Non-malleable Obfuscation

Ran Canetti[1,⋆] and Mayank Varia[2,⋆⋆]

[1] School of Computer Science, Tel Aviv University
canetti@cs.tau.ac.il
[2] Massachusetts Institute of Technology
varia@csail.mit.edu

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.

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 [2,3]. 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 [4,5,6,7]. 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 [8]. Finally, a different definition of obfuscation has been formulated [9,10] in which it is possible to obfuscate the family of re-encryption programs [10].

The original version of this chapter was revised: The copyright line was incorrect. This has been corrected. The Erratum to this chapter is available at DOI: 10.1007/978-3-642-00457-5_36

⋆ Supported by NSF grant 0635297, US-Israel Binational Science Foundation grant 20006317, a European Union Marie Curie grant, and the Check Point Institute for Information Security.

⋆⋆ Supported by the Department of Defense through the NDSEG Program and by NSF grant 0635297.

However, the question of malleability attacks on obfuscated programs 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, 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, the virtual black-box property 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 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. Consider the adversary that outputs its input. Then, a corresponding simulator has oracle access to a program and needs to write the code for this program, which is usually impossible.

In this paper, we demonstrate two different methods 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 Section 3. 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 be able to read the program's 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 alleviate 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 Bob's access to the computer. There exist obfuscators of the three-point circuit such that Alice can succeed in this attack with noticeable probability [8].

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 she can only remove Bob's access to the computer with negligible probability. In this sense, functional non-malleability provides stronger security for Bob because it protects all aspects of his access to the computer, whereas the virtual black-box property only protects his password.

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. Therefore, we allow 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 office-mates 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 a 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. Informally, a *verifier* algorithm V accepts programs if and only if they could have been produced by running the obfuscation algorithm. 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 in the output. As a result, all known impossibility results regarding the virtual black-box property continue to hold for both types of non-malleability [1,2].

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 [6] 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 [8] 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 accepted 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 (CRS) 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 [11,12] 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 [4], 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 CRS 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 [1,2]. In Section 3, we provide rigorous definitions of the two notions of non-malleability. In Sections 4 and 5, we present non-malleable obfuscators of both flavors for the family of multi-point circuits.

2 Obfuscation

In [1], [2], 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 interested in obfuscation with dependent auxiliary information, as defined in [2].

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

Definition 1 (Obfuscation). *Let $\mathcal{C} = \{\mathcal{C}_n\}_{n\in\mathbb{N}}$ be a family of polynomial-size circuits, where $\mathcal{C}_n$ denotes all circuits of input length n. A probabilistic polynomial time (PPT) 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 $x \in \{0,1\}^n$,*

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

  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 $r \in \mathcal{R}_n$, the description length $|\mathcal{O}(C,r)| \le p(|C|)$.*
- Virtual black-box: *For every polynomial ρ 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 $z \in \{0,1\}^*$,*

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

  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 Defining Non-malleable Obfuscation

In this section, we rigorously define 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 [1,7]. 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.

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 $r \in \mathcal{R}_n$, 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 auxiliary information $z \in \{0,1\}^*$, and for all polynomial time computable relations $E : \mathcal{C}_n \times \mathcal{D}_n \to \{0,1\}$ (that may depend on the circuit C),*

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

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 say that $\mathcal{O}$ is a functionally non-malleable obfuscator for *$\mathcal{C}$.*

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 note 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 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 relation $E(I_x, \mathcal{O}(I_y))$ 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 must fail half 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 $w_1, w_2, \ldots, w_m \in \{0,1\}^n$, 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. Note that the w_i need not be distinct, so the circuit $I_{\{w_1,\ldots,w_m\}}$ accepts between 1 and m points. For technical reasons, we may also want to consider the circuit $I_\emptyset$ that immediately rejects all inputs. Let

$$\mathcal{P}_n^m = \{I_{\{w_1,\ldots,w_m\}} : w_1, \ldots, w_m \in \{0,1\}^n\} \cup \{I_\emptyset\}$$

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. Also, let $\mathcal{P}^{m+}$ be the subfamily that does not include $I_\emptyset$.

