Robust Combiners for Software Hardening

Amir Herzberg* and Haya Shulman**

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

Abstract. All practical software hardening schemes, as well as practical encryption schemes, e.g., AES, were 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 combiners for software hardening, specifically for obfuscation [2], and for White-Box Remote Program Execution (WBRPE) [14]. WBRPE and obfuscators are software hardening techniques that are employed to protect execution of programs in remote, hostile environment. WBRPE provides a software only platform allowing secure execution of programs on untrusted, remote hosts, ensuring privacy of the program, and of the inputs to the program, as well as privacy and integrity of the result of the computation. 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 reductions and code manipulation.

Keywords: White-box security, software hardening, obfuscation, robust combiners, cryptographic protocols, two-party computation.

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 for many applications, e.g., [4], they are highly inefficient for practical applications, and due to efficiency considerations, software hardening techniques employed in practice do not have a proof of security. Heuristic implementations are a typical choice in practice, and often implementations gain 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 [9], 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 2, and a robust combiner for White-Box Remote Program Execution (WBRPE) schemes, [14], 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 techniques 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 4, 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. Since the security of existing obfuscators is not proven, robust combiners is a natural choice to enhance security.

WBRPE is a white-box security building block, for remote program execution in hostile environment presented by [14]. The authors suggest to apply a build-break-fix cycle to produce efficient, heuristic WBRPE constructions. This process may result in multiple, incomparable candidate practical and efficient schemes; a robust combiner can combine such candidate schemes, and assure security provided at least one of the candidates is secure.

We illustrate the WBRPE scheme in Figure 1. The WBRPE is comprised of two phases, the generation phase, run by an offline trusted third party, and the protocol execution phase, run by and between the local and the remote hosts. The trusted third party generates the parameters of the scheme, i.e. the hardening key hk which is sent to local host, and the obfuscated virtual machine (OVM), a 'hardened' program, which is transfered to the remote host. The OVM emulates a trusted (software only) platform, and executes the input programs supplied by the local host in a secure manner. The local host uses the hardening key hk to harden programs P which it receives in an input; it sends the 'hardened program' $H_{hk}(P)$ to the remote host for execution. The remote host provides the hardened program and optionally local ('auxiliary') input a into the OVM; it also inputs

[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.

the required running time t and the output length l to the OVM (to prevent side channel attacks). The remote host returns the hardened result to the local host. We require that the local host learns only the result of the computation of its program P on the input a of the remote, while the remote host learns nothing at all.

WBRPE and obfuscator are software only techniques which are employed to harden programs for execution in remote environment, yet depending on the application different security requirements should be ensured. Both WBRPE and obfuscator aim to hide the code of the executing program, yet WBRPE, in contrast to obfuscation, also hides the output of the program, which can then be recovered by the party that has the corresponding secret unhardening key. In obfuscation on the other hand, 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.

WBRPE can be applied to ensure security to applications such as network gaming and online trading center, voice over IP, and more. In these applications the goal not only to protect the program but also the result of the program, to ensure privacy and to prevent tampering and meaningful modification of the result. We next review robust combiners, and then discuss obfuscators and WBRPE. Eventually we present our contribution - robust combiners for WBRPE and for obfuscators.

1.1 Our contribution: Robust Combiners for Obfuscators and for WBRPE schemes

We present robust combiners for software hardening, specifically for obfuscation [2] and for White-Box Remote Program Execution (WBRPE) [14].

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 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 2, 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 $\mathcal{O}''(\mathcal{O}'(P))$.

We also present a robust combiner for White-box RPE. WBRPE cascade combiner is (1,2)-robust; it receives two WBRPE schemes, and produces a third WBRPE scheme that is secure if one of the underlying schemes is secure. In fact, cascade is a robust combiner independently for the confidentiality (indistinguishability) and integrity (unforgeability) properties of WBRPE schems. By cascading two WBRPE schemes, as in Figure 3, if one of the two provides indistinguishability (unforgeability), then the cascade is a WBRPE scheme that provides indistinguishability (unforgeability, respectively). The combiners we present introduce an additional overhead, since the resulting

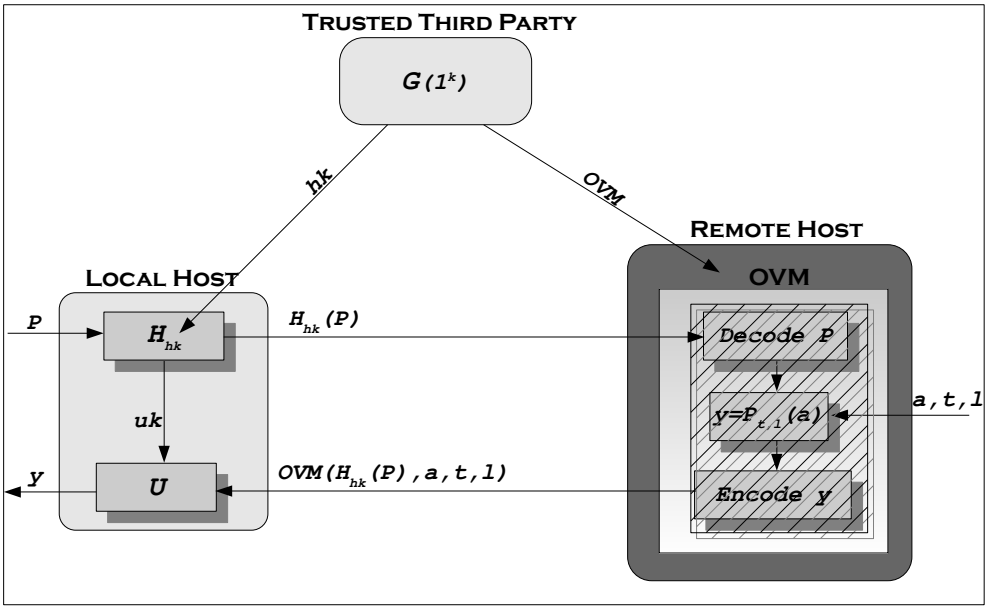

Fig. 1. *WBRPE* scheme is comprised of three parties the software originator, i.e., local host, the evaluator, i.e., the remote host, and the trusted third party. The trusted party is used during the offline generation phase, to produce the OVM and the hardening key hk. The local host obtains the hardening key hk which it uses to harden programs for execution on remote host. The local host applies the hardening algorithm on the program which results in a one time unhardening key uk and a hardened program $\mathcal{H}_{hk}(P)$, which it sends to the remote. The remote host obtains the OVM from the trusted party and uses it to evaluate hardened programs which it receives from the local host. Intuitively, OVM evaluates the program on the input of the remote host, and outputs the result. OVM can be implemented without decoding the result and this is just an abstract, simplified representation.

