Robust Combiners for White-Box Security

Amir Herzberg and Haya Shulman

Bar-Ilan University, Ramat-Gan, 52900, Israel,
{amir.herzberg,haya.shulman}@gmail.com

Abstract. *White-box* security techniques are employed to protect programs so that they can be executed securely in untrusted environments, e.g. for copyright protection. We present the first robust combiner for white-box primitive, specifically for *White-Box Remote Program Execution (WBRPE)* schemes. The *WBRPE* combiner takes two input candidate *WBRPE* schemes, W' and W'', and outputs a third candidate $W = W' \circ W''$. The combiner is $(1, 2)$-*robust*, namely, W is secure as long as either W' or W'' is secure. The security of the combined scheme is established by presenting a reduction to the security of the white-box candidates.
The *WBRPE* combiner is interesting since it presents new techniques of code manipulation, and in addition it provides both properties of confidentiality and authentication, even though it is a $(1, 2)$-robust combiner.
Robust combiners are particularly important for *white-box* security, since no secure candidates are known to exist. Furthermore, robust combiners for white-box primitives, are interesting since they introduce new techniques of reductions.

Keywords: White-box security, Robust combiners, cryptographic protocols, applied cryptography

1 Introduction

White-box security, flourished during the last two decades, due to the abundance of online software distribution, mobile agents technology, licensed software distribution and more. In white-box security, the software is executed in remote, possibly hostile environments, such that once the software leaves the site of the owner, it is at full mercy of the user controlling the execution environment, and the owner has no control over it. In white-box security the program along with the data is located on the remote machine exposed to curious or malicious users. More specifically, in white-box, the attacker can observe the internals of the executing program, i.e. the program's code and data. This is in contrast to black-box security, which assumes a trusted platform, i.e. a black-box, on which the secret data can be stored and computations involving it performed.

Therefore measures should be taken to harden the software so that it can withstand attacks in the white-box security model, i.e. to prevent undetected tampering or exposure of secret information, by providing integrity and confidentiality of the execution and the computations performed.

Enabling secure remote execution of programs in hostile environment is required for many applications, e.g. *Digital Rights Management (DRM)* Liu and Safavi-naini [28], market place Karnik and Tripathi [25], public online databases Verykios et al. [37], Agrawal and Srikant [2], Gertner et al. [16], *Private Information Retrieval* in Chor et al. [8]. Different practical and theoretical techniques to harden software were developed, in order to produce solutions to address the security issues of the applications in white-box security. Theoretical solutions include the *Mobile Code Cryptography* techniques, for details see Related Works in Section 8.However, so far provably secure solutions that achieve white-box security are either inefficient, or restricted to a limited set of functions, and hence are not suitable for practical applications. Furthermore, the existing practical techniques, such as obfuscation, are either proprietary, or lack a clearly stated conjecture of the security properties

of the cryptographic constructions, let alone a rigorous definition of security. Not surprisingly, these (mostly proprietary) solutions are not believed to be secure by the experts, e.g. Collberg and Thomborson [13], Collberg et al. [12]. Since the security of the existing white-box primitives is hard to assess, and in particular none is proven secure (based on hardness assumptions), it is particularly important to construct robust combiners for white-box security primitives. This will ensure that the scheme is at least as secure as the stronger candidate.

We therefore focus on white-box security and in particular we are interested in investigating fortification of software tools, that are employed to harden programs for secure execution in untrusted environments. More specifically, we investigate robust combiners for white-box primitives. In particular for *WBRPE* which was, recently, presented by Herzberg et al. [23].

1.1 White Box Remote Program Execution (WBRPE)

In Herzberg et al. [23], introduced a new white-box security building block, the *White Box Remote Program Execution (WBRPE). WBRPE* can be employed to facilitate a variety of applications, e.g. grid computing, public online databases, Digital Rights Management (DRM) applications.

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*; see Figure 1. The local and the remote hosts may not trust each other, and since the local host loses all control over the program it should be protected, and 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.

The *WBRPE* satisfies these requirements, employing software only techniques without assuming secure hardware, i.e. trusted third party or smartcards. The *WBRPE* scheme is composed of three efficient procedures, generation, hardening and unhardening, see Figure 1:

- The generation procedure produces two keys, (hardening and verification key) and a program, which we call the *obfuscated virtual machine* $\mathcal{OVM}$.
- The hardening key is used by the hardening procedure to harden, e.g. encrypt and/ or authenticate, the input programs.
- The obfuscated virtual machine receives the 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 result of the program applied to the input. The result is encoded, e.g. encrypted and/or authenticated (within the $\mathcal{OVM}$).
- The unhardening procedure unhardens, e.g. decrypts and validates the result received from the remote host.

1.2 Robust Combiners for White-Box Primitives

In cryptography, the security of constructions often depends on the underlying building blocks, e.g. unproven hardness assumptions, or primitives that withstood cryptanalysis attacks, and the security of the basic primitives is not known. 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 are a valuable tool in cryptography, and can be applied to enhance an overall security of cryptographic constructions, i.e. prevent erroneous implementations or design bugs.

Furthermore, combiners are of particular importance when the security of cryptographic primitives is not clear, i.e. especially in the context of white-box security, where no secure candidates are known to exist, i.e. no schemes are provably secure, and no building blocks were standardised. Hence, the safest choice is to enhance white-box security constructions, by combining several candidates into one. As a result, even if vulnerabilities in one of the candidates were discovered due to design or implementation bugs, the overall construction will remain secure as long as one of the candidates is secure. However, it should be noted w.r.t. the stated above, that robust combiners do not provide a guarantee of security, but they ensure that the security of the combined system is at least as that of the strongest primitive.

Robust combiners received considerable attention from the research community, however all in the context of black-box primitives, e.g. block-ciphers Bellare and Rogaway [5], encryption schemes, hash functions, *Message Authentication Code (MAC)* and signature schemes Herzberg [22], Oblivious Transfer Harnik et al. [21], Meier et al. [30]. In Herzberg [22] present several types of robust combiners for various cryptographic primitives, e.g. encryption, signature and commitment schemes, with focus on efficiency. In this work we investigate combiners for white-box primitives. In particular, for *WBRPE* Herzberg et al. [23].

In this work we present the first robust combiner for white-box primitive, specifically for *White-Box Remote Program Execution (WBRPE)*, as defined in Herzberg et al. [23]. We focus on combiners that receive two candidates, also called (1,2)-robust combiners, and produce a third candidate that is secure if one of the underlying candidates is secure.

Our main contribution: Robust Combiners for WBRPE In this work we present the following (1,2)-robust combiners for *WBRPE*: By cascading two *WBRPE* schemes, in Figure 3, such that at least one provides indistinguishability, (resp. unforgeability) we obtain a *WBRPE* scheme that provides indistinguishability (unforgeability). The combiner will remain secure even if one of the candidate schemes is insecure. In the cascade construction, we assume that the given candidates satisfy the correctness property. Namely, that the unhardening procedure returns the result of the computation of $\mathcal{OVM}$ on the hardened input program, the remote input, and the auxiliary t and l parameters used to prevent side channel attacks.

We then present a combined construction of the *WBRPEwV* scheme, in Figure 3.3, i.e. a *WBRPE* scheme that provides remote inputs privacy by employing input programs validation, and prove that this extended construction also ensures indistinguishability (res. unforgeability) if at least one of the candidates provides indistinguishability (unforgeability). Also in this case even if one implementation is insecure, e.g. the external $\mathcal{OVM}$ exposes the remote input, the combined scheme will still be secure.

In the (1,2)-robust *WBRPE* cascade construction that we present, we achieve both security specifications, i.e. confidentiality and authentication. This is in contrast to most existing combiners, which preserve a single property only, e.g. confidentiality or authentication, collision resistance or pseudorandomness, binding or hiding property for commitment schemes, e.g. [22]. In order to ensure all the properties, a (1,3)-robust combiners were constructed. Another recent result for multi-property preserving combiners for hash functions was presented in [15].

On the flip side, the combiner that we present has a substantial overhead, since the resulting complexity is exponential in the number of the of the underlying primitives. Although it seems that this is an optimal combiner for *WBRPE*, we leave it as an open question, to investigate more efficient constructions.

Furthermore, to provide unforgeability specification, (1,2)-robust copy combiner suffices. Copy combiner is much more efficient than the cascade and is more simple, hence the unforgeability property seems much easier to attain. Although it should be noted that its output length is doubled. It is also interesting to construct a cascade combiner employing the candidate, that results from the copy combiner and a candidate *WBRPE* scheme that provides indistinguishability. The resulting scheme should provide both the indistinguishability and the unforgeability property, i.e. by combining these two, we obtain a construction that robust for *WBRPE*.

1.3 Organisation

In Section 2 we present the definition of *WBRPE* and of the security specifications. In Section 3 we present the construction of the combiner for *WBRPE* and in Sections 1, 7.2 we prove that the combiner is robust for the security specifications of *WBRPE* presented in Section 2. We then present a construction of the *WBRPEwV* combiner that provides privacy of the input programs, and analyse its security in Section, 7.4. In Section 8 we presented the existing works in white-box security and robust combiners, and we conclude with open questions in Section 9.

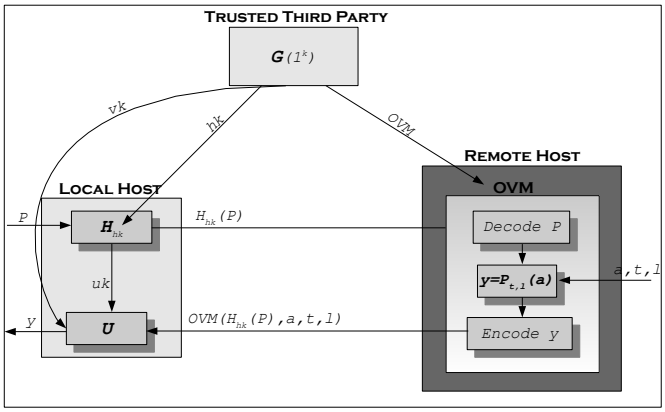

Fig. 1. *WBRPE* scheme.

