Decomposable Obfuscation: A Framework for Building Applications of Obfuscation from Polynomial Hardness

Qipeng Liu(✉) and Mark Zhandry

Princeton University, Princeton, USA
qipengl@cs.princeton.edu

Abstract. There is some evidence that indistinguishability obfuscation (iO) requires either exponentially many assumptions or (sub)exponentially hard assumptions, and indeed, all known ways of building obfuscation suffer one of these two limitations. As such, any application built from iO suffers from these limitations as well. However, for most applications, such limitations do not appear to be inherent to the application, just the approach using iO. Indeed, several recent works have shown how to base applications of iO instead on functional encryption (FE), which can in turn be based on the polynomial hardness of just a few assumptions. However, these constructions are quite complicated and recycle a lot of similar techniques.

In this work, we unify the results of previous works in the form of a weakened notion of obfuscation, called *Decomposable Obfuscation*. We show (1) how to build decomposable obfuscation from functional encryption, and (2) how to build a variety of applications from decomposable obfuscation, including all of the applications already known from FE. The construction in (1) hides most of the difficult techniques in the prior work, whereas the constructions in (2) are much closer to the comparatively simple constructions from iO. As such, decomposable obfuscation represents a convenient new platform for obtaining more applications from polynomial hardness.

1 Introduction

Program obfuscation has recently emerged as a powerful cryptographic concept. An obfuscator is a compiler for programs, taking an input program, and scrambling it into an equivalent output program, but with all internal implementation details obscured. Indistinguishability obfuscation (iO) is the generally-accepted notion of security for an obfuscator, which says that the obfuscations of equivalent programs are computationally indistinguishable.

In the last few years since the first candidate indistinguishability obfuscator of Garg et al. [GGH+13], obfuscation has been used to solve many new amazing tasks such as deniable encryption [SW14], multiparty non-interactive key agreement [BZ14], polynomially-many hardcore bits for any one-way function [BST14], and much more. Obfuscation has also been shown to imply most

Y. Kalai and L. Reyzin (Eds.): TCC 2017, Part I, LNCS 10677, pp. 138–169, 2017.
https://doi.org/10.1007/978-3-319-70500-2_6

traditional cryptographic primitives[1] such as public key encryption [SW14], zero knowledge [BP15], trapdoor permutations [BPW16], and even fully homomorphic encryption [CLTV15]. This makes obfuscation a "central hub" in cryptography, capable of solving almost any cryptographic task, be it classical or cutting edge. Even more, obfuscation has been shown to have important connections to other areas of computer science theory, from demonstrating the hardness of finding Nash equilibrium [BPR15] to the hardness of certain tasks in differential privacy [BZ14,BZ16].

The power of obfuscation in part comes from the power of the underlying tools, but its power also lies in the *abstraction*, by hiding away the complicated implementation details underneath a relatively easy to use interface. In this work, we aim to build a similarly powerful abstraction that avoids some of the limitations of iO.

1.1 The Sub-exponential Barrier in Obfuscation

Indistinguishability obfuscation (iO), as an assumption, has different flavor than most assumptions in cryptography. Most cryptographic assumptions look like

"Distribution A is computationally indistinguishable from distribution B," or
"Given a sample a from distribution A, it is computationally infeasible
to compute a value b such that a, b satisfy some given relation."

Such assumptions are often referred to as falsifiable [Nao03], or more generally as complexity assumptions [GT16]. In contrast, iO has the form

"*For every pair of circuits* C_0, C_1 *that are functionally equivalent*,
$\mathsf{iO}(C_0)$ is computationally indisitnguishable from $\mathsf{iO}(C_1)$."

In other words, for each pair of equivalent circuits C_0, C_1, there is an instance of a complexity assumption: that $\mathsf{iO}(C_0)$ is indistinguishable from $\mathsf{iO}(C_1)$. iO then is really a *collection* of exponentially-many assumptions made simultaneously, one per pair of equivalent circuits. iO is violated if *a single* assumption in the collection is false. This is a serious issue, as the security of many obfuscators relies on new assumptions that essentially match the schemes. To gain confidence in the security of the schemes, it would seem like we need to investigate the iO assumption for every possible pair of circuits, which is clearly infeasible.

Progress has been made toward remedying this issue. Indeed, Gentry et al. [GLSW15] show how to build obfuscation from a single assumption—multilinear subgroup elimination—on multilinear maps. Unfortunately, the security reduction loses a factor exponential in the number of input bits to the program. As such, in order for the reduction to be meaningful, the multilinear subgroup elimination problem must actually be *sub-exponentially* hard. Similarly, Bitansky and Vaikuntanathan [BV15] and Ananth and Jain [AJ15] demonstrate how to construct iO from a tool called functional encryption (FE).

[1] With additional mild assumptions such as the existence of one-way functions.

In turn, functional encryption can be based on simple assumptions on multilinear maps [GGHZ16]. However, while the construction of functional encryption can be based on the polynomial hardness of just a couple multilinear map assumptions, the construction of iO from FE incurs an exponential loss. This means the FE scheme, and hence the underlying assumptions on multilinear maps, still need to be *sub-exponentially* secure.

All current techniques for building iO suffer one of these two limitations: either security is based on an exponential number of assumptions, or the reduction incurs an exponential loss. Unfortunately, this means every application of iO also suffers from the same limitations. As iO is the only known instantiation of many new cryptographic applications, an important research direction is to devise new instantiations that avoid this exponential loss.

1.2 Breaking the Sub-exponential Barrier

A recent line of works starting with Garg et al. [GPS16] and continued by [GPSZ16, GS16] have shown how to break the sub-exponential barrier for certain applications. Specifically, these works show how to base certain applications on functional encryption, where the loss of the reduction is just polynomial. Using [GGHZ16], this results in basing the applications on the polynomial hardness of a few multilinear map assumptions. The idea behind these works is to compose the FE-to-iO conversion of [BV15, AJ15] with the iO-to-Application conversion to get an FE-to-Application construction. While this construction requires an exponential loss (due to the FE-to-iO conversion), by specializing the conversion to the particular application and tweaking things appropriately, the reduction can be accomplished with a polynomial loss. Applications treated in this way include: the hardness of computing Nash equilibria, trapdoor permutations, universal samplers, multiparty non-interactive key exchange, and multi-key functional encryption[2].

While the above works represent important progress, the downside is that, in order to break the sub-exponential barrier, they also break the convenient obfuscation abstraction. Both the FE-to-iO and iO-to-Application conversions are non-trivial, and the FE-to-iO conversion is moreover non-black box. Add to that the extra modifications to make the combined FE-to-Application conversion be polynomial, and the resulting constructions and analyses become reasonably cumbersome. This makes translating the techniques to new applications rather tedious—not to mention potentially repetitive given the common FE-to-iO core—and understanding the limits of this approach almost impossible.

1.3 A New Abstraction: Decomposable Obfuscation

In this work, we define a new notion of obfuscation, called *Decomposable Obfuscation*, or dO, that addresses the limitations above. This notion abstracts away

[2] The kind of functional encryption that is used as a starting point only allows for a single secret key query.

many of the common techniques in [GPS16, GPSZ16, GS16]; we use those techniques to build dO from the *polynomial* hardness of functional encryption. Then we can show that the dO can be used to build the various applications. With our new notion in hand, the dO-to-Application constructions begin looking much more like the original iO-to-Application constructions, with easily identifiable modifications that are necessary to prove security using our weaker notion.

The Idea

Functional Encryption (FE). As in the works of [GPS16, GPSZ16, GS16], we will focus on obtaining our results from the starting point of polynomially-secure functional encryption. Functional encryption is similar to regular public key encryption, except now the secret key holder can produce function keys corresponding to arbitrary functions. Given a function key for a function f and a ciphertext encrypting m, one can learn $f(m)$. Security requires that even given the function key for f, encryptions of m_0 and m_1 are indistinguishable, so long as $f(m_0) = f(m_1)$[3].

The FE-to-iO Conversion. The FE-to-iO conversions of [BV15, AJ15] can be thought of very roughly as follows. To obfuscate a circuit C, we generate the keys for an FE scheme, and encrypt the description of C under the FE scheme's public key, obtaining c. We also produce function keys fk_i for particular functions f_i that we will describe next. The obfuscated program consists of c and the fk_i.

To evaluate the program on input x, we first use fk_1 and c to learn $f_1(C)$. $f_1(C)$ is defined to produce two ciphertexts c_0, c_1, encrypting $(C, 0)$ and $(C, 1)$, respectively. We keep c_{x_1}, discarding the other ciphertext. Now, we actually define fk_1 to encrypt $(C, 0)$ and $(C, 1)$ using the functional encryption scheme itself—therefore, we can continue applying function keys to the resulting plaintexts. We use fk_2 and c_{x_1} to learn $f_2(C, x_1)$. $f_2(C, b)$ is defined to produce two ciphertexts c_{b0}, c_{b1}, encrypting $(C, b0)$ and $(C, b1)$. Again, these ciphertexts will be encrypted using the functional encryption scheme. We will repeat this process until we obtain the encryption c_x of (C, x). Finally, we apply the last function key for the function f_{n+1}, which is the universal circuit evaluating $C(x)$.

This procedure implicitly defines a complete binary tree of all strings of length at most 2^n, where a string x is the parent of the two string $x||0$ and $x||1$. At each node $y \in \{0, 1\}^{\leq n}$, consider running the evaluation above for the first $|y|$ steps, obtaining a ciphertext c_y encrypting (C, y). We then assign the circuit C to the node y, according the circuit that is encrypted in c_y. The root is *explicitly* assigned C by handing out the ciphertext c since we explicitly encrypt C to obtain c. All subsequent nodes are *implicitly* assigned C as c_y is derived from c during evaluation time. Put another way, by explicitly assigning a circuit C to a node (in this case, the root) we implicitly assign the same circuit C to

[3] The two encryptions would clearly be distinguishable if $f(m_0) \neq f(m_1)$ just by decrypting using the secret function key. Thus, this is the best one can hope for with an indistinguishability-type definition.

all of its descendants. The exception is the leaves: if we were to assign a circuit C to a leaf x, we instead assign the output $C(x)$. In this way, the leaves contain the truth table for C.

Now, we start from an obfuscation of C_0 (assigning C_0 to the root of the tree) and we wish to change the obfuscation to an obfuscation of C_1 (assigning C_1 to the root). We cannot do this directly, but the functional encryption scheme does allow us to do the following: un-assign a circuit C from any internal node y[4], and instead *explicitly* assign C to the two children of that node. This is accomplished by changing c_y to encrypt $(\perp, x)$, explicitly constructing the ciphertexts $c_{y||0}$ and $c_{y||1}$, and embedding $c_{y||0}, c_{y||1}$ in the function key $\mathsf{fk}_{|y|}$ in a particular way. If the children are leaves, explicitly assign the outputs of C on those leaves. Note that this process does not change the values assigned to the leaves; as such, the functionality of the tree remains unchanged, so this change cannot be detected by functionality alone. The security of functional encryption shows that, in fact, the change is undetectable to any polynomial-time adversary.

The security reduction works by performing a depth-first traversal of the binary tree. When processing a node y on the way down the tree, we un-assign C_0 from y and instead explicitly assign C_0 to the children of y. When we get to a leaf, notice that by functional equivalence, we actually simultaneously have the output of C_0 and C_1 assigned. Therefore, when processing a node y on our way up the tree from the leaves, we can perform the above process in reverse but for C_1 instead of C_0. We can un-assign C_1 from the children of y, and then explicitly assign C_1 to y. In this way, when the search is complete, we explicitly assign C_1 to the root, which implicitly assigns C_1 to all nodes in the tree. At this point, we are obfuscating C_1. By performing a depth-first search, we ensure that the number of explicitly assigned nodes never exceeds $n + 1$, which is crucial for the efficiency of the obfuscator, as we pay for explicit assignments (since they correspond to explicit ciphertexts embedded in the function keys) but not implicit ones (since they are computed on the fly). Note that while the obfuscator itself is polynomial, the number of steps in the proof is exponential: we need to un-assign and re-assign every internal node in the tree, which are exponential in number. This is the source of the exponential loss.

Shortcutting the Conversion Process. The key insight in the works of [GPS16, GPSZ16, GS16] is to modify the constructions in a way so that it is possible to re-assign certain internal nodes in a single step, without having to re-assign all of its descendants first. By doing this it is possible to shortcut our way across an exponential number of steps using just a few steps.