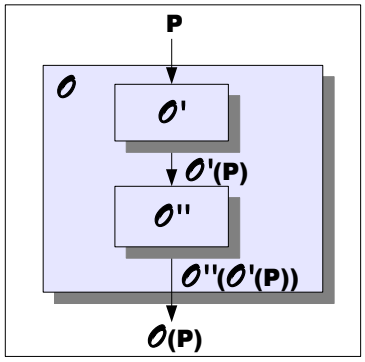

Fig. 2. Cascade obfuscator $\mathcal{O}$ receives a program P, and applies $\mathcal{O}'$ and $\mathcal{O}''$ sequentially. The resulting obfuscated program $\mathcal{O}''(\mathcal{O}'(P))$ is then output.

complexity is the multiple of the complexities of the candidate schemes. Hence when the combiner is applied repeatedly to combine n schemes, the complexity would be exponential in n (although in practice we expect to combine only very few schemes, e.g., two). We leave it as an open question, to investigate more efficient constructions, or to prove lower bounds. In practice, programs are often

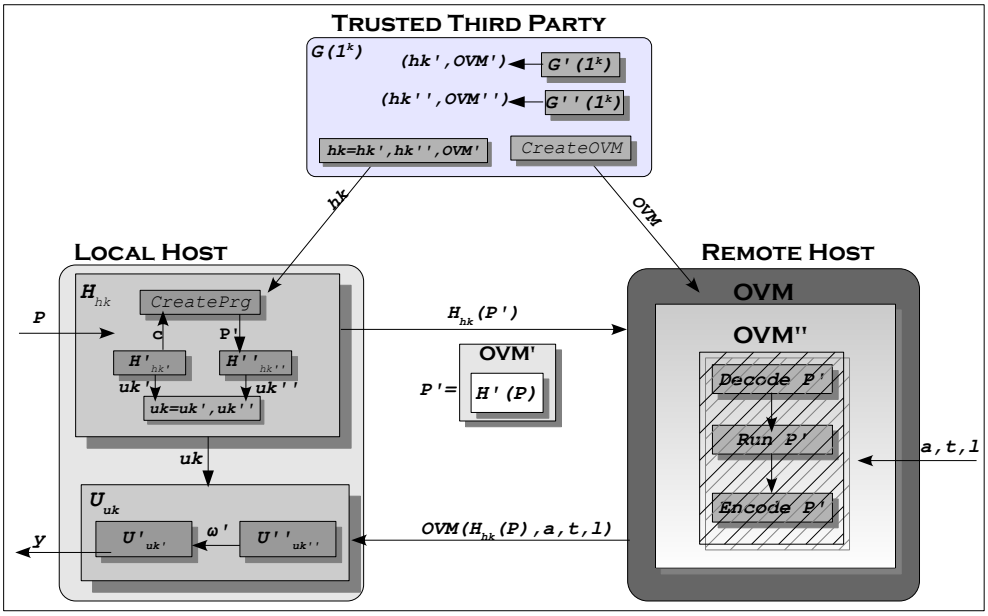

Fig. 3. Cascade *WBRPE* scheme.

run (or emulated) inside other programs, i.e., emulators. The basic idea of the WBRPE combiner is the same, the OVM is executed inside another OVM.

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 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 [10] showed that cascade is a robust combiners of block ciphers, against message recovery attacks. Cascade, and other basic robust combiners, were studied by Herzberg [15], for encryption, *MAC*, signature and commitment schemes. Robust combiners were also studied for other primitives, e.g. hash functions Fischlin and Lehmann [11], Boneh and Boyen [3], private information retrieval (PIR) Meier and Przydatek [17] and oblivious transfer Harnik et al. [13].

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.

1.3 Software Hardening Techniques: White-Box Remote Program Execution and 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 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 not accessible to the attacker. Therefore, alternative tools and techniques have been proposed. We review these next.

Obfuscation A prevailing technique employed by practitioners to harden software for execution in a remote environment is *obfuscation*, e.g. [7, 8]. 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 would. Obfuscated program is then sent off to remote platform, see Figure 4. 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

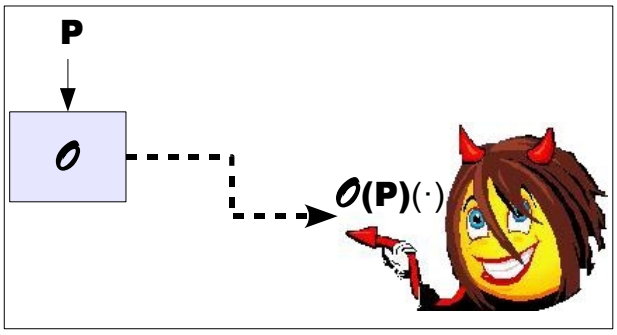

Fig. 4. A program P is obfuscated, and then sent to remote platform. The user controlling the execution environment can execute the obfuscated program on any input of its choice and can observe the outputs. The virtual black-box property requires that the user does not learn more about the program than 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, 18, 16], which are secure according to [2]. However none of these works show how to construct an obfuscator, but construct obfuscated programs from 'scratch'. Those results do not suffice for practical definition of 'obfuscation' but they do exhibit a feasibility.

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.

Although virtual black box obfuscation is a prevalent software hardening technique used in practice, it does not address security requirements of various applications, e.g., applications that require privacy of the result of the obfuscated program's computation. In [14], authors presented the White-Box Remote Program Execution scheme, which ensures privacy of the inputs as well as of the result. We briefly recap it next.

White Box Remote Program Execution (WBRPE) In *Remote Program Execution*, programs are sent by a *local host* (a.k.a. the originator) for execution on a *remote host*, and possibly use some data available to the *remote host*. The local and the remote hosts may be with conflicting interests, therefore the related security issues need to be dealt with. In particular, these include confidentiality and integrity of input programs supplied by the local host and confidentiality of inputs provided by the remote host.

White-box remote program execution (WBRPE) schemes are designed to ensure confidentiality of the input program P and of the output ($P(a)$), and integrity of the output. In addition, WBRPE ensures that the local host does not learn anything about the remote input a, beyond the result of the program applied to it ($P(a)$). All this should be ensured using software only, i.e., without assuming secure hardware, e.g. smartcards. WBRPE schemes can be employed to facilitate a variety of applications. For example, they can be applied in grid computing applications, to send a program for execution on an (untrusted or insecure) remote host. Another example is a mobile agents that traverse the Web, e.g. purchasing goods or searching for specific information. Such agents may include secret data, e.g., signing key, and therefore needs to be protected from a possibly hostile host, which may try to learn the secrets. There are also applications where the 'plain' WBRPE may not suffice, since it does not sufficiently protect the interests of the remote host, i.e., does not restrict the programs that can be executed on the input of the remote host a. The local host can send any program to the remote host; in particular it can learn arbitrary parts of the data a of the remote host or even all of it, by sending a program that computes an identity function. Specifically,

in many cases, the remote host may want to restrict the set of programs it is willing to run, e.g., to limit exposure of the remote input a, to charge differently for different types of queries, or to apply access control policy. Many of these applications fall into the category of online databases kept by the remote host, which needs to restrict access by the local host to the database. This is related to the problem of Private Information Retrieval in [6], but we want to protect the privacy of both parties, the remote host (server) which holds the database and the local host (client) who wishes to query the database. The input 'program' supplied by the local host (client) is a query, and the remote input of the remote host (server) is the database. We want to allow the remote host to define some restrictions (policy) on the allowable queries (or programs).

This can be achieved by validating the inputs supplied by the local host. The remote host provides its validation policy V as input to the generation process run by the trusted third party; The OVM will now also validate that P (supplied by local host) is 'valid', by applying V. As a result, the privacy and the integrity of both inputs of the client and the server are preserved, since the server cannot observe the queries submitted by the client, and the client receives results only for allowed queries.