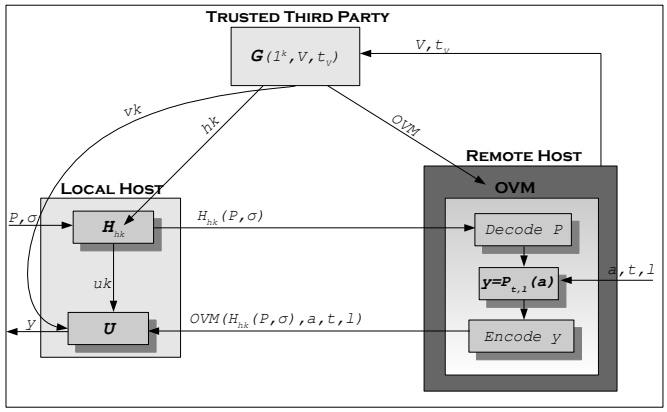

Fig. 2. *WBRPEwV* scheme with programs privacy.

2 White-Box RPE Definitions

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*, the hardening key hk and the public verification key vk. The hardening procedure applied on some input program, hardens the program, e.g. encrypts and/ or authenticates the original program, and produces two outputs, the hardened program and a one time unhardening key. The remote host passes the hardened program, along with the remote input a to the OVM for execution. The OVM has the required keys, and can therefore extract and evaluate the program. Next, the OVM computes the result of P on a, and returns the (hardened) output. The local host,

upon receipt the hardened output, applies the unhardening procedure with the unhardening key, that it received from the hardening procedure, to obtain the final result of the computation.

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 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. The definition follows.

Definition 1 (*WBRPE).* *A White Box RPE (WBRPE) scheme W for programs family $\{P_k\}_{k\in\mathbb{N}}$ consists of a tuple $W = \langle \mathcal{G}, \mathcal{H}, \mathcal{U} \rangle$ of PPT algorithms satisfying the following conditions:*

For all $(hk, vk, OVM) \xleftarrow{R} \mathcal{G}(1^k), a \in \{0,1\}^*, P \in \mathbb{TM}$, $t, l \in \mathbb{N}$ and $(c, uk) \leftarrow \mathcal{H}_{hk}(P)$, holds:
- *$OVM \in PPT$*
- *$P_{t,l}(a) = \mathcal{U}_{uk,vk}(OVM(c, a, t, l))$*

In various scenarios, e.g. when the remote input is a database, e.g. that contains private medical or personal information, it is necessary to limit the information about the remote input that the local host may obtain. To address this, we allow the owner of the remote input to specify the set of valid queries on the database during the generation phase of the scheme, thus the obfuscated virtual machine is confined to executing valid programs only. Defining the valid queries set, also prevents manipulation of the database by the adversary, i.e. deleting or modifying entries.

To restrict the execution to valid programs, we introduce the following supplementary parameters to the definition of *WBRPE* scheme, in Definition 1: the validation program $V \in \mathbb{TM}$ and the number of steps t_V to execute V, which are given to the trusted party that performs the generation phase. On input a program $P \in \mathbb{TM}$ and a validation parameter $\sigma \in \{0,1\}^*$, $V_{t_V,1}(P, \sigma) \in \{0,1\}$ returns 1 if the program is valid and 0 otherwise. The OVM will only execute valid programs. In addition, the signature of the hardening procedure $\mathcal{H}$ is modified and along with the input program P, it will also receive the validation parameter σ. The definition of the *WBRPEwV*, in Figure 2, presented below:

Definition 2 (*WBRPE with Validation (WBRPEwV)).* *A White Box RPE with Validation (WBRPEwV) scheme WV, consists of a tuple $\langle \mathcal{G}, \mathcal{H}, \mathcal{U} \rangle$ of PPT algorithms satisfying the following conditions:*

For all $\langle hk, vk, OVM \rangle \leftarrow \mathcal{G}(1^k, V, t_V)$, s.t. $V \in \mathbb{TM}, t_V \in \mathbb{N}$, $a \in \{0,1\}^*, P \in \mathbb{TM}, \sigma \in \{0,1\}^*$, $t, l \in \mathbb{N}$, and $(c, uk) \leftarrow \mathcal{H}_{hk}(P, \sigma)$, holds:
- *$OVM \in PPT$*
- *if $(V_{t_V,1}(P, \sigma) = 1)$ then $P_{t,l}(a) = \mathcal{U}_{uk,vk}(OVM(c, a, t, l))$*
- *else $\perp = \mathcal{U}_{uk,vk}(OVM(c, a, t, l))$*

In the subsequent subsections we present only a high level description of the security specifications of *WBRPE*, and rigorous game based definitions are in Herzberg et al. [23].

2.1 Indistinguishability of the Local Inputs Specification

The goal is 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 Goldreich [17] is employed.

As its first step the experiment generates the keys and the obfuscated virtual machine. Next it invokes the adversarial algorithm and provides it with an oracle access to the hardening functionality for its hardening queries, passes it the obfuscated virtual machine and the public verification key. Each application of the hardening procedure generates a hardened program and a one time unhardening key, which is used to recover the result. Eventually the adversary outputs two programs of equal size. The experiment tosses a bit b and the appropriate program is subsequently hardened. During the second phase the adversary keeps an oracle access to $\mathcal{HO}$, obtains the hardened challenge program and has to distinguish. If the adversary guesses correctly, the experiment returns 1, i.e. the adversary won, and otherwise returns 0, the adversary lost.

The hardening key can be either public or private; a flag φ is used to differentiate between public and private key schemes, s.t. when φ equals PK, the adversary receives a public hardening key, whereas when φ equals SK the adversary only receives an access to a hardening oracle.

2.2 Unforgeability Specification

In typical scenarios, e.g. shopping mobile agent, the adversary may try to change the programs sent by the originator to programs of his choice, such that instead of looking for the best offer the agent purchases the most expensive item. Further, the adversary may try to change the result of the computation to some other result. Our goal is to circumvent adversarial attempts to forge the result output by the scheme, and this is achieved by the unforgeability specification.

In [23], they extend the definition of *WBRPE* scheme, such that the unhardening procedure $\mathcal{U}$ can obtain additional optional parameters in an input, when validation of the inputs is required. More specifically, the local host can validate the result, i.e. the result of the computation is indeed of the input program P that it supplied for the specified number of steps t. To perform validation, the unhardening procedure $\mathcal{U}$ will use the public verification key vk. The implication is that everyone can validate the result, but only the possessor of the secret unhardening key uk can obtain the final result. The validation of the result is optional and can only be performed when P and t are supplied in addition to ω, i.e. the output of OVM.

To verify the integrity of the result the recipient can apply the unhardening procedure, on ω, an input program P, and the t that was used for programs execution. The unhardening procedure uses the public verification key vk, and the secret unhardening key uk.
We consider two types of forgery of *WBRPE* scheme W: in particular, in [23], they introduce the notion of output forgery, i.e. the result of the computation is an incorrect output and could not have been generated by the input program on any remote input, and of program forgery. Below, we give an intuitive description of the unforgeability experiment, followed by the presentation of both types of forgery.

The experiment applies the generation procedure and obtains hardening key, verification key and OVM. It then invokes the adversary with an oracle access to hardening functionality, the OVM and the verification key in an input. Eventually the adversary outputs the hardened result of the computation ω, an input program P, the number of steps t and the unhardening key uk. The experiment applies the unhardening procedure $\mathcal{U}$ on ω, P and t, and obtains the result of the computation y. If y is valid the experiment checks if it is a forgery and if yes, returns 1, i.e. the adversary successfully generated a forgery, otherwise returns 0, the adversary failed.

Output Forgery Consider a public online database, where a user queries the database with some query, and the adversary changes the result of the computation to some other value of his choice. We prevent this using the "correct output" specification.

This type of 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 tuple (ω, P, t, uk), s.t. the result $y \leftarrow \mathcal{U}_{uk,vk}(\omega)$ could not have been generated by the hardened program P, i.e. $\forall a \; y \neq P_{t,|y|}(a)$.

Program Forgery The forgery of the program is only relevant for the symmetric *WBRPE* scheme, therefore the adversary obtains an oracle access to the hardening functionality and does not obtain the hardening key in an input. In this type of forgery, the legitimate party never queried the hardening oracle with a program for which the result was generated. Instead, the adversary replaces the authentic hardened program with some other program (replay or a forgery). We consider two variants:

- The adversary successfully generated a new unhardening key uk, which was not output by the hardening oracle. Namely, it generated a tuple (ω, P, t, uk), s.t. $y \leftarrow \mathcal{U}_{uk,vk}(\omega)$ and $y = P_{t,|y|}(a)$ for some $a, \; t, \; l$.
- The adversary successfully generated an output (ω, P, t, uk) s.t. $y \leftarrow \mathcal{U}_{uk,vk}(\omega)$, and $y = P_{t,|y|}(a)$, where the unhardening key uk was generated for a different program P'. In both cases, the adversary did not perform a hardening oracle query on P.

2.3 Privacy of Remote Inputs Specification

In the definition of the privacy specification below we require that the adversary gains no additional information about the remote input than what is already a-priori known to it, given the result of the computation and access to the scheme. We formalize this using the simulation paradigm: the *WBRPE* scheme W is semantically secure w.r.t. privacy of remote inputs, if everything an adversary can learn about the remote input given an access to the scheme for the computation of the result, the simulator could learn without any access to the scheme, by resorting to the help of a trusted third party to execute and validate programs on its behalf.
In the definition below the function f is the information about the remote input that the adversary attempts to learn and the function h represents the adversary's a-priori knowledge regarding the remote input. We denote by $\{REM_k\}_{k\in\mathbb{N}}$ the probability ensemble representing the distribution of the remote inputs and there exists a single polynomial $p(\cdot)$ such that for all sufficiently large k's, $|REM_k| \leq p(k)$, which essentially implies that there exists a polynomial bound on the length of the strings in this distribution. The adversary's inability to learn information about the remote input should hold for any distribution of remote inputs.

In the definition of privacy of remote inputs we present two environments, the white box environment which emulates the the real execution, and the black box environment that emulates the ideal execution, such that the black box environment provides no access to the scheme but simulates a trusted third third party which carries out the computations and the validations of the input programs on behalf of the local host, whereas the white box environment simulates an adversary which exploits the scheme to gain some additional information about the remote input. The advantage that the adversary gains in the white box, i.e. real, environment should be almost the same, as the advantage that the simulator gains in the black box, i.e. ideal, environment.

The adversary may have, a possibly limited, control over the remote input, e.g. consider a scenario where the remote input supplied by the executing remote host is a database. Clearly, the adversary may insert or update entries in the database. We model this by introducing the a_{ADV} parameter, which is the part of the remote input that is under the control of the adversary.

