Non-Malleable Obfuscation

Ran Canetti[*] Mayank Varia[†]

Abstract

Existing definitions of program obfuscation do not rule out malleability attacks, where an adversary that sees an obfuscated program is able to generate another (potentially obfuscated) program that is related to the original one in some way.

We formulate two natural flavors of non-malleability requirements for program obfuscation, and show that they are incomparable in general. We also construct non-malleable obfuscators of both flavors for some program families of interest. Some of our constructions are in the Random Oracle model, whereas another one is in the common reference string model. We also define the notion of verifiable obfuscation which is of independent interest.

[*]School of Computer Science, Tel Aviv University (`canetti@cs.tau.ac.il`)
[†]Massachusetts Institute of Technology (`varia@csail.mit.edu`)

Contents

1 Introduction

The problem of program obfuscation has recently received a lot of attention in cryptography. Informally, the goal of obfuscation is to transform a program in such a way that its code becomes unintelligible while its functionality remains the same. This intuitive idea was formalized in [1] using a simulation-based definition.

In [1] it is shown that there do not exist generic algorithms that obfuscate any program family. These results are extended in [7, 8]. However, positive results have been shown for some program families of interest, such as the family of "point circuits," which accept a single input string (that is explicitly given in the circuit description) and reject all other inputs [2, 5, 12, 17]. These constructions can be generalized to form obfuscators for two more families: "multi-point circuits," which accept a constant number of input strings, and "point circuits with multibit output," which store a hidden string that is revealed only for a single input value [4]. Finally, a slightly different definition of obfuscation has been formulated [9, 10] in which it is possible to obfuscate the family of re-encryption programs [10].

However, the question of malleability attacks on obfuscations has not been addressed. In fact, many of the above constructions are malleable. We provide an overview of the definition of obfuscation and then describe its malleability concerns.

Virtual black-box obfuscation. At a high level, the concept of "obfuscating" a program is to produce a new program with the same functionality but with "garbled" code. Of course, it is impossible for the garbled code to hide all useful information, because at the very least one can run the program and observe its input-output behavior. In this way, access to the code of a program must be at least as useful as access to an oracle for the program. At a high level, obfuscation ensures the converse: that access to the code of an obfuscated program is *no more* useful than access to the oracle.

The formalization of this idea provided by [1], called the *virtual black-box* property, considers two different worlds. In the real world, an efficient adversary has access to the code of an obfuscated program, and attempts to learn a single-bit predicate about the underlying program. Now consider an imaginary world in which the code of an obfuscated program is not provided, but rather only oracle access to the program is provided. Obfuscation ensures that there exists an efficient algorithm known as a *simulator* that can learn the same predicate in the imaginary world that the adversary learns in the real world.

Malleability concerns. If an adversary has access to obfuscated code, obfuscation guarantees that she cannot "understand" the underlying program. However, suppose the adversary instead uses the obfuscated code to create a new program in such a way that the she controls the relationship between the input-output functionality of the two programs.

Intuitively, one might expect that virtual black-box obfuscation already prevents malleability attacks. The simulator only has oracle access to the obfuscated code, so any program that it makes can only depend on the input-output functionality of the obfuscated code at a polynomial number of locations. Therefore, obfuscation should guarantee that the adversary is also restricted to these trivial malleability attacks. However, the virtual black-box definition in [1] does not carry this guarantee. Upon close inspection, the problem is that the virtual black-box definition only considers adversaries and simulators that output a single bit, not adversaries and simulators that output programs.

A naïve solution to this problem is to extend the virtual black-box definition to hold even when the adversary and simulator output long strings. However, in this case obfuscation becomes unrealizable for any family of interest: given the adversary that outputs its input, a corresponding simulator has oracle access to a program and needs to write the code for such a program, which is usually impossible.

In this paper, we demonstrate two different ways to incorporate non-malleability guarantees into obfuscation. Both non-malleability definitions extend the virtual black-box definition by allowing the adversary and simulator to produce multiple bit strings, but only in a restricted manner. There are many subtleties involved in constructing a proper definition, such as deciding the appropriate restrictions to impose on the adversary and simulator, and creating relations to test the similarities between the adversary's input and output programs. We defer treatment of these important details to Sections 3.1 and 3.2. Here, we motivate and describe the two definitions at a high level.

Functional non-malleability. Imagine that Alice, Bob, and Charles are three graduate students in an office that receives a new computer. The department's network administrator wishes to configure the computer to allow the grad students root access to the computer. The administrator receives the students' desired passwords, and she needs to write a login program that accepts these three passwords and rejects all other inputs. The administrator knows that she has to be careful in designing the login program because the students will have root access to the computer so they can read the program code. As a result, she forms the login program using an obfuscator for the family of three-point circuits, which ensures that a dictionary attack is the best that the graduate students can do to learn their officemates' passwords.

However, obfuscation does not ally all of the administrator's fears, because the students will have root access to the computer so they can alter the login program as well. The administrator would like to prevent tampering of the login program, but the virtual black-box definition does not provide this guarantee. For instance, suppose Alice wants to remove access to the computer for exactly one of her officemates. There exist obfuscators of the three-point circuit [4] such that Alice can always succeed in this attack. (We describe one such obfuscator in the Constructions section below.)

Intuitively, the goal of obfuscation is to turn a program into a "black box," so the only predicates that Alice can learn from the program are those she could learn from a black box. We extend this intuition to cover modifications as well. We say that an obfuscator is *functionally non-malleable* if the only programs that Alice can create given obfuscated code are the programs she could create given black-box access to the obfuscated code.

This definition provides a guarantee on the possible attacks Alice can apply. For instance, if Alice only has black-box access to the login program, then there is no way for her to remove exactly one of her officemates with certainty. In this sense, functional non-malleability provides stronger security for Bob and Charles because it protects all aspects of their access to the computer, whereas the virtual black-box property only protects their passwords.

We wish to define functional non-malleability using a simulation-based definition: for every adversary that receives obfuscated code and uses it to create a new program, there exists a simulator that only has oracle access to the obfuscated code and produces a program that is functionally equivalent to the adversary's program. However, this definition is too strong: given the trivial adversary that outputs its input, the simulator gets oracle access to a program and needs to output the code of this program, which is usually impossible. But it is unfair to demand that the simulator do this much work. After all, the adversary's input is a program but the simulator's input is just an oracle. At the very least, the adversary can output a program that uses its input program in a black-box manner, and the simulator should have the same ability. We get around this by allowing the program that the simulator outputs to have oracle access to the obfuscated code.

Verifiable non-malleability. Functional non-malleability is a nice property for obfuscators, but there are some scenarios where even this does not suffice. For example, suppose that Alice wishes to play an April fools' prank on her officemates by altering the login program to accept their old passwords appended to the string "Alice is great." Alice only knows her own password, so she cannot run the obfuscator to produce this modified program. Nevertheless, she can write the following program: "on input a string s, check that s begins with 'Alice is great,' and if so, remove it from s and send the rest of the string to the administrator's login program." Functional non-malleability does not prevent this prank. In fact, it is impossible to prevent this prank because Alice only uses the obfuscated login program in a black-box manner.

Still, this attack is not "perfect": after Alice performs this attack, the new login program "looks" very different from a program that the network administrator would create. As a result, we may not be able to prevent Alice from performing her prank, but we may be able to detect Alice's modification afterward and restore the original program. Alice's job is now harder, since she has to modify obfuscated code in such a way that the change is undetectable. We say that an obfuscator is *verifiably non-malleable* if the only programs Alice can create that pass the verification procedure are the programs she could create given black-box access to the obfuscated code. This approach gives us hope to detect attacks that we cannot prevent, although it requires a stronger model in which a verification procedure routinely audits the program.

In the setting of our example, one simple way to achieve non-malleability is for the network administrator to digitally sign every program she makes, and for the verification procedure to check the validity of the signature attached to an obfuscated program before running it. By the existential unforgeability of the signature scheme, Alice cannot make any modifications, so the non-malleability goal is achieved.

However, this solution requires that the verification procedure can find and store the network administrator's verification key, which may not be practical. We want the non-malleability guarantee to be an intrinsic property of the obfuscation, without relying on an external public key infrastructure. As a result, in this paper we consider "public" verifiers that depend only on the obfuscation algorithm, and not on the party performing the obfuscation. Specifically, an algorithm V is a *verifier* for a given obfuscator if V accepts programs generated by the obfuscator. We stress that V does not receive any party-specific information (such as public keys), so it does not depend on the person that runs the obfuscator.

Verifiability has interesting applications in and of itself, as we describe in the Discussion section below, but in this paper we only use it to create a simulation-based definition of verifiable non-malleability. The definition guarantees that an adversary cannot maul obfuscated code into a new program that passes the verification test unless there exists a simulator that can perform the same attack given only oracle access to the obfuscated code. (Note that the adversary must create a new program and not simply output the obfuscated code it receives.) Because we hope to detect attacks that operate in a black-box manner (which we could not hope to prevent in the functional setting), we no longer give the simulator the extra help that we gave it in the definition of functional non-malleability. Instead, the simulator must output a fully-functional program that does not have an oracle.

Comparison. We show that both forms of non-malleability imply the virtual black-box property. Intuitively, this relationship holds because an adversary that outputs programs should easily be able to encode a single bit of information. As a result, all known impossibility results regarding the virtual black-box property continue to hold for both types of non-malleability [1, 7].

Additionally, we compare the two flavors of non-malleability. The goal of functional non-malleability is to *prevent* as many malleability attacks as possible, whereas the goal of verifiable non-malleability is to *detect* as many attacks as possible. Intuitively, these goals are incomparable: the verifiable definition is stronger because we can detect more attacks than we can prevent, but on the other hand it is weaker because the model requires its participants to understand and apply the verification algorithm. We justify this intuition by showing that in the random oracle model, the two definitions of non-malleability are indeed incomparable.

Constructions. In the random oracle model, we show that the obfuscator for point circuits in [12] satisfies both functional and verifiable non-malleability. Next, we study the family of multi-point circuits, which accept a constant number of inputs. One idea to obfuscate the program that accepts values $x_1, \ldots, x_m$ is as follows:

1. Use several instantiations of a single point circuit obfuscator in order to create obfuscated programs P_1, P_2, $\ldots$, P_m, where each P_i accepts only the value x_i

2. Create the program P that contains P_1 through P_m as subroutines, and on input x, iteratively feeds x into the P_i and accepts if any one of these programs accept.

This methodology is known as *concatenation*, and it is shown in [4] that concatenation preserves obfuscation. That is, given any obfuscator for the family of single-point circuits, concatenation produces an obfuscator for the family of multi-point circuits. However, concatenation does not preserve non-malleability. The program P stores the subroutines P_1 through P_m in a readily identifiable way, so an adversary can modify one accepted point by changing one of the subroutines. This is true even if the obfuscator for the family of single point circuits is non-malleable.

In the verifiable setting, we resolve the problem with concatenation by using a self-signing technique to ensure that the subroutines are not modified. The verification algorithm associated with this construction runs the verification algorithm for the signature scheme. This approach does not suffice in the functional setting, where the self-signing technique is useless because there is no guarantee that anybody checks the signature. Instead, we "glue" the acceptable points together in such a way that any attempt to change the obfuscated code destroys information on all of the points simultaneously.

We also give a construction that does not use random oracles. Instead, it uses the common reference string model, in which a sequence of bits is chosen uniformly at random and published in a public location that all participants can access. (Note that this is different from a public key infrastructure because the CRS is not tied to the specific identity of the party performing the obfuscation.) We construct a verifiably non-malleable obfuscation for the family of point circuits by providing any (potentially malleable) obfuscation along with a non-malleable NIZK proof of knowledge [14, 15] that the obfuscator knows the point that is accepted.

Informally, a non-malleable NIZK proof of knowledge considers an adversary that can request multiple proofs for statements of its choice and then produces a new proof. The non-malleability guarantee requires that the adversary knows a witness to its constructed proof, so it cannot simply modify the old proofs to prove a new statement.

Intuitively, our construction is verifiably non-malleable because an adversary can only make a program that passes the verification test if she knows its functionality, so she cannot produce a program that is related to a given obfuscated point circuit or else she would learn information about the obfuscated circuit, which is impossible by the virtual black-box property. However, the actual proof turns out to be delicate. Using proof techniques from [2], we achieve a somewhat weaker variant of verifiable non-malleability. Specifically, we show that for a large class of relations, no adversary can perform a modification that satisfies the relation with noticeable probability.

Discussion on verifiable obfuscation. The concept of verifiable obfuscation is useful even in situations where malleability is not a concern. For example, suppose you create a new computer program that solves an important problem. You wish to profit from your research by selling this program to others, but you also want to protect the algorithm that you discovered. Therefore, you sell an obfuscated version of the program to your customers. This protects your intellectual property, but another problem has presented itself. Your customers do not wish to install the obfuscated program on their computers because they no longer have any guarantees about what this program does. For all they know, the program could contain a virus, and because the program is obfuscated there is no hope for a virus checker to detect the presence of a virus. Hence, you need a verification algorithm that proves to your customers that you are selling them a program from the proper family.

Future work. First, the constructions in this paper use the random oracle model or common reference string model. It remains an open question to construct a non-malleable obfuscator (of either flavor) without trusted setup.

Second, we provide a verifiably non-malleable obfuscation of single point circuits in the common reference string model for a large class of relations. Unfortunately, extending this construction to the multi-point setting is insufficient, as it only succeeds for a small class of relations. It remains open to find a better construction in the multi-point setting.

Organization. In Section 2, we provide an overview of virtual black-box obfuscation [1, 7]. In Section 3, we provide rigorous definitions of the two notions of non-malleability in the standard model, and then extend the definitions to the random oracle and common reference string models. In Sections 4 and 5, we present non-malleable obfuscators of both flavors for the family of multi-point circuits.

2 Obfuscation

In [1], [7], and other works, an obfuscator is defined as a compiler that takes a circuit as input and returns another circuit. The output circuit should be equivalent in functionality to the input circuit, but the output circuit should be unintelligible in the sense that any information that can be obtained from the output circuit can also be obtained with oracle access to the circuit. In this paper we will be mainly interested in obfuscation with dependent auxiliary information, as defined in [7].

Definition 1 (Obfuscation with respect to dependent auxiliary information). Let $\mathcal{C} = \{\mathcal{C}_n\}$ be a family of polynomial-size circuits, where $\mathcal{C}_n$ denotes all circuits of input length n. A probabilistic polynomial time algorithm $\mathcal{O}$ is an *obfuscator* for the family $\mathcal{C}$ operating over randomness $\mathcal{R} = \{\mathcal{R}_n\}$ if the following three conditions are met.

- *Approximate functionality:* There exists a negligible function ε such that for every n, every circuit $C \in \mathcal{C}_n$, and for all $x \in \{0,1\}^n$,

$$\Pr\left[r \leftarrow \mathcal{R}_n, D \leftarrow \mathcal{O}(C, r) : C(x) = D(x)\right] > 1 - \varepsilon(n).$$

  If this probability is always 1, then we say that $\mathcal{O}$ has *exact functionality*.

- *Polynomial slowdown:* There exists a polynomial p such that for every n, circuit $C \in \mathcal{C}_n$, and randomness $r \in \mathcal{R}_n$, the description length $|\mathcal{O}(C, r)| \leq p(|C|)$.

- *Virtual black-box:* For every polynomial q and every PPT adversary A, there exists a PPT simulator S such that for all sufficiently large n, for all $C \in \mathcal{C}_n$, and for all auxiliary information $z \in \{0,1\}^*$,

$$\left|\Pr\left[A(\mathcal{O}(C), z) = 1\right] - \Pr\left[S^C(1^n, z) = 1\right]\right| < \frac{1}{q(n)},$$

  where the first probability is taken over the coin tosses of A and $\mathcal{O}$, and the second probability is taken over the coin tosses of S. Furthermore, we require that A and S operate in time polynomial in the length of their first input.

We define obfuscation without auxiliary information in the same manner, except that the auxiliary information is removed from the virtual black-box definition. Unless otherwise specified, in this paper we assume that obfuscations are secure with respect to dependent auxiliary information.

3 Definitions

In this section, we rigorously define one of the the two variants of non-malleable obfuscation.

3.1 Functionally non-malleable obfuscation

We obtain functionally non-malleable obfuscation by generalizing the virtual black-box definition to allow the adversary and simulator to output programs instead of bits. Intuitively, a functionally non-malleable obfuscation has the property that an adversary, given the obfuscated code to a program, can only make a related program if it could have already done so given only black-box access to the program.

This is problematic in general, because the simulator cannot emulate all programs that the adversary can produce. For example, consider the adversary that outputs its input. Then, the simulator has oracle access to a circuit and has to produce a program that is functionally equivalent to its oracle. This is impossible unless the circuit is learnable with oracle queries, in which case the entire concept of obfuscation is uninteresting. To make the definition meaningful, we allow the program that the simulator produces to make oracle queries to the original circuit as well.

To capture the effectiveness of an adversary's modification, we introduce a polynomial-time computable relation E that receives the adversary's input program and output program. The adversary succeeds in the modification if E accepts it. The definition of non-malleability ensures that for every relation E, the simulator can perform a successful modification with the approximately the same probability as the adversary.

One technical concern about the relation E is the manner in which it receives the adversary's input and output programs. The goal of functional non-malleability is to compare the functionality of these programs, and not their underlying code, so E should operate in the same manner when given functionally equivalent inputs. Our definition resolves this issue by giving the relation a "canonical" member of the family that is equivalent to the adversary's output program. (See the Discussion section below for more detail on this issue.)

Additionally, in many situations, the adversary knows some a-priori useful information on the obfuscated program, so we allow dependent auxiliary information in the definition of non-malleability. For instance, in the motivating example from the Introduction in which Alice wishes to modify a login program, she possesses the knowledge of her own password.

Throughout this work, the adversaries and simulators are assumed to be non-uniform.