In these prior works, the process is different for each application. In this work, we generalize the conditions needed for and the process of shortcutting in a very natural way. To see how shortcutting might work, we introduce a slightly different version of the above assignment setting. Like before, every node can be assigned a circuit. However, now the circuit assigned to a node u of length k must work on inputs of length $n - k$; essentially, it is the circuit that is "left

[4] By assigning $\perp$ instead, which does *not* propagate down the tree.

over" after reading the first k bits and which operates on the remaining $n-k$ bits.

If we explicitly assign a circuit C_y to a node y, its children are implicitly assigned the *partial evaluations* of C_y on 0 and 1. That is, the circuit $C_{y||b}$ assigned to $y||b$ is $C_y(b,\cdot)$. We will actually use $C_y(b,\cdot)$ to denote the circuit obtained by hard-coding b as the first input bit, and then simplifying using simple rules: (1) any unary gate with a constant input wire is replaced with an appropriate input wire, (2) any binary gate with a constant input is replaced with just a unary gate (a passthrough or a NOT) or a hardwired output according to the usual rules, (3) any wire that is not used is deleted, and (4) this process is repeated until there are no gates with hardwired inputs and no unused wires. An important observation is that our rules guarantee that circuits assigned to leaves are always constants, corresponding to the output of the circuit at that point.

Now when we obfuscate by assigning C to the root, the internal nodes are implicitly assigned the simplified partial evaluations of C on the prefix corresponding to that node: node y is assigned $C(y,\cdot)$ (simplified). The move we are allowed to make is now to un-assign C from a node where C was explicit, and instead explicitly assign the simplified circuits $C(0,\cdot)$ and $C(1,\cdot)$ to its children. We call the partial evaluations $C(0,\cdot)$ and $C(1,\cdot)$ *fragments* of C, and we call this process of un-assigning the parent and assigning the fragments to the children *decomposing* the node to its children fragments. The reverse of decomposing is *merging*.

This simple transformation to the binary tree rules allows for, in some instances, the necessary shortcutting to avoid an exponential loss. When transforming C_0 to C_1, the crucial observation is that if any fragment $C_0(x,\cdot)$ is equal to $C_1(x,\cdot)$ *as circuits* (after simplification), it suffices to stop when we explicitly assign a circuit to x; we do not need to continue all the way down to the leaves. Indeed, once we explicitly assign the fragment $C_0(y,\cdot)$ to a node y, y already happens to be assigned the fragment $C_1(y,\cdot)$ as well, and all of its descendants are therefore implicitly assigned the corresponding partial evaluations of C_1 as well. By not traversing all the way to the leaves, we cut out potentially exponentially many steps. For certain circuit pairs, it may therefore be possible to transform C_0 to C_1 in only polynomially-many steps.

Our New Obfuscation Notion. Our new obfuscation notion stems naturally from the above discussion. Consider two circuits C_0, C_1 of the same size, and consider assigning C_0 to the root of the binary tree. Suppose there is a set S of tree nodes of size τ that (1) exactly cover all of the leaves[5], and (2) for every nodes $x \in S$, the (simplified) fragments $C_0(y,\cdot)$ and $C_1(y,\cdot)$ are identical *as circuits*. Then we say the circuits C_0, C_1 are τ-decomposing equivalent. Our new obfuscation notion, called *decomposable obfuscation*, is parameterized by τ and says, roughly, that the obfuscations of two τ-decomposing equivalent circuits must be indistinguishable.

[5] In the sense that for each leaf, the path from root to leaf contains exactly one element in S.

1.4 Our Results

Our results are as follows:

– We show how to use (compact, single key) functional encryption to attain our notion of dO. The construction is similar to the FE-to-iO conversion, with the key difference that each step simplifies the circuit as must as possible; this implements the new tree rules we need for shortcutting.
  The number of steps in the process of converting C_0 to C_1, and hence the loss in the security reduction is proportional to τ. However, we show that by performing the decompose/merge steps in the right order, we can make sure the number of explicitly assigned nodes is always at most $n+1$, independent of τ. This means the obfuscator itself does not depend on τ, and therefore τ can be taken to be an arbitrary polynomial or even exponential and the obfuscator will still be efficient. If we restrict τ to a polynomial, we obtain dO from polynomially secure FE. Our results also naturally generalize to larger τ: we obtain dO for quasipolynomial τ from quasipolynomially secure FE, and we obtain dO for exponential τ from (sub)exponentially secure FE.
– We note that by setting τ to be 2^n, τ-decomposing equivalence corresponds to standard functional equivalence, since we can take the set S of nodes to consist of all leaf nodes. Then dO coincides with the usual notion of indistinguishability obfuscation, giving us iO from sub-exponential FE. This re-derives the results of [BV15, AJ15]. In our reduction, the loss is $O(2^n)$.
– We then show how to obtain several applications of obfuscation from dO with *polynomial* τ. Thus, for all these applications, we obtain the application from the polynomial hardness of FE, re-deriving several known results. In these applications, there is a single input, or perhaps several inputs, for which the computation must be changed from using the original circuit to using a hard-coded value. This is easily captured by decomposing equivalence: by decomposing each node from the root to the leaf for a particular input x, the result is that that the program's output on x is hard-coded into the obfuscation. Applications include:
  - Proving the hardness of finding Nash equilibria (in the full version [LZ17]; Nash hardness from FE was originally shown in [GPS16])
  - Trapdoor Permutations (originally shown in [GPSZ16])
  - Universal Samplers (Sect. 3.3; originally shown in [GPSZ16])
  - Short Signatures (Sect. 3.2; not previously known from functional encryption, though known from obfuscation [SW14])
  - Multi-key functional encryption (in the full version [LZ17]; originally shown in [GS16])

  We note that Nash, universal samplers, and short signatures only require (polynomially hard) dO and one-way functions. In contrast, trapdoor permutations and multi-key functional encryption both additionally require public key encryption. If basing the application on public key functional encryption, this assumption is redundant. However, unlike the case for full-fledged iO, we do not know how to obtain public key functional encryption from just polynomially hard dO and one-way functions (more on this below). We do show that a weaker multi-key *secret key* functional encryption scheme does follow from dO and one-way functions.

Thus, we unify the techniques underlying many of the applications of FE—namely iO, Nash, trapdoor permutations, universal samplers, short signatures, and multi-key FE—under a single concept, dO. The constructions and proofs starting from dO are much simpler than the original proofs using functional encryption, due to the convenient dO abstraction hiding many of the common details. We hope that dO will also serve as a starting point for further constructions based on polynomially-hard assumptions.

1.5 Discussion

A natural question to ask is: what are the limits of these techniques? Could they be used to give full iO from polynomially-hard assumptions? Or at least all known applications from polynomial hardness? Here, we discuss several difficulties that arise.

Difficulties in Breaking the Sub-exponential Barrier. First, exponential loss may be inherent to constructing iO. Indeed, the following informal argument is adapted from Garg et al. [GGSW13]. Suppose we can prove iO from a single fixed assumption. This means that for every pair of equivalent circuits C_0, C_1, we prove under this assumption that $\mathsf{iO}(C_0)$ is indistinguishable from $\mathsf{iO}(C_1)$. Fix two circuits C_0, C_1, and consider the proof for those circuits. If C_0 is equivalent to C_1, then the proof succeeds. However, if C_0 is *not* equivalent to C_1, then the proof *must* fail: let x be a point such that $C_0(x) \neq C_1(x)$. Then a simple adversary with x hard-coded can distinguish $\mathsf{iO}(C_0)$ from $\mathsf{iO}(C_1)$ simply by running the obfuscated program on x.

This intuitively means that the proof must some how decide whether C_0 and C_1 are equivalent. Since the proof consists of an *efficient* algorithm R reducing breaking the assumption to distinguishing $\mathsf{iO}(C_0)$ from $\mathsf{iO}(C_1)$, it seems that R must be efficiently deciding circuit equivalence. Assuming $P \neq NP$, such a reduction should not exist.[6]

The reductions from iO to functional encryption/simple multilinear map assumptions avoid this argument by not being efficient. Indeed, the reductions traverse the entire tree of 2^n nodes as described above. In essence, the proof in each step just needs to check a local condition such as $C_0(x) = C_1(x)$ for some

[6] One may wonder whether the same arguments apply to the seemingly similar setting of zero knowledge, where zero knowledge must hold for true instances, but soundness must hold for false instances. The crucial difference is that soundness does not prevent the zero knowledge simulator from working on false instances. Therefore, a reduction from a hard problem to zero knowledge does not need to determine whether the instance is in the language. In contrast, for iO, the security property must apply to equivalent circuits, but correctness implies that it *cannot* apply to inequivalent circuits.

particular x—which can be done efficiently—as opposed to checking equivalence for all inputs.

While this argument is far from a proof of impossibility, it does represent an significant inherent difficulty in building full-fledged iO from polynomial hardness. We believe that overcoming this barrier, or showing that it is insurmountable, is an important and fascinating open question. For example, imagine translating the arguments above to iO for computational models with unbounded input lengths such as Turing machines. In this case, equivalence is not only inefficient, but *undecidable*. As such, the above arguments demonstrate a barrier to basing Turing machine obfuscation on a finite number of even (sub)exponentially hard assumptions. An important open question is whether it is possible to build iO from Turing machines from iO for circuits; we believe achieving this goal will likely require techniques that can also be used to overcome the sub-exponential barrier.

For the remainder of the discussion, we will assume that building iO from polynomial hardness is beyond reach without significant breakthroughs.

Avoiding the Barrier. We observe that poly-decomposing equivalence is an NP relation: the polynomial-sized set of nodes where the fragments are identical provides a witness that two circuits are equivalent: it is straightforward to check that a collection of nodes covers all of the leaves and that the fragments at those nodes are identical. In contrast, general circuit equivalence is $co\text{-}NP$-complete, and therefore unlikely to be in NP unless the polynomial hierarchy collapses. This distinction is exactly what allows us to avoid the sub-exponential barrier.

Our security reduction has access to the witness for equivalence, which guides how the reduction operates. The reduction can use the witness to trivially verify that the two circuits are equivalent; if the witness is not supplied or is invalid, the reduction does not run. The sub-exponential barrier therefore no longer applies in this setting.

More generally, the sub-exponential barrier will not apply to circuit pairs for which there is a witness proving equivalence; in other words, languages of circuit pairs in $NP \cap co\text{-}NP$[7]. Any languages outside $NP \cap co\text{-}NP$ are likely to run into the same sub-exponential barrier as full iO since witnesses for equivalence do not exist, and meanwhile there remains some hope that languages inside might be obfuscatable without a sub-exponential loss by feeding the witness to the reduction.

In fact, almost all applications of obfuscation we are aware of can be modified so that the pairs of circuits in question have a witness proving equivalence. For example, consider obtaining public key encryption from one-way functions using obfuscation [SW14]. The secret key is the seed s for a PRG, and the public key is the corresponding output x. A ciphertext encrypting message m is an obfuscation of the program $P_{x,m}$, which takes as input a seed s' and checks that $\mathsf{PRG}(s') = x$. If the check fails, it aborts and outputs 0. Otherwise if the check

[7] Circuit equivalence is trivially in $co\text{-}NP$; a point on which the two circuits differ is a witness that they are not equivalent.

passes, it outputs m. To decrypt using s, simply evaluate obfuscated program on s.

In the security proof, iO is used for the following two programs: $P_{x,m}$ where x *is a truly random element in the co-domain of* PRG, and Z, the trivial program that always outputs 0. We note that since PRG is expanding, with high probability x will not have a pre-image, and therefore $P_{x,m}$ will also output 0 everywhere. Therefore, $P_{x,m}$ and Z are (with high probability) functionally equivalent.

For general PRGs, there is no witness for equivalence of these two programs. However, by choosing the right PRG, we can remedy this. Let P be a one-way permutation, and let h be a hardcore bit for P. Now let $\mathsf{PRG}(s) = (P(s), h(s))$. Instead of choosing x randomly, we choose x as $P(s), 1 \oplus h(s)$ for a random seed s[8]. This guarantees that x has no pre-image under PRG. Moreover, s serves as a witness that x has no pre-image. Therefore, the programs $P_{x,m}$ and Z have a witness for equivalence.