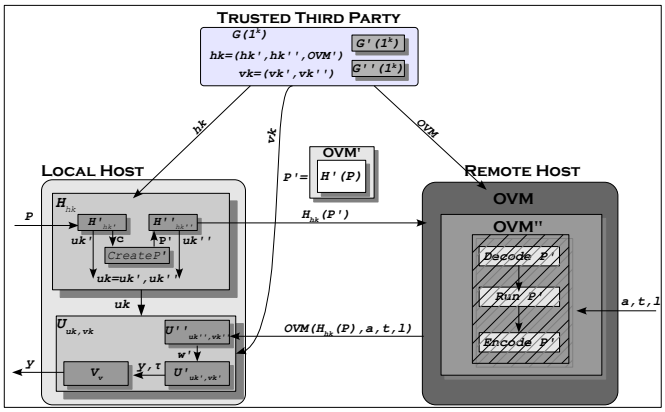

Fig. 3. Cascade *WBRPE* scheme.

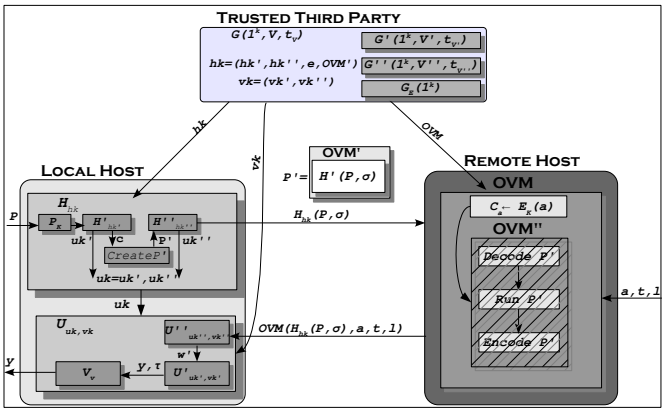

Fig. 4. Cascade *WBRPEwV* scheme.

3 Cascade Combiner for WBRPE Schemes

In this section we present the cascade combiner for *WBRPE* scheme, in Figure 3, and investigate its tolerance for security specifications defined for *WBRPE*. Given two candidate *WBRPE* implementations, we combine them into one scheme, that is secure w.r.t. security specifications of *WBRPE*, provided at least one of the two candidates is secure. More specifically, we encode the $\mathcal{OVM}$ of one of the candidate *WBRPE* schemes as the program to be computed by the other one. We first present a high level presentation of the cascade combiner for *WBRPE* and then proceed to intuitive analysis of its security. The detailed construction is presented in the subsections 3.1, 3.2, 3.3 in the sequel.

On a high level a *WBRPE* combiner operates as follows:

Upon input a program P the local host hardens the program using an internal hardening procedure $\mathcal{H}'$, obtains c' and the unhardening key uk'. Then, generates a program P', defined in Figure 2, which is essentially an $\mathcal{OVM}'$ instantiated with a hard-coded hardened program c'. The P' program returns the result of the execution of $\mathcal{OVM}'$ on a hardened program P and a remote input a. Then the program P' is hardened using the external hardening procedure $\mathcal{H}''$, resulting in a pair c and uk'', such that c is sent to the remote host for execution.

On the remote host there is the $\mathcal{OVM}$, presented in Figure 2. The $\mathcal{OVM}$ is created during the generation phase, it has a hard-coded encoding of the $\mathcal{OVM}''$, supplied by W'' and some additional processing, which we will extend on later. The $\mathcal{OVM}$ upon input a hardened program and a remote input passes them to $\mathcal{OVM}''$ for execution. Intuitively, the $\mathcal{OVM}''$ recovers, i.e. decodes and validates, the program P', and then executes it on the remote input a. The program P', as

presented above, is simply an $\mathcal{OVM}'$ with an instantiated parameter c'. The $\mathcal{OVM}'$ recovers, i.e. validates and decrypts, the hardened program P and runs it on the remote input a. Finally the $\mathcal{OVM}'$ returns the result y of the computation, and hardens it, i.e. encodes and authenticates, to obtain ω'. The $\mathcal{OVM}''$ hardens, i.e. encodes and authenticates, ω' and returns ω. The ω is then sent back to the local host. The local host recovers the result y of the computation of P on a, by applying $\mathcal{U}''$ and $\mathcal{U}'$ on ω.

Intuitively, the resultant scheme preserves indistinguishability, if one of the input candidates preserve indistinguishability. This holds since the inner $\mathcal{OVM}'$ is hidden by an outer $\mathcal{OVM}''$. Therefore, even if the outer $\mathcal{OVM}''$ is not secure, i.e. does not "hide" the programs that it executes, an overall scheme is secure, since the attacker cannot inspect the original input. Alternately, if the inner $\mathcal{OVM}'$ is not secure, the outer $\mathcal{OVM}''$ protects the computations. Identically, the combined scheme preserves unforgeability of program/ output if one of the candidates ensures unforgeability of program/ output.

However, the presented above scenario is exposed to side channel attacks. In particular, the attacker can distinguish programs by their length or by their running time. We therefore introduce the t and l parameters to prevent the ability of the adversary to distinguish the input programs by their running times or output length, for detailed construction refer to Figure 3.3. More specifically, the remote host will specify the running time and the output length for each input program P. In particular, the $\mathcal{OVM}$ upon input t and l will calculate the respective running time t' and output length l as a function of t and l and will supply them to $\mathcal{OVM}''$. These will specify the length of ω' returned by and the running time of the $\mathcal{OVM}'$.

Definition 3 (Cascade of WBRPE schemes). *Given two candidate* WBRPE *schemes W' and W'', where $W'' = (\mathcal{G}'', \mathcal{H}'', \mathcal{U}'')$ and $W' = (\mathcal{G}', \mathcal{H}', \mathcal{U}')$. A cascade* WBRPE *is defined by a tuple of PPT algorithms $(\mathcal{G}, \mathcal{H}, \mathcal{U})$, presented in Sections 3.1,3.2 and 3.3 respectively. We denote the cascade combiner by the binary operator $\circ$, i.e. $(\mathcal{G}'', \mathcal{H}'', \mathcal{U}'') \circ (\mathcal{G}', \mathcal{H}', \mathcal{U}')$.*

3.1 Generation Procedure

Let $T(\cdot)$ and $L(\cdot)$ be two polynomials bounding the running time and the output length of the $\mathcal{OVM}''$. The generation procedure $\mathcal{G}$, in Algorithm 1, applies the generation procedures $\mathcal{G}'$ and $\mathcal{G}''$ of both candidates and returns the hardening key, the verification key and the obfuscated virtual machine, i.e. the tuple $(hk, vk, \mathcal{OVM})$.

- The public verification key vk is comprised of vk' and vk''.
- The hardening key hk consists of the concatenation of the hardening keys generated by both candidates and of the encoding of the $\mathcal{OVM}'$ generated by $\mathcal{G}'$. The the hardened input program P is hard-coded into the $\mathcal{OVM}'$ on the local host, subsequently hardened, and transformed to the remote host for execution. The $\mathcal{OVM}''$ is one of the elements generated by $\mathcal{G}''$, and both $\mathcal{OVM}'$ and $\mathcal{OVM}''$, have the corresponding secret keys for hk',vk' and hk'',vk'' respectively, and can therefore recover the programs, e.g. decrypt and validate, and consequently harden, e.g. encrypt and authenticate, the result.
- Then the $\mathcal{OVM}$ string is generated using the *createOVM* macro, in Algorithm 2, and it constitutes an external envelope for $\mathcal{OVM}''$. More specifically, the $\mathcal{OVM}$ is installed on the remote host, and on each execution, it obtains the input parameters (c, a, t, l), computes the corresponding parameters $t' = T(t), l' = L(l)$ and $a' = (a, t, l)$ for $\mathcal{OVM}''$, and invokes $\mathcal{OVM}''$ with (c, a', t', l').

For the graphical representation of the operations performed during the execution on the remote host refer to Figure 3.3.

Algorithm 1 The cascade *WBRPE* combiner $(\mathcal{G}, \mathcal{H}, \mathcal{U})$.

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

Algorithm 2 External $\mathcal{OVM}$ with hard-coded $\mathcal{OVM}''$, external Program P', supplied as input to $\mathcal{H}''$.

$createOVM\{\mathcal{OVM}''\}$
  `read` c, a, t, l
  $t' = T(t)$
  $l' = L(l)$
  $a' = (a, t, l)$

$createP'\{c'\}$
return `read` a'
  $(a, t, l) \leftarrow a'$
  `return` $\mathcal{OVM}'(\$1, a, t, l)$;

3.2 Hardening Procedure

The hardening procedure $\mathcal{H}$, in Algorithm 3, upon input a program P and a hardening key $hk = \langle hk', hk'', \mathcal{OVM}' \rangle$, performs the following steps:

1. The hardening procedure $\mathcal{H}$ applies ${\mathcal{H}'}_{hk'}$ on the input program P and obtains (c', uk').
2. $\mathcal{H}$ then generates the program P' using $createP'$ macro, in Algorithm 2, with input c'. The P' is essentially an $\mathcal{OVM}'$ with a hard-coded hardened input program, i.e. c'. The program P' reads and parses the input a', and obtains the (a, t, l) parameters. It returns the result of the execution of $\mathcal{OVM}'$ applied on the hardened input c' and (a, t, l).
3. Next it applies ${\mathcal{H}''}_{hk''}$ on P', to obtain the hardening c and the unhardening key uk''.
4. The output is $\langle c, uk \rangle$, where uk is comprised of the unhardening keys (uk', uk''), the random string r, which is later used by the unhardening procedure $\mathcal{U}$, in Algorithm 3, to reconstruct the program P' for validation performed by $\mathcal{U}''$.

3.3 Unhardening Procedure

The unhardening procedure $\mathcal{U}$, in Algorithm 3, receives the ephemeral unhardening and public verification keys, i.e. $uk = \langle uk', uk'', r \rangle$ and $vk = \langle vk', vk'' \rangle$, along with the hardened program P. It applies both unhardening procedures $\mathcal{U}'$ and $\mathcal{U}''$, of the given candidates, along with uk', uk'' and vk', vk'' to recover the result of the computation. Specifically, the unhardening keys uk'', uk' are used to unharden the respective outputs, ω' and y, of $\mathcal{OVM}''$ and $\mathcal{OVM}'$, and vk'', vk' are used to validate ω' and ω, i.e. that ω' is a result of the computation of P after t steps, and ω is a result

Algorithm 3 The cascade *WBRPE* combiner $(\mathcal{G}, \mathcal{H}, \mathcal{U})$.

