mathsf{Enc}, \mathsf{Dec})$.

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

1. Compute $\hat{D} \leftarrow \mathsf{iO.Obfuscate}(1^\lambda, C)$ and output $\hat{D}$.

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

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.

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

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 α.

Figure 2 Circuit A.

Hardcoded-values: $\hat{C}, EK$.

Input: α.

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 α'.

Figure 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').

Figure 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 [9, 4, 24], we can get the size of the composed circuit as $O(|C_0| + \ldots + |C_k|) + \mathsf{poly}(\lambda)$.

5.2 Proof of Security

The following theorem gives our security proof. Note that the proof suffers from an exponential loss in security and currently it does not seem feasible to have a hybrid argument that does not iterate over all its inputs.

▶ **Theorem 2** (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 $D_0, \ldots, D_k$[5] 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 $i \in \{0, \ldots, k\}$.
- **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 $B[C_k^0, DK_{k-1}]$.

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

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

Hardcoded-values: C, EK.

Input: x.

1. Compute $y := C(x)$.
2. Compute $\alpha := \mathsf{Enc}(EK, (x, y))$.
3. Output α.

Figure 5 Circuit A_2.

Hardcoded-values: C, DK, EK'.

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) proceed as follows.
4. Compute $y' := C(y)$.
5. Compute $\alpha' := \mathsf{Enc}(EK', (x, y'))$.
6. Output α'.

Figure 6 Circuit G_2.

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 α.

Figure 7 Circuit A_3.

Hardcoded-values: j, C^0, C^1, DK, EK'.

Input: α.

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 α'.

Figure 8 Circuit G_3.

Hardcoded-values: j, C^0, C^1, DK.

Input: α.

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').

Figure 9 Circuit B_2.

▶ **Lemma 4.** *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 5.** *For any $j < |\mathcal{X}|$, 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 $A[\hat{C}_0^1, EK_0]$.
- 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 6.** *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 5

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

- $\boldsymbol{H_j^{0,0}}$**.** This is the same as H_j.
- $\boldsymbol{H_j^{0,1}}$**.** 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.

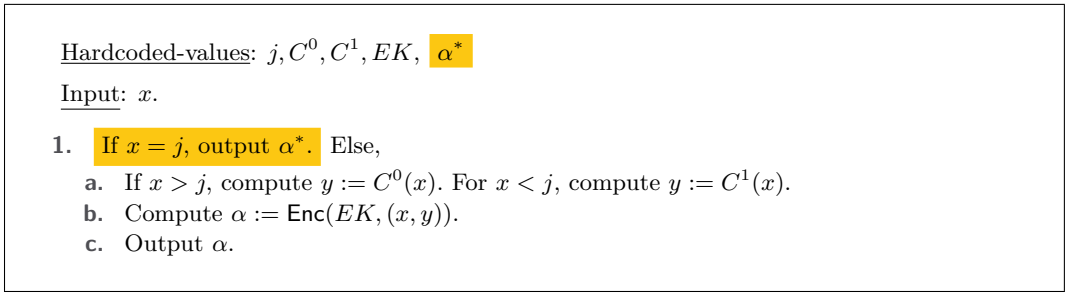

Figure 10 Circuit A_4.

▷ Claim 7. Assuming iO is a secure indistinguishability obfuscator, hybrids $H_j^{0,1}$ and $H_j^{0,0}$ are computationally indistinguishable.

- $\boldsymbol{H_j^{0,2}}$**.** 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 8. 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. ◁

- $\boldsymbol{H_j^{0,3}}$**.** 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 9. 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. ◁

- $\boldsymbol{H_j^{1,1}}$**.** 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: α.

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 α^*. 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 α'.

Figure 11 Circuit G_4.

▷ Claim 10. 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. ◁

- $\boldsymbol{H_j^{i,l}}$**.** 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 $S_i^0 = \{(j, \neq y_i^0)\}$.
- $\boldsymbol{H_j^{k,1}}$**.** 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: α.

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').

Figure 12 Circuit B_3.