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, perhaps $\mathcal{C}$ 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 circuit 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 a functionally non-malleable obfuscator when $\mathcal{D}$ is the family of all circuits, but unfortunately this is impossible. 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. A formal proof can be found in the full version of this paper [13].

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. We show that the functional non-malleability property implies the virtual black-box property (at least for reasonable choices of the circuit family $\mathcal{D}$). This justifies our terminology of using the word “obfuscation” in Definition 3.

Theorem 4. *Let $\mathcal{C}$ and $\mathcal{D}$ be circuit families, and let $\mathcal{O}$ be an obfuscator for $\mathcal{C}$ that is functionally non-malleable over $\mathcal{D}$. Furthermore, suppose that for sufficiently large n, there exist circuits $D_0, D_1 \in \mathcal{D}_n$ such that $D_0 \not\equiv D_1$. Then, $\mathcal{O}$ satisfies the virtual black-box property. As a result, $\mathcal{O}$ is an obfuscator for $\mathcal{C}$.*

This theorem also holds if neither the virtual black-box property nor functional non-malleability allows auxiliary information.

Intuitively, this theorem holds because an adversary that outputs programs can use this channel to transmit a single bit of information b by outputting the program D_b. See the full version of this paper [13] for a rigorous proof.

One consequence of this theorem is that all impossibility results pertaining to the virtual black-box property immediately carry over to the non-malleability setting [1,2].

3.2 Verifiably Non-malleable Obfuscation

In this section, we develop the notion of verifiable obfuscation and use it to define another definition of non-malleability.

Definition 5 (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 $(\mathcal{O}, \mathbb{1})$ 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 [4] 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. This results in a *perfect verifier* that accepts its input program if and only if it has the form of a program produced by the obfuscator.

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 her output 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 6 (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 $\mathcal{C}$. Additionally, for every n and every circuit P with n bits of input such that $V(P) = 1$, there exists $D \in \mathcal{D}_n$ such that $P \equiv D$.*
- 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 $r \in \mathcal{R}_n$, 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 auxiliary information $z \in \{0,1\}^*$, and for all polynomial time computable relations $E : \mathcal{C}_n \times \mathcal{D}_n \to \{0,1\}$,*

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

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

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

It is potentially reasonable to relax the definition by not requiring the simulator to pass the verification text. We choose not to do so because the current definition puts the adversary and simulator on more equal footing and because all of our constructions satisfy the stronger notion.

We also note that the definition requires that V only accept circuits that are equivalent to members of $\mathcal{D}$. The benefit of this restriction is that the adversary can efficiently test whether she outputs a circuit in $\mathcal{D}_n$, which is a requirement for her to succeed under this definition.

Additionally, the remarks pertaining to functional non-malleability also apply here:

1. Restricting to univariate relations $E(D)$ results in an equivalent definition.
2. It is usually impossible to achieve verifiable non-malleability if $\mathcal{D}$ is the family of all circuits.
3. Verifiable non-malleability also implies the virtual black-box property.

Theorem 7. *Let $\mathcal{C}$ and $\mathcal{D}$ be circuit families, and let $(\mathcal{O}, V)$ be an obfuscator for $\mathcal{C}$ that is verifiably non-malleable over $\mathcal{D}$. Then, $\mathcal{O}$ is an obfuscator for $\mathcal{C}$. This theorem also holds if neither definition allows auxiliary information.*

3.3 Comparison

We conclude this section by showing that the two non-malleability definitions are incomparable. Intuitively, functional non-malleability *prevents* more attacks. This is reflected in the definition by the fact that an adversary attacking functional non-malleability does not have to pass a verification test, so the simulator must emulate more potential attacks. A concrete example of this separation is the obfuscator described in Algorithm 2 below, which does not prevent the attack described in the Introduction in which Alice removes Bob's password.