Limits of the dO Approach. Unfortunately, decomposable obfuscation is not strong enough to prove security in many settings. In fact, we demonstrate (Sect. 4) that τ-decomposing equivalence can be decided in time proportional to τ, meaning poly-decomposing equivalence is actually in P. However, for example, the equivalence of programs $P_{x,m}$ and Z above cannot possibly be in P—otherwise we could break the PRG: on input x, check if $P_{x,m}$ is equivalent to Z. A random output will yield equivalence with probability 1/2, whereas a PRG sample will never yield equivalence circuits. In other words, $P_{x,m}$ and Z are provably *not* poly-decomposing equivalent, despite being functionally equivalent programs.

One can also imagine generalizing dO to encompass more general paths through the binary tree of prefixes. For example, one could decompose the circuit into fragments, partially merge some of the fragments back together, decompose again, etc. We show that this seemingly more general *path* decomposing equivalence is in fact equivalent to (standard) decomposing equivalence. Therefore, this path dO is equivalent to (standard) dO, and only works for pairs of circuits that can be easily verified as equivalent.

Unsurprisingly then, all the applications we obtain using poly-decomposable obfuscation obfuscate circuits for which it is easy to verify equivalence. This presents some interesting limitations relative to iO:

– All known ways of getting public key encryption from iO and one-way functions suffer from a similar problem, and cannot to our knowledge be based on poly-dO. In other words, unlike iO, dO might not serve as a bridge between Minicrypt and Cryptomania. Some of our applications—namely multi-key functional encryption and trapdoor permutations—imply public key encryption; for these applications, we actually have to use public key encryption as an additional ingredient. Note that if we are instantiating dO from functional

[8] This is no longer a random element in the codomain of the PRG, but it suffices for the security proof.

encryption, we get public key encryption for free. However, if we are interested in placing dO itself in the complexity landscape, the apparent inability to give public key encryption is an interesting barrier.
More generally, a fascinating question is whether any notion of obfuscation that works only for efficiently-recognizable equivalent circuits can imply public key encryption, assuming additionally just one-way functions.
- While iO itself does not imply one-way functions[9], iO can be used in conjunction with a worst-case complexity assumption, roughly $NP \not\subseteq BPP$, to obtain one-way functions [KMN+14]. The proof works by using a hypothetical inverter to solve the circuit equivalence problem; assuming the circuit equivalence problem is hard, they reach a contradiction. The solver works exactly because iO holds for the equivalent circuits.
This strategy simply does not work in the context of dO. Indeed, dO only applies to circuits for which equivalence is easily decidable anyway, meaning no contradiction is reached. In order to obtain any results analogous to [KMN+14] for restricted obfuscation notions, the notion must always work for at least some collection of circuit pairs for which circuit equivalence is hard to decide. Put another way, dO could potentially exist in Pessiland.
- More generally, dO appears to roughly capture the most general form of the techniques in [GPS16,GPSZ16,GS16], and therefore it appears that these techniques will not extend to the case of non-efficiently checkable equivalence. Many constructions using obfuscation fall in this category of non-checkable equivalence: deniable encryption and non-interactive zero knowledge [SW14], secure function evaluation with optimal communication complexity [HW15], adaptively secure universal samples [HJK+16], and more.

We therefore leave some interesting open questions:

- Build iO for a class of circuit pairs for which equivalence is not checkable in polynomial time, but for which security can be based on the polynomial hardness of just a few assumptions.
- Modify the constructions in deniable encryption/NIZK/function evaluation/etc so that obfuscation is only ever applied on program pairs for which equivalence can be easily verified—ideally, the circuits would be decomposing equivalent.
- Prove that for some applications, obfuscation *must* be applied to program pairs with non-efficiently checkable equivalence.

2 Decomposing Equivalence and dO Definitions

In this section, we define several basic definitions including decomposing equivalence and dO.

[9] If $P = NP$, one-way functions do not exist but circuit minimization can be used to obfuscate.

2.1 Partial Evaluation on Circuits

Definition 1. *Consider a circuit C defined on inputs of length $n > 0$, for any bit $b \in \{0,1\}$, a* ***partial evaluation*** *of C on* ***bit*** *b denoted as $C(b,\cdot)$ is a circuit defined on inputs of length $n-1$, where we hardcode the input bit x_1 to b, and then simplify. To simplify, while there is a gate that has a hard-coded input, replace it with the appropriate gate or wire in the usual way (e.g. $\mathsf{AND}(1,b)$ gets replaced with the pass-through wire b, and $\mathsf{AND}(0,b)$ gets replaced with the constant 0). Then remove all unused wires.*

Also we can define a partial evaluation of a circuit C on a ***string*** *x which is repeatedly applying partial evaluations and simplifying bit by bit.*

From now on, whenever we use the expression $C(x,\cdot)$, we always refer to the result of simplifying C after hardcoding the prefix x.

2.2 Circuit Assignments

A binary tree T_n is a tree of depth $n+1$ where the root is labeled ε (an empty string), and for any node that is not a root whose parent is labeled as x, it is labeled $x||0$ if it is a left-child of its parent; it is labeled as $x||1$ if it is a right-child of its parent.

Definition 2 (Tree Covering). *We say a set of binary strings $\{x_i\}_{i=1}^{\ell}$ is a* ***tree covering*** *for all strings of length n if the following holds: for every string $x \in \{0,1\}^n$, there exists exactly one x_j in the set such that x_j is a prefix of x.*

A tree covering $\{x_i\}_{i=1}^{\ell}$ also can be viewed as a set of nodes in T_n such that for every leaf in the tree, the path from root ε to this leaf will pass exactly one node in the set.

Yet another equivalent formulation is that a tree covering is either (1) a set consisting of the root node of the tree, or (2) the union of two tree coverings for the two subtrees rooted at the children of the root node.

Definition 3 (Circuit Assignment). *We say $L = \{(x_i, C_{x_i})\}_{i=1}^{\ell}$ is a* ***circuit assignment*** *with size ℓ where $\{x_i\}_{i=1}^{\ell}$ is a tree covering for T_n and $\{C_{x_i}\}_{i=1}^{\ell}$ is a set of circuits where C_{x_i} is assigned to the node x_i in the covering.*

We say a circuit assignment is valid if for each C_{x_i}, it is defined on input length $n - |x_i|$.

An evaluation of L on input x is defined as: find the unique x_j which is a prefix of $x = x_j||x_{-j}$ and return $C_{x_j}(x_{-j})$.

We call each circuit in the assignment a ***fragment****. The* ***cardinality*** *of the circuit assignment is the size of the tree covering, and the* ***circuit size*** *is the maximum size of any fragment in the assignment.*

A circuit assignment $L = \{(x_i, C_{x_i})\}_{i=1}^{\ell}$ naturally corresponds to a function: on input $y \in \{0,1\}^n$, scan the prefix of y from left to right until we find the smallest i such that $y_{[i]}$ equals to some x_j, output $C_{x_j}(y_{[i+1\cdots n]})$. We will override the notation and write this function as $L(x)$.

We associate a circuit C with the assignment $L_C = \{(\varepsilon, C)\}$ which assigns C to the root of the tree. Notice that L_C and C are equivalent as functions.

Definition 4 (one shot decomposing equivalent). *Given two circuits C_0, C_1 defined on inputs of length n, we say they are τ-**one shot decomposing equivalent** or simply τ-**decomposing equivalent** if the following hold:*

- *There exists a tree covering $\mathcal{X} = \{x_i\}_i$ of size at most τ;*
- *For all $x_i \in \mathcal{X}$, $C_0(x_i, \cdot) = C_1(x_i, \cdot)$ as circuits (they are exactly the same circuit).*

Definition 5. dO *with two PPT algorithms* {dO.ParaGen, dO.Eval} *is a $\tau(n, s, \kappa)$-decomposing obfuscator if the following conditions hold*

- ***Efficiency:*** dO.ParaGen, dO.Eval *are efficient algorithms;*
- ***Functionality preserving:*** dO.ParaGen *takes as input a security parameter κ and a circuit C, and outputs the description $\hat{C}$ of an obfuscated program. For all κ and all circuit C, for all input $x \in \{0,1\}^n$, we have* $\mathsf{dO.Eval}(\mathsf{dO.ParaGen}(1^\kappa, C), x) = C(x)$*;*
- ***Decomposing indistinguishability:*** *Consider a pair of PPT adversaries $(Samp, D)$ where $Samp$ outputs a tuple (C_0, C_1, σ) where C_0, C_1 are circuits of the same size $s = s(\kappa)$ and input length $n = n(\kappa)$. We require that, for any such PPT $(Samp, D)$, if*

$$\Pr[C_0 \text{ is } \tau\text{-decomposing equivalent to } C_1 : (C_0, C_1, \sigma) \leftarrow Samp(\kappa)] = 1$$

then there exists a negligible function $\mathsf{negl}(\kappa)$ *such that*

$$|\Pr[D(\sigma, \mathsf{dO.ParaGen}(1^\kappa, C_0)) = 1] - \Pr[D(\sigma, \mathsf{dO.ParaGen}(1^\kappa, C_1)) = 1]| \leq \mathsf{negl}(\kappa)$$

Note that the size of parameters generated by dO.ParaGen is bounded by $\mathsf{poly}(n, \kappa, \tau, |C|)$. But however you will see later that τ can always be replaced by n so even if $\tau = \Omega(2^n)$, the size is still bounded by $\mathsf{poly}(n, \kappa, |C|)$ (but you will have τ-loss in the security analysis).

And in the next section we will discuss about the applications of dO and later come back to more discussions about dO including constructions and relations between different iO.

3 Applications

3.1 Notations

Before all the applications, let us first introduce several definitions for convenience.

First let us look at some operations defined on circuits (or circuit assignments).

1. $\mathsf{Decompose}(L, x)$ takes a circuit assignment L and a string x as parameters. This operation is invalid if x is not in the tree covering. The new circuit assignment has a slightly different tree covering: the new tree covering includes $x||0$ and $x||1$ but not x. It decomposes the fragment C_x into two fragments $C_x(0, \cdot)$ and $C_x(1, \cdot)$ and assigns them to $x||0$ and $x||1$ respectively.

2. $\mathsf{CanonicalMerge}(L, x)$ operates on an assignment L where the tree covering includes both children of node x but not x itself. It takes two circuits $C_{x||0}, C_{x||1}$ assigned to the node $x||0$ and $x||1$ and merge them to get the following circuit $C_x(b, y) = (b \wedge C_{x||0}(y)) \vee (\bar{b} \wedge C_{x||1}(y))$ (Here we assume the output length of both circuits is 1. It is straightforward to extend the definition to circuits with any output length). The new tree covering has x but not $x||0$ or $x||1$.
   One observation is that for any circuit assignment whose tree covering has $x||0$ and $x||1$ but not x and $C_{x||0}, C_{x||1}$ can not be simplified any further, $\mathsf{Decompose}(\mathsf{CanonicalMerge}(L, x), x) = L$.
3. $\mathsf{DecomposeTo}(L, TC)$: It takes a circuit assignment L (if the first parameter is a circuit C, then $L = \{(C, \varepsilon)\}$) and a tree covering TC where TC is below the covering in L. This procedure keeps taking the lexicographically first circuit fragment C_x which x is not in TC and do $\mathsf{Decompose}(L, x)$. Because the covering in L is above TC, the procedure halts when the covering in the new circuit assignment is exactly TC.
   We can also define $\mathsf{DecomposeTo}(L, x) = \mathsf{DecomposeTo}(L, TC_x)$ where TC_x is a tree covering that consists all the nodes adjacent to the path from root to node x, in other words, $TC_x = \{\neg x_1, x_1\neg x_2, x_1 x_2 \neg x_3, \cdots, x_{|x|-1}\neg x_{|x|}, x\}$ (a full description is in Sect. 4).
4. $\mathsf{CanonicalMerge}(L)$: it canonically merges all the way to the root. In other words, the procedure keeps taking the lexicographically first circuit fragment pair $C_{x||0}$ and $C_{x||1}$ and doing $\mathsf{CanonicalMerge}(L, x)$ until the tree covering in the circuit assignment is $\{\varepsilon\}$, in other words, it becomes a single circuit.

Note that the functionality of a circuit assignment is preserved under applying any valid operation above.

We now define an decomposing compatible pseudo random function. The construction [GGM86] automatically satisfies the definition below.