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

$\mathcal{U}_{(uk=\langle uk', uk'', r \rangle, vk=\langle vk', vk'' \rangle)}(\omega, P, t)$
if $((P, t) \neq NULL)$ **then**
  $(\tilde{c'}, \tilde{uk}) \leftarrow \mathcal{H}'_{hk'}(P; r)$
  $P' \leftarrow createP'(\tilde{c'})$
  $t' = t + 3$
  $\omega' \leftarrow \mathcal{U}''_{uk'', vk''}(\omega, P', t')$
  $y \leftarrow \mathcal{U}'_{uk', vk'}(\omega', P, t)$
else
  $y \leftarrow \mathcal{U}'_{\langle uk', vk' \rangle}(\mathcal{U}''_{\langle uk'', vk'' \rangle}(\omega))$
return y

of the t' steps computation of the input program P'. The program P' is reconstructed using the $createP'$ macro, in Algorithm 2, using the random string r, which it received as part of the uk key, and the number of steps t' is computed using t. This validation is performed only if P and t were supplied. If P and t were not supplied it does not perform the validation but only unhardens the output ω of $\mathcal{OVM}$, and obtains y.

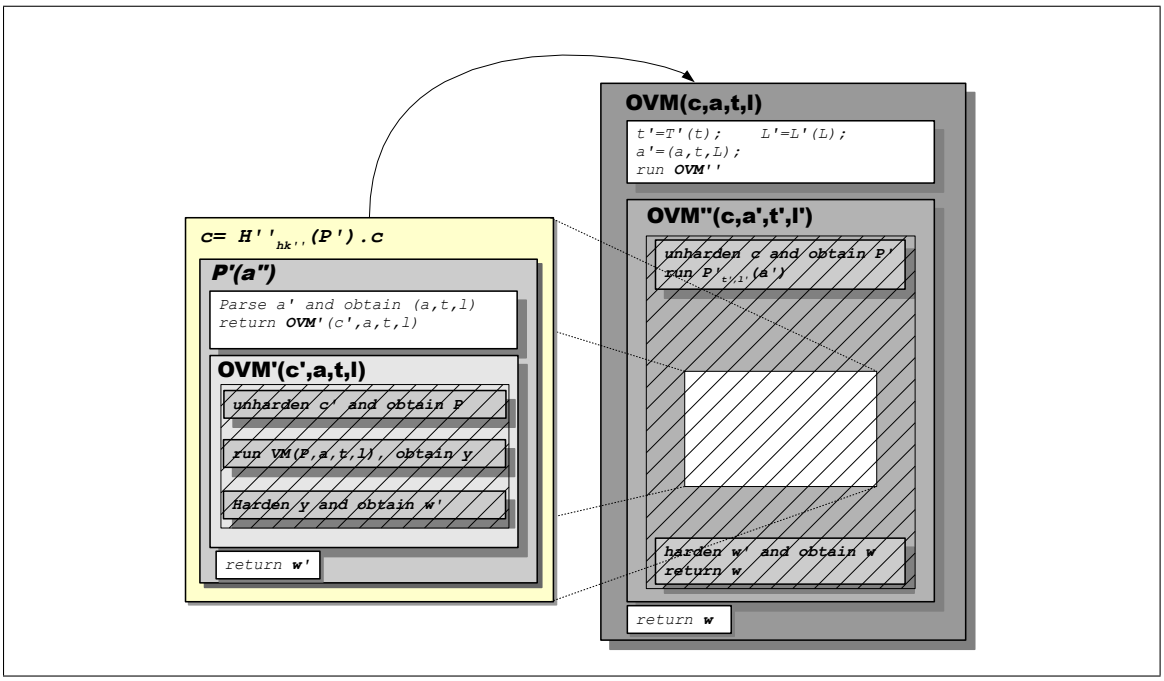

Fig. 5. $\mathcal{OVM}$ on the remote host.

4 Cascade Combiner is Robust for WBRPE

The combined *WBRPE* is robust for the security specifications of *WBRPE* scheme. More specifically if one of the candidates satisfies the security specifications of *WBRPE*, then the cascade satisfies the security specifications of *WBRPE*, i.e. the indistinguishability and the unforgeability of the

input programs and of the result of the computations. We prove this in Theorems 1, 2, 5, in Section 4.

We follow the attack definitions which were presented in Section 2, 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.
The proofs of the theorems 1, 2, 5, in Sections 7.1, 7.2, 7.3 respectively.

Theorem 1 (Cascade is Robust for Indistinguishability). *For $\varphi \in \{PK, SK\}$, a cascade WBRPE scheme $W = W'' \circ W'$, is $WB - IND - CPA - \varphi$ secure if at least one of W' or W'' is $WB - IND - CPA - \varphi$.*

Theorem 2 (Cascade is Robust for Output Forgery). *For $\varphi \in \{PK, SK\}$, a cascade WBRPE scheme $W = W'' \circ W'$, is $UNF - OUT - \varphi$ secure if at least one of W' or W'' is $UNF - OUT - \varphi$ secure.*

Theorem 3 (Cascade is Robust for Program Forgery). *For $\varphi \in \{PK, SK\}$, a cascade WBRPE scheme $W = W'' \circ W'$, is $UNF - PRG$ secure if at least one of W' or W'' is $UNF - PRG$ secure.*

5 Cascade Combiner for WBRPE with Validity (WBRPEwV) Schemes

Cascade combiner for *WBRPEwV* is similar in nature to the cascade combiner of *WBRPE*, in Section 3. It is constructed using two candidate *WBRPEwV* with validation, and in addition to the indistinguishability and the unforgeability specifications of *WBRPE*, it also provides privacy of remote inputs, that are supplied by the remote host. More specifically, provided one of the candidate *WBRPEwV* schemes provides privacy of remote inputs, the cascade provides privacy of remote inputs.

In the construction of the cascade *WBRPEwV* scheme we assume correctness, i.e. that both candidate schemes are correct. This assumption is critical, since the algorithms $\mathcal{H}$ and $\mathcal{U}$ may behave arbitrarily, and in particular, may expose the input program P, or the remote input a, contradicting the security specifications.

We now present a high level overview of the *WBRPEwV* scheme. The input to the hardening procedure $\mathcal{H}$, presented in Algorithm 6, is a pair P, σ, where σ is a validation parameter of P. The hardening procedure $\mathcal{H}$ selects a secret encryption key K and generates a program P_K with P and K. The program P_K, defined in 6, decrypts the remote input and then runs the VM on (P, a, t, l). The encryption of the remote input is performed by the $\mathcal{OVM}$ using the secret encryption key K. The remote input is encrypted in order to prevent its exposure by one of the $\mathcal{OVM}''$ or $\mathcal{OVM}'$, e.g. by concatenating the remote input to the result of the computation. The program P_K is the hardened along with the new σ_K, which includes σ and the input program P, resulting in a pair (c', uk'). Then a program P' is generated, which is an $\mathcal{OVM}'$ with an embedded input c', and an appropriate validation parameter σ' for P' is generated. The validation parameter σ' is to be used by the validation procedure V'' in $\mathcal{OVM}''$ to verify that the program P' is indeed of the required format, and not some other program, e.g. a program that does not perform the invocation of $\mathcal{OVM}'$ with hardening c', but for instance computes an identity function, i.e. given a returns a. Then P' is hardened along with σ' and is sent to the remote host for execution. The key $uk = uk', uk''$ is used to unharden the result of the computation, returned by the remote host.

On the remote host, the $\mathcal{OVM}$ upon input a remote input a, encrypts a, computes the new running time t' and output length l', and invokes the $\mathcal{OVM}''$ on the encrypted a, the hardened pair (P', σ'), and t', l'. Intuitively, the $\mathcal{OVM}''$ unhardens the input to recover the program P' and the validation parameter σ'. Since we assume correctness, the $\mathcal{OVM}''$ will run the validation program V'' on P' and σ'. If the validation program returns 1, the $\mathcal{OVM}'$ will execute P' on a', for t' steps and will truncate pad the result to l' bits. When executed, the program P' is simply an $\mathcal{OVM}'$ with hard-coded c' inside. P' first parses the a', to obtain c_a, t, l and will invoke an $\mathcal{OVM}'$ with (c', c_a, t, l). The $\mathcal{OVM}'$ recovers P_K and σ_K and validates them with V'. If validation passes, it runs P_K, on c_a for $t+3$ steps, then hardens and returns the result of the computation. The $\mathcal{OVM}''$ subsequently hardens and returns the output returned from $\mathcal{OVM}'$.

The local host, upon input ω, unhardens it using the unhardening procedure $\mathcal{U}$, which is presented and defined in Section 3.3, in Algorithm 3.

Definition 4 (Cascade of WBRPEwV schemes). *Given two candidate* WBRPEwV *schemes WV' and WV'', where $WV'' = \langle \mathcal{G}'', \mathcal{H}'', \mathcal{U}'' \rangle$ and $WV' = \langle \mathcal{G}', \mathcal{H}', \mathcal{U}' \rangle$. A cascade* WBRPEwV *is defined by a tuple of PPT algorithms $\langle \mathcal{G}, \mathcal{H}, \mathcal{U} \rangle$, presented in Sections 5.1, 5.2, 3.3 respectively. We denote the cascade combiner by the binary operator $\circ$, i.e. $\langle \mathcal{G}'', \mathcal{H}'', \mathcal{U}'' \rangle \circ \langle \mathcal{G}', \mathcal{H}', \mathcal{U}' \rangle$.*

5.1 Generation Procedure

The generation procedure $\mathcal{G}$, in Figure 4, upon input a validation program V, the number of steps t_V to run V, and the security parameters, generates the validation programs V' and V'', for $\mathcal{OVM}'$ and $\mathcal{OVM}''$ respectively. Applies $\mathcal{G}'$ and $\mathcal{G}''$ to generate the parameters of the scheme, and outputs $\langle hk, vk, \mathcal{OVM}'' \rangle$. More specifically, $\mathcal{G}$ operates as follows:

1. Applies the function *create*V'', in Figure 5, on V, t_V and generates V'', i.e. the validation function that is to be embedded in $\mathcal{OVM}''$, and computes the new number of steps $t_{V''}$ to run V''.
2. Applies $\mathcal{G}''$ and generates the tuple $\langle hk'', vk'', \mathcal{OVM}'' \rangle$. The reason that first $\mathcal{G}''$ is applied and then $\mathcal{G}'$, is because the validation function V' obtains $\mathcal{OVM}''$ in an input.
3. Then the *create*V' function, in Figure 5, is invoked with V, t_V and $\mathcal{OVM}''$ to generate V', and the appropriate number of steps $t_{V'}$ is computed.
4. Applies $\mathcal{G}'$ and generates the tuple $\langle hk', vk', \mathcal{OVM}' \rangle$.
5. A key tuple (e, d) is generated, and d is embedded in the external $\mathcal{OVM}$, that is created using the *createOVM* function, in Algorithm 4. The secret key d will be used by the $\mathcal{OVM}$ to decrypt the shared key K, supplied by the input program P_K. The $\mathcal{OVM}$ will encrypt the remote input a with K before transferring it to $\mathcal{OVM}''$. This is required to prevent possible exposure of the remote input by a malicious $\mathcal{OVM}''$.
6. Then $\mathcal{G}$ returns the tuple $\langle hk, vk, \mathcal{OVM} \rangle$, such that the hardening key is comprised of $\langle hk', hk'', e, \mathcal{OVM}' \rangle$, where e is the corresponding public encryption key, for the secret key d. The public verification key $vk = \langle vk', vk'' \rangle$, and the $\mathcal{OVM}$ is the external obfuscated virtual machine.