On the other hand, verifiable non-malleability *detects* more attacks. This is reflected in the definitions by the fact that the program Q constructed by the simulator is not given oracle access to C in the verifiable definition. A concrete example of this separation is the functionally non-malleable obfuscator for multi-point functions described in the full version of the paper [13], which is vulnerable to a slightly modified form of Alice's April fools' prank described in the Introduction.

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. Algorithm 1 describes an obfuscator $\mathcal{O}_{\mathcal{P}^1}$ for the family $\mathcal{P}^1$ [6]. Informally, $\mathcal{O}_{\mathcal{P}^1}$ obfuscates the point circuit I_w by recording $R(w)$. This provides an information-theoretic way to hide the point w while still making it easy to check whether the input to the obfuscated program is w. However, the obfuscator should not be deterministic [4], so $\mathcal{O}_{\mathcal{P}^1}$ uses some randomness as well.

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

```
Input: a circuit of the form $I_w$ or $I_\emptyset$
1: if the input circuit is $I_\emptyset$ then
2:     choose $r \overset{U}{\leftarrow} \{0,1\}^{3|w|}$ and $t \overset{U}{\leftarrow} \{0,1\}^{4|w|}$ (that is, uniformly at random)
3: else
4:     extract the point $w$ and choose randomness $r \overset{U}{\leftarrow} \{0,1\}^{3|w|}$
5:     set $t = R(w \circ r)$, where $\circ$ denotes the string concatenation operation
6: end if
7: output the circuit $\Phi_{r,t}$ that stores $r$ and $t$ in some clearly identifiable manner, and
   on input a string $x$, outputs 1 if $R(x \circ r) = t$ and 0 otherwise
```

Theorem 8. *In the random oracle model, the algorithm $\mathcal{O}_{\mathcal{P}^1}$ is a functionally non-malleable obfuscator for the family of point circuits $\mathcal{P}^1$.*

Due to space constraints, rigorous proofs of non-malleability for all of our constructions are deferred to the full version of this paper [13].

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 circuit (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 obfuscated 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. Our construction is described in the full version of this paper [13].

5 Constructions of Verifiably Non-malleable Obfuscators

In this section, we present verifiably non-malleable obfuscators for the family of multi-point circuits 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 accepts if and only if its input is a program of the form $\Phi_{r,t}$. It is clear from Algorithm 1 that $V_{\mathcal{P}^1}$ always accepts proper obfuscations of point circuits.

Theorem 9. *In the random oracle model, $(\mathcal{O}_{\mathcal{P}^1}, V_{\mathcal{P}^1})$ is a verifiably non-malleable obfuscator for $\mathcal{P}^1$.*

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 in order to construct the obfuscator $\mathcal{O}_{\mathcal{P}^m}$

described in Algorithm 2. The associated verification algorithm $V_{\mathcal{P}^m}$ checks that its input program has the proper structure and validates the signature.

The self-signing technique ensures that an adversary will be detected if she tries to re-use any of the $\mathcal{O}_{\mathcal{P}^1}$ obfuscations given to her, because she will not be able to forge the required signature. For instance, using the example described in the Introduction, Alice cannot keep the pieces of the obfuscated circuit that accept Charles and herself but remove the part that accepts Bob. However, the scheme does not *prevent* Alice from implementing this attack, so the scheme is malleable in the functional sense.

The one-time signature scheme can be constructed from any one-way function [14] so in particular it can be constructed from the random oracle.