We now give few details on the components of WBRPE scheme. A WBRPE scheme is composed of three efficient procedures, generation, hardening and unhardening, see Figure 1:

- The generation procedure produces a hardening key hk, and a program, which we call the *obfuscated virtual machine* (OVM).
- The hardening key hk is used by the hardening procedure to harden, e.g. encrypt and/ or authenticate, the input programs.
- The obfuscated virtual machine OVM receives a hardened input program along with input from the remote host. It decodes the hardened program, e.g. decrypts and/ or validates it, and returns the encoded result, e.g., encrypted and/or authenticated, of the program applied to the inputs.
- The unhardening procedure 'unhardens', e.g. decrypts and validates the result received from the remote host.

2 Cascade Combiner for Obfuscators

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. More specifically, the proof requires a refined definition of obfuscation, which we give and motivate below.

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.

2.1 Virtual Black-Box Obfuscation: Definition

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 definition 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:*

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

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.

2.2 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''.

```
O(C) {
    C' ← O'(C)
    C'' ← 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 present a formal and detailed proof of the cascade construction (Algorithm 1), and show that cascade is robust for virtual black-box obfuscation.

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. □

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. □

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.

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

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.

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

3 Cascade Combiner for WBRPE Schemes

In this section, we present a 'cascade' combiner for *WBRPE* scheme (see recap of definitions in Appendix, Section A), as illustrated in Figure 5. Given two candidate *WBRPE* schemes W' and W'', we combine them into one WBRPE scheme $W = W' \circ W''$. Essentially, we combine the *OVM* of W' with the input program P, to create the program to be hardened by W''. The construction is presented in Algorithm 4. In the sequel we shall refer to W' as an *internal scheme* and to W'' as an *external scheme*. We next give an intuitive, high level description of the design.

The main idea behind the combiner is that even if one of the schemes is insecure, e.g., one of the *OVM*s does not protect the memory contents, or if one of the schemes is incorrect, e.g., exposes the secret input program, then the overall construction will pertain security and correctness. Namely, the resultant scheme preserves indistinguishability, if one of the input candidates preserve indistinguishability. This holds since the inner *OVM'* is hidden by an outer *OVM"*. Therefore, even if the outer *OVM"* is not secure, i.e., does not 'hide' the programs that it executes, the combined scheme is secure, since the attacker cannot inspect the original input and output. Alternately, if the inner *OVM'* is not secure, the outer *OVM"* protects the computations. Similarly, the combined scheme preserves unforgeability of program and output, if one of the candidates ensures unforgeability of program and output, respectively.

Technicalities

- we include the t and l parameters in the construction, in order to prevent the adversary from distinguishing the input programs by their running times or output length. The t and l specified by the remote host are the bounds on the running time and output length of the input program P. In the combiner we have two programs P and P', such that for each input program P, we generate an external program P' that is to be executed by *OVM*. Hence we need to supply the *OVM* with the correct bound on P' running time and output length. We assume that we are given the polynomial bounds $T'(\cdot)$ and $L'(\cdot)$, on the running time and the output length of the internal *WBRPE* scheme W'. The *OVM* upon input t and l will calculate the respective running time t' and output length l', and supply them to *OVM'*. More specifically, there are $t' = T'(t,k,l)$ steps performed by P', and the output length of P' is $l' = L'(l,k)$. In particular, the t' and l' will specify the length of ω' returned by the inner scheme W', and the running time of P'.
- The macros *createOVM* and *createPrg* receive input parameters and generate a string, which is an encoding of a program. The *createOVM* receives an *OVM"* and a security parameter k. It then generates and returns an encoding of the *OVM* program which will be executed on the remote host (the *OVM* program is encoded as a string). Similarly, the *createPrg* generates a string encoding the program P'.

Definition 4 (Cascade of WBRPE schemes). *Let $T(\cdot)$ and $L(\cdot)$ be two polynomials. Given two candidate WBRPE schemes W' and W'', where $W'' = (\mathcal{G}'', \mathcal{H}'', \mathcal{U}'')$ and $W' = (\mathcal{G}', \mathcal{H}', \mathcal{U}')$, we denote their cascade by $W = W' \circ W''$, where $W = (\mathcal{G}, \mathcal{H}, \mathcal{U})$ with $L(\cdot)$ bounding its output length and $T(\cdot)$ bounding its running time, and $(\mathcal{G}, \mathcal{H}, \mathcal{U})$ are PPT algorithms, presented in Algorithm 4.*

Algorithm 4 The cascade *WBRPE* combiner $(\mathcal{G}, \mathcal{H}, \mathcal{U})$ with *createOVM* and *createPrg* macros, creating the *OVM* of the cascade *WBRPE* and the external program P' supplied as input to $\mathcal{H}''$, respectively. Macros return the code (program) after incorporating their parameters.

$\mathcal{G}(1^k)$ {
  $\langle hk', OVM' \rangle \stackrel{R}{\leftarrow} \mathcal{G}'(1^k)$
  $\langle hk'', OVM'' \rangle \stackrel{R}{\leftarrow} \mathcal{G}''(1^k)$
  $hk = \langle hk', hk'', OVM' \rangle$
  $OVM \leftarrow createOVM(OVM'', k)$
  return $\langle hk, OVM \rangle$
  }

$\mathcal{H}_{\langle hk', hk'', OVM' \rangle}(P)$ {
  $(c', uk') \leftarrow \mathcal{H}'_{hk'}(P)$
  $P' \leftarrow createPrg(c', OVM')$
  $(c, uk'') \leftarrow \mathcal{H}''_{hk''}(P')$
  $uk = \langle uk', uk'', P, P' \rangle$
  return $\langle c, uk \rangle$
  }

