Robust Combiner for Obfuscators

Amir Herzberg[⋆] and Haya Shulman[⋆⋆]

Bar Ilan University
Department of Computer Science
Ramat Gan, 52900, Israel

Abstract. Practical software hardening schemes are heuristic and are not proven to be secure. One technique to enhance security is *robust combiners*. An algorithm C is a robust combiner for specification S, e.g., privacy, if for any two implementations X and Y, of a cryptographic scheme, the combined scheme $C(X,Y)$ satisfies S provided *either* X *or* Y satisfy S.
We present the first robust combiner for software hardening, specifically for obfuscation [2]. Obfuscators are software hardening techniques that are employed to protect execution of programs in remote, hostile environment. Obfuscators protect the code (and secret data) of the program that is sent to the remote host for execution.
Robust combiners are particularly important for software hardening, where there is no standard whose security is established. In addition, robust combiners for software hardening are interesting from software engineering perspective since they introduce new techniques of software only fault tolerance.

Keywords: White-box security, software hardening, obfuscation, robust combiners, fault tolerance, cryptographic protocols.

1 Introduction

Many applications rely on secure execution of programs in untrusted, potentially hostile, environments. *White-box security*, refers to ensuring security of programs running in such untrusted environments. Over the last two decades there is a growing interest in white-box security, in order to enable distributed network applications including on-line software distribution and licensing, mobile agents, grid computing, and others. In white-box security the software is at full control of the platform executing the software. The originator loses all control over her software, which is completely exposed to the hosting environment, and the entity controlling the execution environment obtains full access to the program, and can observe and manipulate the execution, code and data.

White box security stands in contrast to traditional cryptography, which assumes a trusted platform, i.e., a *black-box*, on which secrets, e.g., private keys, can be stored. In black-box security all the computations are performed inside a trusted black-box, and secrets (keys) never leave its boundaries. Attackers can only observe the input/output behaviour, but cannot access the code or data, or observe the execution inside the black-box. To support execution in untrusted environment, this approach requires and relies on an additional tamper-resistant hardware module, e.g., a trusted server as in [1]. In contrast, white box security does not assume a trusted module, and relies on software hardening techniques, rather than depending on (specialized) hardware. In particular, the software is hardened in order to prevent undetected tampering or exposure of secret information, by providing integrity and confidentiality of the execution and of the computations performed.

[⋆] Amir.Herzberg@gmail.com
[⋆⋆] Haya.Shulman@gmail.com

Although provably secure software hardening techniques exist, e.g., [4], they are highly inefficient for practical applications, and due to efficiency considerations, software hardening techniques employed in practice do not rely on provable security. Heuristic implementations are a typical choice in practice, which often gain a reasonable security reputation as a result of failed efforts to cryptanalyse them, and as a result of build-break-fix[1] paradigm. Same approach is also taken in black-box cryptography, e.g., instead of implementing schemes with provable security, cryptanalysis secure standards, such as AES [8], are employed, resulting in efficient and practical implementations. When security of the cryptographic primitive is not proven, robust combiner is a safe choice, to ensure that the overall security of the cryptosystem will be as that of the most secure underlying primitive. In this work we focus on *robust combiners* for software hardening techniques. More specifically, we present a robust combiner for obfuscators, [2], in Section 3. Our approach and constructions may constitute a methodology for future heuristic white-box primitives.

Robust combiners ensure that the scheme is at least as secure as the stronger one of the underlying candidates. Robust combiners are employed for practical constructions to provide security when the security of the underlying primitives is not known, e.g., the primitive is believed to be secure due to failed crypt-analysis. Robust combiners are especially important in white-box security, where mostly heuristic or cryptanalysis secure solutions are employed, since provably secure solutions are inefficient for practical purposes.