Definition 2 (Functional equivalence). We say that two circuits C_1 and C_2 are *functionally equivalent*, and write $C_1 \equiv C_2$, if for all inputs x it holds that $C_1(x) = C_2(x)$.

Definition 3 (Functionally non-malleable obfuscation). Let $\mathcal{C}$ and $\mathcal{D}$ be families of circuits, and let $\mathcal{O}$ be a PPT algorithm. We say that $\mathcal{O}$ is an *obfuscator for $\mathcal{C}$ that is functionally non-malleable over $\mathcal{D}$* if the following three conditions hold:

- *Almost exact functionality:* There exists a negligible function ε such that for every n and every circuit $C \in \mathcal{C}_n$, $\Pr[r \leftarrow \mathcal{R}_n : \mathcal{O}(C, r) \equiv C] > 1 - \varepsilon(n)$.
- *Polynomial slowdown:* There exists a polynomial p such that for every n, circuit $C \in \mathcal{C}_n$, and randomness $r \in \mathcal{R}_n$, the description length $|\mathcal{O}(C, r)| \leq p(|C|)$.
- *Functional non-malleability:* for every polynomial ρ and PPT adversary A, there exists a PPT simulator S such that for all sufficiently large n, for all circuits $C \in \mathcal{C}_n$, for all polynomial time computable relations $E : \mathcal{C}_n \times \mathcal{D}_n \to \{0, 1\}$ (that may depend on the circuit C), and for all auxiliary information

$z \in \{0,1\}^*$,

$$\begin{aligned}|\Pr[P \leftarrow A(\mathcal{O}(C), z) : \exists D \in \mathcal{D}_n \text{ s.t. } D \equiv P \text{ and } E(C, D) = 1] \\ - \Pr\left[Q \leftarrow S^C(1^n, z) : \exists D \in \mathcal{D}_n \text{ s.t. } D \equiv Q^C \text{ and } E(C, D) = 1\right]| < \frac{1}{\rho(n)},\end{aligned}$$

where the probabilities are over the coin tosses of A, $\mathcal{O}$, and S. We require that A and S run in time polynomial in the length of their first inputs.

If $\mathcal{D} = \mathcal{C}$, we simply say that $\mathcal{O}$ is a *functionally non-malleable obfuscator for* $\mathcal{C}$.

3.1.1 Discussion.

We make several remarks about this definition.

Almost exact functionality. The functionality requirement here is stronger than the one used in Definition 1 above. Approximate functionality only guarantees that an obfuscated program $\mathcal{O}(C, r)$ is "close" in functionality to C. However, $\mathcal{O}(C, r)$ might never have the same functionality as C does (for any choice of r). By contrast, almost exact functionality requires that the two circuits have identical functionality for most choices of r.

We remark that most of the constructions in this paper satisfy exact functionality.

Bivariate relation. In this definition, the bivariate relation E is allowed to depend on the choice of circuit $C \in \mathcal{C}_n$. Thus, restricting attention to univariate relations $E(D)$ results in an equivalent definition. We use a bivariate relation only to emphasize the fact that E depends on both C and D.

Possible definitions for E. As mentioned above, an important feature of the definition is that E only depends on the functionality of the adversary's output P and simulator's output Q^C, and not on the code of these circuits. We found three possible ways to enforce this condition on E.

First, we can constrain E to receive only oracle access to the program P or Q^C. As a result, it follows immediately that E only depends on the functionality of these two programs, and not on their underlying code. Unfortunately, this definition is too weak, because there are many natural predicates that cannot be tested by relations of this type.

For instance, consider the family of point circuits, where I_w is the circuit that accepts only the string w. Suppose the adversary is given an obfuscation of the point circuit I_x and wishes to create a new point circuit I_y such that the first bit of x and y are equal. No polynomial-time relation E (even ones that know x, since E can depend on x) can test the adversary's probability of success given only oracle access to I_y. We believe that relations of this type are meaningful, and therefore we want a definition that can test for them.

Second, we can give the relation E full access to the code of P or Q^C, but restrict our attention to relations that have identical output when given two functionally equivalent programs. Specifically, we only consider relations E such that given any programs $C \in \mathcal{C}_n$, $D \in \mathcal{D}_n$, and P, P' such that $D \equiv P \equiv P'$, it follows that $E(C, P) = E(C, P')$. This definition does allow E access to the code of its input programs. The advantage of this definition is that E finally gets access to the code of the programs. The disadvantage is that the condition we impose on relations is very restrictive. As a result, it is still impossible to compute many relations, such as the one described in the previous paragraph.

Specifically, the virtual black-box definition guarantees that any program $E(I_x, \mathcal{O}(I_y))$ cannot compute whether x and y have the same first bit with probability greater than $\frac{1}{2}$. Thus, the condition that we impose on E is that it has the same probability of success even when it is given I_x and I_y as inputs, in which case E

has enough information to perform the computation with probability 1 but has to fail $\frac{1}{2}$ of the time anyway in order to be consistent with the condition.

Third, we can allow all polynomial-time relations E, but instead of providing the code of P or Q^C as input to E, we provide the code of a functionally equivalent member in $\mathcal{D}_n$. This is the option we use in Definition 3 above, because it clearly satisfies the requirement that E only depend on the functionality of the adversary and simulator's output, and it is a stronger definition that can test for many relations that the previous two definitions cannot. For these reasons, we choose to use relations of this type in the definition of functional non-malleability.

One technical point to keep in mind is that the relation E takes the description of circuits in $\mathcal{C}_n$ and $\mathcal{D}_n$ as input. As a result, this definition is dependent upon the representation of the circuits in these families, and not just the functionality of these circuits. Therefore, we should choose a representation of the circuit families that enables relations to extract important information easily from the description of a circuit.

As a result, we define the families of multi-point circuits as follows. Given distinct $w_1, w_2, \ldots, w_m \in \{0,1\}^n$, let $I_{\{w_1,\ldots,w_m\}}$ be the circuit that stores $w_1, \ldots, w_m$ in some canonical, explicit manner, and on input x returns 1 if and only if $x = w_i$ for some i. In particular, relations can extract the strings $w_1, \ldots, w_m$ from the description of $I_{\{w_1,\ldots,w_m\}}$ in polynomial time. Then, let $\mathcal{P}_n^m$ be the set of all m-point circuits on n bits, and let $\mathcal{P}^m = \{\mathcal{P}_n^m\}_{n\in\mathbb{N}}$ be the family of m-point circuits. In particular, $\mathcal{P}^1$ is the family of point circuits.

Output family $\mathcal{D}$. According to our definition, an adversary succeeds only if it outputs a circuit that is equivalent to a circuit in the family $\mathcal{D}$. The most natural family to choose is $\mathcal{D} = \mathcal{C}$, but we allow $\mathcal{D}$ to be different from $\mathcal{C}$ in order to consider a wider range of adversaries. For instance, maybe $\mathcal{C}$ is the family of point circuits, but we are concerned with adversaries that produce two-point circuits as output as well. The definition of functional non-malleability allows us to form a larger family $\mathcal{D}$ to acknowledge this.

Of course, there is no reason to stop there: we may also be concerned with an adversary that produces a three-point circuit, or a four-point circuit, or any function for that matter. In fact, the presence of the circuit family $\mathcal{D}$ in the definition seems restrictive. It would be nice if our definition simultaneously covered all possible outputs of the adversary, and not just those in a specific family. In other words, we would like to have an obfuscator that is functionally non-malleable when $\mathcal{D}$ is the family of all circuits.

Unfortunately, this is not possible for many circuit families of interest, such as the case where $\mathcal{C}$ is the family of point circuits. Intuitively, the family of all circuits is so big that it allows A to output the obfuscated code that it receives as input, which the simulator cannot do.

Theorem 4. *Let $\mathcal{D}_n$ is the set of all circuits with input length n and binary output, and let $\mathcal{D} = \{\mathcal{D}_n\}$. Let $\mathcal{C}$ be a circuit family with the following two properties:*

1. *$\mathcal{C}$ is not learnable, so for every simulator S, there exists a circuit $C \in \mathcal{C}_n$ such that the circuit S, with only oracle access to C, cannot output the description of a circuit that is equivalent in functionality to C except with negligible probability.*

2. *Given a circuit $C \in \mathcal{C}_n$ and a circuit P that is equivalent to a member of $\mathcal{C}_n$, one can test in polynomial time whether $C \equiv P$.*

Then, there are no obfuscators $\mathcal{O}$ for $\mathcal{C}$ that are functionally non-malleable over $\mathcal{D}$.

Proof. Suppose that there exists an obfuscator $\mathcal{O}$ for $\mathcal{C}$ that is functionally non-malleable over $\mathcal{D}$. Informally, we will derive a contradiction by creating an adversary that takes its input program $\mathcal{O}(C)$, views it as a string s, and encodes the string in a circuit $D \in \mathcal{D}_n$ in a readily identifiable way. By unlearnability, the simulator cannot construct the code of any program that is functionally equivalent to C, which establishes the separation.

One simple choice is to make the circuit $D = I_s$, that is, D is the point circuit that accepts only the string s. Unfortunately, this naive idea does not work due to a technical constraint, namely that D is supposed to be a circuit in $\mathcal{D}_n$, and hence D should have n bits of input. However, the circuit I_s requires $|s|$ bits of input, which is usually greater than n. Instead, we form a circuit $D \in \mathcal{D}_n$ whose truth table is determined by s in such a way that s can be recovered by performing a small number of executions of D.

Given a circuit $C \in \mathcal{C}_n$ and the empty auxiliary input $z = \emptyset$, let A be the adversary that takes its input program $P = \mathcal{O}(C)$ and views it as a binary string s. Then, A appends a termination character $\perp$ to s, so s is now a string with a ternary alphabet. Next, A applies a standard ternary-to-binary encoding of the string s into a new string s'. Now, A forms the circuit $D \in \mathcal{D}_n$ whose truth table equals the string s' concatenated with the all 0s string. Note that the truth table for D is 2^n bits long, whereas the length of the string s' is only polynomially large in n, so for sufficiently large n the entire string s' can be encoded into the truth table for D. Finally, A outputs the circuit D.

Let $E(C, D)$ be the relation that does the following:

1. It queries D on enough input values to recover the string s', stopping once it views the encoded version of the $\perp$ symbol.

2. It decodes s' into the string s and views it as the binary representation of a program. In this way, E has recovered the program P.

3. Return 1 if and only if P is equivalent to its first input C, which by assumption it can test in polynomial time.

Note that the adversary always encodes a circuit that is equivalent in functionality to C, so it passes the relation test with probability 1. On the other hand, the circuit family $\mathcal{C}$ is not learnable, so for every simulator S, there exists a circuit $C \in \mathcal{C}_n$ such that S passes the relation test with only negligible probability. Therefore, A does not have a satisfactory simulator, so the functional non-malleability property is not true. Therefore, $\mathcal{O}$ is not an obfuscator for $\mathcal{C}$ that is functionally non-malleable over $\mathcal{D}$, as desired. □

The key step in this proof is showing a bijection between integers s and members of the family of all circuits. Because the integer s is less than $t = \max_{C \in \mathcal{C}_n}\{2^{p(|C|)}\}$, the impossibility result extends to any family $\mathcal{D}$ such that every $\mathcal{D}_n$ has an efficient ordering of at least t circuits (where p is the polynomial from the polynomial slowdown property). If $\mathcal{C}$ is a family of polynomial-size circuits, then $t \ll 2^{2^n}$ so the impossibility result extends to families $\mathcal{D}$ that are much smaller than the family of all circuits.

Comparison to virtual black-box obfuscation. Now that we have introduced a new definition of obfuscation, it is natural to compare it to the old one. Specifically, the new definition introduces one new requirement, the functional non-malleability property. We show that this property implies the virtual black-box property (at least for reasonable choices of the circuit family $\mathcal{D}$), which justifies our terminology of using the word "obfuscation" in Definition 3.

This result is not clear a-priori since the notion of success for the adversary is more restricted in the functional non-malleability definition. The result is significant, however, as it immediately implies that all impossibility results pertaining to the virtual black-box property immediately carry over to the non-malleability setting [1, 7].

Theorem 5. *Let $\mathcal{C}$ and $\mathcal{D}$ be circuit families, and let $\mathcal{O}$ a PPT algorithm such that the following conditions hold:*

1. $\mathcal{O}$ is an obfuscator for $\mathcal{C}$ that is functionally non-malleable over $\mathcal{D}$ (as defined in Definition 3).

2. For sufficiently large n, at least one of the following is true:

- *$\mathcal{D}_n$ is non-trivial, in the sense that there exist two circuits $D_0, D_1 \in \mathcal{D}_n$ such that $D_0 \not\equiv D_1$.*
- *$\mathcal{C}_n$ is trivial, in the sense that all of the circuits in $\mathcal{C}_n$ have the same functionality.*

Then, $\mathcal{O}$ satisfies the virtual black-box requirement for $\mathcal{C}$. As a result, $\mathcal{O}$ is an obfuscator for $\mathcal{C}$ with respect to dependent auxiliary information (as defined in Definition 1).

The outline of the proof is rather simple. Given an adversary A with binary output, we construct a new adversary A' (that is not constrained to binary output) in the following manner. Let D_0 and D_1 be two distinct circuits in $\mathcal{D}_n$, and without loss of generality suppose that there is some input value x such that $D_0(x) = 0$ and $D_1(x) = 1$. Then, A' emulates an execution of A until it returns a bit b, upon which A' outputs the circuit D_b. In essence, the adversary A' uses its ability to output programs in order to communicate a single bit of information.

By functional non-malleability, there exists a simulator S' corresponding to A'. Now we reverse the above procedure and form a simulator S with binary output as follows: S emulates an execution of S' until it outputs a circuit C, and then S outputs the single bit $C(x)$. We claim that S is a simulator for A that satisfies the virtual black-box requirement. This seems plausible because S extracts the single bit of information that A' and S' were trying to communicate. Indeed, if S' was an "ideal" simulator that always outputs D_0 or D_1 with the same probabilities that A' does, then it is trivially easy to show that S satisfies the virtual black-box property.

However, even though A' always outputs one of D_0 or D_1, it is not true that the same must be true of S', so the above proof technique fails. In fact, it is not immediately clear that the output of S' must be equivalent to *any* member of $\mathcal{D}_n$. Luckily, this statement turns out to be true, and it suffices to prove the theorem.

Proof. Let A be an adversary with binary output and let ρ be a polynomial. We need to construct a simulator S for A that satisfies the virtual black-box property. We prove this statement in two cases.

First, consider all n such that $\mathcal{D}_n$ is non-trivial. In this case, there exist two circuits $D_0, D_1 \in \mathcal{D}_n$ that do not have the same functionality. That is, there exists at least one input value x such that $D_0(x) \neq D_1(x)$. Because the outputs are not equal, they differ in at least one bit position i, and without loss of generality we assume that $D_0(x)$ equals 0 in this bit position whereas $(D_1(x))_i = 1$.

We use A to form an adversary A' (that is not constrained to binary output) in the following manner: A' stores A, D_0, and D_1 in some readily identifiable way (by nonuniformity), and on input $\mathcal{O}(C)$ and auxiliary information z, the circuit A' operates as follows.

```
1: emulate an execution of $A(\mathcal{O}(C), z)$ until it returns a bit $b$
2: **return** the program $D_b$
```

The functional non-malleability property applied to the adversary A' and polynomial 2ρ guarantees the existence of a simulator S' such that for all sufficiently large n, for all $C \in \mathcal{C}_n$, for all auxiliary information $z \in \{0,1\}^*$, and for all PPT relations E,

$$\begin{aligned} |\Pr\big[P \leftarrow A'(\mathcal{O}(C), z) : \exists D \in \mathcal{D}_n \text{ s.t. } D \equiv P \text{ and } E(C, D) = 1\big] \\ - \Pr\big[Q \leftarrow (S')^C(1^n, z) : \exists D \in \mathcal{D}_n \text{ s.t. } D \equiv Q^C \text{ and } E(C, D) = 1\big]| < \frac{1}{2\rho(n)}. \quad (1) \end{aligned}$$

Finally, we construct a simulator S with binary output that nonuniformly stores x and i and operates as follows, when given auxiliary information z and access to an oracle C:

```
1: $S$ emulates an execution of $(S')^C(1^n, z)$ until it returns a string $s$
```

```
2: if $s$ is not the valid encoding of a circuit then
3:     return 0
4: end if
5: let $Q$ be the circuit that is represented by $s$
6: return the bit $(Q^C(x))_i$
```

We now show that this simulator S satisfies the virtual black-box property.

In order to analyze this claim, we must understand the behavior of S'. Even though A' always outputs either D_0 or D_1, the simulator S' does not have to do the same. In fact, it is not immediately clear how often S' outputs any member of $\mathcal{D}_n$. Luckily, S' almost always outputs a member of $\mathcal{D}_n$, which we prove by using the inequality (1) on two relations.