$\mathcal{U}_{(uk = \langle uk', uk'' \rangle)}(\omega, P, t)$ {
  $\langle y, P^*, t^* \rangle \leftarrow \mathcal{U}'_{uk'}(\mathcal{U}''_{uk''}(\omega))$
  if $((P = P^*) \wedge (t = t^*))$ return y
  else return $\perp$
  }

$createOVM(\mathbf{OVM"}, \mathbf{k})$ {
  return "$OVM(c, a, t, l)$
    $t' = T'(t, l, \mathbf{k}) + 2$
    $l' = L'(l, \mathbf{k})$
    $a' = (a, t, l)$
    return **OVM"**(c, a', t', l')"
  }

$createPrg(\mathbf{c'}, \mathbf{OVM'})$ {
  return "$P'(a')$
    $(a, t, l) \leftarrow a'$
    return **OVM'**$(\mathbf{c'}, a, t, l)$"
  }

3.1 Generation Procedure $\mathcal{G}$

The generation procedure $\mathcal{G}$ of $W = W' \circ W''$, in Algorithm 4, is performed by a trusted third party, and generates the parameters of *WBRPE* by applying the generation procedures of both candidates ($\mathcal{G}'$ of W' and $\mathcal{G}''$ of W''). The generated parameters are as follows:

- The hardening key hk consists of the concatenation of the hardening keys generated by both candidates W' and W'', and of the encoding of the OVM' generated by $\mathcal{G}'$. The hardening key hk is returned to the local host, and in case of a private *WBRPE* the hk is kept secret on the local and is assumed not to be known to other parties.
- The OVM of the combined scheme W is generated applying the *createOVM* macro, in Subsection 3.1, during the generation phase by the third party. The third party is particularly important due to the fact that the local and the remote may not trust each other, e.g., each may have conflicting interests. As a result, neither the local nor the remote can be trusted to generate the OVM correctly, since the local may try to expose the secret input of the remote, or to disrupt the execution platform of the remote, while the remote host may try to expose the secret input of the local, or to change the computation.

***createOVM* and *createPrg* Macros** These are functions that encode programs as strings, in order to transfer them securely to remote host for execution. More specifically, they generate the programs for W'', in particular the *OVM"* and the program P'. The *createOVM* macro calculates the corresponding running time t' and output length l', for *OVM"*, generates the remote input a' which is comprised of the remote input a and the original t and l parameters for *OVM'*, and runs the *OVM"*. The *createPrg* macro generates the input program P' for W'', which is an *OVM'* with the input parameters (a, t, l).

3.2 Hardening Procedure $\mathcal{H}$

The hardening procedure, $\mathcal{H}$ of $W = W' \circ W''$, in Algorithm 4, applies $\mathcal{H}'$ and then $\mathcal{H}''$. This ensures that the overall construction will still protect the input programs even if one of the schemes is insecure. In particular, $\mathcal{H}$, upon input a program P and a hardening key $hk = \langle hk', hk'', OVM' \rangle$, operates as follows:

- Upon input a program P, the local host applies the 'internal' hardening procedure ${\mathcal{H}'}_{hk'}$ on P, obtains the hardened program c' and the unhardening key uk'.
- The local host then generates a program P', defined in 4, which is essentially an *OVM'* instantiated with a hardwired hardened program c'. The P' program returns the result of the execution of *OVM'* on c' and a remote input a. P' itself is run inside *OVM"*, ensuring that even if one of the obfuscated virtual machines, *OVM'*or *OVM"* are not secure, e.g., leaks information about the program it executes, the overall construction remains secure.
- Then the program P' is hardened using the external hardening procedure ${\mathcal{H}''}_{hk''}$, resulting in a pair c and uk'', such that c is sent to the remote host for execution by OVM, while $\langle uk'', uk' \rangle$ is stored on the local host in order to recover the final result of the computation upon response from the remote host.

3.3 Unhardening Procedure $\mathcal{U}$

The unhardening procedure $\mathcal{U}$ of $W = W' \circ W''$, in Algorithm 4, receives the ephemeral unhardening keys, i.e. $uk = \langle uk', uk'' \rangle$, and applies $\mathcal{U}'$ and $\mathcal{U}''$, of the given candidates, and recovers the result of the computation of P on a. If the input program P, and the number of computations steps t are provided, the local host can also validate the result. Both validations by $\mathcal{U}'$ and by $\mathcal{U}''$ ensure robustness in case one of the candidates is erroneous.

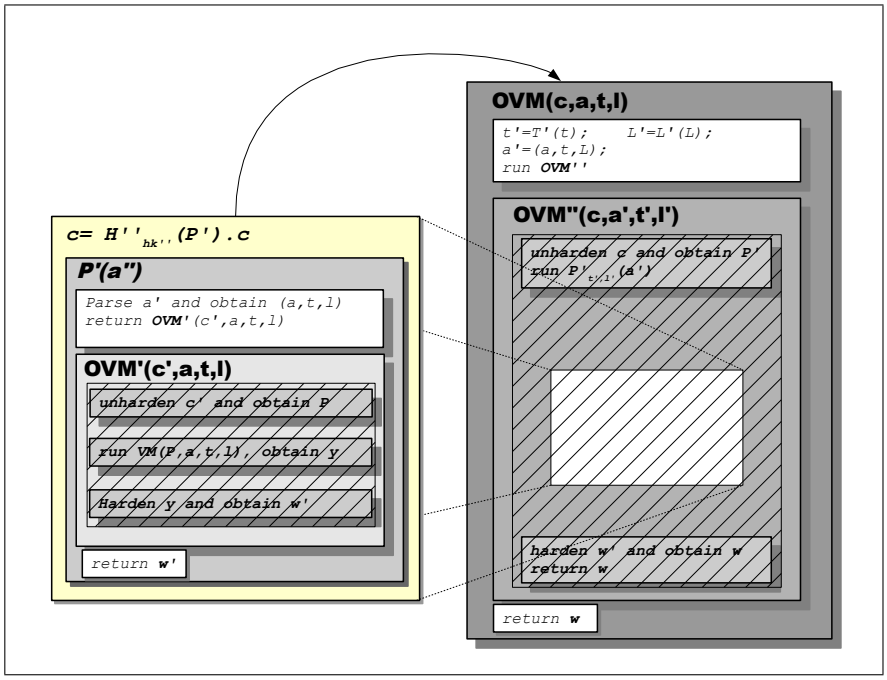

Fig. 5. *OVM* on the remote host.

4 Cascade is a Robust Combiner for WBRPE Schemes

In this section we show that cascade is a (1,2)-robust combiner for the security specifications of *WBRPE*. Namely, if one of the candidates satisfies the security specifications of *WBRPE*, then the cascade satisfies the security specifications of *WBRPE*, i.e. indistinguishability and unforgeability of the program and result. We prove this in Theorem 6 below.

We use $\varphi = PK$ to indicate public key scheme, i.e. the adversary receives the public hardening key hk, and $\varphi = SK$ to define private key scheme, i.e. the adversary obtains oracle access to the hardening functionality.

Theorem 6 (Cascade is a Robust Combiner for *WBRPE* Schemes). *Let W' and W'' be WBRPE schemes. For $\varphi \in \{PK, SK\}$, the combined WBRPE scheme $W = W'' \circ W'$, is:*

- *$WB - IND - CPA - \varphi$ secure if at least one of W' or W'' is $WB - IND - CPA - \varphi$ secure.*

– *$UNF-\varphi$ secure if at least one of W' or W'' is $UNF-\varphi$ secure.*

Proof. Given a PPT algorithm A against W we construct PPT algorithms A' and A'' against W' and W'', respectively, s.t.:

$$\mathbf{Adv}^{WB-IND-CPA-\varphi}_{W',A'}(k) = \mathbf{Adv}^{WB-IND-CPA-\varphi}_{W,A}(k) \tag{7}$$

$$\mathbf{Adv}^{WB-IND-CPA-\varphi}_{W'',A''}(k) = \mathbf{Adv}^{WB-IND-CPA-\varphi}_{W,A}(k) \tag{8}$$

$$\mathbf{Adv}^{WB-UNF-\varphi}_{W',A'}(k) = \mathbf{Adv}^{WB-UNF-\varphi}_{W,A}(k) \tag{9}$$

$$\mathbf{Adv}^{WB-UNF-\varphi}_{W'',A''}(k) = \mathbf{Adv}^{WB-UNF-\varphi}_{W,A}(k) \tag{10}$$

Due to space restrictions full proofs of equations (7), (8), (9), (10), are in Appendix, Sections B.1 and B.2.

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] B. Chor, O. Goldreich, E. Kushilevitz, and M. Sudan. Private Information Retrieval. *Journal of the ACM*, 45 (6):965–982, 1998.

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

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