Obfuscation is a prevalent software hardening technique used in practice, and can be employed as a building block in higher layer protocols. Obfuscator $\mathcal{O}$ is an efficient algorithm, that when applied on some program P, see Figure 2, produces an obfuscated program P', that has the same functionality as P (for any input, $\mathcal{O}(P)$ produces the same result as P), but its code is harder to understand and analyse. There are many practical constructions of obfuscators, yet none is known to be provably secure, and due to result of Barak *et al.* [2] we cannot hope for a universal obfuscator that would turn the code of every program into one that is hard to understand and reverse engineer.

Obfuscators aim to hide the code of the executing program, and once the program is obfuscated and sent to remote host, there is no further interaction with the originator of the program, and the remote environment can execute the obfuscated program on any input of its choice and observe the output. Obfuscator can be used for software protection and licensing, where a program (implementing some secret, proprietary algorithm) is distributed to users, that can evaluate the program on any input and obtain result. For instance, a program can implement an algorithm that finds prime factors of a given composite number. The user purchases the program and can factor any number of its choice. The goal is to prevent the malicious acts targeted at circumventing software protection to recover the secret algorithm, e.g., by competitors. Since the security of existing obfuscators is not proven, robust combiners is a natural choice to enhance security.

1.1 Our contribution: Robust Combiners for Obfuscators

We present robust combiners for software hardening, specifically for obfuscation [2]. Applying obfuscator on an obfuscated program, i.e., combining obfuscators, is a folklore way to enhance security, yet no formal construction prior to this work was given. In addition, whether combining obfuscators indeed contributes to overall security has been controversial among practitioners. In this work we precisely define cascade combiner for obfuscation, and provide a formal proof of security, i.e., we

[1] A software implementation is published for public scrutiny and undergoes extensive efforts to cryptanalyse it. If a weakness is found, it is being fixed, and the software is tested again for security. Eventually, a software implementation is believed to be cryptanalysis secure, due to failed efforts to cryptanalyse it.

show that cascade is a robust combiner for obfuscation. Our combiner for obfuscators is (1,2)-robust, i.e., it receives two obfuscators $\mathcal{O}'$ and $\mathcal{O}''$, and produces a third obfuscator $\mathcal{O}$ that is secure according to virtual black-box property, if one of the underlying obfuscators is virtual black-box secure. The cascade combiner for obfuscator $\mathcal{O}$, see Figure 1, receives a program P in an input, applies $\mathcal{O}'$ and receives $\mathcal{O}'(P)$. It then provides $\mathcal{O}'(P)$ in an input to obfuscator $\mathcal{O}''$, and returns the resulting obfuscated program $\mathcal{O}''(\mathcal{O}'(P))$.

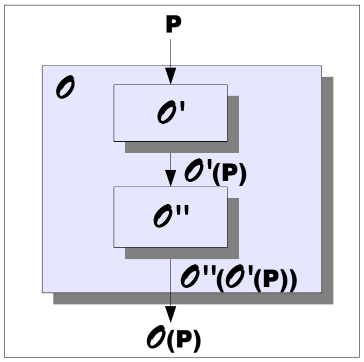

Fig. 1. Cascade obfuscator $\mathcal{O}$ receives a program P, and applies $\mathcal{O}'$ and $\mathcal{O}''$ sequentially. The resulting obfuscated program $\mathcal{O}''(\mathcal{O}'(P))$ is then output.

1.2 Robust Combiners

The security of cryptographic constructions often depends on unproven hardness assumptions, or on the security of primitives that withstood cryptanalysis attacks. A common approach employed to enhance security is to construct robust combiners, by combining two or more cryptographic primitives into one, s.t. the resulting construction is secure even when only some of the candidates are secure. Robust combiners can also be applied for fault tolerance, i.e., to ensure the correctness of the resulting combined scheme and to prevent erroneous implementations or design bugs. Robust combiners for various cryptographic primitives were shown, and alternately, an impossibility of achieving robust constructions for others was presented.