First, let $\tilde{E}$ be the relation that nonuniformly stores x and i and operates as follows: $\tilde{E}(C, D)$ returns 1 if and only if $D(x)_i = 1$. In particular, the output of $\tilde{E}$ is independent of the choice of C. Then, for all n, all $C \in \mathcal{C}_n$, all $z \in \{0,1\}^*$,

$$\begin{aligned}\Pr\left[A(\mathcal{O}(C), z) = 1\right] &= \Pr\left[A'(\mathcal{O}(C), z) \text{ outputs } D_1\right] \\ &= \Pr\left[P \leftarrow A'(\mathcal{O}(C), z) : \exists D \in \mathcal{D}_n \text{ s.t. } D \equiv P \text{ and } E(C, D) = 1\right].\end{aligned}$$

The first equality is an immediate consequence of the construction of A', and the second equality follows from the fact that A' always outputs either D_0 or D_1, and of the two only D_1 satisfies the relation. Using this equality, along with the precise behavior of the relation $\tilde{E}$, functional non-malleability states that for all sufficiently large n, for all $C \in \mathcal{C}_n$, and for all auxiliary information z,

$$\begin{aligned}&|\Pr\left[A(\mathcal{O}(C), z) = 1\right] \\ &\qquad - \Pr\left[Q \leftarrow (S')^C(1^n, z) : \exists D \in \mathcal{D}_n \text{ s.t. } D \equiv Q^C \text{ and } (D(x))_i = 1\right]| < \frac{1}{2\rho(n)}. \quad (2)\end{aligned}$$

Second, let $\hat{E}$ be the relation such that $\hat{E}(C, D) = 1$ if and only if $D(x)_i = 0$. Using the same analysis as above, it follows that

$$\left|\Pr\left[A(\mathcal{O}(C), z) = 0\right] - \Pr\left[Q \leftarrow (S')^C(1^n, z) : \exists D \in \mathcal{D}_n \text{ s.t. } D \equiv Q^C \text{ and } (D(x))_i = 0\right]\right| < \frac{1}{2\rho(n)}.$$

Because A is an adversary with binary output, $\Pr\left[A(\mathcal{O}(C), z) = 1\right] + \Pr\left[A(\mathcal{O}(C), z) = 0\right] = 1$. Applying the triangle inequality to the previous two inequalities, it follows that

$$\begin{aligned}&|\Pr\left[Q \leftarrow (S')^C(1^n, z) : \exists D \in \mathcal{D}_n \text{ s.t. } D \equiv Q^C \text{ and } (D(x))_i = 1\right] \\ &\qquad + \Pr\left[Q \leftarrow (S')^C(1^n, z) : \exists D \in \mathcal{D}_n \text{ s.t. } D \equiv Q^C \text{ and } (D(x))_i = 0\right] - 1| < \frac{1}{2\rho(n)}.\end{aligned}$$

Of course, for any circuit D and input x, it is always the case that either $(D(x))_i = 1$ or $(D(x))_i = 0$. As a result, the above inequality simplifies to

$$\Pr\left[Q \leftarrow (S')^C(1^n, z) : \exists D \in \mathcal{D}_n \text{ s.t. } D \equiv Q^C\right] > 1 - \frac{1}{2\rho(n)}.$$

Therefore, with very large probability, S' outputs a valid encoding of a circuit that is equivalent to a member of $\mathcal{D}_n$.

Finally, by the construction of S from S', it is easy to see that

$$\Pr\left[S^C(1^n, z) = 1\right] = \Pr\left[Q \leftarrow (S')^C(1^n, z) : (Q^C(x))_i = 1\right].$$

Furthermore, we showed that Q fails to be equivalent to a member of $\mathcal{D}_n$ with small probability, so

$$\left|\Pr\left[S^C(1^n, z) = 1\right] - \Pr\left[Q \leftarrow (S')^C(1^n, z) : \exists D \in \mathcal{D}_n \text{ s.t. } D \equiv Q^C \text{ and } (D(x))_i = 1\right]\right| < \frac{1}{2\rho(n)}.$$

Applying the triangle inequality to this inequality and (2) yields

$$\left|\Pr\left[A(\mathcal{O}(C), z) = 1\right] - \Pr\left[S^C(1^n, z) = 1\right]\right| < \frac{1}{\rho(n)},$$

so S satisfies the virtual black-box property as desired.

Now we prove the theorem for values of n such that $\mathcal{C}_n$ and $\mathcal{D}_n$ are both trivial. In this case, we claim that for every adversary A with binary output, for every $z \in \{0,1\}^*$, for sufficiently large n, and for every $C, C' \in \mathcal{C}_n$,

$$\left|\Pr\left[A(\mathcal{O}(C), z) = 1\right] - \Pr\left[A(\mathcal{O}(C'), z) = 1\right]\right|$$

is negligible. If this is true, then it is easy to construct a simulator S for A: the simulator simply chooses a circuit $C \in \mathcal{C}_n$ arbitrarily, obfuscates it, and emulates an execution of A on the obfuscated circuit. It is straightforward to check that this simulator satisfies the virtual black-box property.

Suppose for the sake of contradiction that the claim is not true. Therefore, there exist an adversary A, a polynomial ρ, and auxiliary information $z \in \{0,1\}^*$ such that for infinitely many n, there exist two circuits $C, C' \in \mathcal{C}_n$ such that

$$\left|\Pr\left[A(\mathcal{O}(C), z) = 1\right] - \Pr\left[A(\mathcal{O}(C'), z) = 1\right]\right| > \frac{1}{\rho(n)}.$$

Now form an adversary A' (that is not constrained to binary output) as follows. Arbitrarily choose a circuit $D_1 \in \mathcal{D}_n$, and let D_0 be a circuit that is not equivalent to D_1, and hence not equivalent to any member of $\mathcal{D}_n$. By nonuniformity, the adversary A' stores A, D_0, and D_1 in some readily identifiable way. The circuit A' operates as follows, upon receiving $\mathcal{O}(C)$ and z as input:

```
1: emulate an execution of $A(\mathcal{O}(C), z)$ until it returns a bit $b$
2: **return** $D_b$
```

Let $\bar{E}$ be the relation that always accepts. As a result,

$$\Pr\left[P \leftarrow A'(\mathcal{O}(C), z) : \exists D \in \mathcal{D}_n \text{ s.t. } D \equiv P \text{ and } \bar{E}(C, D) = 1\right] = \Pr\left[A(\mathcal{O}(C), z) = 1\right]$$

because D_1 is equivalent to a member of $\mathcal{D}_n$ but D_0 is not. Hence, by our assumption above, it follows that for infinitely many n, there exist two circuits $C, C' \in \mathcal{C}_n$ and a polynomial ρ such that

$$\begin{aligned} | \Pr&\left[P \leftarrow A'(\mathcal{O}(C), z) : \exists D \in \mathcal{D}_n \text{ s.t. } D \equiv P \text{ and } \bar{E}(C, D) = 1\right] \\ &- \Pr\left[P \leftarrow A'(\mathcal{O}(C'), z) : \exists D \in \mathcal{D}_n \text{ s.t. } D \equiv P \text{ and } \bar{E}(C', D) = 1\right] | > \frac{1}{\rho(n)}. \end{aligned}$$

However, for any simulator S',

$$\begin{aligned} \Pr&\left[Q \leftarrow (S')^C(1^n) : \exists D \in \mathcal{D}_n \text{ s.t. } D \equiv Q^C \text{ and } \bar{E}(C, D) = 1\right] \\ &= \Pr\left[Q \leftarrow (S')^{C'}(1^n) : \exists D \in \mathcal{D}_n \text{ s.t. } D \equiv Q^{C'} \text{ and } \bar{E}(C', D) = 1\right] \end{aligned}$$

information-theoretically because C and C' are functionally equivalent. Applying the triangle inequality to the previous two equations, it must be the case that either

$$\begin{aligned}|\Pr\left[P \leftarrow A'(\mathcal{O}(C), z) : \exists D \in \mathcal{D}_n \text{ s.t. } D \equiv P \text{ and } \bar{E}(C, D) = 1\right] \\ - \Pr\left[Q \leftarrow (S')^C(1^n) : \exists D \in \mathcal{D}_n \text{ s.t. } D \equiv Q^C \text{ and } \bar{E}(C, D) = 1\right]| > \frac{1}{2\rho(n)}\end{aligned}$$

or the same is true of C'. As a result, S' fails to satisfy the functional non-malleability requirement. This is true for all possible S', so $\mathcal{O}$ is not functionally non-malleable over $\mathcal{D}$, which is a contradiction. $\square$

It is simple to check that the proof goes through even if the auxiliary input z is constrained to be the null string. As a result, functional non-malleability without auxiliary information implies virtual black-box without auxiliary information. Similarly, the proof goes through if both the functional non-malleability and virtual black-box properties are modified such that the difference in probabilities of success for the adversary and simulator is required to be negligible.

3.2 Verifiably non-malleable obfuscation

In this section, we formalize another notion of non-malleability as motivated in the Introduction. We begin by developing the concept of verifiable obfuscation.

Definition 6 (Verifier). Given a pair of PPT algorithms $\mathcal{O}$ and V and a circuit family $\mathcal{C}$, we say that V is a *verifier for $\mathcal{O}$ applied to $\mathcal{C}$* if there exists a negligible function ε such that for all n and for all $C \in \mathcal{C}_n$, $\Pr[V(\mathcal{O}(C)) = 1] > 1 - \varepsilon(n)$, where the probability is taken over the randomness of V and $\mathcal{O}$.

If $\mathcal{O}$ is an obfuscator for the family of circuits $\mathcal{C}$, then we say that the pair $(\mathcal{O}, V)$ constitutes a *verifiable obfuscator* for $\mathcal{C}$.

We do not place any restrictions on V when its input is not the result of the obfuscator applied to a circuit in the family. In particular, given any obfuscator $\mathcal{O}$, the pair of algorithms $(\mathcal{O}, \mathbb{1})$ is a verifiable obfuscator, where $\mathbb{1}$ is the algorithm that accepts all inputs. In many cases, however, we can create much better verification algorithms. For example, the (r, r^x) construction of [2] can simply be verified by checking whether r and r^x are elements in the desired group G of prime order, because there is a unique discrete log of r^x so the program does implement a point circuit as desired.

Now we create a definition of non-malleability for verifiable obfuscators. As before, we consider an adversary that takes an obfuscated circuit as input and outputs a program. In this model, the adversary succeeds only if the outputted program passes the verification test and is related to the input program. Our definition of non-malleability requires that a simulator succeeds with approximately the same probability, so it must also output a program that passes the verification test. In particular, we no longer give an oracle to the program constructed by the simulator. A formal definition follows.

Definition 7 (Verifiable non-malleability). Let $\mathcal{C}$ and $\mathcal{D}$ be a families of circuits such that $\mathcal{C} \subseteq \mathcal{D}$, and let $\mathcal{O}$ and V be PPT algorithms. We say that $(\mathcal{O}, V)$ is *an obfuscator for $\mathcal{C}$ that is verifiably non-malleable over $\mathcal{D}$* if the following four conditions hold:

- *Verification:* V is a verifier for $\mathcal{O}$ applied to $\mathcal{C}$.
- *Almost exact functionality:* There exists a negligible function ε such that for every n and every circuit $C \in \mathcal{C}_n$, $\Pr[r \leftarrow \mathcal{R}_n : \mathcal{O}(C, r) \equiv C] > 1 - \varepsilon(n)$.
- *Polynomial slowdown:* There exists a polynomial p such that for every n, circuit $C \in \mathcal{C}_n$, and randomness $r \in \mathcal{R}_n$, the description length $|\mathcal{O}(C, r)| \leq p(|C|)$.

- *Verifiable non-malleability:* for every polynomial ρ and PPT adversary A, there exists a PPT simulator S such that for all sufficiently large n, for all circuits $C \in \mathcal{C}_n$, for all polynomial time computable relations $E : \mathcal{C}_n \times \mathcal{C}_n \to \{0,1\}$, and for all auxiliary information $z \in \{0,1\}^*$,

$$\begin{aligned}|\Pr[P \leftarrow A(\mathcal{O}(C), z) : P \neq \mathcal{O}(C), V(P) = 1, \text{and } \exists D \in \mathcal{D}_n \text{ s.t. } D \equiv P \text{ and } E(C,D) = 1] \\ - \Pr\left[Q \leftarrow S^C(1^n, z) : V(Q) = 1 \text{ and } \exists D \in \mathcal{D}_n \text{ s.t. } D \equiv Q \text{ and } E(C,D) = 1\right]| < \frac{1}{\rho(n)},\end{aligned}$$

  where A and S run in time polynomial in the length of their first inputs.

If $\mathcal{D} = \mathcal{C}$, we simply say that $(\mathcal{O}, V)$ is a *verifiably non-malleable obfuscator for $\mathcal{C}$.*

We remark that the definition subjects the simulator to the verification test, although it is reasonable to weaken the definition by removing this requirement. We choose the stronger definition because it puts the adversary and simulator on more equal footing and because all of our constructions satisfy the stronger notion. Additionally, the remarks pertaining to functional non-malleability also apply here:

1. Restricting attention to univariate relations $E(D)$ results in an equivalent definition.
2. For many circuit families $\mathcal{C}$, it is impossible to achieve verifiable non-malleability if $\mathcal{D}$ is the family of all circuits. The proof of this statement is identical to the proof of Theorem 4 in the functional non-malleability case.
3. Verifiable non-malleability also implies the virtual black-box property. (This holds even if neither property allows auxiliary input, or if both properties require only negligible difference in the success probabilities of the adversary and simulator.)

Theorem 8. *Let $\mathcal{C}$ and $\mathcal{D}$ be circuit families such that $\mathcal{C} \subseteq \mathcal{D}$, and let $\mathcal{O}$ and V be a pair of PPT algorithms such that $(\mathcal{O}, V)$ is an obfuscator for $\mathcal{C}$ that is verifiably non-malleable over $\mathcal{D}$. Then, $\mathcal{O}$ satisfies the virtual black-box requirement for $\mathcal{C}$. As a result, $(\mathcal{O}, V)$ is a verifiable obfuscator for $\mathcal{C}$.*

This proof is very similar to the one in the functional non-malleability setting, with two major differences. First, the condition that $\mathcal{C} \subseteq \mathcal{D}$ makes the technical conditions from Theorem 5 unnecessary here: it is always true that either $\mathcal{D}_n$ is non-trivial or $\mathcal{C}_n$ is trivial. Second, in the previous proof, we created an adversary A' that outputs one of two circuits $D_0, D_1 \in \mathcal{D}_n$. In the verifiable setting, this no longer suffices, because A' only succeeds if it outputs a circuit that passes the verification test. The only circuits that are guaranteed to pass the verification test with high probability are valid obfuscations of circuits in $\mathcal{C}_n$, so we set A' to output circuits of this form.

Proof. Let A be an adversary with binary output and let ρ be a polynomial. We need to construct a simulator S for A that satisfies the virtual black-box property. We prove this statement in two cases.

First, consider all n such that $\mathcal{D}_n$ is non-trivial, in the sense that there exist two circuits in $\mathcal{D}_n$ that do not have the same functionality. Choose a circuit $C_1 \in \mathcal{C}_n$ arbitrarily, and by non-triviality there exists a circuit $D_0 \in \mathcal{D}_n$ such that $D_0 \not\equiv C_1$. That is, there exists at least one input value x and one bit position i such that $(D_0(x))_i \neq (C_1(x))_i$. Let $c = (C_1(x))_i$.

We use A to form an adversary A' that is not constrained to binary output in the following manner. By nonuniformity, A' stores A, D_0, and C_1 in some readily identifiable way. On input $\mathcal{O}(C)$ and auxiliary information z, the circuit A' operates as follows:

```
1: emulate an execution of $A(\mathcal{O}(C), z)$ until it returns a bit $b$
2: **if** $b = 1$ **then**
```

```
3:     return an obfuscation of $C_1$
4: else
5:     return $D_0$
6: end if
```

In particular, if A outputs 1, then A' outputs a well-formed obfuscation of a circuit in $\mathcal{C}_n$, whereas if A outputs 0, then A' outputs a circuit in $\mathcal{D}_n$ with different functionality.

The verifiable non-malleability property applied to the adversary A' and polynomial 2ρ guarantees the existence of a simulator S' such that for all sufficiently large n, for all $C \in \mathcal{C}_n$, for all auxiliary information $z \in \{0,1\}^*$, and for all PPT relations E,

$$\begin{aligned}|\Pr\big[&P \leftarrow A'(\mathcal{O}(C), z) : P \neq \mathcal{O}(C), V(P) = 1, \text{and } \exists D \in \mathcal{D}_n \text{ s.t. } D \equiv P \text{ and } E(C,D) = 1\big] \\ &- \Pr\big[Q \leftarrow (S')^C(1^n, z) : V(Q) = 1 \text{ and } \exists D \in \mathcal{D}_n \text{ s.t. } D \equiv Q \text{ and } E(C,D) = 1\big]\,| < \frac{1}{2\rho(n)}. \quad (3)\end{aligned}$$

Finally, we construct a simulator S with binary output that nonuniformly stores V, x, i, and c and operates as follows, when given auxiliary information z and access to an oracle C:

```
1: emulate an execution of $(S')^C(1^n, z)$ until it returns a string $s$
2: if $s$ is not the valid encoding of a circuit then
3:     return 0
4: end if
5: let $Q$ be the circuit that is represented by $s$
6: if $V(Q) = 0$ or $(Q(x))_i \neq c$ then
7:     return 0
8: else
9:     return 1
10: end if
```

We now show that this simulator S satisfies the virtual black-box property.

Let $\tilde{E}$ be the relation that nonuniformly stores x and i and operates as follows: $\tilde{E}(C,D)$ returns 1 if and only if $D(x)_i = c$. In particular, the output of $\tilde{E}$ is independent of the choice of C. Then, for all n, all $C \in \mathcal{C}_n$, all $z \in \{0,1\}^*$,

$$\Pr[A(\mathcal{O}(C), z) = 1] = \Pr\big[A'(\mathcal{O}(C), z) \text{ outputs an obfuscation of } C_1\big]$$

as an immediate consequence of the construction of A'. As a result, we claim that

$$\begin{aligned}|\Pr&[A(\mathcal{O}(C), z) = 1] \\ &- \Pr[P \leftarrow A'(\mathcal{O}(C), z) : P \neq \mathcal{O}(C), V(P) = 1, \text{and } \exists D \in \mathcal{D}_n \text{ s.t. } D \equiv P \text{ and } \tilde{E}(C,D) = 1]|\end{aligned}$$

is negligible. To see this, first note that by the exact functionality of $\mathcal{O}$ and the fact that $\mathcal{C}_n \subseteq \mathcal{D}_n$, it follows that A' always outputs a program that is equivalent to a member of $\mathcal{D}_n$. Furthermore, by definition the verifier V always accepts an obfuscation of C_1. There is only one small technical problem: in the case that $C \equiv C_1$, the output of A' might equal its input, which is not allowed in the definition of verifiable non-malleability. However, A' uses fresh randomness in its obfuscation of C_1, so it can only output the program $\mathcal{O}(C)$ with negligible probability.