Definition 6. *An decomposing compatible pseudo random function* DPRF *consists the following algorithms* DPRF.KeyGen *and* DPRF.Eval *where*

- DPRF.Eval *takes a input of length n and the output is of length $p(n)$ where p is a fixed polynomial;*
- ***(PRF Security).*** *For any poly sized adversary $\mathcal{A}$, there exists a negligible function* negl*, for any string $y_0 \in \{0,1\}^n$ and any κ,*

$$|\Pr[\mathcal{A}(\mathsf{DPRF.Eval}(S, y_0)) = 1] - \Pr[\mathcal{A}(r) = 1]| \leq \mathsf{negl}(\kappa)$$

  where $S \leftarrow \mathsf{DPRF.KeyGen}(1^\kappa)$ and $r \in \{0,1\}^{p(n)}$ is a uniformly random string.
- ***(EPRF Security).*** *Consider the following game, let $\mathsf{Game}_{\kappa,\mathcal{A},b}$ be*
  - *The challenger prepares $S \leftarrow \mathsf{DPRF.KeyGen}(1^\kappa)$;*
  - *The adversary makes queries about x and gets $\mathsf{DPRF.Eval}(S, x)$ back from the challenger;*
  - *The adversary gives a tree covering TC and $y^* \in TC$ to the challenger where y^* is not a prefix of any x that has been asked;*

- *The challenger sends the distribution D_b back to the adversary $\mathcal{A}$ where*
  - ∗ D_0*: let the circuit D to be $D(\cdot) = \mathsf{DPRF.Eval}(S, \cdot)$ defined on $\{0,1\}^n$, the circuit assignment is $\mathsf{DecomposeTo}(D, TC)$. We observe that the fragment corresponding to y is $\mathsf{DPRF.Eval}(S, y, \cdot)$ defined on $\{0,1\}^{n-|y|}$.*
  - ∗ D_1*: For each $y \neq y^* \in TC$, let the fragment corresponding to y be $D_y(\cdot) = \mathsf{DPRF.Eval}(S, y, \cdot)$ defined on $\{0,1\}^{n-|y|}$ and for y^*, $D_{y^*}(\cdot) = \mathsf{DPRF.Eval}(S', y^*, \cdot)$ defined on $\{0,1\}^{n-|y^*|}$ where $S' \leftarrow \mathsf{DPRF.KeyGen}(1^\kappa)$.*
- *The adversary can keep making queries about x which does not have prefix y^* and gets $\mathsf{DPRF.Eval}(S, x)$ back from the challenger;*
- *The output of this game is the output of $\mathcal{A}$.*

For any poly sized adversary $\mathcal{A}$, there exists a negligible function negl *such that:*

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

Let us define an another operation on a circuit assignment and a circuit.

Definition 7. *By given a circuit C and a circuit assignment L where C takes two inputs x and $L(x)$, $C(\cdot, L(\cdot))$ is a circuit assignment defined below:*

- *Let TC be the tree covering inside $L = \{(x, D_x)\}_{x \in TC}$.*
- *Let $L' = \mathsf{DecomposeTo}(C, TC) = \{(x, C_x)\}_{x \in TC}$.*
- *For each fragment in the output circuit assignment corresponding to $x \in TC$, it is $C_x(\cdot, D_x(\cdot))$ simplified, which is defined on $\{0,1\}^{n-|x|}$.*

We can also define similar operations on several circuit assignments and one circuit as long as these circuit assignments have the same tree covering. In other words, let $L_1, \cdots, L_m (L_i = \{(x, D_x^i)\}$ are circuit assignments with the same tree covering TC, then $C(\cdot, L_1(\cdot), L_2(\cdot), \cdots, L_m(\cdot))$ is a circuit assignment whose fragment corresponding to $y \in TC$ is $C(y, \cdot, D_y^1(\cdot), \cdots, D_y^m(\cdot))$ simplified.

Then we have the following lemma:

Lemma 1. *For any two circuits C, D where D takes a single input x and C takes two inputs x and $D(x)$, for any tree covering TC, we have*

$$\mathsf{DecomposeTo}(C(\cdot, D(\cdot)), TC) = C(\cdot, [\mathsf{DecomposeTo}(D, TC)](\cdot))$$

For $m+1$ circuits $C, D_1, D_2, \cdots, D_m$, where $D_1, \cdots, D_m$ take a single input x and C takes x and $D_1(x) \cdots D_m(x)$ as inputs, we have

$$\begin{aligned}&\mathsf{DecomposeTo}(C(\cdot, D_1(\cdot), \cdots, D_m(\cdot)), TC)\\&\quad = C(\cdot, \mathsf{DecomposeTo}(D_1, TC), \cdots, \mathsf{DecomposeTo}(D_m, TC))\end{aligned}$$

Proof. Let us first look at the left side. It is a circuit assignment with the tree covering TC. For the fragment corresponding to $y \in TC$, it is the partial evaluation of $C(\cdot, D(\cdot))$ on y.

For the right side, we first have a circuit assignment $\mathsf{DecomposeTo}(D, TC)$ where the fragment corresponding to y is $D(y,\cdot)$. So by the definition of our operation, the fragment corresponding to y in the right side is $C(y,\cdot,D(y,\cdot))$ simplified.

Since each pair of fragments are the same, the left side is equal to the right side.

3.2 Short Signatures

Here, we show how to use dO to build short signatures, following [SW14]. As in [SW14], we will construct statically secure signatures.

The signature is simply of the following form $f(\mathsf{DPRF.Eval}(S, m))$ where f is a one-way function.

Definition 8. *A signature scheme* SS *consists of the following algorithms:*

- $\mathsf{SS.Setup}(1^\kappa)$*: it outputs a verification key* vk *and a signature key* sk*;*
- $\mathsf{SS.Sign}(\mathsf{sk}, m)$*: it is a deterministic procedure; it takes a signature key and a message, then outputs a signature* σ*;*
- $\mathsf{SS.Ver}(\mathsf{vk}, m, \sigma)$*: it is a deterministic algorithm; it takes a verification key, a message* m *and a signature* σ*, it outputs* 1 *if it accepts;* 0 *otherwise.*

We say a short signature scheme is correct if for any message $m \in \{0,1\}^\ell$*:*

$$\Pr\left[\mathsf{SS.Ver}(\mathsf{vk}, m, \sigma) = 1 \;\middle|\; \begin{array}{r}(\mathsf{vk}, \mathsf{sk}) \leftarrow \mathsf{SS.Setup}(1^\kappa) \\ \sigma \leftarrow \mathsf{SS.Sign}(\mathsf{sk}, m)\end{array}\right] = 1$$

We now define security for short signatures.

Definition 9. *We denote* $\mathsf{Game}_{\kappa,\mathcal{A}}$ *to be the following where* κ *is the security parameter and* $\mathcal{A}$ *is an adversary:*

- *First* $\mathcal{A}$ *announces a message* m^* *of length* ℓ*;*
- *The challenger gets* m^* *and prepares two keys* sk *and* vk*; it then sends* vk *back to* $\mathcal{A}$*;*
- $\mathcal{A}$ *can keep making queries* m' *to the challenger and gets* $\mathsf{Sign}(\mathsf{sk}, m')$ *back for any* $m' \neq m^*$*;*
- *Finally* $\mathcal{A}$ *sends a forged signature* σ^* *and the output of the game is* $\mathsf{Ver}(\mathsf{vk}, m^*, \sigma^*)$*.*

We say SS is secure if for any polysized $\mathcal{A}$, there exists a negligible function negl,

$$\Pr[\mathsf{Game}_{\kappa,\mathcal{A}} = 1] \le \mathsf{negl}(\kappa)$$

Algorithm 1. Verification Algorithm

```
procedure V(m, σ, DPRF.Eval(S, m))
    it computes σ' ← DPRF.Eval(S, m)
    if f(σ) = f(σ') then
        return 1
    else
        return 0
    end if
end procedure
```

Construction. We now give a signature scheme where signatures are short. The construction is similar with that in [SW14] but we use dO instead of iO. Our SS has the following algorithms:

- $\mathsf{SS.Setup}(1^\kappa)$: it takes a security parameter κ and prepares a key $S \leftarrow \mathsf{DPRF.KeyGen}(1^\kappa)$. S is the secret key sk. Then it computes the verification key as $\mathsf{vk} \leftarrow \mathsf{dO.ParaGen}(1^\kappa, V(\cdot, \mathsf{DPRF.Eval}(S, \cdot)))$ where V is given in Algorithm 1 (we will pad programs to a length upper bound before applying dO).
- $\mathsf{SS.Sign}(\mathsf{sk}, m) = \mathsf{DPRF.Eval}(S, m)$
- $\mathsf{SS.Ver}(\mathsf{vk}, m, \sigma) = \mathsf{dO.Eval}(\mathsf{vk}, \{m, \sigma\})$

It is straightforward to see that the construction satisfies correctness.

Security

Theorem 1. *If* dO *is a secure* $\mathsf{poly\text{-}dO}$, DPRF *is a secure decomposing compatible* PRF, *and f is a one-way function, then the construction above is a short secure signature scheme.*

Proof. Now prove security through a sequence of hybrid experiments.

- **Hyb 0:** In this hybrid, we are in $\mathsf{Game}_{\kappa,\mathcal{A}}$;
- **Hyb 1:** In this hybrid, since the challenger gets m^* before it releases vk, we decompose the circuit to get $L = \mathsf{DecomposeTo}(V(\cdot, \mathsf{DPRF.Eval}(S, \cdot)), m^*)$. By Lemma 1, the circuit assignment is $V(\cdot, \mathsf{DecomposeTo}(\mathsf{DPRF.Eval}(S, \cdot), m^*))$.
  Therefore we have that the distributions $\mathsf{dO.ParaGen}(1^\kappa, V(\cdot, \mathsf{DPRF.Eval}(S, \cdot)))$ and $\mathsf{dO.ParaGen}(1^\kappa, \mathsf{CanonicalMerge}(L))$ are indistinguishable, since these two circuits are $\ell + 1$-decomposing equivalent by applying dO.
- **Hyb 2:** This is the same as **Hyb 1**, except that we replace the fragment in $\mathsf{DecomposeTo}(\mathsf{DPRF.Eval}(S, \cdot), m^*)$ corresponding to m^*—which is "**return** $\mathsf{DPRF.Eval}(S, m^*)$"—by "**return** $\mathsf{DPRF.Eval}(S', m^*)$" where $S' \leftarrow \mathsf{DPRF.KeyGen}(1^\kappa)$ is a fresh random DPRF key that is independent of S. We call the new circuit assignment L'. **Hyb 1** and **Hyb 2** are indistinguishable because of the DPRF security.

– **Hyb 3:** This is the same as **Hyb 2**, except that we replace the fragment in L', which is "**return** $\mathsf{DPRF.Eval}(S', m^*)$" by "**return** r^*" where r^* is a uniformly random string. We call the new circuit assignment L''. As we don't have S' in the program anywhere except this fragment, **Hyb 2** and **Hyb 3** are indistinguishable because of the PRF security.
  We find that in $\mathsf{CanonicalMerge}(L'')$, the fragment corresponding to m^* is: on input σ, it returns 1 if $f(\sigma) = v^*$; 0 otherwise, where $v^* = f(r^*)$ for a uniformly random r^*.

Lemma 2. *If there exists a poly sized adversary $\mathcal{A}$ for Hyb 3, then we can break one-way function f.*

Proof. Given z^* which is $f(r^*)$ for a truly random r^*, we can actually simulate **Hyb 3**. If we successfully find a forged signature for **Hyb 3** with non-negligible probability, it is actually a pre-image of z^* which means we break one-way function with non-negligible probability.

This completes the security proof.

3.3 Universal Samplers

Here we construct universal samplers from dO. For the sake of simplicity, we will show how to construct samplers meeting the one-time static definition from [HJK+16]. However, note that the same techniques also can be used to construct the more complicated k-time interactive simulation notion of [GPSZ16].

Let US denote an universal sampler. It has the following procedures:

– $\mathsf{params} \leftarrow \mathsf{US.Setup}(1^\kappa, 1^\ell, 1^t)$: the Setup procedure takes a security parameter κ, a program size upper bound ℓ and a output length t and outputs an parameter params;
– $\mathsf{US.Sample}(\mathsf{params}, C)$ is a deterministic procedure that takes a params and a sampler C of length at most ℓ where C outputs a sample of length t. This procedure outputs a sample s;
– $\mathsf{params}' \leftarrow \mathsf{US.Sim}(1^\kappa, 1^\ell, 1^t, C^*, s^*)$ takes a security parameter κ, a program size upper bound ℓ and a output length t, also a circuit C^* and a sample s^* in the image of C^*.