The most well-known combiner is the cascade combiner, which is a sequential application of two cryptographic primitives. Even and Goldreich [9] showed that cascade is a robust combiners of block ciphers, against message recovery attacks. Cascade, and other basic robust combiners, were studied by Herzberg [12], for encryption, *MAC*, signature and commitment schemes. Robust combiners were also studied for other primitives, e.g. hash functions Fischlin and Lehmann [10], Boneh and Boyen [3], private information retrieval (PIR) Meier and Przydatek [14] and oblivious transfer Harnik et al. [11].

Robust combiners are especially important in the context of white-box security, where security of practical candidates is not proven. Furthermore, the existing provably secure white-box primitives are either restricted to a limited class of functions or inefficient and as a result not applicable to practical implementations. Therefore, practitioners have to use heuristic constructions, and currently there isn't even a candidate whose security is sufficiently established; therefore robust combining of candidates is highly desirable.

In principle, robust combiners ensure also fault tolerance: the combined output will meet the spec, even if the code (or hardware) running one mechanism has some fault, and as a result the output of this mechanism does not meet its specifications. However, we caution that many constructions of robust combiners, including the one we present in Section 3, only ensure robustness to the security properties, and assume that some basic functionality is preserved by all modules. In such cases, where functionality is not preserved, the robust combiner does not ensure also fault tolerance (for arbitrary hardware/software faults). We leave it as an open question, whether it is possible to design a robust combiner for obfuscators, which will be tolerant for arbitrary faults (not just faults on security specifications).

1.3 Software Hardening Techniques: Virtual Black-Box Obfuscation

In white-box security the attacker obtains full access to the implementation. This is in contrast to traditional cryptography where a *black-box* (such as a trusted hardware) is assumed to exist, on which secrets can be stored. An attacker cannot access this black-box but can only observe the input-output behavior of the cryptographic implementation, e.g. a server performing signature computations on request. The inherent distinction in the attacker's abilities between the two models implies that traditional cryptographic tools are not applicable to remote environments, since they rely on the fact that the secrets used by the software do not reside on the same execution platform as the malicious host, and are thus not accessible to the attacker. Therefore, alternative tools and techniques have been proposed, with obfuscation being a prevailing technique employed by practitioners to harden software for execution in a remote environment, e.g., [6, 7]. There are practical and theoretical works on obfuscation, including formal definitions and negative and positive results, which we briefly survey below. In [2] *Barak et al.* formalised the notion of obfuscation: an obfuscator $\mathcal{O}(\cdot)$, is a probabilistic polynomial time algorithm, which on an input a program P generates an obfuscated program $\mathcal{O}(P)$, such that $\mathcal{O}(P)$ has the same functionality as P, is at most polynomially slower than P, but leaks no more information about P than a black-box access to P would. Obfuscated program is then sent off to remote platform, see Figure 2. Intuitively this means that the resulting obfuscated program should be harder to reverse engineer and to analyse. That is, the goal of obfuscation is to prevent reverse engineering, and to make it hard to extract information from the binary code. Consider a

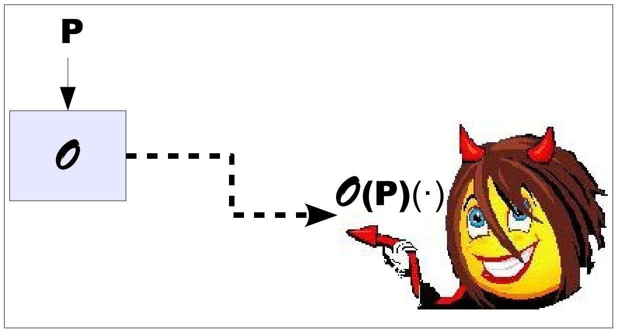

Fig. 2. A program P is obfuscated, and then sent to remote platform. The user controlling the execution environment can execute the obfuscated program on any input of its choice and can observe the outputs. The virtual black-box property requires that the user does not learn more about the program than an oracle access to the program would reveal.

mobile agent purchasing goods on behalf of its originator. The agent purchases best offer based on its internal parameters and signs the purchase with an embedded secret signing key. If the platform hosting the agent could extract the secret key, it could use it to sign any document of its choice. However given a secure obfuscator, it would be possible to obfuscate the agent's code, thus preventing key extraction and analysis of agent's code and data.