Next, using (3) and the behavior of the relation $\tilde{E}$, it follows that for all sufficiently large n, for all $C \in \mathcal{C}_n$, and for all auxiliary information z,

$$\begin{aligned}|\Pr&[A(\mathcal{O}(C), z) = 1] \\ &- \Pr\big[Q \leftarrow (S')^C(1^n, z) : V(Q) = 1 \text{ and } \exists D \in \mathcal{D}_n \text{ s.t. } D \equiv Q \text{ and } (D(x))_i = c\big]\,| < \frac{1}{2\rho(n)}. \quad (4)\end{aligned}$$

Second, let $\hat{E}$ be the relation such that $\hat{E}(C, D) = 1$ if and only if $D(x)_i = 1 - c$. Using the same analysis as above, it follows that

$$\left|\Pr\left[A(\mathcal{O}(C), z) = 0\right] - \Pr\left[Q \leftarrow (S')^C(1^n, z) : \exists D \in \mathcal{D}_n \text{ s.t. } D \equiv Q \text{ and } (D(x))_i = 1 - c\right]\right| < \frac{1}{2\rho(n)},$$

where the only difference from before is that the behavior of $V(D_0)$ is unclear, whereas the behavior of $V(\mathcal{O}(C_1))$ is known. Applying the triangle inequality to the previous two inequalities, and using the fact that A always outputs either 0 or 1, it follows that

$$\Pr\left[Q \leftarrow (S')^C(1^n, z) : \exists D \in \mathcal{D}_n \text{ s.t. } D \equiv Q\right] > 1 - \frac{1}{2\rho(n)}.$$

Finally, by the construction of S from S', it is easy to see that

$$\Pr\left[S^C(1^n, z) = 1\right] = \Pr\left[Q \leftarrow (S')^C(1^n, z) : V(Q) = 1 \text{ and } (Q(x))_i = c\right].$$

Furthermore, we showed that Q fails to be equivalent to a member of $\mathcal{D}_n$ with small probability, so

$$\begin{aligned} &| \Pr\left[S^C(1^n, z) = 1\right] \\ &\qquad - \Pr\left[Q \leftarrow (S')^C(1^n, z) : V(Q) = 1 \text{ and } \exists D \in \mathcal{D}_n \text{ s.t. } D \equiv Q \text{ and } (D(x))_i = c\right] | < \frac{1}{2\rho(n)}. \end{aligned}$$

Applying the triangle inequality to this inequality and (4) yields

$$\left|\Pr\left[A(\mathcal{O}(C), z) = 1\right] - \Pr\left[S^C(1^n, z) = 1\right]\right| < \frac{1}{\rho(n)},$$

so S satisfies the virtual black-box property as desired.

Now we prove the theorem for all n such that $\mathcal{D}_n$ is trivial, meaning that all of the circuits in $\mathcal{D}_n$ have the same functionality. In this case, we claim that for every adversary A with binary output, for every $z \in \{0,1\}^*$, for sufficiently large n, and for every $C, C' \in \mathcal{C}_n$,

$$\left|\Pr\left[A(\mathcal{O}(C), z) = 1\right] - \Pr\left[A(\mathcal{O}(C'), z) = 1\right]\right|$$

is negligible. If this is true, then it is easy to construct a simulator S for A: the simulator simply chooses a circuit $C \in \mathcal{C}_n$ arbitrarily, obfuscates it, and emulates an execution of A on the obfuscated circuit. It is straightforward to check that this simulator satisfies the virtual black-box property.

Suppose for the sake of contradiction that the claim is not true. Therefore, there exist an adversary A, a polynomial ρ, and auxiliary information $z \in \{0,1\}^*$ such that for infinitely many n, there exist two circuits $C, C' \in \mathcal{C}_n$ such that

$$\left|\Pr\left[A(\mathcal{O}(C), z) = 1\right] - \Pr\left[A(\mathcal{O}(C'), z) = 1\right]\right| > \frac{1}{\rho(n)}.$$

Now form an adversary A' (that is not constrained to binary output) as follows. Arbitrarily choose a circuit $C_1 \in \mathcal{C}_n$, and let D_0 be a circuit that is not equivalent to C_1, and hence not equivalent to any member of $\mathcal{D}_n$. By nonuniformity, the adversary A' stores A, D_0, and C_1 in some readily identifiable way. The circuit A' operates as follows, upon receiving $\mathcal{O}(C)$ and z as input:

```
1: emulate an execution of $A(\mathcal{O}(C), z)$ until it returns a bit $b$
2: **if** $b = 1$ **then**
```

```
3:     return an obfuscation of $C_1$
4: else
5:     return $D_0$
6: end if
```

Let $\bar{E}$ be the relation that always accepts. As a result,

$$\begin{aligned}&|\Pr[A(\mathcal{O}(C), z) = 1]\\&\quad - \Pr\left[P \leftarrow A'(\mathcal{O}(C), z) : P \neq \mathcal{O}(C), V(P) = 1, \text{and } \exists D \in \mathcal{D}_n \text{ s.t. } D \equiv P \text{ and } \bar{E}(C, D) = 1\right]|\end{aligned}$$

is negligible, because $\mathcal{C}_1$ is equivalent to a member of $\mathcal{D}_n$ and passes the verification test, whereas D_0 is not equivalent to a member of $\mathcal{D}_n$. The only problem is that the output of A' might equal its input, but this occurs with negligible probability.

Hence, by our assumption above, it follows that for infinitely many n, there exist two circuits $C, C' \in \mathcal{C}_n$ and a polynomial ρ such that

$$\begin{aligned}&|\Pr\left[P \leftarrow A'(\mathcal{O}(C), z) : P \neq \mathcal{O}(C), V(P) = 1, \text{and } \exists D \in \mathcal{D}_n \text{ s.t. } D \equiv P \text{ and } \bar{E}(C, D) = 1\right]\\&-\Pr\left[P \leftarrow A'(\mathcal{O}(C'), z) : P \neq \mathcal{O}(C), V(P) = 1, \text{and } \exists D \in \mathcal{D}_n \text{ s.t. } D \equiv P \text{ and } \bar{E}(C', D) = 1\right]| > \frac{1}{2\rho(n)}.\end{aligned}$$

However, for any simulator S',

$$\begin{aligned}&\Pr\left[Q \leftarrow (S')^C(1^n) : V(Q) = 1 \text{ and } \exists D \in \mathcal{D}_n \text{ s.t. } D \equiv Q \text{ and } \bar{E}(C, D) = 1\right]\\&\qquad = \Pr\left[Q \leftarrow (S')^{C'}(1^n) : V(Q) = 1 \text{ and } \exists D \in \mathcal{D}_n \text{ s.t. } D \equiv Q \text{ and } \bar{E}(C', D) = 1\right]\end{aligned}$$

information-theoretically because C and C' are functionally equivalent. Applying the triangle inequality to the previous two equations, it must be the case that either

$$\begin{aligned}&|\Pr\left[P \leftarrow A'(\mathcal{O}(C), z) : P \neq \mathcal{O}(C), V(P) = 1, \text{and } \exists D \in \mathcal{D}_n \text{ s.t. } D \equiv P \text{ and } \bar{E}(C, D) = 1\right]\\&\qquad - \Pr\left[Q \leftarrow (S')^C(1^n) : V(Q) = 1 \text{ and } \exists D \in \mathcal{D}_n \text{ s.t. } D \equiv Q \text{ and } \bar{E}(C, D) = 1\right]| > \frac{1}{4\rho(n)}\end{aligned}$$

or the same is true of C'. As a result, S' fails to satisfy the functional non-malleability requirement. This is true for all possible S', so $\mathcal{O}$ is not functionally non-malleable over $\mathcal{D}$, which is a contradiction. □

3.3 Non-malleability in models with setup assumptions

In this section, we modify the non-malleability definitions to fit within the random oracle and common reference string models.

3.3.1 Random oracle model

In the random oracle model, every algorithm has access to a length-preserving permutation $R : \{0,1\}^n \rightarrow \{0,1\}^n$ chosen uniformly at random from the set of such permutations. The changes to the functional and verifiable non-malleability properties from Definitions 3 and 7 are detailed below.

Definition 9 (Functional non-malleability in the random oracle model). An algorithm $\mathcal{O}$ satisfies the functional non-malleability property in the random oracle model if for every polynomial ρ and PPT adversary A, there exists a PPT simulator S such that for all sufficiently large n, for all circuits $C \in \mathcal{C}_n$, for all polynomial time computable relations $E : \mathcal{C}_n \times \mathcal{D}_n \to \{0,1\}$ (that may depend on the circuit C), and for all auxiliary information $z \in \{0,1\}^*$,

$$\Big| \Pr\big[P \leftarrow A^R(\mathcal{O}(C)^R, z) : \exists D \in \mathcal{D}_n \text{ s.t. } D \equiv P^R \text{ and } E^R(C,D) = 1\big]$$
$$- \Pr\big[Q \leftarrow S^{C,R}(1^n, z) : \exists D \in \mathcal{D}_n \text{ s.t. } D \equiv Q^{C,R} \text{ and } E^R(C,D) = 1\big] \Big| < \frac{1}{\rho(n)},$$

where the probabilities are over the coin tosses of A, $\mathcal{O}$, and S, as well as the choice of $R : \{0,1\}^n \to \{0,1\}^n$ from the uniform distribution over all permutations. As before, we require that A and S run in time polynomial in the length of their first inputs.

Definition 10 (Verifiable non-malleability in the random oracle model). The pair $(\mathcal{O}, V)$ satisfies the verifiable non-malleability property in the random oracle model if for every polynomial ρ and PPT adversary A, there exists a PPT simulator S such that for all sufficiently large n, for all circuits $C \in \mathcal{C}_n$, for all polynomial time computable relations $E : \mathcal{C}_n \times \mathcal{C}_n \to \{0,1\}$, and for all auxiliary information $z \in \{0,1\}^*$,

$$\Big| \Pr\big[P \leftarrow A^R(\mathcal{O}(C)^R, z) : P \neq \mathcal{O}(C)^R, V^R(P^R) = 1, \text{ and } \exists D \in \mathcal{D}_n \text{ s.t. } D \equiv P^R \text{ and } E^R(C,D) = 1\big]$$
$$- \Pr\big[Q \leftarrow S^{C,R}(1^n, z) : V(Q^R) = 1 \text{ and } \exists D \in \mathcal{D}_n \text{ s.t. } D \equiv Q^R \text{ and } E^R(C,D) = 1\big] \Big| < \frac{1}{\rho(n)},$$

where A and S run in time polynomial in the length of their first inputs.

Note that the relation E and verifier V in these definitions receives the same random oracle that the adversary and simulator receive. As a result, the definitions impose a nonprogrammability restraint on the simulator because E and V collectively serve as an "environment" (in the UC-sense [3]) that can try to detect any differences in the behavior of the adversary and simulator.

3.3.2 Common reference string model

In the common reference string model, a long string Σ is generated by a trusted party with each bit being chosen independently and uniformly at random. In this paper, we only explore verifiable non-malleability in the CRS model. We augment the definition to allow all algorithms in the real world access to the string Σ. In the simulated world, S is allowed to choose Σ.

Definition 11 (Verifiable non-malleability in the common reference string model). The pair $(\mathcal{O}, V)$ satisfies the verifiable non-malleability property in the common reference string model if for every polynomial ρ and PPT adversary A, there exists a PPT simulator S such that for all sufficiently large n, for all circuits $C \in \mathcal{C}_n$, for all polynomial time computable relations $E : \mathcal{C}_n \times \mathcal{C}_n \to \{0,1\}$, and for all auxiliary information $z \in \{0,1\}^*$,

$$\Big| \Pr\left[P \leftarrow A(\mathcal{O}(C), z, \Sigma) : P \neq \mathcal{O}(C), V(P,\Sigma) = 1, \text{ and } \exists D \in \mathcal{D}_n \text{ s.t. } D \equiv P \text{ and } E(C,D) = 1\right]$$
$$-\Pr\big[(Q,\Sigma) \leftarrow S^C(1^n, z) : |\Sigma| = l(n), V(Q,\Sigma) = 1, \text{ and } \exists D \in \mathcal{D}_n \text{ s.t. } D \equiv Q \text{ and } E(C,D) = 1\big] \Big| < \frac{1}{\rho(n)},$$

where A, S, and V run in time polynomial in the length of their first inputs. The first probability is taken over the random coin tosses of A and $\mathcal{O}$ along with the uniformly random choice of Σ, and the second probability is taken over the random coin tosses of S.

Note that in the real execution of the adversary, the verifier receives the same common reference string as the adversary, but in the simulated world, S chooses the CRS for the verifier.

3.4 Comparison of non-malleability definitions

The virtual black-box property can be incorporated into the random oracle model in the same way as the non-malleability properties were, and both flavors of non-malleability still imply the virtual black-box property in the random oracle model. However, the the two types of non-malleability are fundamentally different extensions of the virtual black-box definition.

Theorem 12. *In the random oracle model, the definitions of functional and verifiable non-malleability are incomparable. Specifically,*

1. *There exists a circuit family $\mathcal{C}$ and a functionally non-malleable obfuscator $\mathcal{O}$ for $\mathcal{C}$ such that no verifier V makes the pair $(\mathcal{O}, V)$ a verifiably non-malleable obfuscator for $\mathcal{C}$.*
2. *There exists a circuit family $\mathcal{C}$ and a verifiable obfuscator $(\mathcal{O}, V)$ for $\mathcal{C}$ such that $\mathcal{O}$ is not a functionally non-malleable obfuscator for $\mathcal{C}$.*

Proof. The two statements are immediate consequences of Theorems 15 and 18, respectively. □

4 Constructions of functionally non-malleable obfuscators

In this section, we present a functionally non-malleable obfuscator for the family of multi-point circuits in the random oracle model. We begin with the single-point case.

4.1 Single-point circuits

Algorithm 1 describes an obfuscator $\mathcal{O}_{\mathcal{P}^1}$ for the family of point circuits $\mathcal{P}^1$ [12].

Algorithm 1 Obfuscator $\mathcal{O}_{\mathcal{P}^1}$ for the family of point circuits

```
Input: a circuit of the form $I_w$
1: extract the point $w$ and choose randomness $r \leftarrow \{0,1\}^{3|w|}$
2: compute $t = R(w \circ r)$, where $\circ$ denotes the string concatenation operation
3: output a description of the following circuit, which stores $r$ and $t$ in some clearly identifiable manner
   and which we denote $\Phi_{r,t}$: "on input a string $x$, output 1 if $R(x \circ r) = t$ and 0 otherwise"
```

Theorem 13. *In the random oracle model, the algorithm $\mathcal{O}_{\mathcal{P}^1}$ is a functionally non-malleable obfuscator for the family of point circuits $\mathcal{P}^1$.*

Proof. The exact functionality of $\mathcal{O}_{\mathcal{P}^1}$ follows from the fact that R is a permutation. Polynomial slowdown clearly holds because the runtime of $\Phi_{r,t}$ is linear in its input length. It remains to prove the functional non-malleability property.

To prove that $\mathcal{O}_{\mathcal{P}^1}$ is functionally non-malleable, we need to construct a simulator S for any adversary A. We informally explain S here and give a complete description of S in Algorithm 2.

The simulator has auxiliary information z and oracle access to the point circuit I_w. It makes a fake obfuscated program $\Phi_{r,t}$ by choosing r and t uniformly at random, and then emulates an execution of $A^R(\Phi_{r,t}, z)$. In addition, S examines all of A's random oracle queries. There are three cases to consider:

1. If A queries R on the input $w \circ r$, then the adversary A found the hidden point w, but now the simulator has too. As a result, the simulator aborts this emulation of A, forms a valid obfuscation of I_w, runs A on it, and outputs the resulting program.

Algorithm 2 Algorithm for the simulator $S^{R,I_w}(1^n, z)$ in the proof of Theorem 13

```
Input: inputs $N$ and $z$, along with access to oracles $R$ and $C$
 1: set $n \leftarrow |N|$
 2: choose random $r \leftarrow \{0,1\}^{3n}$ and $t \leftarrow \{0,1\}^{4n}$
 3: repeat
 4:   emulate one step of the execution of $A(\Phi_{r,t}, z)$
 5:   if $A$ queries its random oracle on input value $q$ then
 6:     if $R(q) = t$ then
 7:       return $\perp$
 8:     else if $|q| \neq 4n$ or the last $3n$ bits of $q$ are not equal to $r$ then
 9:       respond to the query with $R(q)$
10:     else
11:       set $y$ to the first $n$ bits of $q$
12:       if $C(y) = 1$ then
13:         abort this emulation
14:         emulate $A(\Phi_{r,R(y \circ r)})$, responding to all oracle queries accurately, then goto step 21
15:       else
16:         respond to the query with $R(q)$
17:       end if
18:     end if
19:   end if
20: until $A$ halts
21: set $P$ to the output of the emulation $A(\Phi_{r,t}, z)$ from step 20, or the output of $A(\Phi_{r,R(x \circ r)})$ from step 12
22: use $P$, $r$, and $t$ to develop the program $Q$, which for legibility is written below
23: return a description of $Q$
```

Algorithm 3 Algorithm for program Q^{R,I_w} in the proof of Theorem 13

```
Input: a string $x$
 1: repeat
 2:   emulate one step of the execution of $P^R(x)$
 3:   if $A$ queries its random oracle on input value $q$ then
 4:     if $R(q) = t$ then
 5:       choose a random value $s \leftarrow \{0,1\}^{4n}$ and respond to the oracle query with $s$
 6:     else if $|q| \neq 4n$ or the last $3n$ bits of $q$ are not equal to $r$ then
 7:       respond to the query with $R(q)$
 8:     else
 9:       set $y$ to the first $n$ bits of $q$
10:       if $C(y) = 1$ then
11:         respond to the query with $t$
12:       else
13:         respond to the query with $R(q)$
14:       end if
15:     end if
16:   end if
17: until $P$ halts
18: return the value that the emulation of $P$ outputs
```

2. If A queries the value $s = R^{-1}(t)$, then A has discovered that our obfuscation is fake. In this case, the simulator fails. However, t is chosen uniformly at random and independently of A and z, so this case only occurs with negligible probability.