Correctness. For any C^* and s^* in the image of C^*, and for any $\ell \geq |C^*|$, and t is a upper bound for C^*'s outputs, we have

$$\Pr\left[\mathsf{US.Sample}(\mathsf{params}', C^*)] = s^* \mid \mathsf{params}' \leftarrow \mathsf{US.Sim}(1^\kappa, 1^\ell, 1^t, C^*, s^*)\right] = 1$$

Security. For any ℓ and t, for any C^* of size at most ℓ and output size at most t, for any poly sized adversary $\mathcal{A}$, there exists a negligible function negl, such that

$$\Big| \Pr[\mathcal{A}(\mathsf{params}, C^*) = 1 \mid \mathsf{params} \leftarrow \mathsf{US.Setup}(1^\kappa, 1^\ell, 1^t)] - \Pr\left[\mathcal{A}(\mathsf{params}', C^*) = 1 \,\Big|\, \begin{array}{l}\mathsf{params}' \leftarrow \mathsf{US.Sim}(1^\kappa, 1^\ell, 1^t, C^*, s^*), \\ s^* \xleftarrow[R]{} C^*(\cdot)\end{array}\right] \Big| \leq \mathsf{negl}(\kappa)$$

where $s^* \xleftarrow[R]{} C^*(\cdot)$ means s^* is a truly random sample from $C^*(\cdot)$.

Construction. Now we give the detailed construction for our universal sampler:

- Define U to be the size upper bound among all the circuits being obfuscated in our proof (not the size of circuits fed into the universal sampler). It is straightforward to see that $U = \mathsf{poly}(\kappa, \ell, t)$; Whenever we mention $\mathsf{dO.ParaGen}(1^\kappa, C)$, we will pad C to have size U.
- For simplicity, we will assume circuits C fed into the universal sampler will always be padded to length ℓ so that we can consider only circuits of a fixed size.
- $\mathsf{US.Setup}(1^\kappa, 1^\ell, 1^t)$ randomly samples a key $S \leftarrow \mathsf{DPRF.KeyGen}(1^\kappa)$, and constructs a circuit $\mathsf{Sampler}$ (see Algorithm 2) as follows: on input circuit C of size ℓ, it outputs a sample based on the randomness generated by DPRF; and the output of the procedure $\mathsf{US.Setup}$ is $\mathsf{params} = \mathsf{dO.ParaGen}(1^\kappa, \mathsf{Sampler}(\cdot, \mathsf{DPRF.Eval}(S, \cdot)))$.

Algorithm 2. Sampler Algorithm

```
procedure Sampler(C = c_1 c_2 ··· c_ℓ, DPRF.Eval(S, C))
    r_C ← DPRF.Eval(S, C)
    return C(; r_C)
end procedure
```

- $\mathsf{US.Sample}(\mathsf{params}, C)$: it simply outputs $\mathsf{dO.Eval}(\mathsf{params}, C)$;
- $\mathsf{US.Sim}(1^\kappa, 1^\ell, 1^t, C^*, s^*)$: it randomly samples a key $S \leftarrow \mathsf{DPRF.KeyGen}(1^\kappa)$, let L be a circuit assignment $\mathsf{Sampler}(\cdot, \mathsf{DecomposeTo}(\mathsf{DPRF.Eval}(S, \cdot), C^*))$. And finally it replaces the fragment corresponding to C^* in L with "**return** s^*" instead of returning $C^*(; \mathsf{DPRF.Eval}(S, C^*))$. Let $\mathsf{Sampler}' = \mathsf{CanonicalMerge}(L)$ and the output of $\mathsf{US.Sim}$ is $\mathsf{params}' = \mathsf{dO.ParaGen}(1^\kappa, \mathsf{Sampler}')$.

Theorem 2. *If* dO *and one-way functions exist, then there exists an universal sampler.*

Proof. First, it is straightforward that correctness is satisfied. Next we prove security. Fix a circuit C^* and suppose there is an adversary $\mathcal{A}$ for the sampler security game for C^*. We prove the indistinguishability through a sequence of hybrids:

- **Hyb 0:** Here, the adversary receives $\mathsf{params} \leftarrow \mathsf{US.Setup}(1^\kappa, 1^\ell, 1^t)$;
- **Hyb 1:** In this hybrid, let $s^* \leftarrow C^*(; \mathsf{DPRF.Eval}(S, C^*))$. We get $\mathsf{params}_1 \leftarrow \mathsf{US.Sim}(1^\kappa, 1^\ell, 1^t, C^*, s^*)$ where $\mathsf{Sampler}_1$ is the circuit constructed in $\mathsf{US.Sim}$ where we are using the same S in **Hyb 0**.

  It is straightforward that $\mathsf{Sampler}_1$ and $\mathsf{Sampler}$ are $\ell + 1$-decomposing equivalent. Therefore $\mathsf{params}_1 = \mathsf{dO.ParaGen}(1^\kappa, \mathsf{Sampler}_1)$ and $\mathsf{params} = \mathsf{dO.ParaGen}(1^\kappa, \mathsf{Sampler})$ are indistinguishable by dO security, meaning **Hyb 0** and **Hyb 1** are indistinguishable.

– **Hyb 2:** This is the same as **Hyb 1**, except we replace the fragment in $\mathsf{DecomposeTo}(\mathsf{DPRF.Eval}(S, \cdot), C^*)$ corresponding to C^* with the fragment "**return** $\mathsf{DPRF.Eval}(S', C^*)$" where $S' \leftarrow \mathsf{DPRF.KeyGen}(1^\kappa)$ is a new key generated by a uniformly random string. We call the new circuit assignment L'. The indistinguishability between **Hyb 1** and **Hyb 2** follows from the DPRF security.
– **Hyb 3:** In this hybrid, since the fragment in L' corresponding to C^* is now returning $C^*(; \mathsf{DPRF.Eval}(S', C^*))$ and we don't have S' in the program, by PRF security, we can replace the return value with $C(; r^*)$ where r^* is a truly random string. This is equivalent to the adversary receiving $\mathsf{params} \leftarrow \mathsf{US.Sim}(1^\kappa, 1^\ell, 1^t, C^*, s^*)$ for a fresh sample $s^* \leftarrow C^*$.

4 Constructions of dO

In this section, we give more discussions about decomposing equivalence and dO. And finally we give the constructions of dO from compact functional encryption schemes.

4.1 New Notions of Equivalence for Circuits

We will define a partial order $\preceq$ on nodes in a binary tree. We say that $x \preceq y$ (alternatively, x is **above** y) if x is a prefix of y. We also extend our partial order $\preceq$ to tree coverings. We say a tree covering $TC_0 \preceq TC_1$, or TC_0 is **above** TC_1, if for every node u in TC_1, there exists a node v in TC_0 such that $v \preceq u$ (that is, v is equal to u or an ancestor of u). A tree covering TC_0 is **below** TC_1 if TC_1 is above TC_0. It is straightforward that if $TC_0 \preceq TC_1$, then $|TC_0| \leq |TC_1|$ where $|TC_0| = |TC_1|$ if and only if $TC_0 = TC_1$. We can also extend $\preceq$ to compare tree coverings to nodes. We have $u \preceq TC$ if there is a node $v \in TC$ such that $u \preceq v$. $TC \preceq u$ if there exists a $v \in TC$ such that $v \preceq u$.

We give more operations defined on circuits and circuit assignments for convenience.

– $\mathsf{Decompose}(L, x)$: mentioned in Sect. 3.1.
– $\mathsf{CanonicalMerge}(L, x)$: mentioned in Sect. 3.1.
– $\mathsf{TargetedMerge}(L, x, C)$ operates on an assignment L where the tree covering includes both children of node x but not x itself. This operation is invalid if either $C(0, \cdot) \neq C_{x||0}$ or $C(1, \cdot) \neq C_{x||1}$ as circuits. It takes the two circuits $C_{x||0}, C_{x||1}$ assigned to the node $x||0$ and $x||1$ and merges them to get $C_x = C$. The new tree covering has x but not $x||0$ or $x||1$.
  We observe that
  - $\mathsf{Decompose}(\mathsf{TargetedMerge}(L, x, C), x) = L$ where $C_{x||0}$ and $C_{x||1}$ in L can not be simplified any further, and all the operations are valid
  - $\mathsf{TargetedMerge}(\mathsf{Decompose}(L, x), x, C) = L$ where C is the fragment at node x in L (as long as the operations are valid).

- DecomposeTo(L, x): takes a circuit assignment L and a string x as parameters. The operation is valid if $TC \preceq x$, where TC is the tree covering for L. Let u be ancestor of x in TC. Let $p_0 = u, p_1, \dots, p_t = x$ be the path from u to x. DecomposeTo first sets $L_0 = L$, and then runs $L_i \leftarrow$ Decompose(L_{i-1}, p_{i-1}) for $i = 1, \dots, t$. The output is the new circuit assignment $L' = L_t$. The new tree covering TC' for L' is the minimal TC' that is both below TC and contains x.
  We will also extend DecomposeTo to operate on circuits in addition to assignments, by first interpreting the circuit as an assignment, and performing DecomposeTo on the assignment.
- DecomposeTo(L, TC): mentioned in Sect. 3.1.
- CanonicalMerge(L, TC): It takes a circuit assignment L and a tree covering TC where TC is below the covering in L. It repeatedly performs CanonicalMerge(L, x) at different x until the tree covering in the assignment becomes TC. To make the merging truly canonical, we need to specify an order that nodes are merged in. We take the convention that the lowest nodes in the tree are merged first, and between nodes in the same level, the leftmost nodes are merged first.
- CanonicalMerge(L) = CanonicalMerge$(L, \{\varepsilon\})$: mentioned in Sect. 3.1.

4.2 Locally, Path, One Shot Decomposing Equivalence

We define two new equivalence notions for circuits based on the decomposing and merging operations defined above. First, we define a local equivalence condition on circuit assignments:

Definition 10 (locally decomposing equivalent). *We say two circuit assignments $L_1 = \{(x_i, C_{x_i})\}, L_2 = \{(y_i, C'_{y_i})\}$ are (ℓ, s)-**locally decomposing equivalent** if the following hold:*

- *The circuit size of L_1, L_2 is at most s;*
- *The cardinality of L_1, L_2 is at most ℓ;*
- *L_1 can be obtained from L_2 by applying* Decompose(L_2, x) *for some x or by applying* TargetedMerge(L_2, x, C) *for some x and C is the fragment assigned in L_1 to the string (node) x;*

Local decomposing equivalence (Local DE) means that we can transform L_1 into L_2 by making just a single local change, namely decomposing a node or merging two nodes. Notice that since decomposing a node does not change functionality, local DE implies that L_1 and L_2 compute equivalent functions. For any ℓ, s, (ℓ, s)-local decomposing equivalence forms a graph, where nodes are circuit assignments and edges denote local decomposing equivalence. Next, we define a notion of *path* decomposing equivalence for circuits (which can be thought of as nodes in the graph), which says that two circuits are equivalent if they are connected by a reasonably short path through the graph.

Definition 11 (path decomposing equivalent). *We say two circuits C_1, C_2 are (ℓ, s, t)-**path decomposing equivalent** if there exists at most $t-1$ circuit assignments $L'_1, L'_2, \cdots, L'_{t-1}$ such that, for any $1 \leq i \leq t$, L'_{i-1} and L'_i are (ℓ, s)-locally decomposing equivalent, where $L'_0 = \{(\varepsilon, C_1)\}$ and $L'_t = \{(\varepsilon, C_2)\}$.*

Now let's recall the definition of one shot decomposing equivalent which allows for exactly two steps to get between C_1 and C_2. Now the steps are not confined to be local, but instead the first step is allowed to decompose the root to a given tree covering, and the second then merges the tree covering all the way back to the root.