- $\boldsymbol{H_j^{k+1,1}}$ 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 11. Assuming iO is a secure indistinguishability obfusctor 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 the full version. ◁

- $\boldsymbol{H_j^{i,l}}$**.** 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 $\alpha_{i-1}^1 = \mathsf{Enc}(EK_{i-1}, (j, y_{i-1}^1))$. For $l = 7$, we change puncturing of the decryption key DK_{i-1} inside D_i from the set U to the set $S_{i-1}^1 = \{(j, \neq y_{i-1}^1)\}$.
  For $l = 8$, we remove the hardwired ciphertext α_{i-1}^1 inside D_i. For $l = 9$, we change D_i so that it uses the circuit C_i^1 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.
- $\boldsymbol{H_j^{0,4}}$**.** In this hybrid, we change the first circuit D_0 so that it uses the circuit C_0^1 for the input $x = j$. Note that this hybrid is the same as H_{j+1}. ◀

6 Applications and Extensions

Due to space constraints, applications and extensions of homomorphic indistinguishability obfuscation are deferred to the full version of the paper.

References

1. Prabhanjan Ananth, Aloni Cohen, and Abhishek Jain. Cryptography with updates. In *Advances in Cryptology – EUROCRYPT 2017*, pages 445–472, 2017.
2. Prabhanjan Ananth, Apoorvaa Deshpande, Yael Tauman Kalai, and Anna Lysyanskaya. Fully homomorphic nizk and niwi proofs. In *Theory of Cryptography*, 2019.
3. Prabhanjan Ananth and Abhishek Jain. Indistinguishability obfuscation from compact functional encryption. Cryptology ePrint Archive, Report 2015/173, 2015.
4. Prabhanjan Ananth, Abhishek Jain, and Amit Sahai. Indistinguishability obfuscation for turing machines: Constant overhead and amortization. In *Advances in Cryptology – CRYPTO 2017*, 2017.
5. Prabhanjan Ananth, Abhishek Jain, and Amit Sahai. Patchable indistinguishability obfuscation: io for evolving software. In *Advances in Cryptology – EUROCRYPT 2017*, pages 127–155, 2017.
6. 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.
7. 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.
8. 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.
9. Nir Bitansky and Vinod Vaikuntanathan. Indistinguishability obfuscation from functional encryption. In *2015 IEEE 56th Annual Symposium on Foundations of Computer Science*, pages 171–190, 2015.
10. Elette Boyle, Niv Gilboa, Yuval Ishai, Huijia Lin, and Stefano Tessaro. Foundations of homomorphic secret sharing. Cryptology ePrint Archive, Paper 2017/1248, 2017.
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, 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.
15. Ran Canetti, Huijia Lin, Stefano Tessaro, and Vinod Vaikuntanathan. Obfuscation of probabilistic circuits and applications. Cryptology ePrint Archive, Report 2014/882, 2014.
16. Wutichai Chongchitmate and Rafail Ostrovsky. Circuit-private multi-key fhe. Cryptology ePrint Archive, Paper 2017/010, 2017.
17. 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.
18. Sanjam Garg, Craig Gentry, Amit Sahai, and Brent Waters. Witness encryption and its applications. In *Symposium on Theory of Computing Conference, STOC'13, Palo Alto, CA, USA, June 1-4, 2013*, STOC '13, pages 467–476, 2013.
19. Sanjam Garg and Omkant Pandey. Incremental program obfuscation. In *Advances in Cryptology – CRYPTO 2017*, pages 193–223, 2017.
20. Craig Gentry. Fully homomorphic encryption using ideal lattices. In *Proceedings of the Forty-First Annual ACM Symposium on Theory of Computing*, STOC '09, pages 169–178, 2009.

21 Craig Gentry, Shai Halevi, and Vinod Vaikuntanathan. i-hop homomorphic encryption and rerandomizable yao circuits. In *Advances in Cryptology – CRYPTO 2010*, 2010.

22 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.

23 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.

24 Aayush Jain, Huijia Lin, and Ji Luo. On the optimal succinctness and efficiency of functional encryption and attribute-based encryption. In *Advances in Cryptology – EUROCRYPT 2023*, 2023.

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, pages 419–428, 2015.

26 R L Rivest, L Adleman, and M L Dertouzos. On data banks and privacy homomorphisms. *Foundations of Secure Computation, Academia Press*, pages 169–179, 1978.

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, pages 475–484, 2014.