The result in [2] also showed that there are programs which cannot be obfuscated. Namely, they showed that there does not exist an obfuscator that can protect every program's code from exposing more than the program's input-output behavior. Thus the goal is to investigate obfuscators for specific function families. Positive results in obfuscation presented constructions of obfuscated functions e.g. [5, 15, 13], which are secure according to [2]. However, none of these works show how to construct an obfuscator that could be applied on programs to harden them, but they instead construct obfuscated programs from 'scratch'. Those results do not suffice for practical definition of 'obfuscation' but they do exhibit a feasibility, paving way to future work in this direction.

As for practical and commercial mechanisms based on obfuscation, those lack a clearly stated security analysis, and rely on security by obscurity, which contradicts basic cryptographic principles, and assume limited patience of the attackers when reverse engineering. Other proposed obfuscations may rely on program analysis that is known to be pragmatically difficult, however, are very susceptible to continuing improvements in program analysis. Unfortunately, to date, no secure practical obfuscation candidates are known to exist, e.g. it is not known how to obfuscate simple programs that upon receipt of an encrypted input, decrypt and execute it, without revealing anything of the inputs or computation.

1.4 Organisation

We first present the definition of obfuscation and virtual black-box obfuscation in Section 2, where we extend the existing virtual black-box definition of [2]. We subsequently define cascade of obfuscators in Section 3.1, and then in Section 3.2 we prove that cascade is robust for obfuscation.

2 Obfuscation: Definitions

We initiate with refined definition of obfuscation, which is based on the definition given by Barak *et al.* in [2], and then present cascade combiner construction for obfuscation and prove its security.

According to [2], universal obfuscator for every program does not exist, therefore we use the definition of obfuscation w.r.t. a family of programs. To simplify exposition, in definitions below we define obfuscation w.r.t. a family of circuits (instead of programs). The difference between circuits and programs, is that circuits are defined for a specific input size, therefore no need to explicitly refer to running time and other technicalities. We next give two definitions of obfuscation: Definition 1 says nothing of security, but only specifies correctness and functionality properties; Definition 2, in addition to correctness and functionality, also defines security of the obfuscator. The distinction between the two definitions, 1 and 2, is essential to emphasise that not every obfuscator delivers on the 'expected' security.

Definition 1 (Obfuscation). *Let $\mathcal{C} = \{\mathcal{C}_k\}$ be a family of polynomial size circuits of input length k. A polynomial time algorithm $\mathcal{O}$ on $\mathcal{C}$ is an obfuscator that takes as input a circuit C from $\mathcal{C}_k$, and outputs an obfuscated circuit $\mathcal{O}(C) \in \mathcal{C}_k$, s.t.:*

- (Preserving Functionality) *For every k, and every $C \in \mathcal{C}_k$, $\mathcal{O}(C)$ is a circuit that computes the same function as C.*
- (Polynomial Slowdown) *Obfuscated circuit $\mathcal{O}(C)$ is roughly as efficient as a circuit C, i.e., there exists a polynomial $p(\cdot)$, such that for sufficiently large k's, for every $C \in \mathcal{C}_k$ holds: $|\mathcal{O}(C)| \leq |p(|C|)|$*

Definition 2 (Virtual Black-Box Obfuscation). *Let $\mathcal{C} = \{\mathcal{C}_k\}$ be a family of polynomial size circuits of input length k. A polynomial time algorithm $\mathcal{O}$ on $\mathcal{C}$ is an obfuscator that takes as input a circuit C from $\mathcal{C}_k$, and outputs an obfuscated circuit $\mathcal{O}(C) \in \mathcal{C}_k$, s.t.:*