5.2 Hardening Procedure

The hardening procedure $\mathcal{H}$ receives the input program P along with the validation parameter σ, and the hardening key hk which is comprised of $\langle hk', hk'', e, \mathcal{OVM}' \rangle$. The $\mathcal{H}$ operates as follows:

Algorithm 4 The generator of the cascade *WBRPEwV* scheme $(\mathcal{G}, \mathcal{H}, \mathcal{U})$.

$\mathcal{G}(1^k)$
- $V'' \leftarrow createV''(V, t_V)$
- $t_{V'} = t_V + 7$
- $\langle hk'', vk'', \mathcal{OVM}'' \rangle \overset{R}{\leftarrow} \mathcal{G}''(1^k, V'', t_{V''})$
- $V' \leftarrow createV'(V, t_V, \mathcal{OVM}'')$
- $t_{V'} = t_V + 5$
- $\langle hk', vk', \mathcal{OVM}' \rangle \overset{R}{\leftarrow} \mathcal{G}'(1^k, V', t_{V'})$
- $(e, d) \leftarrow \mathcal{G}_E(1^k)$
- $\mathcal{OVM} \leftarrow createOVM(\mathcal{OVM}'', d)$
- $hk = \langle hk', hk'', e, \mathcal{OVM}' \rangle$
- $vk = \langle vk', vk'' \rangle$

return $\langle hk, vk, \mathcal{OVM} \rangle$

$createOVM\{\mathcal{OVM}''\}$

return "`read` c, a, t, l
- $(c'', c_K) \leftarrow c$
- $K \leftarrow \mathcal{D}_d(c_K)$
- $c_a \leftarrow \mathcal{E}'_K(a)$
- $t'' = T(t);\ \ l'' = L(l)$
- $t' = t + 3;\ \ l' = l$
- $a' = (c_a, t, l)$
- $a'' = (a', t', l')$
- `return` $\mathcal{OVM}''(c'', a'', t'', l'')$"

Algorithm 5 Validation of the input programs P to W'.

$createV'(V, t_V, \mathcal{OVM}'')$

return "read $(P_K, \tilde{\sigma})$
- $(P, \sigma) \leftarrow \tilde{\sigma}$
- if $((P_K = decryptionPrg(P)) \wedge (V_{t_V,1}(P, \sigma) = TRUE))$
  - return 1
- return 0"

$createV''(V, t_V)$

return "read (P', σ')
- $(\tilde{P}') \leftarrow \sigma'$
- if $(\tilde{P}' = P')$
  - return 1
- return 0"

1. Generates the P_K program by applying *decryptionPrg* macro on input program P, in Algorithm 6.
   – The P_K parses the input and obtains (P, c_a, t, l), decrypts the remote input c_a using K, runs the virtual machine VM with (P, a, t, l) and outputs y, i.e. the result of the computation.
2. The P_K program is hardened along with the validation parameter σ_K, using $\mathcal{H}'$.
3. A new program P' is created using the *createP'* macro, in Algorithm 2, which is essentially an $\mathcal{OVM}'$ with the hard-coded input c', along with the appropriate validation parameter σ'.
4. Both P' and σ' are hardened using $\mathcal{H}''$.
5. The $\mathcal{H}$ returns the pair (c, uk), such that, c is comprised of the encrypted secret key c_K is concatenated to the hardened program P', and uk is comprised of the unhardening keys uk', uk'', the verification key v and the hardened program P to be used by $\mathcal{U}$.

Algorithm 6 $\mathcal{H}_{\langle hk', hk'', e, \mathcal{OVM}' \rangle}(P, \sigma)$

$\mathcal{H}_{\langle hk', hk'', \mathcal{OVM}' \rangle}(P, \sigma)$
- $K \in_R \{0,1\}^k$
- $P_K \leftarrow decryptionPrg\{P, K\}$
- $c_K \leftarrow \mathcal{E}_e(K)$
- $\tilde{\sigma} \leftarrow (\sigma, P)$
- $(c', uk') \leftarrow \mathcal{H}'_{hk'}(P_K, \tilde{\sigma}; r)$
- $P' \leftarrow createP'(c')$
- $\sigma' \leftarrow (P')$
- $(c'', uk'') \leftarrow \mathcal{H}''_{hk''}(P', \sigma')$
- $c \leftarrow (c'', c_K)$
- $uk = \langle uk', uk'', r \rangle$

return (c, uk)

$decryptionPrg'\{P, K\}$

return `read` a'
- $(c_a, t, l) \leftarrow a'$
- $a \leftarrow \mathcal{D}_{\$1}(c_a)$
- $\mathtt{y} \leftarrow VM(\$2, a, t, l)$
- `return` y;

6 Cascade Combiner is Robust for WBRPEwV

The combined *WBRPE* is robust for the security specifications of *WBRPE* scheme. More specifically the cascade satisfies the privacy of remote input a, in addition to the indistinguishability and the unforgeability of the input programs and of the result of the computation. We present a proof that the cascade satisfies privacy of the remote inputs in Appendix, Section 7.4, and we present an intuition, showing that the *WBRPEwV* satisfies the indistinguishability and the unforgeability specifications, identically to 1, 2, 5, in Appendix, Section 4.
The proof of the theorem 7.4, in Appendix, Section 7.4.

Theorem 4 (Cascade is Robust for Privacy). *A combined WBRPEwV scheme WV, s.t. $WV = WV' \circ WV''$ is $PRV - VLD$ secure if at least one of WV' or WV'' is PRV-VLD secure.*

7 Security Analysis of the WBRPE Combiner

Below we present an analysis of the security specifications of the *WBRPE* combiner. More specifically, we show that even if one candidate is insecure, i.e. does not satisfy the security specifications of *WBRPE*, the overall construction is secure. We provide separate treatment for each security property of *WBRPE* in Sections 7.1, 7.2, 7.3.

7.1 Cascade is Robust for WBRPE w.r.t. Indistinguishability Specification

We specify the indistinguishability definition w.r.t. a PPT adversary $A = (A_1, A_2)$, denoting by $\mathcal{HO}$ the oracle which the adversary A obtains access to, during the experiment.
Proof of Theorem 1 For $\varphi \in \{SK, PK\}$, let $W = (\mathcal{G}, \mathcal{H}, \mathcal{U})$, s.t. $W = W' \circ W''$. Given $A = (A_1, A_2)$ against W we construct $A' = (A'_1, A'_2)$ and $A'' = (A''_1, A''_2)$ against W' and W'' respectively, s.t.

$$\mathtt{Adv}^{WB-IND-CPA-\varphi}_{W',A'}(k) = \mathtt{Adv}^{WB-IND-CPA-\varphi}_{W,A}(k) \quad (1)$$

$$\mathtt{Adv}^{WB-IND-CPA-\varphi}_{W'',A''}(k) = \mathtt{Adv}^{WB-IND-CPA-\varphi}_{W,A}(k) \quad (2)$$

We prove equations (1), (2) in lemmas 1, 2 respectively. The theorem 1 follows. □

Lemma 1. *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 1 holds for $\varphi \in \{PK, SK\}$.*

Proof. Let $A = (A_1, A_2)$ be a PPT algorithm against the combined *WBRPE* scheme W, we construct a PPT algorithm $A' = (A'_1, A'_2)$, in 7, against a candidate *WBRPE* scheme W', that uses A and W' as black boxes. A' simulates the indistinguishability experiment, in Section 2.1, of the combined *WBRPE* scheme for A, by generating the corresponding parameters, and supplying responses to its hardening oracle queries. Eventually A' returns A's answer.

In Figure 8, we present the implementation of the hardening procedure $\mathcal{HP}$ accessed by A. For $\varphi = PK$ holds $\mathcal{HP}_{hk''}(P) = (\mathcal{HO}'_{hk'}(P), hk'', \mathcal{OVM}') = (hk', hk'', \mathcal{OVM}')$.

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 if A' simulates the execution environment for A according to the steps specified

Algorithm 7 Algorithm $A' = (A'_1, A'_2)$, that reduces $WB - IND - CPA - \varphi$ of $W = W' \circ W''$ to that of W'.

$A'_1{}^{\mathcal{HO}'_{hk'}(\cdot,\varphi;r_{H'_1})}(1^k, \mathcal{OVM}', vk'; r_{A'_1})$
- $\left\langle r_{G''}, r_{H''_1}, r_{A_1} \right\rangle \leftarrow r_{A'_1}$
- $\langle hk'', vk'', \mathcal{OVM}'' \rangle \leftarrow \mathcal{G}''(1^k; r_{G''})$
- $\mathcal{OVM} \leftarrow createOVM\{\mathcal{OVM}''\}$
- $vk = \langle vk', vk'' \rangle$
- $(r^{(1)}_{H''_1}, ..., r^{(p(k))}_{H''_1}) \leftarrow r_{H''_1}$
- $\langle P_0, P_1, s \rangle \leftarrow A_1{}^{\mathcal{HP}_{hk''}(\cdot,\mathcal{OVM}',\varphi;r^{(i)}_{H''_1})}(1^k, \mathcal{OVM}, vk; r_{A_1})$
- $s' \leftarrow \langle hk'', vk, \mathcal{OVM}', \mathcal{OVM} \rangle$

return $(P_0, P_1, \langle s', s \rangle)$

$A'_2{}^{\mathcal{HO}'_{hk'}(\cdot,\varphi;r_{H'_2})}(c'_b, \langle s', s \rangle; r_{A'_2})$
- $\left\langle r_{H''_b}, r_{H''_2}, r_{A_2} \right\rangle \leftarrow r_{A'_2}$
- $\langle hk'', vk, \mathcal{OVM}', \mathcal{OVM} \rangle \leftarrow s'$
- $P'_b \leftarrow createP'(c'_b)$
- $(c_b, uk''_b) \leftarrow \mathcal{H}''{}_{hk''}(P'_b; r_{H''_b})$
- $(r^{(1)}_{H''_2}, ..., r^{(p(k))}_{H''_2}) \leftarrow r_{H''_1}$
- $b' \leftarrow A_2{}^{\mathcal{HP}_{hk''}(\cdot,\mathcal{OVM}',\varphi;r_{H''_2})}(c_b, s; r_{A_2})$

return b'

Algorithm 8 $\mathcal{HP}_{hk''}$ and $\mathcal{HP}_{hk'}$ given as oracle to $A = (A_1, A_2)$ by A' and by A'' respectively.