3. On any other query q, S responds honestly with $R(q)$.

Note that S either responds to oracle queries honestly or aborts the emulation of A.

Once the emulation of A terminates, it outputs a program P. Then, the simulator S has to output a program Q^{R,I_w}. We explain here how Q^{R,I_w} operates and provide a complete description in Algorithm 3.

The circuit Q^{R,I_w} runs P^R but continues to examine oracle calls. Once again, there are three cases to consider:

1. If P queries R on the input $w \circ r$, then Q answers the query with t.

2. If P queries the value $s = R^{-1}(t)$, then Q answers the query with a random value in $\{0,1\}^{4n}$.

3. Otherwise, Q gives the correct response to P.

Now we analyze the above simulator and show that it satisfies the functional non-malleability requirement . We break the analysis of the simulator into cases based on the random oracle queries made by the emulated adversary. If the emulated adversary ever queries the correct password, then the emulation becomes perfect. Additionally, the emulated adversary runs in polynomial time, so it can only invert the random oracle and make the query $s = R^{-1}(t)$ with negligible probability. Therefore, it suffices to restrict our attention to the case in which neither of these situations occurs.

Let E be any relation that only makes polynomially-many oracle queries, and define

$$p_A = \Pr\left[C \leftarrow A^R(\mathcal{O}(I_w), z) : \exists I_{w'} \in \mathcal{P}^1 \text{ s.t. } I_{w'} \equiv C^R \text{ and } E^R(I_w, I_{w'}) = 1\right]$$
$$p_S = \Pr\left[D \leftarrow S^{R,I_w}(1^n, z) : \exists I_{w'} \in \mathcal{P}^1 \text{ s.t. } I_{w'} \equiv D^{R,I_w} \text{ and } E^R(I_w, I_{w'}) = 1\right]$$

as the probabilities of success for the real adversary and simulator, respectively. We will show that $|p_A - p_S|$ is negligible in n.

Perform the following "mental experiment:" let R' be a different random oracle permutation in which

$$R'(w \circ r) = t, \quad R'(s) = R(w \circ r),$$

and R' agrees with R on all other values. (See Figure 1 for a diagram.) If R' really were the random oracle, then the "fake" obfuscation $\Phi_{r,t}$ would actually be a valid obfuscation of I_w. As a result,

$$p_A = \Pr\left[C \leftarrow A^{R'}(\Phi_{r,t}, z) : \exists I_{w'} \in \mathcal{P}^1 \text{ s.t. } I_{w'} \equiv C^{R'} \text{ and } E^{R'}(I_w, I_{w'}) = 1\right].$$

Furthermore, by assumption A never queries the oracle at the two locations in which R and R' differ. Hence, A must act the same in both cases, so the program P returned by the emulation of A^R on the "fake" obfuscation is identical to the program returned by $A^{R'}$ when given a real obfuscation.

Consider a new simulator S' that operates just like S, except that after it finishes its emulation of the adversary, it simply returns the program P that the emulated adversary outputs. Then,

$$p_A \approx \Pr\left[P \leftarrow (S')^{R,I_w}(1^n, z) : \exists I_{w'} \in \mathcal{P}^1 \text{ s.t. } I_{w'} \equiv P^{R'} \text{ and } E^{R'}(I_w, I_{w'}) = 1\right],$$

where the symbol $\approx$ is used to denote the fact that the two probabilities may differ by the negligible probability that the emulated adversary queries its random oracle on $s = R^{-1}(t)$, forcing the simulator to abort.

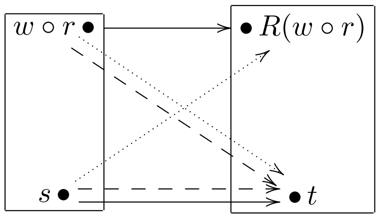

Figure 1: Pictorial representation of the actions of the actual random oracle R (solid line) and the two imaginary oracles R' (dotted line) and R'' (dashed line). The domain and range are both $\{0,1\}^{4|w|}$. The three random oracles operate identically on all other values.

Of course, the simulator S constructed above does not return P, but rather a related program Q that makes a few modifications to random oracle queries. These modifications are precisely the ones that make $P^{R'} \equiv Q^R$, except in the case that P queries the value $s = R^{-1}(t)$. Because this happens with negligible probability,

$$p_A \approx \Pr\left[Q \leftarrow S^{R,I_w}(1^n, z) : \exists I_{w'} \in \mathcal{P}^1 \text{ s.t. } I_{w'} \equiv Q^R \text{ and } E^{R'}(I_w, I_{w'}) = 1\right].$$

Next, let R'' be an oracle in which $R''(w \circ r) = t$, and R'' agrees with R on all other values. (See Figure 1 for a diagram.) In other words, R'' and R' differ only on the input s. We consider the difference in functionality between $E^{R'}$ and $E^{R''}$. Because the adversary, simulator, and relation E all run in polynomial time, they only query the value s with negligible probability. Therefore,

$$p_A \approx \Pr\left[Q \leftarrow S^{R,I_w}(1^n, z) : \exists I_{w'} \in \mathcal{P}^1 \text{ s.t. } I_{w'} \equiv Q^R \text{ and } E^{R''}(I_w, I_{w'}) = 1\right].$$

Finally, we claim that there is only a negligible difference in functionality between E^R and $E^{R''}$. This claim is information-theoretic: R and R'' only differ on input $w \circ r$, which has at least $3n$ bits of entropy, but the relation E only receives $2n$ bits of information as input (namely, the values w and w'). As a result, using any list encoding scheme, the adversary and simulator only have negligible probability of transmitting the value r to the relation E. If the relation does not query the value $w \circ r$, then the behavior of E^R and $E^{R''}$ is identical. As a result,

$$p_A \approx \Pr\left[Q \leftarrow S^{R,I_w}(1^n, z) : \exists I_{w'} \in \mathcal{P}^1 \text{ s.t. } I_{w'} \equiv Q^R \text{ and } E^R(I_w, I_{w'}) = 1\right] = p_S,$$

as desired.

Hence, $\mathcal{O}_{\mathcal{P}^1}$ is a functionally non-malleable obfuscator for $\mathcal{P}^1$, and in fact the definition is achieved in a strong sense in which the simulator S is independent of the choice of the polynomial ρ. $\square$

4.2 Multi-point circuits

Constructing a functionally non-malleable obfuscator for the family of multi-point circuits $\mathcal{P}^m$ is significantly more difficult. Roughly speaking, the principal issue is that the obfuscated program must "bundle together" the m points in a way that would prevent the adversary from changing any point in the bundle without applying the exact same change to all points in the bundle. For instance, simply concatenating m obfuscations of a single-point function (even obfuscations that are individually non-malleable) does not suffice because an adversary will be able to change some of the points at will. Instead, the code of the program must be a "house of cards" in the sense that an adversary cannot change the code without destroying information about all of the accepted points simultaneously.

This is a difficult task to achieve, so we first demonstrate some warm-up examples in which the obfuscator receives special abilities.

First warm-up. In this example, we assume the existence of an m-to-1 random oracle, although this may easily be constructed from a permutation. Additionally, we augment the model by making two unrealistic assumptions.

1. Rather than receiving a list of accepted points $\{w_1, \ldots, w_m\}$, the obfuscator chooses these points on her own from the uniform distribution. An obfuscator of this type may still be useful in several situations, such as the motivating scenario from the Introduction in which m graduate students wish to share a computer. The students may allow the network administrator to choose passwords for them, as long as the passwords are chosen uniformly.

2. The obfuscator receives a special auxiliary input consisting of a random point t and all m preimages of t under the random oracle. Nobody else in the world receives this auxiliary input (which can be thought of as "trapdoor" information on the random oracle).

In this setting, the obfuscator can do the following.

```
1: choose a target point $t \leftarrow \{0,1\}^n$ uniformly at random
2: compute the $m$ inverses of $t$, and denote them as $w_1, \ldots, w_m$
3: output the following program, which stores $t$ in some easily identifiable way: "on input $x$, accept iff
   $R(x) = t$
4: distribute passwords to the appropriate entities
```

Note that the output of this algorithm is clearly an m-point circuit, since the random oracle is m-to-1. Furthermore, the only information stored by the program is t. This value "glues" together all m passwords, in the sense that an adversary cannot modify t without destroying all of the information about every password simultaneously.

Second warm-up. While the above protocol is clean and simple, it operates using unrealistic assumptions, which we would like to remove. In the next warm-up, we return to the setting in which the obfuscator receives a set of accepted points $\{w_1, \ldots, w_m\}$, rather than being able to choose them on her own. However, the obfuscator is still able to invert the random oracle at a single point t.

Suppose the obfuscator starts as before by computing the m preimages of t under the random oracle, which we denote by $w'_1, w'_2, \ldots, w'_m$. It would be nice if the real accepted points w_i were equal to the w'_j, for then the obfuscator could use the first warm-up to complete her task, but unfortunately this is not the case.

As it stands, the obfuscator needs a way to "link" the real accepted points to the fake ones. One such link is a polynomial. Specifically, view the w_i and w'_i as elements of a finite field, and let q be the unique degree $m-1$ polynomial over this field such that $q(w_i) = w'_i$ for all i. Then, the obfuscator creates the program "on input x, accept if and only if $R(q(x)) = t$." We show the algorithm pictorially in Figure 2.

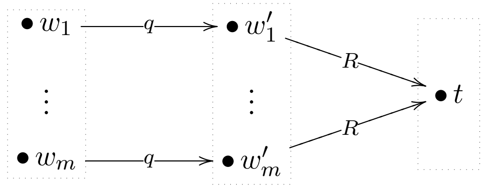

Figure 2: Pictorial representation of the obfuscated m-point program in the second warm-up.

In this algorithm, the polynomial q "glues" together m passwords, in the sense that it is not possible for an adversary to change the polynomial without destroying information on all the points simultaneously, except for those points that the adversary already knows. There are two technical issues that prevent this algorithm from being an obfuscator.

1. There is no randomness in the obfuscation. That is, if the obfuscator receives the same set of passwords twice, then she will output the same program both times. As shown in [2], a deterministic algorithm inherently cannot be an obfuscator.

2. The obfuscator's program does not preserve functionality. By construction, it accepts all of the w_i, but it accepts additional values too. In particular, it accepts all points x such that $q(x)$ equals any of the w'_i, and there are up to $m(m-1)$ such points.

Luckily, both of these issues can be resolved simultaneously with a simple change to the algorithm. The obfuscator needs to choose a long random string r, and apply the previous algorithm to the strings $\{w_1 \circ r, \ldots, w_m \circ r\}$ instead of just the passwords $\{w_1, \ldots, w_m\}$. As a result, the obfuscator outputs a program of the form "on input x, accept if and only if $R(q(x \circ r)) = t$." With overwhelming probability, the other inverses of the polynomial q will not be strings that end in r, so the obfuscation will have exact functionality.

Recall that the obfuscator still receives special "trapdoor" information about the inverses of t under the random oracle. As a result, an adversary cannot run the obfuscation algorithm on her own, so her only hope of creating an obfuscated multi-point circuit is to modify a program provided by the obfuscator. Fortunately, it can be proved that this algorithm is a functionally non-malleable obfuscator for the family of multi-point circuits $\mathcal{P}^m$ in the "trapdoor" random oracle model.

Full protocol for $m = 2$. Now we remove the second unrealistic assumption and return to the traditional random oracle model, in which the obfuscator does not receive any special information about the inverses of the random oracle. We formally present a functionally non-malleable obfuscator in the $m = 2$ case, and then extend it to arbitrary values of m.

In the second warm-up, R played two roles. In addition to its use as a random oracle, it also serves as a sort of trapdoor one-way function because the obfuscator receives a special inversion power. In this protocol, we separate the two duties into the following objects:

1. A random oracle permutation R that is invertible, so every algorithm has access to R^{-1} as well. This can be constructed within the standard random oracle model [6].

2. A trapdoor family of 2-to-1 one-way functions that also satisfies "second preimage resistance," which means that an adversary given a function $f \in T$ and an element x in the domain of f cannot find the element y such that $f(y) = f(x)$. One example is the family of Rabin one-way functions [13]

$$\{f_N(x) = x^2 \bmod N : \exists p, q \text{ prime s.t. } N = pq\}.$$

   Every $f_N : \mathbb{Z}_N^* \to \mathbb{Z}_N^*$ is a 4-to-1 function, although only two of the preimages of a point are really "independent" (and the other two preimages are their additive inverses). As a result, we restrict the domain of f_N to the set $\{1, \ldots, \lfloor \frac{N}{2} \rfloor\}$, in which case the function is 2-to-1.

The one-wayness and second preimage resistance of the Rabin family is based on the intractability of the factoring problem.

Definition 14 (Factoring assumption). Let $D(1^n)$ be a sampling algorithm that chooses two primes p, q uniformly at random from the set of n bit integers and outputs $N = pq$. Then, for every PPT algorithm A, the probability that $A(D(1^n))$ outputs a prime factor of its input is negligible (over the random coin tosses of A and D).

Additionally, we make the following change to the model. Whereas the single-point obfuscation operates in the nonprogrammable random oracle model (as defined in Section 3.3), this algorithm requires a programmable random oracle. In the single-point construction, the simulator S and the program Q that it makes trap on random oracle queries and potentially make a change to the oracle. Fortunately, it is easy to detect that a modification is necessary whenever the oracle query is made, so S and Q do not have to program the random oracle.

In this construction, due to the fact that the oracle is now invertible, it is not always easy to detect ahead of time that a random oracle query must be trapped. For instance, suppose that the simulator traps on the oracle query $R(x)$ and modifies its value to be y. Then, whenever an oracle query of the form $R^{-1}(y)$ is made, the simulator must remember its previous modification and respond consistently with the value x. Fortunately, the simulator can keep a table of all of its changes and pass this table to Q.

However, there is a problem whenever the circuit Q traps on a random oracle query and modifies its value. The circuit Q may be invoked multiple times, and it keeps no state between invocations. As a result, it has no way to remember previous modifications, so it cannot respond to future random oracle queries consistently.

As a result, we augment the model to give Q this extra capability. Specifically, the simulator S now outputs two programs: a circuit Q as before, and a small module M^R that interfaces with the random oracle. The module M is "always on" and has a small amount of storage space that enables it to remember any changes made to the random oracle. Then, the circuit Q has oracle access to M instead of interfacing with R directly.

An equivalent model is to augment the capability of Q directly by giving it a small amount of storage space that persists between executions, with the restriction that this memory can only be used to store changes made to the random oracle, and as a result the memory is only accessed during oracle queries. We use this notion in our construction for simplicity, and note that Q only requires $2n$ bits of persistent memory.

Define $\mathbb{F}^{(n)}$ to be the field $\mathbb{Z}/p\mathbb{Z}$ where p is the first prime bigger than 2^{6n}. We embed strings of length $\{0,1\}^{6n}$ into $\mathbb{F}^{(n)}$ (and vice-versa) in the natural way. We provide a pictorial representation of the obfuscator $\bar{\mathcal{O}}_{\mathcal{P}^2}$ in Figure 3 and then describe the full construction in Algorithm 4.

$$\begin{tikzcd}
\bullet\, w_1 \arrow[r, "l(-\circ r)"] & \bullet\, x_1 \arrow[r, "R"] & \bullet\, y_1 \arrow[dr, "\mathrm{Rabin}_N"] & \\
 & & & \bullet\, t \\
\bullet\, w_2 \arrow[r, "l(-\circ r)"] & \bullet\, x_2 \arrow[r, "R"] & \bullet\, y_2 \arrow[ur, "\mathrm{Rabin}_N"'] &
\end{tikzcd}$$

Figure 3: Pictorial representation of the action of an obfuscated two-point program created by $\bar{\mathcal{O}}_{\mathcal{P}^2}$. As before, t is the "glue" that binds the two accepted points together.

Algorithm 4 Obfuscator $\bar{\mathcal{O}}_{\mathcal{P}^2}$ for the family of two-point circuits

```
Input: a circuit of the form $I_{w_1,w_2,\dots,w_m}$
1: extract the points $w_1, w_2, \dots, w_m$ and randomize their order
2: choose strings $r \leftarrow \{0,1\}^{5n}$ and $t \leftarrow \{0,1\}^{6n}$ uniformly
3: choose two $6n$ bit primes $p, q$ uniformly at random (by sampling random integers and testing for primality), and let $N \leftarrow pq$
4: let $y_1, y_2$ be two square roots of $t \bmod N$ such that $y_1 \not\equiv \pm y_2 \bmod N$
5: let $x_i \leftarrow R^{-1}(y_i)\ \forall i$
6: in the finite field $\mathbb{F}^{(n)}$, find the line $l$ such that $l(w_i \circ r) = x_i\ \forall i$
7: output the program $\Psi_{r,l,N,t}$ that operates as follows: "on input $x$, compute $x' \leftarrow l(x \circ r)$ in the field $F^(n)$, $x'' \leftarrow R(x')^2$ in the group $Z_N^*$, and accept iff $x'' = t$"
```

Theorem 15. *Assume that factoring is intractable. In the random oracle model, the obfuscator $\bar{\mathcal{O}}_{\mathcal{P}^2}$ is a functionally non-malleable obfuscator for $\mathcal{P}^2$. However, $\bar{\mathcal{O}}_{\mathcal{P}^2}$ is malleable in the verifiable sense, meaning that no verifier algorithm $\bar{V}_{\mathcal{P}^2}$ makes the pair $(\bar{\mathcal{O}}_{\mathcal{P}^2}, \bar{V}_{\mathcal{P}^2})$ verifiably non-malleable.*

Many of the ideas in the proof follow that in the single-point case, and these details are only sketched here in order to focus on the new ideas, so the reader is encouraged to examine the proof of Theorem 13 before continuing.