- (Preserving Functionality) *For every k, and every $C \in \mathcal{C}_k$, $\mathcal{O}(C)$ is a circuit that computes the same function as C.*
- (Polynomial Slowdown) *Obfuscated circuit $\mathcal{O}(C)$ is roghly as efficient as a circuit C, i.e., there exists a polynomial $p(\cdot)$, such that for sufficiently large k's, for every $C \in \mathcal{C}_k$ holds: $|\mathcal{O}(C)| \leq |p(|C|)|$*
- (Virtual Black-Box) *For every polynomial $p(\cdot)$ and every probabilistic polynomial time algorithm A, there exists a probabilistic polynomial time simulator S, such that for all sufficiently large k's, for all $C \in \mathcal{C}_k$, holds:*

$$\Big| \Pr[A(\mathcal{O}(C)) = 1] - \Pr[S^C(1^k) = 1] \Big| < \frac{1}{p(k)}$$

Where the probabilities are taken over the random coins of A and S.

Several comments related to above definitions are in order:

- The definition presented in [2] is a special case of virtual black-box definition given in 2, where the family of circuits $\mathcal{C}_k$ is defined to be all the circuits.
- Defining two types of obfusctors, one w.r.t. functionality and correctness, and another that is also secure (which is also true in reality, an obfuscator can be correct, but not deliver on its security guarantees) is important for cascade combiner, which we show below, to show that it suffices for only one of the underlying obfuscators, input to the combiner, to be secure according to Definition 2, so that the cascade obfuscator results in a secure obfuscator, according to Definition 2.
- Another subtlety which should be explicitly dealt with when defining obfuscation (and cascade combiner thereof) w.r.t. family of circuits, is that the family of circuits should be closed for obfuscation and for cascade of obfuscators. Namely, upon input a circuit C from family of circuits $\mathcal{C}_k$, the result is an obfuscated circuit $\mathcal{O}(C)$ which is also an element in the family $\mathcal{C}_k$. The obfuscated circuit $\mathcal{O}(C)$ can thus be supplied in an input to obfuscator $\mathcal{O}$ (since $\mathcal{O}$ is defined for $\mathcal{C}_k$) again, and will result in $\mathcal{O}(\mathcal{O}(C))$, which is also an element in $\mathcal{C}_k$. This technicality is motivated by the fact that practical obfuscators are typically written in the same language as the programs on which they are applied.

3 Cascade Combiner for Obfuscators: Construction

In this section we present the first robust cascade combiner for obfuscation. The combiner we construct is natural and folklore, yet was not formally defined and proved secure prior to this work. The construction of the combiner is simple yet the security is not directly implied, and needs to be explicitly proven. In addition, the proof requires our refined definition of obfuscation, given in Section 2.

3.1 Cascade Combiner for Obfuscation: Construction

Definition 3 (Cascade of Obfuscators). *Let $\mathcal{O}'$ and $\mathcal{O}''$ be two obfuscators for circuits family $\mathcal{C}_k$. Their cascade obfuscator $\mathcal{O} = \mathcal{O}' \circ \mathcal{O}''$ is presented in Algorithm 1.*

Algorithm 1 Construction of cascade obfuscator $\mathcal{O}$.On input a circuit $C \in \mathcal{C}_k$, $\mathcal{O}$ first applies $\mathcal{O}'$ on C and obtains $C' \in \mathcal{C}_k$. Then $\mathcal{O}$ runs $\mathcal{O}''$ on C' and as a result receives $C'' \in \mathcal{C}_k$; $\mathcal{O}$ outputs C''.

```
$\mathcal{O}(C)$ {
    $C' \leftarrow \mathcal{O}'(C)$
    $C'' \leftarrow \mathcal{O}''(C')$
    return $C''$
    }
```

Intuitively, the resulting combined obfuscator $\mathcal{O}$ is virtual black-box secure, according to Definition 2, if at least one of the underlying obfuscators, $\mathcal{O}'$ or $\mathcal{O}''$, is virtual black-box secure, i.e., satisfies the properties in Definition 2, and the other is obfuscator according to Definition 1. Assume that $\mathcal{O}'$ is virtual black-box secure, then the circuit is hidden and even if the external $\mathcal{O}''$ exposes it, the security relies on the security of $\mathcal{O}'$. A similar argument holds for $\mathcal{O}''$. We next, in Section 3.2, present a formal and detailed proof of the cascade construction (Algorithm 1), and show that cascade is robust for virtual black-box obfuscation.