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

[10] 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.

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

[12] O. Goldreich. *Foundations of Cryptography: Volume 2, Basic Applications*. Cambridge University Press New York, NY, USA, 2004.

[13] 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.

[14] A. Herzberg, H. Shulman, A. Saxena, and B. Crispo. Towards a theory of white-box security. In *Emerging Challenges for Security, Privacy and Trust: 24th Ifip Tc 11 International Information Security Conference, SEC 2009, Pafos, Cyprus, May 18-20, 2009, Proceedings*, page 342. Springer, 2009.

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

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

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

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

A White-Box RPE Definitions

In this section, we recap the definitions of WBRPE from [14]. A WBRPE scheme W, in Figure 1, is comprised of three efficient algorithms, $(\mathcal{G},\mathcal{H},\mathcal{U})$ for generation, hardening and unhardening, respectively. The generation procedure $\mathcal{G}$ generates the obfuscated virtual machine OVM and the hardening key hk. The hardening procedure $\mathcal{H}_{hk}$ is applied by the local host on the input program P, and produces two outputs: the 'hardened' program $\mathcal{H}_{hk}(P)$, which is sent to the remote host for execution, and a one time unhardening key uk, which is to be used by the local host to unharden, i.e., recover, the result of the program executed by the remote host. The remote host passes the hardened program, along with the remote input a to the OVM for execution. The OVM has the required corresponding keys, and can therefore extract and evaluate the program. Next, the OVM computes the result of P on a, and returns the (hardened) output ω. The local host, upon receipt of the hardened output ω, applies the unhardening procedure $\mathcal{U}$ with the unhardening key uk, that it received from the hardening procedure, to obtain the final result of the computation. The definition follows.

Given a Turing machine $P \in \mathbb{TM}$, let $P(a)$ denote a value of the computation of P on a. In order to prevent side channel attacks, we bound the computation time of P on a to t steps and restrict the output length to l bits. Let $P_{t,l}(a) = P_t(a)[1...l]$ denote an l bit value of the t step computation of P on input a.

Definition 5 (WBRPE). *A White Box RPE scheme W for programs family $\{P_k\}_{k\in\mathbb{N}}$ is a tuple $W = \langle\mathcal{G},\mathcal{H},\mathcal{U}\rangle$ of PPT algorithms, s.t. for all $(hk, \text{OVM}) \xleftarrow{R} \mathcal{G}(1^k)$, $P \in \mathbb{TM}$, $ovm \in PPT$, $a \in \{0,1\}^*$, $t,l \in \mathbb{N}$ and $(c, uk) \leftarrow \mathcal{H}_{hk}(P)$ holds:*

$$P_{t,l}(a) = \mathcal{U}_{uk}(\text{OVM}(c, a, t, l))$$

In the subsequent subsections we present the security specifications of *WBRPE*.

Discussion

- The bound on the number of execution steps may be of commercial value in many practical scenarios, where the remote host charges the local host per execution cycles. In this case the local would want an assurance that the remote could not have faked the result, or the number of execution steps.
- The hardening key hk can be either public or private. We present a general definition, and robust combiners capturing both options. When hk is secret, there is a unique OVM for every local host, and only the possessor of the corresponding unhardening key uk can recover the result of the computation. Alternately, when the hardening key hk is public, everyone can harden programs and send to OVM for execution. The obvious advantage of the public key WBRPE is in its flexibility, i.e. new hosts can join the system without any effort, e.g. a marketplace scenario where everyone can work with one central remote host and the same OVM.
- In the security specifications we use a flag φ to differentiate between public and private key schemes, s.t. when $\varphi = PK$, the adversary receives a public hardening key hk, whereas when $\varphi = SK$ the adversary only receives an access to a hardening oracle.

A.1 Confidentiality (Indistinguishability) of the Local Inputs

The program supplied by the local host, may contain secret information, e.g. secret encryption or signature key, therefore it is necessary to hide the contents of the input programs from the remote host. To ensure local inputs privacy, a variation of the indistinguishability experiment for encryption schemes is employed.

More specifically, the experiment generates the hardening key hk and the obfuscated virtual machine OVM, and invokes the adversary with an OVM in an input, and with oracle access to hardening procedure $\mathcal{H}_{hk}$. The adversary selects two programs of equal size, and specifies the number of steps to execute them, and the required output length. One of the programs is selected and hardened by the experiment and the adversary obtains the hardened program (but does not obtain the unhardening key required to recover the result of the computation). The adversary can execute the program on any input of its choice polynomially many times, and eventually it returns a bit b', specifying which of the two programs it selected was hardened. If the adversary guessed correctly, i.e., $b = b'$, then the experiment returns 1, in which case the adversary won, otherwise the experiment returns 0, i.e. the adversary lost. The scheme is secure if any adversary cannot distinguish (but with a negligible probability) between executions of two different programs, that are limited to the same number of execution steps, and producing outputs of same length (the result can be padded or truncated to reach same length). Formal definition is in full version of the paper.

A.2 Unforgeability of the Result Specification

When a program leaves the local host it is subject to tampering by the remote host, which may attempt to change the result of the computation. In some scenarios, e.g. shopping mobile agent, a remote host may try to change the result of the programs sent by the originator, e.g. such that instead of looking for the best offer the agent purchases the most expensive item. Our goal is to circumvent adversarial attempts to forge the result output by the scheme. This is captured by the unforgeability specification, based on unforgeability experiment for e.g., signature scheme Goldreich [12].