Proof. The almost exact functionality of $\bar{\mathcal{O}}_{\mathcal{P}^2}$ follows for the same reason as in the second warm-up. There are four square roots of t in $\mathbb{Z}_N$, and R^{-1} and l^{-1} are bijections so there exist at most four strings s such that $R(l(s))^2 = t$ (where all operations are formed in their appropriate groups). Two of these strings are $w_1 \circ r$ and $w_2 \circ r$, whereas the other two are uniformly random over the choice of the random oracle, and with overwhelming probability these strings do not contain r as a substring so they do not lead to additional accepted points. Additionally, the polynomial slowdown property is clear. It remains to prove functional non-malleability.

Let A be an adversary, and we construct a simulator S for A. Given an input length n, auxiliary input z, and access to an oracle for a two-point circuit $I_{\{w_1,w_2\}}$, the simulator S constructs a "fake" obfuscated program by choosing $w_1', w_2' \in_U \{0,1\}^n$ uniformly at random and running the obfuscator on $I_{\{w_1',w_2'\}}$ to construct a program of the form $\Psi_{r,l,N,t}$ together with the primes p, q such that $N = pq$. Then, $S^{R,R^{-1},I_{\{w_1,\ldots,w_m\}}}$ emulates an execution of $A^{R,R^{-1}}(\Psi_{r,l,N,t}, z)$ and traps on A's oracle queries.

When A makes an oracle query of the form $R(x)$, the simulator computes $v = l^{-1}(x)$ and queries its oracle $I_{\{w_1,w_2\}}$ on the first n bits of the string v. If the oracle accepts, then S has found an accepted point, so it aborts this emulation of the adversary, replaces one of the fake accepted points with the real one, and restarts the emulation. Otherwise, S responds to the random oracle query honestly. Similarly, when A makes an oracle query of the form $R^{-1}(y)$, the simulator computes $v = R^{-1}(l^{-1}(x))$ and queries its oracle to test whether v encodes a new accepted point.

Note that S restarts the adversary at most twice. Eventually, the emulation of A halts and outputs a program $P^{R,R^{-1}}$. If S found both accepted points w_1 and w_2, then the final "fake" obfuscation is in fact a real obfuscation so the emulation of the adversary is perfect. Therefore, S simply outputs the program $Q = P$. Otherwise, S outputs the program Q that continues to run P but traps on its random oracle queries in the same manner as described above. If a new accepted point w is found, then Q alters the random oracle so that $R(l(w \circ r))$ is sent to one of the fake square roots y_1' or y_2' (whichever has not been used yet as an image of a real accepted point). Additionally, Q stores the value w so it can respond to future random oracle queries $R(l(w \circ r))$ or $R^{-1}(y_i')$ consistently. This requires $2n$ bits of persistent memory.

This concludes the explanation of the simulator S and its output circuit Q. Now we analyze these circuits and prove that they satisfy functional non-malleability for the adversary A. Let $\Psi_{r,l,N,t}$ be a fake obfuscation constructed by the simulator (who also knows y_1 and y_2 as a result) in its final emulation of the adversary, and consider a "mental experiment" in which we replace the random oracle R with a fake oracle R' such that R' differs from R only on the four values

$$R'(l(w_i \circ r)) = y_i, \quad R'(R^{-1}(y_i)) = R(l(w_i \circ r)).$$

Then, R' has the uniform distribution (since there is a bijection between choices for R and R'), and we claim that the emulated adversary cannot distinguish whether its random oracle is R or R'.

If the adversary has already queried on both accepted points (and thus the simulator had to restart twice), then $R = R'$. Now suppose A has been restarted exactly once because it made a query that involved w_1. Then, R and R' agree on the values involving w_1 so they only differ on two values. By assumption, this is the last restart so A never makes a query of the form $R(l(w_2 \circ r))$. It remains to show that A never makes the query $R(l(w_2 \circ r))$ or $R^{-1}(y_2)$. The worrisome issue here is that A may possess some auxiliary information on w_2, so we prove the claim in a very strong setting in which we assume that A knows the value w_2 exactly, but just does not bother to query the random oracle on this value.

Lemma 16. *Given $w_1 \in \{0,1\}^n$, let $\mathcal{C}_{w_1}$ be the distribution over obfuscated two-point circuits $C = \bar{\mathcal{O}}_{\mathcal{P}^2}(I_{\{w_1, w_2'\}})$ for $w_2' \in_U \{0,1\}^n$ chosen uniformly. Suppose that there exists a choice of $w_1, w_2 \in \{0,1\}^n$ and a PPT algorithm B such that*

$$\Pr_{C \leftarrow \mathcal{C}_{w_1}, R}[B^{R,R^{-1}}(w_1, w_2, C, y_1) = y_2]$$

is noticeable, where $C = \Psi_{r,l,N,t}$ is chosen from $\mathcal{C}_{w_1}$ and y_1, y_2 are two square roots of $t \mod N$ such that $y_1 \not\equiv \pm y_2 \mod N$ and B never queries its random oracle on $R(l(w_2 \circ r))$. Then, there exists a PPT algorithm F (in the standard model) that can break the factoring assumption.

Proof. Let F be a PPT algorithm that receives as input an $6n+1$-bit integer N chosen from the distribution $D(1^{6n})$ in Definition 14. By nonuniformity, F stores w_1 and w_2. and creates a circuit C' as follows.

```
1: choose $w_2' \leftarrow \{0,1\}^n$, $r \leftarrow \{0,1\}^{5n}$, and $x_1, x_2, y_1 \leftarrow \{0,1\}^{6n}$ independently and uniformly
2: set $t \leftarrow (y_1)^2 \mod N$
3: in the finite field $\mathbb{F}^{(n)}$, choose the line $l$ determined by $l(w_1 \circ r) = x_1$ and $l(w_2' \circ r) = x_2$
4: output the program $\Psi_{r,l,N,t}$
```

Recall that F operates in the standard model, but B requires a random oracle so F constructs one on the fly, beginning with $R(l(w_1 \circ r)) = y_1$ (which is a random constraint because y_1 is chosen from the uniform distribution). It is easy to see that the circuit C' is exactly distributed according to $\mathcal{C}_{w_1}$ subject to this constraint on the random oracle.

Then, F emulates an execution of $B^{R,R^{-1}}(C, w_1, w_2, y_1)$, responding to B's oracle queries randomly and building up a table of previous queries so it can make consistent responses, with the exception that F fails if B ever queries $R(l(w_2 \circ r))$. Note that the oracle is built according to the same distribution as B would face in the actual random oracle model.

When B halts and outputs y_2, F outputs $\gcd\{y_1 - y_2, N\}$. If B succeeds (meaning y_2 really is a square root of t that is not $\pm y_1$), then F outputs a nontrivial factor of N, as desired [13] . $\square$

Returning to the proof of the theorem, the lemma shows that the emulated adversary A cannot distinguish between R and R' in the case that A queried exactly one accepted point w_1. The proof in the case that A does not query any any accepted points is analogous (it uses a weakening of Lemma 16 in which a fake point w_1' is chosen randomly as well). Therefore, the operation of A must be information-theoretically independent of whether the random oracle is R or R'. But if the oracle is R', then the "fake" obfuscation that A receives is actually correct, so the functionality of $P^{R'}$ has the same distribution as the functionality of the output of the real adversary. Furthermore, the construction of Q modifies P in precisely the manner to ensure $Q^R \equiv P^{R'}$.

It remains to prove that the relation E^R cannot distinguish between the real and fake random oracles, even with access to the real random oracle. This follows by an information-theoretic argument identical to the one used in the proof of Theorem 13 (and explained in more detail there). All of the points in which R and R' differ include the string r. Therefore, the relation can only distinguish the two oracles if it knows the value of r. However, $|r| = 5n$, but the relation only receives $4n$ bits of input, so the adversary has only negligible probability of transmitting enough information to the relation for it to reconstruct r. (This holds even if the adversary and relation agree on a list-encoding scheme beforehand.) This completes the proof of functional non-malleability.

Finally, we show that $\bar{\mathcal{O}}_{\mathcal{P}^2}$ is malleable in the verifiable sense, meaning that for every verifier $\bar{V}_{\mathcal{P}^2}$, the pair $(\bar{\mathcal{O}}_{\mathcal{P}^2}, \bar{V}_{\mathcal{P}^2})$ is not verifiably non-malleable. We show this in the case that the auxiliary information $z = \varnothing$ is the empty string and that $E(I_{\{w_1,w_2\}}, I_{\{w_1',w_2'\}})$ is the polynomial time computable relation that accepts if and only if $(w'_1 \circ 0^{4n}) = (w_1 \circ 0^{4n}) + 1$ and $(w'_2 \circ 0^{4n}) = (w_2 \circ 0^{4n}) + 1$, where the $6n$-bit strings are embedded into $\mathbb{F}^{(n)}$ in the obvious way and the additions are performed over this field.

Let c be the embedding of the $6n$-bit string $0^{n-1}10^{4n}$ into the field $\mathbb{F}^{(n)}$. When A is given the program $\Psi_{r,l,N,t}$, it computes the line $l'(x) = l(x - c)$ and returns $\Psi_{r,l',N,t}$. It is easy to check that for almost all $\{w_1, w_2\}$, $\Psi_{r,l',N,t}$ is an obfuscation of the circuit $I_{\{w'_1,w'_2\}}$ satisfying the relation.

Furthermore, the program $\Psi_{r,l',N,t}$ is the output of the obfuscator when it is given the circuit $I_{\{w'_1,w'_2\}}$ and chooses random coins r, t, and N. As a result, any verification algorithm $\bar{V}_{\mathcal{P}^2}$ must accept $\Psi_{r,l',N,t}$.

In order for verifiable non-malleability to hold, there must exist a simulator S such that $S^{R,R^{-1},I_{\{w_1,w_2\}}}(1^n)$ outputs a valid obfuscation of the circuit $I_{\{w'_1,w'_2\}}$. But the circuit family $\mathcal{P}^2$ is unlearnable, so a simulator that is given oracle access to $I_{\{w_1,w_2\}}$ cannot create any circuit that computes $I_{\{w_1,w_2\}}$ for almost all $\{w_1, w_2\}$ (except for a polynomial-sized set in which it guesses w_1 and w_2 correctly). As a result, the simulator cannot make any circuit that computes $I_{\{w'_1,w'_2\}}$ either. Therefore, every simulator only succeeds with negligible probability on almost every $\{w_1, w_2\}$, so the adversary A breaks the verifiable non-malleability property on $(\bar{\mathcal{O}}_{\mathcal{P}^2}, \bar{V}_{\mathcal{P}^2})$. This is true for all verifiers, so the proof is complete. □

General m. Most of $\bar{\mathcal{O}}_{\mathcal{P}^2}$ generalizes immediately to the m-point case. The main issue is with the trapdoor one-way function. In the $m = 2$ case we used the Rabin function because it had a special property that allowed us to prove Lemma 16. This property was needed to prove security even for adversaries that knew some of the w_i.

In the general case, we need a trapdoor family $\mathcal{T}$ of m-to-1 one-way functions with a special property that we call "m^{th} preimage resistance": an adversary that is given a function $f \in_R \mathcal{T}$ and inputs $x_1, \ldots, x_{m-1}$ such that $f(x_1) = \cdots = f(x_{m-1})$ still cannot find the final value x_m such that $f(x_m) = f(x_1)$. Unfortunately, we do not know of any functions with this property. Most m-to-1 one-way functions simply iterate 2-to-1 functions using a hash-tree-like structure, and as a result it is possible to break the m^{th} preimage resistance property.

Note that m^{th} preimage resistance is needed solely in order to prove Lemma 16, which is used in the proof to show that the adversary cannot distinguish the real oracle R from a fake oracle R'. Note that if the adversary does not have auxiliary information, then Lemma 16 is not needed because we can prove the claim directly: if an adversary can distinguish R and R', then it has found one of the values $f^{-1}(t)$ so it has broken the one-way function.

Hence, using any trapdoor family of m-to-1 functions, we can prove that the generalization of $\bar{\mathcal{O}}_{\mathcal{P}^2}$ to the multi-point case satisfies functional non-malleability without auxiliary information. The full construction of the obfuscator $\bar{\mathcal{O}}_{\mathcal{P}^m}$ is given in Algorithm 5.

Algorithm 5 Obfuscator $\bar{\mathcal{O}}_{\mathcal{P}^m}$ for the family of m-point circuits

```
Input: a circuit of the form $I_{w_1,w_2,\ldots,w_m}$
1: extract the points $w_1, w_2, \ldots, w_m$ and randomize their order
2: choose strings $r \leftarrow \{0,1\}^{2mn+n}$ and $t \leftarrow \{0,1\}^{2mn+2n}$ uniformly
3: choose a function-trapdoor pair $(f, \tau) \leftarrow \mathcal{T}$
4: let $y_1, \ldots, y_m$ be the $m$ preimages of $t$ under $f$, and let $x_i \leftarrow R^{-1}(y_i)\ \forall i$
5: in the finite field $\mathbb{F}^{[n]}$, find the unique $m-1$ degree polynomial $q$ such that $q(w_i \circ r) = x_i\ \forall i$
6: output the program that stores $r$, $q$, $f$, and $t$ in a readily identifiable manner and on input x, accepts iff
   $f(R(q(x \circ r))) = t$
```

Note how all the variables generalize from the two-point case.

- The randomness r grows to $2mn + n$ bits so we can still use the information-theoretic argument in the proof that the relation E cannot distinguish between the real and fake oracles. Everything else grows in size appropriately (in particular, $\mathbb{F}^{[n]}$ is the finite field $\mathbb{Z}_p$ for the first prime $p \geq 2^{2mn+2n}$).

- The line l from the two-point case generalizes to the unique degree $m-1$ polynomial that goes through the desired m points.
- The amount of persistent memory required for Q increases, but remains linear in n. Additionally, this memory is still used solely for programming of the random oracle.

The proof of functional non-malleability for $\bar{\mathcal{O}}_{\mathcal{P}^m}$ follows the two-point case exactly, except for the caveat described above, so we do not repeat the proof.

5 Constructions of verifiably non-malleable obfuscators

In this section, we present verifiably non-malleable obfuscators for $\mathcal{P}^1$ and $\mathcal{P}^m$ in the random oracle and common reference string models.

5.1 Random oracle model

In the single-point case, the obfuscator $\mathcal{O}_{\mathcal{P}^1}$ from Algorithm 1 can also be used to create a verifiably non-malleable obfuscation. Let $V_{\mathcal{P}^1}$ be the verification algorithm that checks whether its input is a program of the form $\Phi_{r,t}$ for some r and t such that $|t| = 4/3\,|r|$, and accepts if and only if this is true. It is clear from Algorithm 1 that for any $w \in \{0,1\}^n$, $V_{\mathcal{P}^1}$ always accepts a proper obfuscation of the circuit I_w.

Theorem 17. *In the random oracle model, $(\mathcal{O}_{\mathcal{P}^1}, V_{\mathcal{P}^1})$ is a verifiably non-malleable obfuscation for the family of point circuits $\mathcal{P}^1$.*

Proof. The functionality and polynomial slowdown properties are clear, so it remains to prove verifiable non-malleability. Let A be an adversary, and we construct the simulator S for A as follows. When given auxiliary information z and oracle access to the point circuit I_w, the simulator $S^{R,I_w}(1^n, z)$ forms a fake obfuscation by choosing $r \in_U \{0,1\}^{3n}$ and $t \in_U \{0,1\}^{4n}$ at random. Then, S emulates an execution of $A(\Phi_{r,t}, z)$ while monitoring its random oracle calls. There are three cases to consider:

1. If A queries R on the input $w \circ r$, then the adversary A found the hidden point w, but now the simulator has too. As a result, the simulator can simply form an obfuscation of I_w and run A on it. In this case, the simulation is perfect.
2. If A queries the value $s = R^{-1}(t)$, then A has discovered that our obfuscation is fake. In this case, the simulator returns an abort symbol $\perp$.
3. On any other query q, S allows A to access the random oracle and continue its execution.

At the end of its execution, the emulated adversary outputs a program P. Then, the simulator outputs the same program $Q = P$.

Because the second case only occurs with negligible probability, and the simulation becomes perfect if the first case ever occurs, the interesting scenario is that in which only the third case occurs. In this scenario, the adversary A never queries the oracle on the hidden point w, nor does it discover that the obfuscation is a forgery.

Let E be any relation that only makes polynomially-many oracle queries, and define

$$p_A = \Pr\left[P \leftarrow A^R(\mathcal{O}(I_w), z) : P \neq \mathcal{O}(C), V(P) = 1, \text{ and } \exists I_{w'} \in \mathcal{P}^1 \text{ s.t. } I_{w'} \equiv P^R \text{ and } E^R(I_w, I_{w'}) = 1\right]$$
$$p_S = \Pr\left[Q \leftarrow S^{R,I_w}(1^n, z) : V(Q) = 1 \text{ and } \exists I_{w'} \in \mathcal{P}^1 \text{ s.t. } I_{w'} \equiv Q^{R,I_w} \text{ and } E^R(I_w, I_{w'}) = 1\right]$$

as the probabilities of success for the real adversary and simulator, respectively. We will show that $|p_A - p_S|$ is negligible in n.

Perform the following "mental experiment:" let R' be a different random oracle permutation in which

$$R'(w \circ r) = t, \quad R'(s) = R(w \circ r),$$

and R' agrees with R on all other values. (See Figure 1 for a diagram.) If R' really were the random oracle, then the "fake" obfuscation $\Phi_{r,t}$ would actually be a valid obfuscation of I_w. As a result,

$$p_A = \Pr\left[P \leftarrow A^{R'}(\Phi_{r,t}, z) : P \neq \Phi_{r,t}, V(P) = 1, \text{ and } \exists I_{w'} \in \mathcal{P}^1 \text{ s.t. } I_{w'} \equiv P^{R'} \text{ and } E^{R'}(I_w, I_{w'}) = 1\right].$$

Furthermore, by assumption A never queries the oracle at the two locations in which R and R' differ. Hence, A must act the same in both cases, so the program Q returned by the simulator after its emulation of A^R on the "fake" obfuscation is identical to the program P returned by $A^{R'}$ when given a real obfuscation. Therefore,

$$p_A \approx \Pr\left[Q \leftarrow S^{R,I_w}(1^n, z) : Q \neq \Phi_{r,t}, V(Q) = 1 \text{ and } \exists I_{w'} \in \mathcal{P}^1 \text{ s.t. } I_{w'} \equiv Q^{R'} \text{ and } E^{R'}(I_w, I_{w'}) = 1\right],$$

where now the simulator itself chooses r and t (rather than before, where they were provided to the adversary).