3.2 Cascade Combiner is Robust for Obfuscation: Proof

Lemma 1 (Cascade is Robust for Obfuscation) *Let $\mathcal{O}'$ and $\mathcal{O}''$ be two obfuscators for circuits family $\mathcal{C}$, the cascade $\mathcal{O} = \mathcal{O}' \circ \mathcal{O}''$ is a virtual black-box obfuscator for circuits family $\mathcal{C}$, according to Definition 2, if at least one of $\mathcal{O}'$ or $\mathcal{O}''$ is virtual black-box secure for $\mathcal{C}$.*

Proof. In order to show that cascade is robust for obfuscators, we prove that $\mathcal{O}$ satisfies the functionality and the security requirements in Definition 2. We first show that $\mathcal{O}$ is indeed an obfuscator, i.e., satisfies the efficiency and correctness requirements in Definition 1.
Preserving Functionality: Let $C \in \mathcal{C}_k$. Then according to functionality requirement of $\mathcal{O}'$, the obfuscated circuit $\mathcal{O}'(C) = C'$ is also an obfuscator, i.e., $C' \in \mathcal{C}_k$, and has the same functionality as C. Obfuscator $\mathcal{O}''$ also satisfies a functionality requirement, and therefore when applied on C' it produces an obfuscated circuit C'', i.e., $C'' = \mathcal{O}''(C')$, such that C'' is also a circuit $C'' \in \mathcal{C}_k$, and has the same functionality as C'. Since C, C' and C'' all compute the same function, cascade obfuscator preseves the functionality of the original circuit C.

Polynomial Slow Down: Cascade obfuscator $\mathcal{O}$ applies $\mathcal{O}'$ on C, then $\mathcal{O}''$ on $\mathcal{O}'(C)$, and outputs the result. According to definition, there exists a polynomial $p'(\cdot)$ bounding the encoding of the obfuscated circuit $\mathcal{O}'(C)$ (and thus its running time), and there exists a polynomial $p''(\cdot)$, bounding the size of $\mathcal{O}''(\mathcal{O}'(C))$; composition of polynomials $p'(\cdot)$ and $p''(\cdot)$ is a polynomial, i.e., $|\mathcal{O}(C)| \leq p''(|p'(|C|)|)$, thus $\mathcal{O}$ is efficient if $\mathcal{O}'$ and $\mathcal{O}''$ are.

We next show that obfuscator $\mathcal{O}$ is virtual black-box secure, according to Definition 2.

Virtual Black-Box: Assume towards contradiction that there exists a PPT algorithm A and a polynomial $p(\cdot)$, such that for every simulator S holds:

$$\left|\Pr[A(\mathcal{O}(C)) = 1] - \Pr[S^C(1^k) = 1]\right| \geq \frac{1}{p(k)}$$

Given a PPT algorithm A against $\mathcal{O}$ we construct PPT algorithms A' and A'' against $\mathcal{O}'$ and $\mathcal{O}''$ respectively, s.t. for every $C \in \mathcal{C}_k$ holds:

$$\Pr[A'(\mathcal{O}'(C)) = 1] = \Pr[A(\mathcal{O}(C)) = 1] \tag{1}$$

$$\forall C, \exists C',\ s.t.\ \Pr[A''(\mathcal{O}''(C')) = 1] = \Pr[A(\mathcal{O}(C)) = 1] \tag{2}$$