$\mathcal{HP}^{\mathcal{HO}'_{hk'}(\cdot,\varphi;r^{(i)}_{H'})}_{hk''}(P, \mathcal{OVM}', \varphi; r^{(i)}_{H''})$
- **if** $(\varphi = PK)$ **then**
  - **return** $(\mathcal{HO}'_{hk'}(P, \varphi; r^{(i)}_{H'}), hk'', \mathcal{OVM}')$
- **else**
  - $(c', uk') \leftarrow \mathcal{HO}'_{hk'}(P, \varphi; r^{(i)}_{H'})$
  - $P' \leftarrow createP'(c')$
  - $(c, uk'') \leftarrow \mathcal{H}''{}_{hk''}(P'; r^{(i)}_{H''})$
  - $uk = \left\langle uk', uk''; r^{(i)}_{H'}, r^{(i)}_{H''} \right\rangle$
  - **return** (c, uk)

$\mathcal{HP}^{\mathcal{HO}''_{hk''}(\cdot,\varphi;r^{(i)}_{H'})}_{hk'}(P, \mathcal{OVM}', \varphi; r^{(i)}_{H''})$
- **if** $(\varphi = PK)$ **then**
  - **return** $(hk', \mathcal{HO}''_{hk''}(P, \varphi; r^{(i)}_{H'}), \mathcal{OVM}')$
- **else**
  - $(c', uk') \leftarrow \mathcal{H}'_{hk'}(P; r^{(i)}_{H'})$
  - $P' \leftarrow createP'(c')$
  - $(c, uk'') \leftarrow \mathcal{HO}''_{hk''}(P', \varphi; r^{(i)}_{H''})$
  - $uk = \left\langle uk', uk''; r^{(i)}_{H'}; r^{(i)}_{H''} \right\rangle$
  - **return** (c, uk)

in the indistinguishability experiment 2.1, and implements the hardening procedure, in Algorithm 8, for A identically to the construction 3, then for any random string 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: $r' = (r_{G'}, r_{H'_1}, r_{A'_1} = \left\langle r_{G''}, r_{H''_1}, r_{A_1} \right\rangle, r_b, r_{H'_b}, r_{H'_2}, r_{A'_2} = \left\langle r_{H''_b}, r_{H''_2}, r_{A_2} \right\rangle)$ To simplify matters, we show how 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, use the random string $\mathbf{r}$, making it clear that if r' is constructed as above, the view of the algorithm A when invoked by A', in the experiment against *WBRPE* scheme W', in Algorithm 10, is distributed identically to its view in the indistinguishability experiment against the *WBRPE* scheme W, in Algorithm 9.

Therefore A's advantage in the simulation run by A' during the $\mathbf{Exp}^{WB-IND-CPA-\varphi}_{W',A'}(k; \mathbf{r'})$ is equal to its advantage in the $\mathbf{Exp}^{WB-IND-CPA-\varphi}_{W,A}(k; \mathbf{r})$ experiment. Hence we obtain, that for any r, equation 1 holds. □

Lemma 2. *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 2 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 $\mathbf{Exp}^{WB-IND-CPA-\varphi}_{W,A}(k;\mathbf{r})$.

$(r_{G'}, r_{G''}, r_{H'_1}, r_{H''_1}, r_{A_1}, r_b, r_{H'_b}, r_{H''_b}, r_{H'_2}, r_{H''_2}, r_{A_2}) \leftarrow \mathbf{r}$

$(hk, vk, \mathcal{OVM}) \leftarrow \mathcal{G}(1^k; r_{G'}, r_{G''})$
$(r^{(1)}_{H'_1}, ..., r^{(p(k))}_{H'_1}) \leftarrow r_{H'_1}$
$(r^{(1)}_{H''_1}, ..., r^{(p(k))}_{H''_1}) \leftarrow r_{H''_1}$
$\langle P_0, P_1, s\rangle \leftarrow A_1^{\mathcal{HO}_{\langle hk', hk'', \mathcal{OVM}'\rangle}(\cdot,\varphi;r^{(i)}_{H'_1}, r^{(i)}_{H''_1})}(1^k, \mathcal{OVM}, vk; r_{A_1})$

$b \leftarrow r_b$
$(c, uk) \leftarrow \mathcal{H}_{hk}(P_b; r_{H'_b}, r_{H''_b})$
$(r^{(1)}_{H'_2}, ..., r^{(p(k))}_{H'_2}) \leftarrow r_{H'_2}$
$(r^{(1)}_{H''_2}, ..., r^{(p(k))}_{H''_2}) \leftarrow r_{H''_2}$
$b' \leftarrow A_2^{\mathcal{HO}_{\langle hk', hk'', \mathcal{OVM}'\rangle}(\cdot,\varphi;r^{(i)}_{H'_2}, r^{(i)}_{H''_2})}(c, s; r_{A_2})$
return b'

Algorithm 10 $\mathbf{Exp}^{WB-IND-CPA-\varphi}_{W',A'}(k;\mathbf{r'})$.

$(r_{G'}, r_{H'_1}, r_{A'_1}, r_b, r_{H'_b}, r_{H'_2}, r_{A'_2}) \leftarrow \mathbf{r}$
$r_{A'_1} = \left\langle r_{G''}, r_{H''_1}, r_{A_1} \right\rangle$
$r_{A'_2} = \left\langle r_{H''_b}, r_{H''_2}, r_{A_2} \right\rangle$
$(hk, vk, \mathcal{OVM}) \leftarrow \mathcal{G}'(1^k; r_{G'})$
$(r^{(1)}_{H'_1}, ..., r^{(p(k))}_{H'_1}) \leftarrow r_{H'_1}$
$\langle P_0, P_1, s\rangle \leftarrow A'_1{}^{\mathcal{HO}'_{\langle hk', hk'', \mathcal{OVM}'\rangle}(\cdot,\varphi;r^{(i)}_{H'_1})}(1^k, \mathcal{OVM}, vk; r_{A'_1})$

$b \leftarrow r_b$
$(c, uk) \leftarrow \mathcal{H}'_{hk'}(P_b; r_{H'_b})$
$(r^{(1)}_{H'_2}, ..., r^{(p(k))}_{H'_2}) \leftarrow r_{H'_2}$
$b' \leftarrow A'_2{}^{\mathcal{HO}'_{\langle hk', hk'', \mathcal{OVM}'\rangle}(\cdot,\varphi;r^{(i)}_{H'_2})}(c, s; r_{A'_2})$
return b'

Figure 11, against W'', that uses A and W'' as black boxes. Specifically A'' simulates the indistinguishability experiment, in Section 2.1, 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.

In Figure 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 $\mathtt{Exp}^{WB-IND-CPA-\varphi}_{W'',A''}(k)$.
For $\varphi = PK$ holds $\mathcal{HP}^{\mathcal{HO}''_{hk''}(\cdot,\varphi)}(P) = (hk', \mathcal{HO}''_{hk''}(P, \varphi), \mathcal{OVM}') = (hk', hk'', \mathcal{OVM}')$.
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 2.1, and implements the hardening procedure 8, for A identically to the construction 3, then for any random string $\mathbf{r}$, it holds that

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

Algorithm 11 Algorithm $A'' = (A''_1, A''_2)$ that reduces $WB - IND - CPA - \varphi$ of $W = W' \circ W''$ to that of W.

$A''_1{}^{\mathcal{H}\mathcal{O}''_{hk''}(\cdot,\varphi;r_{H''_1})}(1^k, \mathcal{OVM}'', vk''; r_{A''_1})$
$(r_{G'}, r_{H'_1}, r_{H'_b}, r_{A_1}) \leftarrow r_{A''_1}$
$\langle hk', vk', \mathcal{OVM}' \rangle \leftarrow \mathcal{G}'(1^k; r_{G'})$
$\mathcal{OVM} \leftarrow createOVM\{\mathcal{OVM}''\}$
$vk = \langle vk', vk'' \rangle$
$\langle P_0, P_1, s \rangle \leftarrow A_1^{\mathcal{HP}_{hk'}(\cdot,\mathcal{OVM}',\varphi;r_{H'_1})}(1^k, \mathcal{OVM}'', vk; r_{A_1})$

$(c'_0, uk'_0) \leftarrow \mathcal{H}'_{hk'}(P_0; r_{H'_b})$
$(c'_1, uk'_1) \leftarrow \mathcal{H}'_{hk'}(P_1; r_{H'_b})$
$P'_0 \leftarrow createP'(c'_0)$
$P'_1 \leftarrow createP'(c'_1)$
$s' \leftarrow \langle hk', \mathcal{OVM}' \rangle$
return $(P'_0, P'_1, \langle s', s \rangle)$

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

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 9.

□

7.2 Cascade is Robust for WBRPE w.r.t. Output Unforgeability Specification

Proof of Theorem 2 For $\varphi \in \{SK, PK\}$, let $W = (\mathcal{G}, \mathcal{H}, \mathcal{U})$, s.t. $W = W' \circ W''$. Given $A = (A_1, A_2)$ against W we construct $A' = (A'_1, A'_2)$ and $A'' = (A''_1, A''_2)$ against W' and W'' respectively, s.t.

$$\mathtt{Adv}^{UNF-OUT-\varphi}_{W',A'}(k) = \mathtt{Adv}^{UNF-OUT-\varphi}_{W,A}(k) \tag{3}$$

$$\mathtt{Adv}^{UNF-OUT-\varphi}_{W'',A''}(k) = \mathtt{Adv}^{UNF-OUT-\varphi}_{W,A}(k) \tag{4}$$

We prove equations (3), (4) in lemmas 3, 4 respectively. The theorem 5 follows. □

Lemma 3. *Given a PPT algorithm A against a combined* WBRPE *scheme W, there exists a PPT algorithm $A' = (A'_1, A'_2)$ against the candidate* WBRPE *scheme W' s.t. for infinitely many k's, equation 3 holds for $\varphi \in \{SK, PK\}$.*

Proof. Let $A = (A_1, A_2)$ be a PPT algorithm against the combined *WBRPE* scheme W, we construct a PPT algorithm $A' = (A'_1, A'_2)$, in Algorithm 12, against W' that uses A and W' as black boxes. A' simulates the output 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 $y \leftarrow \mathcal{U}'_{uk',vk'}(\omega', P, t)$ and y could not have been a result of P for 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'' and vk'', on ω and obtains an ω'. Specifically, A' operates as defined in Algorithm 12

Algorithm 12 Algorithms $A' = (A'_1, A'_2)$ and $A'' = (A''_1, A''_2)$, that reduce $WB{-}UNF{-}OUT{-}\varphi$ of $W = W' \circ W''$ to that of W' and W''.