Next, we claim that if $Q \neq \Phi_{r,t}$ $V(Q) = 1$, and $Q^{R'}$ is a point circuit, then Q^R and $Q^{R'}$ have the same functionality. The verification test ensures that Q has the form of a program in $\mathcal{P}^1$, so $Q = \Phi_{r',t'}$ for some r' and t', and Q accepts the single point x such that $x \circ r'$, when applied to its oracle, yields t. Furthermore, the functionality of Q depends solely on the value of the random oracle at $x \circ r'$.

Recall that R and R' differ in only two input values, namely s and $w \circ r$. As a result, Q^R and $Q^{R'}$ have different functionalities if and only if $x \circ r' = s$ or $x \circ r' = w \circ r$.

1. The first case is easy to dismiss: it is infeasible for the simulator to find s, so it cannot output the final $3n$ bits of s as the value of r'.
2. In the second case, $x = w$ and $r' = r$. Because $\Phi_{r',t'} \neq \Phi_{r,t}$ but $r' = r$, it must be the case that $t' \neq t$. As a result, the program $Q^{R'}$ does not accept any points, because there is no $R'(x \circ r') \neq t'$. This contradicts the assumption that $Q^{R'}$ is a point circuit.

As a result, Q^R and $Q^{R'}$ have different functionality with only negligible probability, so it follows that

$$p_A \approx \Pr\left[Q \leftarrow S^{R,I_w}(1^n, z) : V(Q) = 1 \text{ and } \exists I_{w'} \in \mathcal{P}^1 \text{ s.t. } I_{w'} \equiv Q^R \text{ and } E^{R'}(I_w, I_{w'}) = 1\right].$$

Finally, it remains to show that the relation E behaves similarly when given random oracle R or R'. This is an information-theoretic argument that proceeds in exactly the same manner as in the proof of Theorem 13.

Let R'' be another random oracle in which $R''(w \circ r) = t$, and R'' agrees with R on all other values. (See Figure 1 for a diagram.) In other words, R'' and R' differ only on the input s. Because the adversary, simulator, and relation E all run in polynomial time, they only query the value s with negligible probability, and therefore the difference in functionality between $E^{R'}$ and $E^{R''}$ is negligible. Furthermore, R and R''

only differ on input $w \circ r$, which has at least $3n$ bits of entropy, but the relation E only receives $2n$ bits of information as input (namely, the values w and w'). As a result, using any list encoding scheme, the adversary and simulator only have negligible probability of transmitting the value r to the relation E. If the relation does not query the value $w \circ r$, then the behavior of E^R and $E^{R''}$ is identical. As a result,

$$p_A \approx \Pr\left[Q \leftarrow S^{R,I_w}(1^n, z) : V(Q) = 1 \text{ and } \exists I_{w'} \in \mathcal{P}^1 \text{ s.t. } I_{w'} \equiv Q^R \text{ and } E^R(I_w, I_{w'}) = 1\right] = p_S,$$

as desired.

Hence, $(\mathcal{O}_{\mathcal{P}^1}, V_{\mathcal{P}^1})$ is a verifiably non-malleable obfuscator for $\mathcal{P}^1$, and in fact the definition is achieved in a strong sense in which the simulator S is independent of the choice of the polynomial ρ. □

In the multi-point setting, we can concatenate m copies of $\mathcal{O}_{\mathcal{P}^1}$ and "glue" them together using a self-signing technique, as shown in Algorithm 6. The one-time signature scheme can be constructed from any one-way function [11] so in particular it can be constructed from the random oracle. Additionally, we assume the existence of m independent random oracle permutations, but they can easily be constructed from a single one.

Algorithm 6 Obfuscator $\mathcal{O}_{\mathcal{P}^m}$ for the family of m-point circuits

```
Input: a circuit of the form $I_{w_1,\dots,w_m}$
1: extract the accepted points $w_1, \dots, w_m$ and randomize their order
2: set $n \leftarrow |w_1|$
3: choose randomness $r_1, \dots, r_m \leftarrow \{0,1\}^{3mn}$ independently and uniformly
4: choose a signature-verification key pair $(s, v)$ for a one-time signature scheme with security parameter
   $n$
5: for $i = 1$ to $m$ do
6:     set $t_i \leftarrow R_i(w_i \circ r_i \circ v)$
7: end for
8: compute the signature $\sigma = \mathrm{sign}_s(t_1, \dots, t_m)$
9: output circuit $\Gamma_{r_1,\dots,r_m,t_1,\dots,t_m,v,\sigma}$ defined below
```

The circuit $\Gamma_{r_1,\dots,r_m,t_1,\dots,t_m,v,\sigma}$ stores the r_i, t_i, v, and σ in plain sight and computes the following: "on input x, for all i from 1 to m, compute $R_i(x \circ r_i \circ v)$ and accept if this value equals t_i." The associated verification algorithm $V_{\mathcal{P}^m}$ checks the validity of the signature.

Theorem 18. *In the random oracle model, $(\mathcal{O}_{\mathcal{P}^m}, V_{\mathcal{P}^m})$ is a verifiably non-malleable obfuscation for $\mathcal{P}^m$. However, $\mathcal{O}_{\mathcal{P}^m}$ is malleable in the functional sense.*

Proof. It is easy to see that $\mathcal{O}_{\mathcal{P}^m}$ satisfies functionality and polynomial slowdown. To prove verifiable non-malleability, let A be an adversary and we construct a simulator S. Given auxiliary information z and oracle access to a circuit of the form $I_{\{w_1,\dots,w_m\}}$, the simulator $S^{R_1,\dots,R_m,I_{\{w_1,\dots,w_m\}}}(1^n, z)$ creates a "fake" obfuscation by choosing a signature-verification key pair (s, v) from the key generation algorithm of a one-time signature scheme, choosing $r_1, \dots, r_m \in_U \{0,1\}^{3mn}$ and $t_1, \dots, t_m \in_U \{0,1\}^{n+3mn+|v|}$ at random, computing a signature $\sigma = \mathrm{sign}_s(t_1, \dots, t_m)$, and constructing $\Gamma_{r_1,\dots,r_m,t_1,\dots,t_m,v,\sigma}$. Then, the simulator emulates an execution of $A^{R_1,\dots,R_m}(\Gamma_{r_1,\dots,r_m,t_1,\dots,t_m,v,\sigma}, z)$. Whenever A makes a query q to the random oracle R_i, S acts as follows:

1. If $R_i(q) = t_i$, then the adversary has discovered one of our "fake" accepted points. In this case, the simulator fails. However, the t_i are chosen uniformly at random and independently of A or z, so the adversary can only find the inverse of t_i under the random oracle with negligible probability.

2. If the query q is $n + 3mn + |v|$ bits long, then let x denote the first n bits of q. The simulator queries its oracle to find the value $I_{\{w_1,\dots,w_m\}}(x)$. If the oracle returns 0, then S allows A to access the random oracle and find $R(q)$. If the oracle returns 1, then the simulator has found an accepted point. The simulator aborts the current emulation of A, forms a new "fake" obfuscation that uses all of the currently known accepted points along with fake values, and starts an emulation of A on this new program.

3. On any other query q, S allows A to access the random oracle and continue its execution.

Note in particular that the simulator S either responds to A's oracle queries honestly or aborts the emulation of A.

We note that case 2 can happen at most m times, so the simulator will complete the emulation of A in polynomial time. Let P denote the program that the emulation of A outputs. Then, S outputs the same program P.

It remains to analyze the probability of success for this simulator. Although our emulation of A was on a fake obfuscation, note that there is a set of random oracles on which the obfuscation is correct. In particular, let $R'_1, \dots, R'_m$ be random oracles defined as follows:

1. If the simulation found the point w_i, then set $R'_i = R_i$.

2. If the simulation did not find the point w_i, then set

$$R'_i(w_i \circ r_i \circ v) = t_i, \quad R'_i(R^{-1}(t_i)) = R(w_i \circ r_i \circ v),$$

   and set R'_i equal to R_i on all other inputs.

If the relation E and the program P that S outputs have access to the random oracles $R'_1, \dots, R'_m$, then the simulation is perfect. It remains to show that substituting the real random oracles $\{R_i\}$ for the fake $\{R'_i\}$ only has negligible effect on the success probability of the simulator.

As long as the program P passes the verification test, it must have the form $\Gamma_{r'_1,\dots,r'_m,t'_1,\dots,t'_m,v',\sigma'}$ and the signature test must pass. Because the adversary can only forge signatures with negligible probability, it follows that $v \neq v'$ with overwhelming probability. In this case, the differences between the R_i and R'_i random oracles are meaningless for the execution of P, so $P^{R_1,\dots,R_m} \equiv P^{R'_1,\dots,R'_m}$.

Additionally, the relation E receives $2mn$ bits of information, but the randomness values $r_1, \dots, r_m$ each have $3mn$ bits of entropy. As a result, using any list encoding scheme, the adversary and simulator have only negligible probability of transmitting any of the r_i to the relation E. Without r_i, the relation cannot distinguish between R_i and R'_i. Hence, the probability of success for the simulator changes only by a negligible amount when we substitute $E^{R_1,\dots,R_m}$ for $E^{R'_i,\dots,R'_m}$. Therefore, $\mathcal{O}_{\mathcal{P}^m}$ is a verifiably non-malleable obfuscator for $\mathcal{P}^m$, as desired.

Next, we show that $\mathcal{O}_{\mathcal{P}^m}$ is malleable in the functional sense. We do so in the case that the auxiliary input $z = \varnothing$ is the empty string. Let A be the adversary that when given $\Gamma_{r_1,\dots,r_m,t_1,\dots,t_m,v,\sigma}$ as input, chooses a random value $w^* \in_U \{0,1\}^n$, sets $t^* = R_1(w^* \circ r_1 \circ v)$, and outputs the program $\Gamma_{r_1,\dots,r_m,t^*,t_2,\dots,t_m,v,\sigma}$. Note the following facts about A.

1. The signature σ is now incorrect, but in the functional setting this is okay because nobody checks the signature.

2. If the input $\Gamma_{r_1,\dots,r_m,t_1,\dots,t_m,v,\sigma}$ was the obfuscation of the m-point circuit $I_{\{w_1,\dots,w_m\}}$, then the output of A accepts $m-1$ of the points in $\{w_1, \dots, w_m\}$. However, one of the accepted values is removed and replaced by a random point w^*.

Let E be the polynomial time computable relation such that

$$E(I_{\{w_1,\ldots,w_m\}}, I_{\{w'_1,\ldots,w'_m\}}) = 1 \iff \{w_1,\ldots,w_m\} \cap \{w'_1,\ldots,w'_m\} = m-1.$$

Then, A outputs a program that passes this relation with overwhelming probability: the only way that A fails is by choosing $t^* = t_1$ so its output circuit is equal to its input circuit, but t^* is chosen randomly so this only occurs with negligible probability. However, any simulator $S^{R_1,\ldots,R_m,I_{\{w_1,\ldots,w_m\}}}(1^n)$ only has oracle access to $I_{\{w_1,\ldots,w_m\}}$, and the only hope that the simulator has to create a program of the form $I_{\{w^*,w_2,\ldots,w_m\}}$ is to guess points at random and hope that it finds an accepted point w_i which it can then replace. Therefore, every simulator succeeds with only negligible probability, so the adversary A breaks the functional non-malleability property as desired. $\square$

5.2 Common reference string model

In the common reference string model, we can only prove a slightly weaker form of verifiable non-malleability. We first present an obfuscator in the single-point setting, where we believe the weaker non-malleability property is meaningful. Then, we generalize the obfuscator to operate in the multi-point setting, but we also show that the weaker form of non-malleability is insufficient in this setting.

Single-point circuits. Our construction uses two building blocks:

1. Let $\hat{\mathcal{O}}_{\mathcal{P}^1}$ be any obfuscator for $\mathcal{P}^1$ without auxiliary information, such as those found in [2, 5, 17].
2. Let Π be a non-malleable NIZK proof of knowledge system [14, 15]. Informally, the proof system Π has the property that given an adversary that sees a proof π and then creates a proof π' with $\pi' \neq \pi$, there exists an extractor that extracts the witness to the proof of π'.

Using these building blocks, we form the obfuscator $\tilde{\mathcal{O}}_{\mathcal{P}^1}(I_w)$ that outputs $\hat{\mathcal{O}}_{\mathcal{P}^1}(I_w)$ along with a proof that it knows the point w that the obfuscation hides. The verification algorithm $\tilde{V}_{\mathcal{P}^1}$ associated to $\tilde{\mathcal{O}}_{\mathcal{P}^1}$ runs the verification algorithm of the proof system Π to check the validity of the proof.

More formally, we define an NP relation $R_{\hat{\mathcal{O}}_{\mathcal{P}^1}}$ based on $\hat{\mathcal{O}}_{\mathcal{P}^1}$ as follows:

$$R_{\hat{\mathcal{O}}_{\mathcal{P}^1}}(P, w) = 1 \text{ if and only if } \exists r \text{ s.t. } P = \hat{\mathcal{O}}_{\mathcal{P}^1}(I_w, r).$$

That is, the first input to the relation must be a valid output of the obfuscator $\hat{\mathcal{O}}_{\mathcal{P}^1}$, and the second input must be the unique point that is accepted by this circuit. The obfuscator $\tilde{\mathcal{O}}_{\mathcal{P}^1}$, described in Algorithm 7, uses a non-malleable NIZK proof of knowledge system [15] on this NP relation.

Algorithm 7 Obfuscator $\tilde{\mathcal{O}}_{\mathcal{P}^1}$ for the class of point circuits in the CRS model

```
Input: a circuit of the form $I_w$ and a common reference string $\Sigma$
1: extract the accepted point $w$
2: choose randomness $r$ and compute the circuit $\hat{I}_w = \hat{\mathcal{O}}_{\mathcal{P}^1}(I_w)$
3: use the NIZK proof of knowledge $\Pi$ to prove that the obfuscator knows a witness $w$ to the statement
   that $R_{\hat{\mathcal{O}}_{\mathcal{P}^1}}(\hat{I}_w, w) = 1$, and call the resulting proof $\pi_w$
4: output the circuit $\tilde{I}_w$ that is equal to $\hat{I}_w$ except that it also stores $\pi_w$ in some clearly visible way
```

Intuitively, the non-malleability of the obfuscation follows from the non-malleability of the NIZK. Unfortunately, the proof turns out to be quite delicate, and we can only prove a weaker version of verifiable non-malleability.

Definition 19 (Weakly verifiable non-malleability in the common reference string model). Let $\mathcal{C}$ be a family of circuits and $(\mathcal{O}, V)$ be a pair of algorithms. We say that $(\mathcal{O}, V)$ is *weakly verifiably non-malleable for relation* E if for every polynomial q and PPT adversary A, there exists a PPT simulator S such that for all sufficiently large n and for all circuits $C \in \mathcal{C}_n$,

$$\begin{aligned} &|\Pr\left[P \leftarrow A(\mathcal{O}(C), \Sigma) : P \neq \mathcal{O}(C), V(P, \Sigma) = 1, \text{and } \exists D \in \mathcal{C}_n \text{ s.t. } D \equiv P \text{ and } E(C, D) = 1\right] \\ &\qquad - \Pr\left[(Q, \Sigma) \leftarrow S^C(1^n) : V(Q, \Sigma) = 1 \text{ and } \exists D \in \mathcal{C}_n \text{ s.t. } D \equiv Q \text{ and } E(C, D) = 1\right] | < \frac{1}{q(n)}, \end{aligned}$$

where the first probability is taken over the coin tosses of A and $\mathcal{O}$, along with the uniformly random choice of the common reference string Σ, and the second probability is taken over the coin tosses of S.

Note that this definition is weaker than Definition 11 in two ways: the simulator S is allowed to depend on the relation E, and there is no auxiliary information in this definition. We can prove that our construction satisfies this weaker variant of non-malleability for many interesting relations E.

Definition 20 (Invertible relation). A bivariate relation E is *invertible* if there exists a polynomial time algorithm $\bar{E}$ such that for every y, $\bar{E}(y)$ returns a list of all x such that $E(x, y) = 1$.

In particular, because E is a polynomial time algorithm, it can only output a list that is polynomially long in length. Therefore, for every y, there must be only polynomially many x such that $E(x, y) = 1$.

Theorem 21. *Let E be an invertible relation. In the common reference string model, the pair $(\tilde{\mathcal{O}}_{\mathcal{P}^1}, \tilde{V}_{\mathcal{P}^1})$ is a weakly verifiably non-malleable obfuscation for E.*

First, we sketch the proof that $\tilde{\mathcal{O}}_{\mathcal{P}^1}$ is an obfuscation. As usual, functionality and polynomial slowdown are clear, so it remains to prove the virtual black-box property. Note that we are only going to prove a weak form of verifiable non-malleability, and Theorem 8 does not apply for the weaker notion so we have to prove the virtual black-box property directly.