Where C' is a result application of $\mathcal{O}'$ on C. We prove equations (1) and (2) in Claims 2 and 3. We then show, in Claims 4, and 5, that there do not exist simulators S' and S'' for A' and A'' respectively. Namely, given S' (respectively S'') we show how to construct a simulator S for A, thus obtaining:

$$\Pr[S'^C(1^k) = 1] = \Pr[S^C(1^k) = 1] \tag{3}$$

$$\Pr[S''^{C'}(1^k) = 1] = \Pr[S^C(1^k) = 1] \tag{4}$$

However, existence of a simulator S for A contradicts the initial assumption, that there does not exist a simulator for A. More specifically, if there exists an A for which no simulator exists, we can use it to construct A' and A'' against $\mathcal{O}'$ and $\mathcal{O}''$. Thus the advantage of A' and A'' over S' and S'' respectively, is equivalent to the advantage of A over S:

$$\left|\Pr[A'(\mathcal{O}'(C)) = 1] - \Pr[S'^C(1^k) = 1]\right| = \left|\Pr[A(\mathcal{O}(C)) = 1] - \Pr[S^C(1^k) = 1]\right| \tag{5}$$

$$\left|\Pr[A''(\mathcal{O}''(C')) = 1] - \Pr[S''^{C'}(1^k) = 1]\right| = \left|\Pr[A(\mathcal{O}(C)) = 1] - \Pr[S^C(1^k) = 1]\right| \tag{6}$$

Therefore, cascade is robust for virtual black-box obfuscation, and $\mathcal{O}$ is a secure obfuscator. □

Claim 2 *Given a PPT algorithm A, there exists a PPT algorithm A', s.t., for infinitely many k's equation 1 holds.*

Proof. Let A be a PPT algorithm against $\mathcal{O}$. We use A to construct A'. The algorithm A', in Algorithm 2, will receive an obfuscated program $\mathcal{O}'(C)$, and will emulate the execution of A providing it with $\mathcal{O}''(\mathcal{O}'(C))$ and will output whatever A outputs. Since A' applies $\mathcal{O}''$ on $\mathcal{O}'(C)$ and invokes A with the resulting program, it is efficient if A is efficient, and achieves the same probability as A, i.e., Equation 1 holds. □

Claim 3 *Given a PPT algorithm A, there exists a PPT algorithm A'', s.t., for infinitely many k's equation 2 holds.*

Proof. Let A be a PPT algorithm against $\mathcal{O}$. We use A to construct A''. The algorithm A'', in Algorithm 2, will receive an obfuscated program $\mathcal{O}''(\mathcal{O}'(C))$, will invoke A with $\mathcal{O}''(\mathcal{O}'(C))$ and will output whatever A outputs. Since A'' only invokes A, it is efficient if A is efficient, and achieves the same probability as A, i.e., Equation 2 holds. □

Algorithm 2 Given an algorithm A against $\mathcal{O}$, we construct algorithms A' and A'', against $\mathcal{O}'$ and $\mathcal{O}''$ respectively, that achieve the same advantage as A.

```
$A'(C' = \mathcal{O}'(C))$ {
    $C'' \leftarrow \mathcal{O}''(C')$
    return $A(C'')$
    }
```

```
$A''(C'' = \mathcal{O}''(\mathcal{O}'(C)))$ {
    return $A(C'')$
    }
```

Claim 4 *Given a PPT simulator S' there exists a PPT simulator S such that Equation 3 holds.*

Proof. Let S' be a PPT simulator for which A' cannot achieve a non-negligible advantage against $\mathcal{O}'$. We use S' to construct S, such that A will not be able to achieve non-negligible advantage against S contradicting the assumption. The algorithm S is given access to C, it simply invokes S' and provides it with an oracle access to C, in Algorithm 3. For each input query of S', S queries C and returns the result to S', precisely simulating the real execution to S'; thus Equation 3 holds. □

Claim 5 *Given a PPT simulator S'' there exists a PPT simulator S such that Equation 4 holds.*

Proof. Let S'' be a PPT simulator for which A'' cannot achieve a non-negligible advantage against $\mathcal{O}'$. We use S'' to construct S such that A will not be able to achieve non-negligible advantage against S contradicting the assumption. The algorithm S is given access to C, it invokes S'' and provides it with an oracle access to C, in Algorithm 3. For each input query of S'', S queries C and returns the result to S', precisely simulating the real execution to S'; thus Equation 4 holds. □

Note that the simulator S is implemented identically when using S' or S''. This is due to the fact that the simulator S obtains an oracle access to C, and has to simulate execution for S' and S'', which receive an oracle access to C and C'. Yet according to definition of obfuscator, the functionality is preserved, thus all S should do, is query its own oracle upon requests from S' (resp. S'') and return the responses as is.