Recall Definition 4 (one shot decomposing equivalent). *Given two circuits C_0, C_1 defined on inputs of length n, we say they are τ-**one shot decomposing equivalent** or simply τ-**decomposing equivalent** if the following hold:*

- *There exists a tree covering $\mathcal{X} = \{x_i\}_i$ of size at most τ;*
- *For all $x_i \in \mathcal{X}$, $C_0(x_i, \cdot) = C_1(x_i, \cdot)$ as circuits.*

An equivalent definition for "τ-one shot decomposing equivalent" is that there exists a tree covering $\mathcal{X}$ of size at most τ, such that $\mathsf{DecomposeTo}(\{(\varepsilon, C_0)\}, \mathcal{X}) = \mathsf{DecomposeTo}(\{(\varepsilon, C_1)\}, \mathcal{X})$*, in other words, the tree coverings are the same and the corresponding fragments for each node are the same.*

We note that since the operations defining path and one shot decomposing equivalence all preserve functionality, we have that these notions imply standard functional equivalence for the circuits:

Lemma 3. *If C_0, C_1 are (ℓ, s, t)-path decomposing equivalent for any ℓ, s, t, or if C_0, C_1 are τ-one shot decomposing equivalent for any τ, then C_0, C_1 compute equivalent functions ($C_0(x) = C_1(x), \forall x \in \{0,1\}^n$).*

We also observe a partial converse:

Lemma 4. *Two circuits C_0, C_1 (defined on n bits string) are 2^n-one shot decomposing equivalent if and only if they are functionally equivalent ($C_0(x) = C_1(x), \forall x \in \{0,1\}^n$).*

Proof. We only need to show the case that functional equivalence implies 2^n-one shot decomposing equivalence. If C_0, C_1 are functionally equivalent, we can let the tree covering be $\mathcal{X} = \{0,1\}^n$. Because $C_0(x) = C_1(x)$ for all $x \in \{0,1\}^n = \mathcal{X}$, we have $\mathsf{DecomposeTo}(\{(\varepsilon, C_0)\}, \mathcal{X}) = \mathsf{DecomposeTo}(\{(\varepsilon, C_1)\}, \mathcal{X})$. Therefore C_0, C_1 are 2^n-one shot decomposing equivalent.

4.3 Locally, One Shot dO

Here, we will recall decomposing obfuscation (dO) and give one more definition. Let us recall the definition of dO. Decomposable obfuscator, roughly, is an indistinguishability obfuscator, but where the indistinguishability security requirement only applies to pairs of circuits that are decomposing equivalent (as opposed to applying to all equivalent circuits).

Recall Definition 5. *dO wtih two PPT algorithms $\{\mathsf{dO.ParaGen}, \mathsf{dO.Eval}\}$ is a $\tau(n, s, \kappa)$-decomposable obfuscator if the following conditions hold*

- ***Efficiency:*** *$\mathsf{dO.ParaGen}, \mathsf{dO.Eval}$ are efficient algorithms;*
- ***Functionality preserving:*** *$\mathsf{dO.ParaGen}$ takes as input a security parameter κ and a circuit C, and outputs the description $\hat{C}$ of an obfuscated program. For all κ and all circuit C, for all input $x \in \{0,1\}^n$, we have $\mathsf{dO.Eval}(\mathsf{dO.ParaGen}(1^\kappa, C), x) = C(x)$;*
- ***Decomposing indistinguishability:*** *Consider a pair of PPT adversaries $(Samp, D)$ where Samp outputs a tuple (C_0, C_1, σ) where C_0, C_1 are circuits of the same size $s = s(\kappa)$ and input length $n = n(\kappa)$. We require that, for any such PPT $(Samp, D)$, if*

  $$\Pr[C_0 \text{ is } \tau(n,s,\kappa)\text{-decomposing equivalent to } C_1 : (C_0, C_1, \sigma) \leftarrow Samp(\kappa)] = 1$$

  then there exists a negligible function $\mathsf{negl}(\kappa)$ such that

  $$|\Pr[D(\sigma, \mathsf{dO.ParaGen}(1^\kappa, C_0)) = 1] - \Pr[D(\sigma, \mathsf{dO.ParaGen}(1^\kappa, C_1)) = 1]| \leq \mathsf{negl}(\kappa)$$

Since 2^n-equivalence corresponds to standard equivalence, 2^n-dO is equivalent to the standard notion of iO. In this work, we will usually consider a much weaker setting, where τ is restricted to a polynomial.

The following tool, called *local* dO (ldO), will be used to help us build dO. Roughly, ldO is an obfuscator for *circuit assignments* with the property that local changes to the assignment (that is, decomposing operations) are computationally undetectable.

Definition 12. *ldO with two PPT algorithms $\{\mathsf{ldO.ParaGen}, \mathsf{ldO.Eval}\}$ is a locally decomposable obfuscator if the following conditions hold*

- ***Efficiency:*** *$\mathsf{ldO.ParaGen}, \mathsf{ldO.Eval}$ are efficient algorithms;*
- ***Functionality preserving:*** *$\mathsf{ldO.ParaGen}$ takes as input a security parameter κ, a circuit assignment L, a cardinality bound ℓ, and a circuit size bound s. For all κ and all circuit assignment L with cardinality at most ℓ and circuit size at most s, for all input $x \in \{0,1\}^n$, we have $\mathsf{ldO.Eval}(\mathsf{ldO.ParaGen}(1^\kappa, L, \ell, s), x) = L(x)$;*
- ***Local decomposing indistinguishability:*** *Consider polynomials $\ell = \ell(\kappa)$ and $s = s(\kappa)$. For any such polynomials, and any pair of PPT adversaries $(Samp, D)$, we require that if*

  $$\Pr[L_0 \text{ is } (\ell(\kappa), s(\kappa))\text{-local decomp. equiv. to } L_1 : (L_0, L_1, \sigma) \leftarrow Samp(\kappa)] = 1$$

  then there exists a negligible function $\mathsf{negl}(\kappa)$ such that

  $$|\Pr[D(\sigma, \mathsf{ldO.ParaGen}(1^\kappa, L_0, \ell, s)) = 1] - \Pr[D(\sigma, \mathsf{ldO.ParaGen}(1^\kappa, L_1, \ell, s)) = 1]| \leq \ell \cdot \mathsf{negl}(\kappa)$$

We will also consider a stronger variant, called sub-exponentially secure local dO, where in the definition of local decomposing indistinguishability, the negligible function negl is replaced by a subexponential function subexp.

4.4 Locally dO Implies One Shot dO

Lemma 5. *If two circuits C_0, C_1 are $(t/2+1)$-one shot decomposing equivalent, then they are $(n+1, s, t)$-path decomposing equivalent where $s = \max\{|C_0|, |C_1|\}$.*

Proof. We start from the covering that has C_0 assigned to the root. We perform a depth-first traversal of the binary search tree consisting of the "bad" nodes: nodes for which the partial evaluations of C_0 and C_1 are different. Equivalently, we search over the ancestors of nodes in the tree covering. There are $t/2$ such nodes. When we first visit a node on our way down the tree, we Decompose the fragment at that node to its children. When we visit a node x for the second time after processing both children, we merge the fragments in the two children, using a TargetedMerge toward the circuit $(C_1)_x$. This operation is always valid since for each child either: (1) the child is a "good" node, in which case the partial evaluations at that node is identical to the partial evaluation of $(C_1)_{x||b}$; or (2) the child is a "bad" node, in which case it was, by induction, already processed and replaced with the partial evaluation of $(C_1)_{x||b}$. The cardinality of any circuit assignment in this path is at most $n+1$ since we will only have fragments adjacent to the path from the root to the node we are visiting. The circuit size is moreover always bounded by $s = \max\{|C_0|, |C_1|\}$ because all the intermediate fragments are partial evaluations of either C_0 or C_1. Finally, the path performs an Decompose and TargetedMerge for each "bad" node, corresponding to t operations.

Now we show that the existence of ldO implies the existence of dO.

Lemma 6. *If* ldO *exists, then* τ-dO *exists, where the loss in the security reduction is* $2(\tau - 1)$. *In particular, if polynomially secure* ldO *exists, then* τ-dO *exists for any polynomial function* τ. *Moreover, if subexponentially secure* ldO *exists, then* 2^n-dO, *and hence* iO, *exists.*

Proof. The construction of ldO from dO is the natural one: to obfuscate a circuit C, we simply consider the circuit as a circuit assignment with C assigned to the root node, and obfuscate this circuit assignment. We take the maximum cardinality for ldO to be $n+1$ and the circuit size to be $|C|$.

- $\mathsf{dO.ParaGen}(1^\kappa, C) = \mathsf{ldO.ParaGen}(1^\kappa, \{(\varepsilon, C)\}, n+1, |C|)$;
- $\mathsf{dO.Eval}(\mathsf{params}, x) = \mathsf{ldO.Eval}(\mathsf{params}, x)$;

Efficiency and functionality preservation are straightforward to prove. Now we focus on security. Let (Samp, D) be two PPT adversaries, and s, n be polynomials in κ. Suppose the circuits C_0, C_1 outputted by $\text{Samp}(\kappa)$ always have the same size $s(\kappa)$, same input length $n(\kappa)$, and are $\tau(n, s, \kappa)$-decomposing equivalent with probability 1. Then C_0 and C_1 are also $(n+1, s, 2(\tau-1))$-path decomposing equivalent by Lemma 5. By the definition of path decomposing equivalence and Lemma 8 (which states that the minimum tree covering is efficiently computable), there exist $L'_1, L'_2, \cdots, L'_{2(\tau-2)}, L'_{2(\tau-1)-1}$ and

$L'_0 = \{(\varepsilon, C_0)\}, L'_{2(\tau-1)} = \{(\varepsilon, C_1)\}$ such that any two adjacent circuit assignments are $(n+1, s)$-locally decomposing equivalent. So we have that

$$\begin{aligned}
&|\Pr[D(\mathsf{dO.ParaGen}(1^\kappa, C_0))] - \Pr[D(\mathsf{dO.ParaGen}(1^\kappa, C_1))]| \\
&\leq \sum_{i=1}^{2(\tau-1)} \left| \begin{array}{l} \Pr[D(\mathsf{ldO.ParaGen}(1^\kappa, L'_{i-1}), n+1, |C_0|)] \\ \qquad - \Pr[D(\mathsf{ldO.ParaGen}(1^\kappa, L'_i), n+1, |C_0|)] \end{array} \right| \\
&\leq 2(\tau-1) \cdot \epsilon(\kappa)
\end{aligned}$$

Here, ϵ is the advantage of the following adversary pair (Samp', D) in the local dO security game (where D is from above). Samp' runs $(C_0, C_1, \sigma) \leftarrow \mathrm{Samp}'$, computes the path $L'_0, \cdots, L'_{2(\tau-1)}$, chooses a random $i \in [2(\tau-1)]$, and outputs (L'_{i-1}, L'_i, σ).

Therefore, as desired, we get an adversary for the local dO where the loss is $2(\tau-1)$. If we assume the polynomial hardness of ldO, the adversary (Samp', D) must have negligible advantage ϵ, and so we get $\tau - \mathsf{dO}$ for any polynomial τ. If we assume the subexponential hardness of ldO, we can set κ so that $\epsilon = 2^{-n}\mathsf{negl}(\kappa)$ for some negligible function negl. In this case, we even get 2^n-dO, which is equivalent to iO. In the regime of subexponential hardness, we can even set $\epsilon = 2^{-n}\mathsf{subexp}(\kappa)$ for some subexponential function subexp, in which case we get subexponentially secure 2^n-dO and hence subexponentially secure iO. □

Next, we focus on constructing ldO, which we now know is sufficient for constructing dO.

4.5 Compact FE Implies dO

Theorem 3. *If compact single-key selective secure functional encryption schemes exist, then there exists local decomposable obfuscators* ldO.

With Theorem 3 and Lemma 6, we have the following Theorem 4.

Theorem 4. *If compact single-key selective secure functional encryption schemes exist, then there exist decomposable obfuscators* dO.

Now we prove Theorem 3.

Proof. Let us first give the construction of our $\mathsf{ldO.ParaGen}$ (see Algorithm 3) where FE is a compact functional encryption scheme, SKE is a symmetric key encryption scheme and PRG is a pseudo random generator.

For each function $f_i^{b, Z_i^b} (1 \leq i \leq n)$, it basically computes a partial evaluation of an input circuit and encrypts it under two different functional encryption schemes (See Algorithm 5). But instead of doing this, this function also allows us to cheat and output a result given a secret key.

For each function f_{n+1}^b, it is given a circuit with no input, and simply evaluates it (see Algorithm 6).