Given an adversary $\tilde{A}$ in the common reference string model that receives a program from $\tilde{\mathcal{O}}_{\mathcal{P}^1}$, we construct an adversary $\hat{A}$ that receives a program from $\hat{\mathcal{O}}_{\mathcal{P}^1}$ as follows. The adversary $\hat{A}$ generates a CRS Σ along with a trapdoor that allows $\hat{A}$ to make false proofs, and then feeds $\tilde{A}$ its input circuit along with a fake proof that it knows the hidden acceptable point. When the emulation of $\tilde{A}$ halts and returns a bit, $\hat{A}$ outputs this bit too. It is easy to see that $\hat{A}$ successfully breaks the virtual black-box property if and only if $\tilde{A}$ does. By the virtual black-box property of $\hat{\mathcal{O}}_{\mathcal{P}^1}$, there exists a simulator $\hat{S}$ corresponding to $\hat{A}$. It follows that $\hat{S}$ also satisfies the virtual black-box property for $\tilde{A}$.

Now we show that $\tilde{\mathcal{O}}_{\mathcal{P}^1}$ satisfies the weaker verifiably non-malleable property for invertible relations E. The proof is inspired by [2] and proceeds in two stages. First, we describe an intermediate property and show that it implies weakly verifiable non-malleability. Second, we prove the intermediate property using specific facts about $\mathcal{P}^1$, $\hat{\mathcal{O}}_{\mathcal{P}^1}$, Π, and the invertibility property of E.

Definition 22 (Almost everywhere non-malleability). Let $\mathcal{C}$ be a circuit family and $(\mathcal{O}, V)$ be a verifiable obfuscation for $\mathcal{C}$. The obfuscation is *almost everywhere non-malleable for relation* E if for every polynomial ρ and PPT adversary A, there exists a family of polynomial size sets $\mathcal{L} = \{L_n\}$ (where $L_n \subseteq \mathcal{C}_n$) such that for sufficiently large n and for all $C \in \mathcal{C}_n \setminus L_n$,

$$\Pr\left[P \leftarrow A(\mathcal{O}(C)) : V(P) = 1 \text{ and } \exists D \in \mathcal{C}_n \text{ s.t. } P \equiv D \text{ and } E(C, D) = 1\right] < \frac{1}{\rho(n)}.$$

That is, any adversary that tries to modify an obfuscated program in accordance with the relation E can only succeed on a polynomial size set of circuits.

Now we prove that this property implies the weaker notion of verifiable non-malleability for many interesting circuit families, such as the family of point circuits $\mathcal{P}^1$.

Lemma 23. *Let $\mathcal{C}$ be a circuit family and $(\mathcal{O}, V)$ be a verifiable obfuscation for $\mathcal{C}$. Suppose $\mathcal{C}$ has the property that given a circuit $C \in \mathcal{C}$ and oracle access to a circuit $D \in \mathcal{C}$, one can test in polynomial time whether $C \equiv D$. Then, almost everywhere non-malleability for relation E implies weakly verifiable non-malleability for relation E.*

Proof. Let E be a relation and assume that $(\mathcal{O}, V)$ is almost everywhere non-malleable for relation E. Given a polynomial ρ and an adversary A, we construct the simulator S necessary to prove weakly verifiable non-malleability for relation E.

By almost everywhere non-malleability, there is a polynomial size family of sets $\mathcal{L}$ associated to A. We construct a simulator S that knows $\mathcal{L}$. (Remember that we work in the non-uniform setting, so this is feasible.) The simulator $S^D(1^n)$ tests whether there exists $C \in L_n$ such that $C \equiv D$. This is feasible because L_n is a polynomial size set and each test can be done in polynomial time by the assumption of the lemma. If one of the tests succeeds, say when testing $C \in L_n$, then S emulates an execution of $A(\mathcal{O}(C))$. Otherwise, the simulator halts and outputs a failure symbol $\perp$.

It remains to analyze the performance of the simulator S. We consider two cases.

1. If S^D finds some $C \in L_n$ such that $C \equiv D$, then it emulates a real execution of A, so the simulation is perfect in this case.
2. Otherwise, then the simulator succeeds with probability zero. However, in this case the real adversary runs $A(\mathcal{O}(D))$ for some D such that $D \notin L_n$. Therefore, the almost everywhere non-malleability property says that the adversary succeeds with at most $\frac{1}{\rho(n)}$ probability.

Hence, in both cases the difference in success probabilities between the adversary and simulator is less than $\frac{1}{\rho(n)}$, as desired. $\square$

Now we demonstrate that the obfuscator $\tilde{\mathcal{O}}_{\mathcal{P}^1}$ satisfies the almost everywhere non-malleability property. Whereas the above proof was generic in nature, now we focus on our specific construction. It is given that $\hat{\mathcal{O}}_{\mathcal{P}^1}$ is an obfuscator for the family of point circuits. As a result, [2] shows that $\hat{\mathcal{O}}_{\mathcal{P}^1}$ has the following property.

Definition 24 (Oracle indistinguishability). For any polynomial ρ and any PPT distinguisher D that outputs a single bit, there exists a polynomial size family $\mathcal{L} = \{L_n\}$ of sets such that for sufficiently large n and for all $I_w, I_{w'} \in \mathcal{P}^1_n \setminus L_n$,

$$\left|\Pr\left[D(\hat{\mathcal{O}}_{\mathcal{P}^1}(I_w)) = 1\right] - \Pr\left[D(\hat{\mathcal{O}}_{\mathcal{P}^1}(I_{w'})) = 1\right]\right| < \frac{1}{q(n)}.$$

We use this property to demonstrate that almost everywhere non-malleability holds.

Lemma 25. *Let E be an invertible relation. Then, $(\tilde{\mathcal{O}}_{\mathcal{P}^1}, \tilde{V}_{\mathcal{P}^1})$ is almost everywhere non-malleable for relation E.*

Proof. Let E be an invertible relation and $\bar{E}$ be the extraction algorithm associated to E. Assume for the sake of contradiction that $(\tilde{\mathcal{O}}_{\mathcal{P}^1}, \tilde{V}_{\mathcal{P}^1})$ is *not* almost everywhere non-malleable for relation E, and we will show a distinguisher that breaks the oracle indistinguishability property of $\hat{\mathcal{O}}_{\mathcal{P}^1}$. Since $\hat{\mathcal{O}}_{\mathcal{P}^1}$ is assumed to be an obfuscator for the family of point circuits, and [2] shows that all such obfuscators obey oracle indistinguishability, we establish a contradiction.

Suppose the adversary A breaks almost everywhere non-malleability for relation E. We use A to construct a distinguisher D. Note that A outputs long strings, but D is only allowed to output a single bit. Distinguisher D operates as follows.

```
1: on input a circuit $P$, form a CRS-trapdoor pair $(\Sigma, \tau)$
2: use the trapdoor $\tau$ to create a false proof $\pi$ claiming you know a witness $w$ such that $R_{\hat{O}_{\mathcal{P}^1}}(P, w) = 1$
3: set $P'$ to be the circuit that is equal to $P$ except that it also stores $\pi$ in some clearly visible way (the
   same way that the obfuscator $\tilde{O}_{\mathcal{P}^1}$ does in Algorithm 7)
4: emulate $A(P')$ with common reference string $\Sigma$, and set $Q$ to be the output of the emulation of $A$
5: if $\tilde{V}_{\mathcal{P}^1}(Q) = 1$ then
6:     set $\pi'$ to be the proof embedded in $Q$ (there is such a proof because the verifier checks for its existence)
7:     use the proof system's knowledge extractor to extract the witness $w'$ in the proof $\pi'$
8:     set $S \leftarrow \bar{E}(w')$ to be the set of all $x$ such that $E(x, w) = 1$
9:     for all $x \in S$ do
10:        if $P(x) = 1$ then
11:            return the first bit of $x$
12:        end if
13:    end for
14: end if
15: return a uniformly random bit
```

This distinguisher runs in polynomial time because A, $\tilde{V}_{\mathcal{P}^1}$, $\bar{E}$, and the proof system's knowledge extractor are all PPT algorithms. Now we analyze the distinguisher's output when it is given $P = \hat{\mathcal{O}}_{\mathcal{P}^1}(I_w)$ as input. We consider two cases:

1. If the adversary A succeeds in creating a program P' such that there exists w' where P' looks like a valid obfuscation of $I_{w'}$ and $E(I_w, I_{w'}) = 1$, then the distinguisher correctly finds w and outputs the first bit of w.

2. Otherwise, the distinguisher outputs a random bit.

By assumption, A breaks almost everywhere non-malleability for relation E, so there exists a polynomial ρ such that for every family of polynomial size sets $\{L_n\}$, there exist infinitely many n and a circuit $I_w \in \mathcal{P}_n^1 \setminus L_n$ such that

$$\Pr\left[P' \leftarrow A(\mathcal{O}(I_w)) : V(P') = 1 \text{ and } \exists I_{w'} \text{ s.t. } P' \equiv I_{w'} \text{ and } E(I_w, I_{w'}) = 1\right] > \frac{1}{\rho(n)}.$$

By our analysis, if the distinguisher is given an obfuscation of this special circuit I_w, case 1 occurs with probability at least $\frac{1}{\rho(n)}$, so the distinguisher correctly outputs the first bit of w with probability at least $\frac{1}{2} + \frac{\rho(n)}{2}$.

Alternatively, suppose the distinguisher is given the obfuscation of any circuit $I_{\tilde{w}} \in \mathcal{P}_n^1 \setminus L_n$ such that the first bit of $\tilde{w}$ is different from the first bit of w. We no longer know how often case 1 occurs and how often case 2 occurs, but in either case the distinguisher will output the first bit of $\tilde{w}$ with probability at least $\frac{1}{2}$. This is enough to complete the proof.

Set the polynomial $\rho' = \frac{\rho}{2}$, and we have shown the following: for every family of polynomial size sets $\{L_n\}$, there exist infinitely many n and some $I_w, I_{\tilde{w}} \in \mathcal{P}_n^1 \setminus L_n$ such that

$$\Pr\left[D(\mathcal{O}(I_w)) = \text{first bit of } w\right] \geq \frac{1}{2} + \rho'(n)$$

and

$$\Pr\left[D(\mathcal{O}(I_{\tilde{w}})) = \text{first bit of } \tilde{w}\right] \geq \frac{1}{2}.$$

But the first bits of w and $\tilde{w}$ are different, so it follows that

$$|\Pr[D(\mathcal{O}(I_w)) = 1] - \Pr[D(\mathcal{O}(I_{w'})) = 1]| \geq \frac{1}{\rho'(n)},$$

so the distinguisher D and polynomial ρ' break oracle indistinguishability. □

Finally, the composition of Lemmas 23 and 25 yield Theorem 21, as desired.

Multi-point circuits. The obfuscator $\tilde{\mathcal{O}}_{\mathcal{P}^1}$ can be generalized to the multi-point setting, as follows. Let $\tilde{\mathcal{O}}_{\mathcal{P}^m}$ be the obfuscator that, when given $I_{\{w_1,\dots,w_m\}}$ as input, outputs m single point obfuscations $\hat{\mathcal{O}}_{\mathcal{P}^1}(I_{w_1})$, ..., $\hat{\mathcal{O}}_{\mathcal{P}^1}(I_{w_m})$ followed by a non-malleable NIZK proof of knowledge that it knows all of the accepted points $w_1, \dots, w_m$. As before, let $\tilde{V}_{\mathcal{P}^m}$ be the verification algorithm that checks the validity of the proof. It is shown in [4] that $\tilde{\mathcal{O}}_{\mathcal{P}^m}$ is an obfuscator. Using a proof similar to that in Theorem 21, we show that this obfuscator is weakly verifiably non-malleable for invertible relations.

Theorem 26. *Let E be an invertible relation. In the common reference string model, $(\tilde{\mathcal{O}}_{\mathcal{P}^m}, \tilde{V}_{\mathcal{P}^m})$ is a weakly verifiably non-malleable obfuscation for E.*

Proof. The conditions of Lemma 23 hold in the multi-point setting, so it suffices to prove that $(\tilde{\mathcal{O}}_{\mathcal{P}^m}, \tilde{V}_{\mathcal{P}^m})$ is almost everywhere non-malleable for E. Suppose for the sake of contradiction that the adversary A breaks almost everywhere non-malleability for E. Then, we construct a distinguisher that breaks the oracle indistinguishability property for $\hat{\mathcal{O}}_{\mathcal{P}^1}$. The distinguisher D uses the proof system's knowledge extractor K and the extraction algorithm $\bar{E}$ associated with E. The distinguisher operates as follows, when it receives the obfuscation of a *single* point circuit as input.

```
1: on input a circuit $P$, set $n$ to be the input length $P$
2: form a CRS-trapdoor pair $(\Sigma, \tau)$, and choose $w_1, \dots, w_{m-1} \leftarrow \{0,1\}^n$ uniformly at random
3: for $i = 1$ to $m - 1$ do
4:     form the obfuscation $\hat{O}_{\mathcal{P}^1}(I_{w_i})$ using fresh randomness
5: end for
6: let $P'$ be the concatenation of $P$ with the $m - 1$ obfuscations $\hat{O}_{\mathcal{P}^1}(I_{w_i})$ in a random order
7: create a false proof $\pi$ claiming you know the $m$ points that $P'$ accepts, and append this proof to $P'$
8: emulate $A(P')$ with common reference string $\Sigma$, and set $Q$ to be the output of the emulation of $A$
9: if $\tilde{V}_{\mathcal{P}^m}(Q) = 1$ then
10:    set $\pi'$ to be the proof embedded in $Q$
11:    run $K(\pi')$ to obtain the witness $w'_1, \dots, w'_m$ consisting of the $m$ accepted points in $Q$
12:    initialize $S$ to be the empty set, and compute $\bar{E}(w'_1, \dots, w'_m)$
13:    for all $I_{w_1,\dots,w_m}$ such that $E(I_{w_1,\dots,w_m}, I_{w'_1,\dots,w'_m}) = 1$, as found by $\bar{E}$ do
14:        insert $w_1, \dots, w_m \in S$
15:    end for
16:    for all $x \in S$ do
17:        if $P(x) = 1$ then
18:            return the first bit of $x$
19:        end if
20:    end for
21: end if
22: return a uniformly random bit
```

The adversary succeeds with non-negligible probability, so using the same analysis as in the proof of Lemma 25, it follows that the distinguisher $D(\mathcal{O}(I_x))$ has a noticeable advantage in finding the first bit of x for super-polynomially many x, so D breaks oracle indistinguishability as desired. □

Unfortunately, in the multi-point setting, the set of invertible relations is too small. For example, the simple relation $E(I_{\{w_1,\dots,w_m\}}, I_{\{w'_1,\dots,w'_m\}})$ that accepts if any of the w_i equal one of the w'_j is not invertible. As a result, Theorem 26 is a promising result but still unsatisfactory. Future research is needed to find an obfuscator that is verifiably non-malleable for a wider class of relations.

Acknowledgment. We thank Ronny Dakdouk for his useful comments. In particular, an improvement in Theorem 8 and an error in an earlier version of Theorem 17 were found by Ronny.

References

[1] Boaz Barak, Oded Goldreich, Russell Impagliazzo, Steven Rudich, Amit Sahai, Salil Vadhan, and Ke Yang. On the (im)possibility of obfuscating programs. In *Proceedings of CRYPTO 2001*. Springer LNCS 2139:1–18, 2001.

[2] Ran Canetti. Towards realizing random oracles: Hash functions that hide all partial information. In *Proceedings of Crypto 1997*. Springer LNCS 1294:455–469, 1997.

[3] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *FOCS*, pages 136–145, 2001.

[4] Ran Canetti and Ronny Ramzi Dakdouk. Obfuscating point functions with multibit output. In *EUROCRYPT*, pages 489–508, 2008.

[5] Ran Canetti, Daniele Micciancio, and Omer Reingold. Perfectly one-way probabilistic hash functions. In *Proceedings of the 30th ACM Symposium on Theory of Computing*, pages 131–140, 1998.

[6] Jean-Sébastien Coron, Jacques Patarin, and Yannick Seurin. The random oracle model and the ideal cipher model are equivalent. In David Wagner, editor, *CRYPTO*, volume 5157 of *Lecture Notes in Computer Science*, pages 1–20. Springer, 2008.

[7] Shafi Goldwasser and Yael Tauman Kalai. On the impossibility of obfuscation with auxiliary input. In *FOCS 2005: 46th Annual IEEE Symposium on Foundations of Computer Science*, Pittsburgh, PA, October 2005.

[8] Shafi Goldwasser and Guy N. Rothblum. On best-possible obfuscation. In Vadhan [16], pages 194–213.

[9] Dennis Hofheinz, John Malone-Lee, and Martijn Stam. Obfuscation for cryptographic purposes. In Vadhan [16], pages 214–232.

[10] Susan Hohenberger, Guy N. Rothblum, Abhi Shelat, and Vinod Vaikuntanathan. Securely obfuscating re-encryption. In Vadhan [16], pages 233–252.

[11] Leslie Lamport. Constructing digital signatures from a one-way function. Technical Report SRI-CSL-98, SRI International Computer Science Laboratory, 1979.

[12] Ben Lynn, Manoj Prabhakaran, and Amit Sahai. Positive results and techniques for obfuscation. In *EUROCRYPT*, volume 3027 of *Lecture Notes in Computer Science*, pages 20–39. Springer, 2004.

[13] Michael Rabin. Digitalized signatures and public-key functions as intractable as factorization. MIT Laboratory for Computer Science, 1979.

[14] Amit Sahai. Non-malleable non-interactive zero knowledge and adaptive chosen-ciphertext security. In *FOCS*, pages 543–553, 1999.

[15] Alfredo De Santis, Giovanni Di Crescenzo, Rafail Ostrovsky, Giuseppe Persiano, and Amit Sahai. Robust non-interactive zero knowledge. In *Proceedings of CRYPTO 2001*. Springer LNCS 2139:566–598, 2001.

[16] Salil P. Vadhan, editor. *Theory of Cryptography, 4th Theory of Cryptography Conference, TCC 2007, Amsterdam, The Netherlands, February 21-24, 2007, Proceedings*, volume 4392 of *Lecture Notes in Computer Science*. Springer, 2007.

[17] Hoeteck Wee. On obfuscating point functions. In *Proceedings of the 37th ACM Symposium on Theory of Computing*, pages 523–532, 2005.