Algorithm 3 Given a simulator S' (respectively, S'') we show how to construct S that will achieve the same advantage when given a black-box access to C.

```
$S^C(1^k)$ {
    return $S'^C(1^k)$
    }
```

```
$S^C(1^k)$ {
    return $S''^{C'}(1^k)$
    }
```

References

[1] Joy Algesheimer, Christian Cachin, Jan Camenisch, and Gunter Karjoth. Cryptographic security for mobile code. In *SP '01: Proceedings of the 2001 IEEE Symposium on Security and Privacy*, page 2, Washington, DC, USA, 2001. IEEE Computer Society.

[2] Boaz Barak, Oded Goldreich, Russell Impagliazzo, Steven Rudich, Amit Sahai, Salil Vadhan, and Ke Yang. On the (im)possibility of obfuscating programs. In *CRYPTO '01: Proceedings of the 21st Annual International Cryptology Conference on Advances in Cryptology*, London, UK, 2001. Springer-Verlag. ISBN 3-540-42456-3.

[3] D. Boneh and X. Boyen. On the impossibility of efficiently combining collision resistant hash functions.

[4] Christian Cachin, Jan Camenisch, Joe Kilian, and Joy Muller. One-round secure computation and secure autonomous mobile agents. In *Automata, Languages and Programming*, pages 512–523, 2000. URL `citeseer.ist.psu.edu/article/cachin00oneround.html`.

[5] Ran Canetti. Towards realizing random oracles: Hash functions that hide all partial information. In Burton S. Kaliski, Jr., editor, *Advances in Cryptology – CRYPTO ' 97*, volume 1294 of *Lecture Notes in Computer Science*, pages 455–469. International Association for Cryptologic Research, Springer-Verlag, Berlin Germany, 1997. URL `http://philby.ucsd.edu/psfiles/97-07.ps(longerToCLversion,June2,97)`.

[6] C. Collberg, C. Thomborson, and D. Low. A taxonomy of obfuscating transformations. *University of Auckland Technical Report*, 170, 1997.

[7] CS Collberg and C. Thomborson. Watermarking, tamper-proofing, and obfuscation-tools for software protection. *Software Engineering, IEEE Transactions on*, 28(8):735–746, 2002.

[8] J. Daemen and V. Rijmen. *The Design of Rijndael: AES–the Advanced Encryption Standard.* Springer, 2002.

[9] S. Even and O. Goldreich. On the power of cascade ciphers. In D. Chaum, editor, *Proc. CRYPTO 83*, pages 43–50, New York, 1984. Plenum Press.

[10] M. Fischlin and A. Lehmann. Multi-Property Preserving Combiners for Hash Functions.

[11] D. Harnik, J. Kilian, M. Naor, O. Reingold, and A. Rosen. On robust combiners for oblivious transfer and other primitives. *Proc. EUROCRYPT 2005*, pages 96–113, 2005.

[12] Amir Herzberg. Folklore, practice and theory of robust combiners. Cryptology ePrint Archive, Report 2002/135, 2002. `http://eprint.iacr.org/`.

[13] S. Hohenberger and G.N. Rothblum. Securely Obfuscating Re-Encryption. *TCC*, pages 233–252, 2007.

[14] R. Meier and B. Przydatek. On robust combiners for private information retrieval and other primitives. *Proc. CRYPTO 2006*, pages 555–569, 2006.

[15] Wee. On obfuscating point functions. In *STOC: ACM Symposium on Theory of Computing (STOC)*, 2005.