Algorithm 3. Locally decomposable obfuscator $\mathsf{ldO.ParaGen}$

procedure $\mathsf{ldO.ParaGen}(1^\kappa, L = \{(x_i, C_{x_i})\}, \ell, s)$
  for $i = 1, 2, \cdots, n, n+1$ **do**
    $(\mathsf{mpk}_i^b, \mathsf{msk}_i^b) \leftarrow \mathsf{FE.Gen}(1^\kappa)$ for $b \in \{0,1\}$
  end for
  prepare a list of secret keys $\mathsf{sk}_{i,j}^b \leftarrow \mathsf{SKE.KeyGen}(1^\kappa)$ for $1 \le i \le n, 1 \le j \le \ell$ and $b \in \{0,1\}$
  prepare $Z_i^b = Z_{i,1}^b, Z_{i,2}^b, \cdots, Z_{i,\ell}^b$ for $1 \le i \le n$ and $b \in \{0,1\}$ where $Z_{i,j}^b = \mathsf{SKE.Enc}(\mathsf{sk}_{i,j}^b, 0^{t_1})$ and t_1 is a length bound specified later;
  generate c_0, c_1 by calling a recursive algorithm $\mathsf{CGen}(\varepsilon, L)$
  for $i = 1, 2, \cdots, n$ **do**
    $\mathsf{fsk}_i^b \leftarrow \mathsf{FE.KeyGen}(\mathsf{msk}_i^b, f_i^{b, Z_i^b})$ for $b \in \{0,1\}$
  end for
  $\mathsf{fsk}_{n+1}^b \leftarrow \mathsf{FE.KeyGen}(\mathsf{msk}_{n+1}^b, f_{n+1}^b)$ for $b \in \{0,1\}$
  return the parameters $\{c_0, c_1, \{\mathsf{mpk}_i^0, \mathsf{mpk}_i^1\}_{i=1}^{n+1}, \{\mathsf{fsk}_i^0, \mathsf{fsk}_i^1\}_{i=1}^{n+1}\}$
end procedure

Algorithm 4. Generating c_0, c_1 recursively

procedure $\mathsf{CGen}(x, L)$
  if L only contains one pair, it must be (x, C_x) **then**
    Generate $K^b \leftarrow \{0,1\}^\kappa$ for $b \in \{0,1\}$
    $c_b \leftarrow \mathsf{FE.Enc}(\mathsf{mpk}_d^b, \langle C_x, K^b, 0, 0^{t_2}\rangle)$ for $b \in \{0,1\}$, and $d = |x| + 1$
    return c_0, c_1
  end if
  Split L into L_0, L_1 where L_0 contains all the pairs (y, C_y) where y starts with $x||0$ and L_1 contains all the pairs (y, C_y) where y starts with $x||1$
  $(c_0', c_1') \leftarrow \mathsf{CGen}(x||0, L_0)$ and $(c_0'', c_1'') \leftarrow \mathsf{CGen}(x||1, L_1)$
  Choose an integer j_0 randomly from 1 to ℓ that has not been used yet in Z_d^0 and replace Z_{d,j_0}^0 with $\mathsf{SKE.Enc}(\mathsf{sk}_{d,j_0}^0, \langle c_0', c_1'\rangle)$
  Choose j_1 in the same way and replace Z_{d,j_1}^1 with $\mathsf{SKE.Enc}(\mathsf{sk}_{d,j_1}^1, \langle c_0'', c_1''\rangle)$
  return c_0, c_1 where $c_0 = \mathsf{FE.Enc}(\mathsf{mpk}_d^0, \langle \perp, \perp, j_0, \mathsf{sk}_{d,j_0}^0\rangle)$ and $c_1 = \mathsf{FE.Enc}(\mathsf{mpk}_d^1, \langle \perp, \perp, j_1, \mathsf{sk}_{d,j_1}^1\rangle)$
end procedure

Evaluation and Correctness. Now let us look at how $\mathsf{ldO.Eval}$ works. By fixing the first two ciphers and keys, given a input $x \in \{0,1\}^n$,

- It begins with c_0, c_1;
- For $i = 1, 2, \cdots, n$, it picks the function key $\mathsf{fsk}_i^{x_i}$ and c_{x_i}; then does the update: $(c_0, c_1) \leftarrow \mathsf{FE.Dec}(\mathsf{fsk}_i^{x_i}, c_{x_i})$;
- Finally we can either output $\mathsf{FE.Dec}(\mathsf{fsk}_{n+1}^0, c_0)$ or $\mathsf{FE.Dec}(\mathsf{fsk}_{n+1}^1, c_1)$;

$\mathsf{ldO.Eval}(c_0, c_1, \{\mathsf{mpk}_i^0, \mathsf{mpk}_i^1\}_{i=1}^{n+1}, \{\mathsf{fsk}_i^0, \mathsf{fsk}_i^1\}_{i=1}^{n+1}, \cdots)$ actually has the same functionalities with the circuit assignment L since basically on input x, it finds a fragment corresponding to a prefix y of $x = y||x'$ and keeps doing partial evaluations on each input bit of x'. Since the cardinality is at most ℓ, ℓ different $Z_{i,j}^b$ in Z_i^b are enough for use.

Algorithm 5. f_i^{b,Z_i^b} for $1 \leq i \leq n$

procedure $f_i^{b,Z_i^b}(C, K, \sigma, \mathsf{sk})$
  Hardcoded : Z_i^b
  if $\sigma \neq 0$ **then**
    return $\mathsf{SKE.Dec}(\mathsf{sk}, Z_{i,\sigma}^b)$
  else
    $C' \leftarrow C(b, \cdot)$ and pad C' to have length s
    return $\{\mathsf{FE.Enc}(\mathsf{mpk}_{i+1}^0, \langle C', K_{i+1}^0, 0, 0^{t_2}\rangle; r_1),$
      $\mathsf{FE.Enc}(\mathsf{mpk}_{i+1}^1 \langle C', K_{i+1}^1, 0, 0^{t_2}\rangle; r_2)\}$ where
      $K_{i+1}^0 \leftarrow r_3$
      $K_{i+1}^1 \leftarrow r_4$
      using randomness $r_1, r_2, r_3, r_4 \leftarrow \mathsf{PRG}(K)$
  end if
end procedure

Algorithm 6. f_{n+1}^b

procedure $f_{n+1}^b(C, K, \sigma, \mathsf{sk})$
  return the evaluation of C on an empty input
end procedure

Efficiency. Let us look at the parameter size. All the master keys $\{\mathsf{mpk}_i^0, \mathsf{mpk}_i^1\}_{i=1}^{n+1}$ are of length $\mathsf{poly}(\kappa)$. t_2 is the length of a secret key for SKE scheme so it is also of $\mathsf{poly}(\kappa)$. And we assume FE is a compact functional encryption scheme which means the size of ciphers c_0, c_1 is bounded by $O(\mathsf{poly}(s, \log \ell, \kappa))$ and also the size of f circuit is bounded by $O(\mathsf{poly}(s, \ell, \kappa))$ which implies the size $\{\mathsf{fsk}_i^b\}$ is bounded by $O(\mathsf{poly}(s, \ell, \kappa))$. Finally t_1 is bounded by $O(\mathsf{poly}(s, \log \ell, \kappa))$.

So $\mathsf{IdO.ParaGen}$ and $\mathsf{IdO.Eval}$ run in time $\mathsf{poly}(s, \ell, n, \kappa)$.

Security. Without loss of generality, we have two circuit assignments L_0 and L_1 where $\mathsf{Decompose}(L_0, x) = L_1$. We are going to prove the indistinguishability when we are given either L_0 or L_1.

- **Hyb 0:** Here, an adversary is given an instance $\mathsf{IdO.ParaGen}(1^\kappa, L_0, \ell, s)$. In the process of generating c_0, c_1, we will get to CGen on x and L' where L' is the current partial circuit assignment. Since L' only contains (x, C_x), CGen will return $\mathsf{FE.Enc}(\mathsf{mpk}_d^b, \langle C_x, K^b, 0, 0^{t_2}\rangle)$ for $b \in \{0,1\}$ and $d = |x| + 1$; we denote them as $\hat{c}_0, \hat{c}_1$.
- **Hyb 1:** In this hybrid, we change Z_d^b. Assume $\hat{c}_{b,0}, \hat{c}_{b,1} = \mathsf{FE.Dec}(\mathsf{fsk}_d^b, \hat{c}_b)$. In $\mathsf{IdO.ParaGen}$, Z_d^b are assigned to an array of encryptions of 0^{t_1} before calling CGen. We instead choose random j_0, j_1 from the unused indices (not used in CGen process) and change Z_{d,j_0}^0 and Z_{d,j_1}^1 to encryptions of $\langle \hat{c}_{b,0}, \hat{c}_{b,1}\rangle$. Since an adversary does not have any secret key $\mathsf{sk}_{i,j}^b$, SKE security means **Hyb 0** and **Hyb 1** are indistinguishable.

– **Hyb 2:** In this hybrid, we change the ciphertexts $\hat{c}_0, \hat{c}_1$ to

$$\hat{c}_b = \mathsf{FE.Enc}(\mathsf{mpk}_d^b, \langle \bot, \bot, j_b, \mathsf{sk}_{d,j_b}^b \rangle)$$

where $\bot$ means filling it with zeroes and j_b are the indices chosen in **Hyb 1**. Notice that

$$f_d^{b,Z_d^b}(\bot, \bot, j_b, \mathsf{sk}_{d,j_b}^b) = f_d^{b,Z_d^b}(C_x, K^b, 0, 0^{t_2})$$

Therefore, FE security means **Hyb 1** and **Hyb 2** are indistinguishable.

– **Hyb 3:** In this hybrid, we change Z_{d,j_0}^0 and Z_{d,j_1}^1. In **Hyb 1**, $\hat{c}_{b,0}, \hat{c}_{b,1}$ were computed using the randomness from a pseudo random generator. In **Hyb 2**, we removed the seed feed to PRG. Therefore we can replace $\hat{c}_{b,0}, \hat{c}_{b,1}$ to be the values computed using uniformly chosen randomness. Indistinguishability from **Hyb 2** easily follows from PRG security. We observe that the distribution of the instances in **Hyb 3** is identical to the distribution of $\mathsf{IdO.ParaGen}(1^\kappa, L_1, \ell, s)$.

This completes our proof for Theorem 3.

5 Discussion

5.1 Deciding Decomposing Equivalence

Definition 13. *A tree covering TC is a* witness that $C_0 \equiv C_1$ *if TC satisfies* $\mathsf{DecomposeTo}(\{(\varepsilon, C_0)\}, \mathcal{X}) = \mathsf{DecomposeTo}(\{(\varepsilon, C_1)\}, \mathcal{X})$. *In other words, decomposing C_0 and C_1 to TC give the same circuit assignment (as in, the circuit fragments themselves are identical).*

TC is an minimal *witness if, for all other TC' that are witnesses to $C_0 \equiv C_1$, we have that $TC \preceq TC'$. In particular, this means that TC is strictly smaller than all other witnesses.*

We define a node x as "good" for C_0, C_1 if $C_0(x, \cdot) = C_1(x, \cdot)$ as circuits. Notice that the children of a good node are also good. We say that a good node x is "minimal" if its parent is not good.

Lemma 7. *For any two equivalent circuits C_0, C_1, there is always exactly one minimal witness TC^*, and it consists of all of the minimal good nodes for C_0, C_1.*

Proof. Since $C_0 \equiv C_1$, all the leaves are good, and at least the set of leaves form a tree covering that is a witness. Now, for each leaf, consider the path from the leaf to the root. There will be some node x on the path such that all nodes in the path before x are not good, but x and all nodes after x are good. Therefore, that x is an minimal good node. Moreover, no minimal good node can be a descendant of any other minimal good node (since no minimal good node can be the descendant of *any* good node). Therefore, the set of minimal good nodes form a tree covering.

Lemma 8. *τ-one shot decomposing equivalence can be decided deterministically in time $\tau \times \mathsf{poly}(n, \max\{|C_0|, |C_1|\})$. Moreover, if $C_0 \equiv C_1$, then the optimal witness TC^* can also be computed in this time.*

Proof. The algorithm is simple: process the nodes in a depth-first manner, keeping a global list R. When processing a node x, if $C_0(x, \cdot) = C_1(x, \cdot)$ as circuits, add x to R, and then do not recurse. Otherwise, recurse on the children as normal. If the list R every grows to exceed τ elements, abort the search and report non-decomposing equivalence. If the search finishes with $|R| \leq \tau$, then report decomposing equivalence and output R.