$A'^{\mathcal{HO}'_{hk'}(\cdot,\varphi)}(1^k, \text{OVM'}, vk')$
- $\langle hk'', vk'', \text{OVM"}\rangle \leftarrow \mathcal{G}''(1^k)$
- $\mathcal{OVM} \leftarrow createOVM\{\mathcal{OVM}''\}$
- $vk = \langle vk', vk''\rangle$
- $(\omega, P, t, uk) \leftarrow A^{\mathcal{HP}_{hk''}(\cdot,\varphi)}(1^k, \mathcal{OVM}'', vk)$
- $\langle uk', uk'', r\rangle \leftarrow uk$
- $\omega' \leftarrow \mathcal{U}''_{uk'', vk''}(\omega)$

return (ω', P, t, uk')

$A''^{\mathcal{HO}''_{hk''}(\cdot,\varphi)}(1^k, \text{OVM"}, vk')$
- $\langle hk', vk', \text{OVM'}\rangle \leftarrow \mathcal{G}'(1^k)$
- $\mathcal{OVM} \leftarrow createOVM\{\mathcal{OVM}''\}$
- $vk = \langle vk', vk''\rangle$
- $(\omega, P, t, uk) \leftarrow A^{\mathcal{HP}_{hk'}(\cdot,\varphi)}(1^k, \text{OVM"}, vk)$
- $\langle uk', uk'', r\rangle \leftarrow uk$
- $(c', uk') \leftarrow \mathcal{H}'_{hk'}(P; r)$
- $P' \leftarrow createP'(c')$
- $t' = t + 3$

return (ω, P', t', uk'')

In Algorithm 8 we present the implementation of the hardening procedure $\mathcal{HP}$, identical to the hardening used in the indistinguishability reduction, accessed by A using the oracle $\mathcal{HO}'_{hk'}(\cdot)$. The hardening oracle $\mathcal{HO}'$ is available to A' during the unforgeability experiment, $\mathtt{Exp}^{UNF-OUT-\varphi}_{W',A'}(\cdot)$. A' applies $\mathcal{G}''$ and generates the keys and the OVM. Since $\mathcal{G}''$ is efficient, this step is polynomial. Next, A' invokes A and simulates for it the unforgeability experiment of the combined WBRPE scheme W, which essentially involves performing hardening computations for A in response to its queries. Since the $\mathcal{HO}$ oracle's query computation is considered as one step, and $\mathcal{H}''$ is efficient, so is the overall computation performed by $\mathcal{HP}$. Therefore if A is efficient, i.e. a PPT algorithm, then A' is also efficient.
Intuitively, if the simulation run by A' is identical to the unforgeability experiment of the combined scheme, and the hardening procedure is constructed according to the combined construction in Section 3, then A's advantage in the simulation is equal to its advantage in the experiment. Furthermore, given an output 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', since A' applies the unhardening procedure once on the result returned by A and then returns the resulting tuple. In particular, the unforgeability experiment upon input a tuple (ω', P, t, uk') from A', unhardens the result $y \leftarrow \mathcal{U}'_{uk', vk'}(\omega', P, t)$ 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. □

Lemma 4. *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 4 holds for any value of φ.*

Proof. Let $A = (A_1, A_2)$ be a PPT algorithm against the combined *WBRPE* scheme W, we construct a PPT algorithm $A'' = (A''_1, A''_2)$ against W'' that uses A and W'' as black boxes. Specifically, A'' operates as defined in Algorithm 12. 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 2.2.
A'' obtains from A a tuple (ω, P, t, uk), parses uk to extract uk'' and a random string r which was used by the hardening procedure to harden the input program P. Next A'' hardens P and creates a program P' using the hardened P and the random string r. Computes the new number of steps

$t' = p(t)$ 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 and obtains $\omega' \leftarrow {\mathcal{U}''}_{uk'',vk''}(\omega, P', t')$ and then performs the following test

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

Since the probability of this event is $\frac{1}{2^{\omega'}}$, i.e. negligible, this is the same requirement as was defined in the unforgeability security specification, therefore the result computed by A'' constitutes a successful forgery according to the definition of $UNF - OUT$ requirement.

Consequently, if A succeeds in the $\mathtt{Exp}_{W,A}^{WB-UNF-OUT-\varphi}(\cdot)$ 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 $WB - UNF - OUT - \varphi$ secure scheme, therefore the lemma follows. □

7.3 Cascade is Robust for WBRPE w.r.t. Programs Unforgeability Specification

Lemma 5. *A cascade* WBRPE *scheme $W = W' \circ W''$, is $UNF - PRG$ secure if at least one of W' or W'' is $UNF - PRG$ secure.*

Proof. Let $W = (\mathcal{G}, \mathcal{H}, \mathcal{U})$, s.t. $W = W' \circ W''$. Given $A = (A_1, A_2)$ against W we construct $A' = (A'_1, A'_2)$ and $A'' = (A''_1, A''_2)$ against W' and W'' respectively s.t.

$$\mathtt{Adv}_{W',A'}^{UNF-PRG}(k) = \mathtt{Adv}_{W,A}^{UNF-PRG}(k) \tag{5}$$

$$\mathtt{Adv}_{W'',A''}^{UNF-PRG}(k) = \mathtt{Adv}_{W,A}^{UNF-PRG}(k) \tag{6}$$

We prove equations (5), (6) in lemmas 6, 7 respectively. The theorem 5 follows. □

Lemma 6. *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 5 holds for any value of φ.*

Proof sketch: The proof of lemma 6 is identical to proof of lemma 3. We will only give an intuition as to the fact that given an output forgery $(\omega, P, t, \langle uk', uk'' \rangle)$ 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 $y \leftarrow {\mathcal{U}'}_{uk',vk'}(\omega', P, t)$ and then performs the following test

$$(y \neq \perp)\ \wedge\ ((uk' \notin UK)\ \vee\ (P \notin P[uk']))$$

Clearly, this is the same requirement as was defined in the unforgeability security specification, therefore the result computed by A' constitutes a successful forgery according to the definition of $UNF - PRG$ requirement, Section 2.2. □

Lemma 7. *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 5 holds for any value of φ.*

Proof sketch: The proof of lemma 7 is identical to proof of lemma **??**. Given an output forgery $(\omega, P, t, \langle uk', uk''\rangle)$ 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 and obtains $\omega' \leftarrow \mathcal{U}''_{uk'',vk''}(\omega, P', t')$ and then performs the following test

$$(\omega' \neq \perp) \ \wedge \ ((uk'' \notin UK) \ \vee \ (P' \notin P'[uk'']))$$

This holds since the program P' generated by A'' is based on c', i.e. the hardening of the program P returned by A. Since the probability of two different programs resulting in the same c' is negligible we conclude that given a forgery for *WBRPE* scheme W, A'' successfully generates a tuple which is a forgery for *WBRPE* scheme W'' according to the definition of $UNF-PRG$ requirement. Therefore, this is the same requirement as was defined in the unforgeability security specification. □

7.4 Cascade is Robust for WBRPEwV w.r.t. Privacy Specification

Proof of Theorem 7.4 Let $WV = (\mathcal{G}, \mathcal{H}, \mathcal{U})$ be a *WBRPEwV* scheme, s.t. $WV = WV' \circ WV''$. Given $A = (A_1, A_2, A_3)$ against WV we present a construction of $A' = (A'_1, A'_2, A'_3)$ (respectively $A'' = (A''_1, A''_2, A''_3)$) against WV' (respectively WV'') s.t. the following holds

$$\mathtt{Adv}^{PRV-VLD}_{W',A'}(k) = \mathtt{Adv}^{PRV-VLD}_{W,A}(k) \tag{7}$$

$$\mathtt{Adv}^{PRV-VLD}_{W'',A''}(k) = \mathtt{Adv}^{PRV-VLD}_{W,A}(k) \tag{8}$$

We prove equations (7), (8) in lemmas 8, 9 respectively. The theorem 7.4 follows. □

Lemma 8. *Given a PPT algorithm $A = (A_1, A_2, A_3)$, we construct a PPT algorithm $A' = (A'_1, A'_2, A'_3)$ s.t. for infinitely many k's equation 7 holds.*

Proof. Let $A = (A_1, A_2, A_3)$ be a PPT algorithm against the combined *WBRPEwV* scheme WV, we construct a PPT algorithm $A' = (A'_1, A'_2, A'_3)$ in 13, against a candidate *WBRPEwV* scheme WV', that uses A and WV' as black boxes. A' simulates the privacy experiment defined in Section 2.3, of the combined *WBRPEwV* scheme WV, by generating the corresponding parameter. Since A' returns the response of A it learns the same information about the remote input, and therefore has the same advantage of success. □

Lemma 9. *Given a PPT algorithm $A = (A_1, A_2)$ against WV, we construct a PPT algorithm $A'' = (A''_1, A''_2, A''_3)$ against WV'' s.t.*

$$\mathtt{Adv}^{PRV-VLD}_{WV'',A''}(k) = \mathtt{Adv}^{PRV-VLD}_{WV,A}(k)$$

Proof. Let $A = (A_1, A_2, A_3)$ be PPT algorithm against the combined *WBRPEwV* scheme WV'', we construct a PPT algorithm $A'' = (A''_1, A''_2, A''_3)$, in 14, against a candidate *WBRPE* scheme WV'' that uses A and WV'' as black boxes. A'' simulates the privacy experiment, in Section 2.3, of the combined *WBRPEwV* scheme WV for A, by generating the corresponding parameters. □

Algorithm 13 Algorithm $A' = (A'_1, A'_2, A'_3)$, that reduces $WB - PRV$ of $WV = WV' \circ WV''$ to that of WV'.