The unforgeability experiment applies the generation procedure and obtains hardening key hk, and OVM. It then invokes the adversary $A = (A_1, A_2)$ with an oracle access to hardening procedure, and with the OVM in an input. The adversary returns a challenge program for which it has to produce a forgery. The experiment applies the hardening procedure on the program producing two outputs, i.e. the hardened program (which the adversary A_2 obtains) and the unhardening key which is used to recover the final result of the computation. The adversary is invoked with a hardened program and the state from a previous invocation. Eventually, the adversary outputs a forgery ω. The experiment applies the unhardening procedure $\mathcal{U}$ along with the unhardening key uk in order to recover the result of the computation y. If y is valid, then the experiment checks if it is a forgery for a given P for any t and a, and if yes, returns 1, i.e. the adversary successfully generated a forgery, otherwise returns 0, the adversary failed.

In private WBRPE the adversary is given an oracle access to hardening functionality for its queries, therefore it is possible to keep track of the programs the adversary hardened. In contrast, in public WBRPE the hardening key hk is public and the OVM has the corresponding secret unhardening key, i.e. everyone can harden programs for execution. The implication is that after recovering the result by applying the unhardening procedure, it cannot be known what input program was hardened to produce the result. As a result, it is impossible to impose any assumption on the input

program, and it can only be guaranteed that the output is a result of the computation of the input program on some remote input. A possible solution is to supply the program to the local host as part of the unhardening key uk and then to compare the program returned by the adversary to the program supplied as part of uk. However in case there are other legitimate recipients in addition to the originator, it may be desirable to keep the input program secret and to allow others to obtain the result of the execution only. Therefore, we prefer a more general security specification.

We require, that given some input program P, the result that the local host obtains is for the program P that it supplied. We model this requirement by ensuring that the adversary can produce an output ω which is not the result of the computation of P on some remote input a, with only a negligible probability. Forgery means that the output is not a result of the computation of the input program on any remote input. Specifically, the adversary successfully generated an output ω, s.t. the result $y \leftarrow \mathcal{U}_{uk}(\omega)$ is valid but could not have been generated by the hardened program P, i.e. $\forall a\ y \neq P_{t,|y|}(a)$. Formal definition in full version of the paper.

B Cascade is Robust for WBRPE

B.1 Cascade is Robust for WBRPE w.r.t. Indistinguishability Specification

In this subsection we show that cascade is (1,2)-robust combiner for indistinguishability specification of the *WBRPE*, i.e. the equations 7,8 in Lemmas 7,8, respectively.

Lemma 7 *Given a PPT adversary $A = (A_1, A_2)$, there exists a PPT algorithm $A' = (A'_1, A'_2)$, s.t. for infinitely many k's equation 7 holds for $\varphi \in \{PK, SK\}$.*

Proof. Let $A = (A_1, A_2)$ be a PPT algorithm against the combined *WBRPE* scheme W. The algorithm A has an infinite length random tape, $\mathbf{r}$, such that during the execution A reads random bits of the random tape. We construct a PPT algorithm $A' = (A'_1, A'_2)$, in Algorithm 7, against a candidate *WBRPE* scheme W', that uses A and W' as black boxes. A' simulates the indistinguishability experiment, described in Section A.1, of the combined *WBRPE* scheme for A, by generating the corresponding parameters, and supplying responses to its hardening oracle queries. A' supplies A with the random coins for its execution. Eventually A' returns A's answer.
In order to simplify the reasoning behind the random tapes and their usage, we re-write both experiments, i.e. the indistinguishability experiment of the candidate *WBRPE* scheme W' against A', and the indistinguishability experiment of the combined *WBRPE* scheme W against A, using the vectors of random tapes $\mathbf{r}'$ and $\mathbf{r}$ respectively, see Experiments 5, refwbrpe:cascade:expt:ind:a'.

The indistinguishability experiment against W, Experiment 5, is supplied with five infinite random tapes, $\mathbf{r}$, $r_{H'_1}$, $r_{H''_1}$, $r_{H'_2}$ and $r_{H''_2}$, where $r_{H'_1}$ and $r_{H''_1}$ are supplied to the hardening oracle during the first stage of the experiment and $r_{H''_2}$ and $r_{H''_2}$ during the second stage. During each execution step it reads the corresponding bits off the random tapes. The random tape $r_{H'}$ and $r_{H''}$ are provided to the hardening oracle $\mathcal{HO}$ directly, which requires the random bits in order to respond to the hardening queries performed by the algorithm A. As specified in the construction, the $\mathcal{HO}$ applies $\mathcal{HO}'$ and then $\mathcal{HO}''$ supplying them with the corresponding random tapes $r_{H'}$ and $r_{H''}$ respectively. The random tape $\mathbf{r}$ is used for the other tasks performed by the experiment.

We present A' in Algorithm 7. In Algorithm 8, we present the implementation of the hardening procedure $\mathcal{HP}$ accessed by A.

Experiment 5 $\mathbf{Exp}^{WB-IND-CPA-\varphi}_{W,A}(k;\mathbf{r})$.

$(r[0],r[1],r[2],r[3],r[4],r[5],r[6],r[7],r[8],r[9],r[10]) \leftarrow$
$\mathbf{r}$
$(hk,\mathcal{OVM}) \leftarrow \mathcal{G}(1^k; r[0], r[1])$
$\langle P_0, P_1, s\rangle \leftarrow A_1^{\mathcal{HO}_{\langle hk', hk'', \mathcal{OVM}'\rangle}(\cdot,\varphi;r[2],r[3])}(1^k, \mathcal{OVM}; r[4])$

$b \leftarrow r[5]$
$(c, uk) \leftarrow \mathcal{H}_{hk}(P_b; r[6], r[7])$
$b' \leftarrow A_2^{\mathcal{HO}_{\langle hk', hk'', \mathcal{OVM}'\rangle}(\cdot,\varphi;r[8],r[9])}(c, s; r[10])$
return b'

Experiment 6 $\mathbf{Exp}^{WB-IND-CPA-\varphi}_{W',A'}(k;\mathbf{r}')$.

$(r[0],r[1],r[2],r[3],r[4],r[5],r[6]) \leftarrow \mathbf{r}'$
$r_[2] = \left\langle r_{G''}, r_{H_1''}, r_{A_1} \right\rangle$
$r_[6] = \left\langle r_{H_b''}, r_{H_2''}, r_{A_2} \right\rangle$
$(hk,\mathcal{OVM}) \leftarrow \mathcal{G}'(1^k; r_{G'})$