The total running time is bounded by $O(n\tau \cdot \mathsf{poly}(\max\{|C_0|, |C_1|\}))$: at most $n\tau$ nodes are processed (the up to τ nodes in R, plus their ancestors), and processing each node takes time proportional to the sizes of C_0, C_1.

5.2 One Shot DE Is Equivalent to Path DE

We have already proved that path DE implies one shot DE. Now let us prove the converse.

Lemma 9. *If two circuits C_0, C_1 are (ℓ, s, t)-path decomposing equivalent, then they are $(t/2 + 1)$-one shot decomposing equivalent*

Proof. If C_0, C_1 are (ℓ, s, t)-path decomposing equivalent, there exists a minimal tree covering TC^*. We observe that, for each of the ancestors of nodes in TC^*, there must be a step in the path where that node is decomposed, and there must also be a step in the path where that node is merged. It is straightforward to show that the number of ancestors for any tree covering is exactly one less than the size of the covering. From this, we deduce that $|TC^*| \leq t/2 + 1$. Since TC^* exists and the size is bounded by $t/2 + 1$, these two circuits are $(t/2 + 1)$-one shot decomposing equivalent.

We emphasize that the above lemma and proof were independent of the bounds ℓ and s. Putting together Lemmas 5 and 9, we find that the path equivalence definition is independent of the parameters ℓ, s.

We also see that path decomposing equivalence can be computed efficiently, following Lemmas 5, 8, and 9.

5.3 One Shot DE Is Strictly Stronger Than Functional Equivalence

We then show that path/one-shot decomposing equivalence is a strictly stronger notion than standard functional equivalence, when a reasonable bound is placed on the path length/witness size. The rough idea is the use the fact that, say, polynomial decomposing equivalence can be decided in polynomial time, whereas in general deciding equivalence is hard.

Lemma 10. *For any n, there exist two circuits on n bit inputs $C_0 \equiv C_1$ that are not $2^{n-1} - 1$-one-shot decomposing equivalent.*

Proof. Let D_0, D_1 be two equivalent but non-identical circuits on 2 input bits (for example, two different circuits computing the XOR). Let TC^* be the tree covering consisting of all 2^{n-1} nodes in the layer just above the leaves. Let L_b for $b = 0, 1$ be the circuit assignment assigning D_b to every node in TC^*. Finally, Let C_b be the result of canonically merging L_b all the way to the root node.

Now, TC^* is clearly the optimal witness that $C_0 \equiv C_1$. Therefore, any witness must have size at least $|TC^*| = 2^{n-1}$. Therefore, C_0, C_1 are not $2^{n-1}-1$ one-shot decomposing equivalent.

Note that the above separation constructed exponentially-large C_0, C_1. We can even show a similar separation in the case where C_0, C_1 have polynomial size, assuming $P \neq NP$. Indeed, since poly-one shot decomposing equivalence is decidable in polynomial time, but functional equivalence is not (assuming $P \neq NP$), there must be circuits pairs that are equivalent but not poly-one shot decomposing equivalent.

Next, we even demonstrate an explicit ensemble of circuit pairs that are equivalent but not poly-decomposing equivalent, assuming one-way functions exist.

Lemma 11. *Assuming one-way functions exist, there is an explicit family of circuit pairs* (C_0, C_1) *that are equivalent, but are not* $\mathsf{poly}(n)$*-decomposing equivalent for any polynomial* $\mathsf{poly}(n)$*.*

Proof. Let PRG be a length-doubling pseudorandom generator (which can be constructed from any one-way function). Let $C_0(x) =$ "**return** 0" and $C_1(x) =$ '**return** 1 if $\mathsf{PRG}(x) = v$; 0 otherwise'' where v is uniformly chosen from $\{0,1\}^{2\kappa}$. When v is uniformly chosen, except with probability $\frac{1}{2^\kappa}$, v has no pre-image under PRG. Therefore, with probability $1 - \frac{1}{2^\kappa}$, C_0 and C_1 are functionally equivalent.

Next, assume there exists a polynomial τ and a non-negligible probability δ such that C_0 and C_1 are τ-decomposing equivalent with probability δ. Now let us build an adversary $\mathcal{B}$ for this length-doubling PRG:

- The adversary $\mathcal{B}$ gets u from the challenger;
- $\mathcal{B}$ prepares the following two circuits: $C_0(x) =$"**return** 0" and $C_1(x) =$ "**return** 1 if $\mathsf{PRG}(x) = u$; 0 otherwise".
- $\mathcal{B}$ runs the algorithm to see if they are τ-decomposing equivalent. If the algorithm returns **true**, $\mathcal{B}$ guesses u is a truly random string; otherwise it guesses u is generated by PRG.

When u is generated by PRG, it will always return the correct answer since C_1 does not return 0 at some point but C_0 does; when u is truly random, the probability that $\mathcal{B}$ is correct equal to the probability C_0 and C_1 are τ-decomposing equivalent which is a non-negligible δ. So $\mathcal{B}$ has non-negligible advantage δ in breaking PRG.

References

[AJ15] Ananth, P., Jain, A.: Indistinguishability obfuscation from compact functional encryption. In: Gennaro, R., Robshaw, M. (eds.) CRYPTO 2015. LNCS, vol. 9215, pp. 308–326. Springer, Heidelberg (2015). doi:10.1007/978-3-662-47989-6_15

[BP15] Bitansky, N., Paneth, O.: ZAPs and non-interactive witness indistinguishability from indistinguishability obfuscation. In: Dodis, Y., Nielsen, J.B. (eds.) TCC 2015. LNCS, vol. 9015, pp. 401–427. Springer, Heidelberg (2015). doi:10.1007/978-3-662-46497-7_16

[BPR15] Bitansky, N., Paneth, O., Rosen, A.: On the cryptographic hardness of finding a nash equilibrium. In: 2015 IEEE 56th Annual Symposium on Foundations of Computer Science (FOCS), pp. 1480–1498. IEEE (2015)

[BPW16] Bitansky, N., Paneth, O., Wichs, D.: Perfect structure on the edge of chaos. In: Kushilevitz, E., Malkin, T. (eds.) TCC 2016. LNCS, vol. 9562, pp. 474–502. Springer, Heidelberg (2016). doi:10.1007/978-3-662-49096-9_20

[BST14] Bellare, M., Stepanovs, I., Tessaro, S.: Poly-many hardcore bits for any one-way function and a framework for differing-inputs obfuscation. In: Sarkar, P., Iwata, T. (eds.) ASIACRYPT 2014. LNCS, vol. 8874, pp. 102–121. Springer, Heidelberg (2014). doi:10.1007/978-3-662-45608-8_6

[BV15] Bitansky, N., Vaikuntanathan, V.: Indistinguishability obfuscation from functional encryption. In: 2015 IEEE 56th Annual Symposium on Foundations of Computer Science (FOCS), pp. 171–190. IEEE (2015)

[BZ14] Boneh, D., Zhandry, M.: Multiparty key exchange, efficient traitor tracing, and more from indistinguishability obfuscation. In: Garay, J.A., Gennaro, R. (eds.) CRYPTO 2014. LNCS, vol. 8616, pp. 480–499. Springer, Heidelberg (2014). doi:10.1007/978-3-662-44371-2_27

[BZ16] Bun, M., Zhandry, M.: Order-revealing encryption and the hardness of private learning. In: Kushilevitz, E., Malkin, T. (eds.) TCC 2016. LNCS, vol. 9562, pp. 176–206. Springer, Heidelberg (2016). doi:10.1007/978-3-662-49096-9_8

[CLTV15] Canetti, R., Lin, H., Tessaro, S., Vaikuntanathan, V.: Obfuscation of probabilistic circuits and applications. In: Dodis, Y., Nielsen, J.B. (eds.) TCC 2015. LNCS, vol. 9015, pp. 468–497. Springer, Heidelberg (2015). doi:10.1007/978-3-662-46497-7_19

[GGH+13] Garg, S., Gentry, C., Halevi, S., Raykova, M., Sahai, A., Waters, B.: Candidate indistinguishability obfuscation and functional encryption for all circuits. In: Proceedings of the 2013 IEEE 54th Annual Symposium on Foundations of Computer Science, FOCS 2013, Washington, DC, USA, pp. 40–49. IEEE Computer Society (2013)

[GGHZ16] Garg, S., Gentry, C., Halevi, S., Zhandry, M.: Functional encryption without obfuscation. In: Kushilevitz, E., Malkin, T. (eds.) TCC 2016. LNCS, vol. 9563, pp. 480–511. Springer, Heidelberg (2016). doi:10.1007/978-3-662-49099-0_18

[GGM86] Goldreich, O., Goldwasser, S., Micali, S.: How to construct random functions. J. ACM (JACM) **33**(4), 792–807 (1986)

[GGSW13] Garg, S., Gentry, C., Sahai, A., Waters, B.: Witness encryption and its applications. In: Proceedings of the Forty-Fifth Annual ACM Symposium on Theory of Computing, STOC 2013, pp. 467–476. ACM, New York (2013)

[GLSW15] Gentry, C., Lewko, A.B., Sahai, A., Waters, B.: Indistinguishability obfuscation from the multilinear subgroup elimination assumption. In: Proceedings of the 2015 IEEE 56th Annual Symposium on Foundations of Computer Science (FOCS), FOCS 2015, Washington, DC, USA, pp. 151–170. IEEE Computer Society (2015)

[GPS16] Garg, S., Pandey, O., Srinivasan, A.: Revisiting the cryptographic hardness of finding a nash equilibrium. In: Robshaw, M., Katz, J. (eds.) CRYPTO 2016. LNCS, vol. 9815, pp. 579–604. Springer, Heidelberg (2016). doi:10.1007/978-3-662-53008-5_20

[GPSZ16] Garg, S., Pandey, O., Srinivasan, A., Zhandry, M.: Breaking the subexponential barrier in obfustopia. Technical report, Cryptology ePrint Archive, Report 2016/102 (2016). http://eprint.iacr.org/2016/102

[GS16] Garg, S., Srinivasan, A.: Single-key to multi-key functional encryption with polynomial loss. In: Hirt, M., Smith, A. (eds.) TCC 2016. LNCS, vol. 9986, pp. 419–442. Springer, Heidelberg (2016). doi:10.1007/978-3-662-53644-5_16

[GT16] Goldwasser, S., Tauman Kalai, Y.: Cryptographic assumptions: a position paper. In: Kushilevitz, E., Malkin, T. (eds.) TCC 2016. LNCS, vol. 9562, pp. 505–522. Springer, Heidelberg (2016). doi:10.1007/978-3-662-49096-9_21

[HJK+16] Hofheinz, D., Jager, T., Khurana, D., Sahai, A., Waters, B., Zhandry, M.: How to generate and use universal samplers. In: Cheon, J.H., Takagi, T. (eds.) ASIACRYPT 2016. LNCS, vol. 10032, pp. 715–744. Springer, Heidelberg (2016). doi:10.1007/978-3-662-53890-6_24

[HW15] Hubacek, P., Wichs, D.: On the communication complexity of secure function evaluation with long output. In: Proceedings of the 2015 Conference on Innovations in Theoretical Computer Science, ITCS 2015, pp. 163–172. ACM, New York (2015)

[KMN+14] Komargodski, I., Moran, T., Naor, M., Pass, R., Rosen, A., Yogev, E.: One-way functions and (im)perfect obfuscation. In: 2014 IEEE 55th Annual Symposium on Foundations of Computer Science (FOCS), pp. 374–383, October 2014

[LZ17] Liu, Q., Zhandry, M.: Exploding obfuscation: a framework for building applications of obfuscation from polynomial hardness. Cryptology ePrint Archive, Report 2017/209 (2017). http://eprint.iacr.org/2017/209

[Nao03] Naor, M.: On cryptographic assumptions and challenges. In: Boneh, D. (ed.) CRYPTO 2003. LNCS, vol. 2729, pp. 96–109. Springer, Heidelberg (2003). doi:10.1007/978-3-540-45146-4_6

[SW14] Sahai, A., Waters, B.: How to use indistinguishability obfuscation: deniable encryption, and more. In: Proceedings of the 46th Annual ACM Symposium on Theory of Computing, pp. 475–484. ACM (2014)