Algorithm 2. Obfuscator $\mathcal{O}_{\mathcal{P}^m}$ for the family of m-point circuits

```
Input: a circuit of the form $I_{\{w_1,\ldots,w_m\}}$ or $I_\emptyset$
 1: let $k$ be the number of distinct accepted points for the input circuit (possibly 0)
 2: extract the $k$ distinct points $w_1, \ldots, w_k$
 3: choose randomness $r_1, \ldots, r_m \overset{U}{\leftarrow} \{0,1\}^{3mn}$
 4: choose a signature-verification key pair $(s, v)$ for a one-time signature scheme
 5: for $i = 1$ to $k$ do
 6:    set $t_i = R(w_i \circ r_i \circ v)$
 7: end for
 8: for $i = k + 1$ to $m$ do
 9:    choose $t_i \overset{U}{\leftarrow} \{0,1\}^{n+3mn+|v|}$
10: end for
11: choose a random permutation $\pi$ on $m$ elements, and permute the $r_i$ and $t_i$ by $\pi$
12: compute the signature $\sigma = \mathrm{sign}_s(t_1, \ldots, t_m)$
13: output the circuit that stores the $r_i$, $t_i$, $v$, and $\sigma$ in a clearly identifiable manner,
    and on input $x$ does the following: "for $i$ from 1 to $m$, accept if $R(x \circ r_i \circ v) = t_i$"
```

Theorem 10. *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.*

The self-signing technique can also be applied to the functionally non-malleable obfuscator for the family of multi-point circuits described in the full version of the paper [13], producing an obfuscator that simultaneously satisfies both forms of non-malleability.

5.2 Common Reference String Model

In the common reference string (CRS) model, we provide verifiably non-malleable obfuscators for the family $\mathcal{P}^{m+}$ of multi-point circuits that does not include $I_\emptyset$. This is a slightly different family than we used in the random oracle constructions, because the constructions in this section have the property that it is easy to tell that obfuscated circuits accept at least one input, which was not the case in the random oracle constructions.

Also, in this section 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, along with a perfect verifier $\hat{V}_{\mathcal{P}^{1+}}$ for $\hat{\mathcal{O}}_{\mathcal{P}^{1+}}$, such as the obfuscator of [4] described in Section 3.2.
2. Let Π be a non-malleable non-interactive zero-knowledge (NIZK) proof of knowledge system [11,12]. 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. The verification algorithm $\tilde{V}_{\mathcal{P}^{1+}}$ associated to $\tilde{\mathcal{O}}_{\mathcal{P}^{1+}}$ runs $\hat{V}_{\mathcal{P}^{1+}}$ and 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 $\hat{\mathcal{O}}_{\mathcal{P}^{1+}}$ (which can be efficiently verified by $\hat{V}_{\mathcal{P}^{1+}}$), 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 3, uses the NIZK Π on this NP relation.