$\langle P_0, P_1, s\rangle \leftarrow A'^{\mathcal{HO}'_{\langle hk', hk'', \mathcal{OVM}'\rangle}(\cdot,\varphi;r_{H_1'})}_1(1^k, \mathcal{OVM}; r_{A_1'})$

$b \leftarrow r_b$
$(c, uk) \leftarrow \mathcal{H}'_{hk'}(P_b; r_{H_b'})$
$b' \leftarrow A'^{\mathcal{HO}'_{\langle hk', hk'', \mathcal{OVM}'\rangle}(\cdot,\varphi;r_{H_2'})}_2(c, s; r_{A_2'})$
return b'

Algorithm 7 Algorithm $A' = (A_1', A_2')$, that reduces $WB - IND - CPA - \varphi$ of $W = W' \circ W''$ to that of W'.

$A_1'^{\mathcal{HO}'_{hk'}(\cdot,\varphi;r_{H_1'})}(1^k, \mathcal{OVM}'', k; r_{A_1'})$
  $\left\langle r_{G''}, r_{H_1''}, r_{A_1} \right\rangle \leftarrow r_{A_1'}$
  $\langle hk'', \mathcal{OVM}''\rangle \leftarrow \mathcal{G}''(1^k; r_{G''})$
  $\mathcal{OVM} \leftarrow createOVM\{\mathcal{OVM}'', k\}$
  $\langle P_0, P_1, s\rangle \leftarrow A_1^{\mathcal{HP}_{hk''}(\cdot,\mathcal{OVM}',\varphi;r_{H_1''})}(1^k, \mathcal{OVM}; r_{A_1})$

  $s' \leftarrow \langle hk'', \mathcal{OVM}', \mathcal{OVM}\rangle$
return $(P_0, P_1, \langle s', s\rangle)$

$A_2'^{\mathcal{HO}'_{hk'}(\cdot,\varphi;r_{H_2'})}(c_b', \langle s', s\rangle; r_{A_2'})$
  $\left\langle r_{H_b''}, r_{H_2''}, r_{A_2} \right\rangle \leftarrow r_{A_2'}$
  $\langle hk'', \mathcal{OVM}', \mathcal{OVM}\rangle \leftarrow s'$
  $P_b' \leftarrow createP'\{c_b', \mathcal{OVM}'\}$
  $(c_b, uk_b'') \leftarrow \mathcal{H}''_{hk''}(P_b'; r_{H_b''})$
  $b' \leftarrow A_2^{\mathcal{HP}_{hk''}(\cdot,\mathcal{OVM}',\varphi;r_{H_2''})}(c_b, s; r_{A_2})$
return b'

Algorithm 8 Procedures $\mathcal{HP}_{hk''}$ and $\mathcal{HP}_{hk'}$, given as oracles to A and A'' by A' respectively.

$\mathcal{HP}_{hk''}(P, \mathcal{OVM}', \varphi; r_{H''})$
if $(\varphi = PK)$ **then**
  return $(\mathcal{HO}'_{hk'}(P, \varphi; r_{H'}), hk'', \mathcal{OVM}')$
else
  $(c', uk') \leftarrow \mathcal{HO}'_{hk'}(P, \varphi; r_{H'})$
  $P' \leftarrow createP'\{c', \mathcal{OVM}'\}$
  $(c, uk'') \leftarrow \mathcal{H}''_{hk''}(P'; r_{H''})$
  $uk = \langle uk', uk'', P, P'\rangle$
  return (c, uk)

$\mathcal{HP}_{hk'}(P, \mathcal{OVM}', \varphi; r_{H'})$
if $(\varphi = PK)$ **then**
  return $(hk', \mathcal{HO}''_{hk''}(P, \varphi; r_{H''}), \mathcal{OVM}')$
else
  $(c', uk') \leftarrow \mathcal{H}'_{hk'}(P; r_{H'})$
  $P' \leftarrow createP'\{c', \mathcal{OVM}'\}$
  $(c, uk'') \leftarrow \mathcal{HO}''_{hk''}(P', \varphi; r_{H''})$
  $uk = \langle uk', uk'', P, P'\rangle$
  return (c, uk)

Clearly, if A is efficient, i.e. a PPT algorithm, then A' is also efficient, since in addition to invoking A, it applies $\mathcal{G}''$, $\mathcal{H}''$ that are efficient, and performs constant operations on strings.
We next show that for any string $\mathbf{r}$:

$$\mathbf{Exp}^{WB-IND-CPA-\varphi}_{W',A'}(k;\mathbf{r}'(\mathbf{r})) = \mathbf{Exp}^{WB-IND-CPA-\varphi}_{W,A}(k;\mathbf{r}) \tag{11}$$

Where $\mathbf{r}'(\mathbf{r})$.

Then if $\mathbf{r}'$ is constructed as above, the view of the algorithm A when invoked by A', in the experiment against *WBRPE* scheme W', in Algorithm 6, is distributed identically to its view in the indistinguishability experiment against the *WBRPE* scheme W, in Algorithm 5.

This holds for any $\mathbf{r}$. Since if $\mathbf{r}$ is uniformly distributed, then so is $\mathbf{r}'$. Hence the advantage of A' is equivalent to the advantage of A, i.e. equation 7 holds. □

Lemma 8 *Given a PPT adversary $A = (A_1, A_2)$, there exists a PPT algorithm $A'' = (A''_1, A''_2)$, s.t. for infinitely many k's equation 8 holds for $\varphi \in \{PK, SK\}$.*

Proof. Let $A = (A_1, A_2)$ be a PPT algorithm against the combined *WBRPE* scheme W, we present a construction of a PPT algorithm $A'' = (A''_1, A''_2)$ against the candidate *WBRPE* scheme W'', in Algorithm 9, against W'', that uses A and W'' as black boxes. Specifically A'' simulates the indistinguishability experiment, in Section A.1, of the combined *WBRPE* scheme W for A, by generating the corresponding parameters, and supplying responses to its hardening oracle queries. Eventually A'' returns A's answer.

Algorithm 9 Algorithm $A'' = (A''_1, A''_2)$ that reduces $WB - IND - CPA - \varphi$ of $W = W' \circ W''$ to that of W.

$A''_1{}^{\mathcal{HO}''_{hk''}(\cdot,\varphi;r_{H''_1})}(1^k, \mathcal{OVM}''; r_{A''_1})$
- $(r_{G'}, r_{H'_1}, r_{H'_b}, r_{A_1}) \leftarrow r_{A''_1}$
- $\langle hk', \mathcal{OVM}' \rangle \leftarrow \mathcal{G}'(1^k; r_{G'})$
- $\mathcal{OVM} \leftarrow createOVM\{\mathcal{OVM}'', k\}$
- $\langle P_0, P_1, s \rangle \leftarrow A_1^{\mathcal{HP}_{hk'}(\cdot,\mathcal{OVM}',\varphi;r_{H'_1})}(1^k, \mathcal{OVM}''; r_{A_1})$
- $(c'_0, uk'_0) \leftarrow \mathcal{H}'_{hk'}(P_0; r_{H'_b})$
- $(c'_1, uk'_1) \leftarrow \mathcal{H}'_{hk'}(P_1; r_{H'_b})$
- $P'_0 \leftarrow createP'(c'_0, \mathcal{OVM}')$
- $P'_1 \leftarrow createP'(c'_1, \mathcal{OVM}')$
- $s' \leftarrow \langle hk', \mathcal{OVM}' \rangle$

return $(P'_0, P'_1, \langle s', s \rangle)$

$A''_2{}^{\mathcal{HO}''_{hk''}(\cdot,\varphi;r_{H''_2})}(c_b, \langle s', s \rangle; r_{A''_2})$
- $(r_{H'_2}, r_{A_2}) \leftarrow r_{A''_2}$
- $\langle hk', \mathcal{OVM}' \rangle \leftarrow s'$

return $A_2^{\mathcal{HP}_{hk'}(\cdot,\mathcal{OVM}',\varphi;r_{H'_2})}(c_b, s; r_{A_2})$

In Algorithm 8, we present the implementation of the hardening procedure $\mathcal{HP}$ accessed by A using the oracle $\mathcal{HO}''$ that is available to A' during the indistinguishability experiment $\mathtt{Exp}^{WB-IND-CPA-\varphi}_{W'',A''}(k)$.
If A is efficient, i.e. a PPT algorithm, then A'' is also efficient.
If A'' simulates the execution environment for A according to the steps specified in the indistinguishabbility experiment, in Section A.1, and implements the hardening procedure in Algorithm 8, for A identically to design of the hardening procedure in the construction of the combined scheme, in 3, then for any random string $\mathbf{r}$, it holds that

$$\mathbf{Exp}^{WB-IND-CPA-\varphi}_{W'',A''}(k;\mathbf{r}'') = \mathbf{Exp}^{WB-IND-CPA-\varphi}_{W,A}(k;\mathbf{r})$$

Where $\mathbf{r} = (r_{G'}, r_{G''}, r_{H'_1}, r_{H''_1}, r_{A_1}, r_b, r_{H'_b}, r_{H''_b}, r_{H'_2}, r_{H''_2}, r_{A_2})$ and **r"** is constructed as follows: **r"** $= (r_{G''}, r_{H''_1}, r_{A''_1} = \left\langle r_{G'}, r_{H'_1}, r_{A_1} \right\rangle, r_b, r_{H''_b}, r_{H''_2}, r_{A''_2} = \left\langle r_{H'_b}, r_{H'_2}, r_{A_2} \right\rangle)$ If **r"** is constructed as above, the view of the algorithm A when invoked by A'', in the experiment against *WBRPE* scheme W'' is distributed identically to its view in the indistinguishability experiment against the *WBRPE* scheme W, in Algorithm 5.