Algorithm 3. Obfuscator $\tilde{\mathcal{O}}_{\mathcal{P}^{1+}}$ for the family of point circuits in the CRS model

```
Input: a circuit of the form $I_w$ and a common reference string $\Sigma$
1: produce the obfuscated circuit $\hat{I}_w = \hat{\mathcal{O}}_{\mathcal{P}^{1+}}(I_w)$
2: use $\Pi$ 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 $\pi_w$
3: 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 the verifiable non-malleability property.

Definition 11 (Weakly verifiable non-malleability in the CRS 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*

ρ and PPT adversary A, there exists a PPT simulator S such that for all sufficiently large n and for all circuits $C \in \mathcal{C}_n$,

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

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 the verifiable non-malleability property in Definition 6 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 12 (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 13. *Let E be an invertible relation. In the common reference string model, $(\tilde{\mathcal{O}}_{\mathcal{P}^{1+}}, \tilde{V}_{\mathcal{P}^{1+}})$ is a weakly verifiably non-malleable obfuscator for E.*

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 $I_{\{w_1,\ldots,w_m\}}$ as input, outputs the concatenation of 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, ..., w_m. As before, let $\tilde{V}_{\mathcal{P}^{m+}}$ be the verification algorithm that checks the structure of the program and the validity of the proof. It is shown in [8] that $\tilde{\mathcal{O}}_{\mathcal{P}^{m+}}$ is an obfuscator, and we show that this obfuscator is weakly verifiably non-malleable for invertible relations.

Theorem 14. *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 obfuscator for E.*

Unfortunately, in the multi-point setting, the set of invertible relations is too small. For example, the simple relation $E(I_{\{w_1,\ldots,w_m\}}, I_{\{w'_1,\ldots,w'_m\}})$ that accepts if any of the w_i equal any of the w'_j is not invertible. As a result, Theorem 14 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 7 and an error in an earlier proof of Theorem 9 were found by Ronny.

References

1. Barak, B., Goldreich, O., Impagliazzo, R., Rudich, S., Sahai, A., Vadhan, S., Yang, K.: On the (im)possibility of obfuscating programs. In: Kilian, J. (ed.) CRYPTO 2001. LNCS, vol. 2139, pp. 1–18. Springer, Heidelberg (2001)
2. Goldwasser, S., Kalai, Y.T.: On the impossibility of obfuscation with auxiliary input. In: FOCS, pp. 553–562. IEEE Computer Society, Los Alamitos (2005)
3. Goldwasser, S., Rothblum, G.N.: On best-possible obfuscation. In: Vadhan, S.P. (ed.) TCC 2007. LNCS, vol. 4392, pp. 194–213. Springer, Heidelberg (2007)
4. Canetti, R.: Towards realizing random oracles: Hash functions that hide all partial information. In: Kaliski Jr., B.S. (ed.) CRYPTO 1997. LNCS, vol. 1294, pp. 455–469. Springer, Heidelberg (1997)
5. Canetti, R., Micciancio, D., Reingold, O.: Perfectly one-way probabilistic hash functions. In: Proceedings of the 30th ACM Symposium on Theory of Computing, pp. 131–140 (1998)
6. Lynn, B., Prabhakaran, M., Sahai, A.: Positive results and techniques for obfuscation. In: Cachin, C., Camenisch, J.L. (eds.) EUROCRYPT 2004. LNCS, vol. 3027, pp. 20–39. Springer, Heidelberg (2004)
7. Wee, H.: On obfuscating point functions. In: Proceedings of the 37th ACM Symposium on Theory of Computing, pp. 523–532 (2005)
8. Canetti, R., Dakdouk, R.R.: Obfuscating point functions with multibit output. In: Smart, N.P. (ed.) EUROCRYPT 2008. LNCS, vol. 4965, pp. 489–508. Springer, Heidelberg (2008)
9. Hofheinz, D., Malone-Lee, J., Stam, M.: Obfuscation for cryptographic purposes. In: Vadhan, S.P. (ed.) TCC 2007. LNCS, vol. 4392, pp. 214–232. Springer, Heidelberg (2007)
10. Hohenberger, S., Rothblum, G.N., Shelat, A., Vaikuntanathan, V.: Securely obfuscating re-encryption. In: Vadhan, S.P. (ed.) TCC 2007. LNCS, vol. 4392, pp. 233–252. Springer, Heidelberg (2007)
11. Sahai, A.: Non-malleable non-interactive zero knowledge and adaptive chosen-ciphertext security. In: FOCS, pp. 543–553 (1999)
12. De Santis, A., Di Crescenzo, G., Ostrovsky, R., Persiano, G., Sahai, A.: Robust non-interactive zero knowledge. In: Kilian, J. (ed.) CRYPTO 2001. LNCS, vol. 2139, pp. 566–598. Springer, Heidelberg (2001)
13. Canetti, R., Varia, M.: Non-mallable obfuscation. Cryptology ePrint Archive, Report 2008/495 (2008), http://eprint.iacr.org/2008/495
14. Lamport, L.: Constructing digital signatures from a one-way function. Technical Report SRI-CSL-98, SRI International Computer Science Laboratory (1979)