□

B.2 Cascade is Robust for WBRPE w.r.t. Unforgeability Specification

Lemma 9 *Given a PPT algorithm A against the combined WBRPE scheme $W = W' \circ W''$, there exists a PPT algorithm A' against W', s.t. for infinitely many k's, equation 9 holds, for $\varphi \in \{SK, PK\}$.*

Proof. Given A, we construct a PPT algorithm A', in Algorithm 10. Algorithm A' simulates the unforgeability experiment, of the combined WBRPE scheme W for A, by generating the corresponding parameters, and supplying responses to its hardening queries. Eventually A returns a forgery tuple (ω, P, t, uk). Next A' constructs a forgery for the experiment against W'. Namely, it has to return a tuple (ω', P, t, uk'), where $y \leftarrow {\mathcal{U}'}_{uk'}(\omega', P, t)$ and y could not have been generated by program P on any remote input a. A' parses the unhardening key uk, extracts the uk'' key, and applies the unhardening procedure $\mathcal{U}''$ with the keys uk'', on ω and obtains an ω'. Specifically, A' operates as defined in Algorithm 10.

If the parameters $(\mathcal{OVM})$ and the implementation of the hardening procedure $\mathcal{HP}$, that A' generates for A are identical to the parameters generates by the unforgeability experiment of the combined scheme, then for every $\mathbf{r}$, the advantage of A against the combined *WBRPE* is identical to the advantage of A' against the candidate *WBRPE*. Namely, the forgery generated of W by A can be reduced to the forgery of W' generated by A'. In particular, given a forgery (ω, P, t, uk) of the combined *WBRPE* scheme W generated by A, the tuple (ω', P, t, uk') output by A' constitutes a forgery of the *WBRPE* scheme W'. The unforgeability experiment upon input a tuple (ω', P, t, uk') from A', unhardens the result $y \leftarrow {\mathcal{U}'}_{uk'}(\omega', P, t)$ and then performs the following test

$$\forall a,\ y \neq P_{t,|y|}(a)$$

This is the same requirement as was defined in the output unforgeability security specification, therefore the result computed by A' constitutes a successful forgery according to the definition of $UNF - OUT$ requirement. This holds due to an observation, that A' essentially returns the result output by A after applying an external unhardening procedure $\mathcal{U}''$ to it. In other words, if ω is a forgery for input program P, then $\omega' \leftarrow {\mathcal{U}''}_{uk''}(\omega)$ is also a forgery for P. □

Algorithm 10 Algorithms A' and A'', that reduce $WB - UNF - \varphi$ of $W = W' \circ W''$ to that of W' and W'', respectively. See Lemmas 9,10.

${A'}^{\mathcal{HO'}_{hk'}(\cdot,\varphi)}(1^k, \mathcal{OVM}')$	${A''}^{\mathcal{HO''}_{hk''}(\cdot,\varphi)}(1^k, \mathcal{OVM}'')$
$\langle hk'', \mathcal{OVM}'' \rangle \leftarrow \mathcal{G}''(1^k)$	$\langle hk', \mathcal{OVM}' \rangle \leftarrow \mathcal{G}'(1^k)$
$\mathcal{OVM} \leftarrow createOVM\{\mathcal{OVM}'', k\}$	$\mathcal{OVM} \leftarrow createOVM\{\mathcal{OVM}'', k\}$
$(\omega, P, t, uk) \leftarrow A^{\mathcal{HP}_{hk''}(\cdot,\mathcal{OVM}',\varphi)}(1^k, \mathcal{OVM})$	$(\omega, P, t, uk) \leftarrow A^{\mathcal{HP}_{hk'}(\cdot,\mathcal{OVM}',\varphi)}(1^k, \mathcal{OVM})$
$\langle uk', uk'', P, P' \rangle \leftarrow uk$	$\langle uk', uk'', P, P' \rangle \leftarrow uk$
$\omega' \leftarrow {\mathcal{U}''}_{uk''}(\omega)$	$t' = T'(t) + 2$
return (ω', P, t, uk')	**return** (ω, P', t', uk'')

This is the same requirement as was defined in the output unforgeability security specification, therefore the result computed by A' constitutes a successful forgery according to the definition of $UNF - OUT$ requirement. This holds due to an observation, that A' essentially returns the result output by A after applying an external unhardening procedure $\mathcal{U}''$ to it. In other words, if ω is a forgery for input program P, then $\omega' \leftarrow {\mathcal{U}''}_{uk''}(\omega)$ is also a forgery for P. □

Lemma 10 *Given a PPT algorithm $A = (A_1, A_2)$ there exists a PPT algorithm $A'' = (A''_1, A''_2)$ s.t. for infinitely many k's equation 10 holds for any value of φ.*

Proof. Let A be a PPT algorithm against the combined *WBRPE* scheme W, we construct a PPT algorithm A'' against W'' that uses A and W'' as black boxes. Specifically, A'' operates as defined in Algorithm 10. In Algorithm 8, we present the implementation of the hardening procedure $\mathcal{HP}$ accessed by A using the oracle $\mathcal{HO}''(\cdot)$ that is available to A'' during the unforgeability experiment $\mathtt{Exp}^{UNF-OUT-\varphi}_{W'',A''}(k)$, Section A.2.
A'' obtains from A a tuple (ω, P, t, uk), parses uk to extract uk'', P and P'. Next A'' computes the number of steps $t' = T'(t, l, k) + 2$ for P' and returns a tuple (ω, P', t', uk''). Given an output forgery (ω, P, t, uk) for the combined *WBRPE* scheme W generated by A, the tuple (ω, P', t', uk'') output by A'' constitutes a forgery for the *WBRPE* scheme W''. The unforgeability experiment upon input a tuple (ω, P', t', uk'') from A'', unhardens the result, obtains $\omega' \leftarrow {\mathcal{U}''}_{uk''}(\omega, P', t')$ and then performs the following test

$$\forall a',\ \omega' \neq {P'}_{t',|\omega'|}(a')$$

Let ω be a forgery for W returned by A. To reduce the forgery for W to the forgery for W'', we assume by contradiction that $\exists a$, s.t. ${\mathcal{U}''}_{uk''}(\omega) = \mathcal{OVM}'({\mathcal{H}'}_{hk'}(P), a, t, l)$. Namely, ω' is a valid result and is not a forgery for W'. We then show that ω' is also a valid result and not a forgery for W. Observe that ${\mathcal{U}'}_{uk'}({\mathcal{U}''}_{uk''}(\omega)) = \mathcal{U}_{uk}(\omega) = P_{t,l}(a)$, this simply follows from ${\mathcal{U}'}_{uk'}(\omega') = \mathcal{OVM}'({\mathcal{H}'}_{hk'}(P), a, t, l))$. Then ω is not a forgery, but this contradicts the assumption.
Consequently, if A succeeds in the $\mathtt{Exp}^{WB-UNF-\varphi}_{W,A}(\cdot)$ with non-negligible advantage, it succeeds in the simulation run by A'' with non-negligible advantage, and since A''s success probability is related to that of A, A'' gains the same advantage as A in the unforgeability experiment with W'', contradicting the assumption of W'' being a $WB - UNF - \varphi$ secure scheme, therefore the lemma follows. □