${A'_1}^{REM_k}(1^k, \tau)$
- $(V, t_V, s) \leftarrow A_1^{REM_k}(1^k, \tau)$
- $V'' \leftarrow createV''(V, t_V)$
- $t_{V''} = t_V + 2$
- $\langle hk'', vk'', \mathcal{OVM}'' \rangle \leftarrow \mathcal{GV}''(1^k, V'', t_{V''})$
- $V' \leftarrow createV'(V, t_V, \mathcal{OVM}'')$
- $t_{V'} = t_V + 7$
- $s' \leftarrow (hk'', vk'', \text{OVM"}, V', t_{V'}, V'', t_{V''}, 1^k, \tau)$

return $(V', t_{V'}, \langle s', s \rangle)$

${A'_3}^{REM_k}(\tilde{\omega}, \langle s', s \rangle)$
- $(s', hk, vk, uk', uk'', \tilde{uk'}, V'', t_{V''}, \mathcal{OVM}'') \leftarrow s'$
- $\omega \leftarrow {\mathcal{U}'}_{\tilde{uk'}, vk'}(\tilde{\omega})$

return $A_3^{REM_k}(\omega, \langle uk', uk'' \rangle, s)$

${A'_2}^{REM_k}(hk', vk', \mathcal{OVM}', \langle s', s \rangle)$
- $(hk'', vk'', \mathcal{OVM}'', V', t_{V'}, V'', t_{V''}, 1^k, \tau) \leftarrow s'$
- $hk = \langle hk', hk'', \mathcal{OVM}' \rangle$
- $vk = \langle vk', vk'' \rangle$
- $(c, a_{ADV}, t, l, uk, s) \leftarrow A_2^{REM_k}(\mathcal{OVM}'', hk, vk, s)$
- $\tilde{P} \leftarrow createP'(c)$
- $t' = t + 3$
- $\tilde{\sigma} = (\langle uk', uk'' \rangle, \langle vk', vk'' \rangle, c, t, l)$
- $(\tilde{c'}, \tilde{uk'}) \leftarrow {\mathcal{H}'}_{hk'}(\tilde{P}, \tilde{\sigma})$
- $s' \leftarrow s' \cup (hk, vk, uk', uk'', \tilde{uk'}, V'', t_{V''}, \mathcal{OVM}'')$

return $(\tilde{c'}, \langle uk', uk'' \rangle, a_{ADV}, t', l, \langle s', s \rangle)$

Algorithm 14 Algorithm $A'' = (A''_1, A''_2)$, that reduces $WB - PRV$ of $WV = WV' \circ WV''$ to that of WV''.

${A''_1}^{REM_k}(1^k, \tau)$
- $(V, t_V, s) \leftarrow A_1(1^k, \tau)$
- $V'' \leftarrow createV''(V, t_V)$
- $t_{V''} = t_V + 2$
- $s' \leftarrow (V, t_V, 1^k, \tau)$

return $(V'', t_{V''}, \langle s', s \rangle)$

${A''_3}^{REM_k}(\omega, uk'', \langle s', s \rangle)$
- $(s', hk, vk, uk', uk'', \tilde{uk'}, V'', t_{V''}, \mathcal{OVM}'') \leftarrow s'$

return $A_3^{REM_k}(\omega, uk = \langle uk', uk'' \rangle, s)$

${A''_2}^{REM_k}(hk'', vk'', \mathcal{OVM}'', \langle s', s \rangle)$
- $(V, t_V, 1^k, \tau) \leftarrow s'$
- $V' \leftarrow createV'(V, t_V, \mathcal{OVM}'')$
- $t_{V'} = t_V + 7$
- $\langle hk', vk', \mathcal{OVM}' \rangle \leftarrow \mathcal{GV}'(1^k, V', t_{V'})$
- $hk = \langle hk', hk'', \mathcal{OVM}' \rangle$
- $vk = \langle vk', vk'' \rangle$
- $(c, a_{ADV}, t, l, uk, s) \leftarrow A_2^{REM_k}(\mathcal{OVM}'', hk, vk, s)$
- $uk', uk'' \leftarrow uk$
- $s' \leftarrow s' \cup (hk, vk, uk, \tilde{uk'}, V'', t_{V''}, \mathcal{OVM}'')$

return $(c, a_{ADV}, t, l, uk'', \langle s', s \rangle)$

8 Related Works

In this section we review the existing works in white-box security, particularly focusing on obfuscation, white-box encryption and secure functions evaluation. Obfuscation is the main practical tool employed to harden software for remote execution. White-box encryption constitutes a special case of obfuscation and is targeted at hiding secret keys in cryptographic implementations. Secure function evaluation (encrypted computation), presents theoretical constructions, that are not applicable for practical implementations. We give main results in each of the aforementioned fields, and discuss suitability of each for practical implementations.

8.1 Obfuscation

Software obfuscation is widely deployed by practitioners when trying to harden software for execution in remote untrusted environments, and is aimed at protecting the executable program against reverse engineering, i.e. hides the secrets, e.g. algorithm, data, keys, in software implementations. For instance, in *DRM* protected applications, the secret decryption key has to remain hidden from the user that may attempt to analyse or to reverse engineer the software, e.g. Collberg and Thomborson [13]. On a high level, obfuscation uses heuristic techniques to Obfuscation the program, such that the resulting obfuscated program computes the same function as the original program, however it is hard to analyse. The problem with this approach is that it is difficult to estimate the security of the resulting obfuscated program, i.e. how much harder is the resulting obfuscated program as opposed to the original one. Furthermore, obfuscation techniques do not assume a clear attack model, and are not based on rigorous cryptographic principles. Obfuscation is a controversial technique among computer scientists for its lack of ability to provide provable security and the fact that it currently relies on "fuzzy security" only. In addition, it is not known whether obfuscation can protect basic modular programs.

It is conjectured that obfuscation alone does not provide a solution to remote program execution concept. For instance, obfuscation does not hide the output, therefore allowing the host to learn the result of the computation, which as claimed by Algesheimer et al. [3], and should be used in tandem with other techniques to obtain provably secure protocols and frameworks for execution of programs in remote untrusted environments, and in particular it is not a substitution to these techniques.

Program obfuscation was formally defined by Barak et al. [4] based on black box simulation, which also proved that obfuscation is theoretically impossible for general purpose functions and for some specific functions. Following the definition presented by Barak *et al.*, Wee [39], presented a positive result for point functions obfuscation. The definition presented by Barak et al. [4] was further extended to include auxiliary inputs by Goldwasser and Kalai [18], which in addition proved that the construction presented by Wee [39] holds in their model. Furthermore, the prominent impossibility result of Barak et al. [4] also holds in their weaker model.

In contrast to the impossibility result of Barak et al. [4] and Goldwasser and Kalai [18], there are other positive solutions, e.g. an NP-hardness result of Wang [38], Ogiso et al. [31], a PSpace hardness result of Chow et al. [9]. There are also alternative weaker definitions of obfuscation, e.g. Hohenberger and Rothblum [24], that present a positive obfuscation result for cryptographic re-encryption functionality, obfuscation for access control Lynn et al. [29], also a weaker definition of Best Possible Obfuscation in Goldwasser and Rothblum [19], which makes a relaxed requirement

that the obfuscated program leaks as little information as any other program with the same functionality, present a separation between black box and best possible obfuscation, and show tasks which can be achieved under this new definition.

8.2 White-Box Encryption

In recent years, a number of cryptographic implementations have appeared for symmetric key ciphers such as *Data Encryption Standard (DES)* Standard [36] and *Advanced Encryption Standard (AES)* Daemen and Rijmen [14], that have claimed to be secure in a white-box model. More specifically, the white box *AES* Chow et al. [11] and the white-box *DES* Chow et al. [10]. The approach is to integrate the key into the encryption algorithm so that the algorithm performs the encryption properly but the key is never made explicit. For a system that protects keys from white-box attackers, the secret key is inaccessible.

It is mostly employed in the protection of multimedia players, which mainly employ symmetric cipher schemes, e.g. AES Daemen and Rijmen [14], DES Standard [36], and decode the content with a secret key, that is embedded in the player. In order to use a file protected by such an algorithm, the attacker must either use the authorized program or reverse engineer the entire algorithm. Since reverse engineering an entire algorithm is a more involved task than finding a secret key, this makes circumventing copy protection much more difficult.

However, the white-box cryptography has a limited applicability since it requires a large memory space, and imposes a heavy performance overhead. In addition, the proposed white box cryptography solutions were subsequently broken Billet et al. [6], Goubin et al. [20].

8.3 Mobile Code Cryptography (Encrypted Computation)

Mobile code cryptography, also called the encrypted function computation technique, aims at providing black box security by employing provably secure cryptographic techniques. This approach allows the program to be executed securely on a remote untrusted host by transforming the mobile code into an encrypted form, thus obtaining encrypted executable program that consist of instructions and operate on encrypted inputs, such that the remote host cannot inspect the original program. This is an extension of the idea of computing with encrypted data (CED), presented in Abadi and Feigenbaum [1] and is a basic technique for secure distributed computing, where the involved parties wish to perform a computation of their private inputs, such the inputs remain secret and not revealed to other parties participating in the computation.

Sander and Tschudin [34], S and Tschudin [32], initiated mobile code research and were the first to identify the possibility of securely executing a code on a remote untrusted host, by proposing an application of encrypted computation techniques to the problem of protecting intellectual property, secret functions and mobile code from malicious hosts. They pointed out that protocols for secure multi-party computation could be useful in the design of a software only solution to protect mobile code against malicious hosts. In particular, they found that polynomial functions can be encrypted for non-interactive evaluation if an algebraic homomorphic trapdoor one-way functions exist.

More specifically, Sander and Tschudin presented a non-interactive Computing with Encrypted Functions protocol for computing polynomials and rational functions, using homomorphic encryption scheme. However their solution is limited to evaluation of polynomials and rational functions and is highly inefficient for practical use. Similar solutions were presented in Lee et al. [27], Kotzanikolaou et al. [26].

This approach can be further employed by function hiding as is extended by Sander and Tschudin [33], which essentially means that the result of the computation is returned to the remote host upon receipt of the encrypted result by the originator. A subsequent work by Sander *et. al.* Sander et al. [35] presented a non-interactive computing with encrypted data protocol for all NC^1 functions. Based on that, a non-interactive computing with encrypted functions protocol can be implemented by letting client's private input be its function f and server's function be a universal circuit. However due to the logarithmic limitation on the depth of the circuit being evaluated, its application is limited.

Although mobile code cryptography provides black box security, full protection is difficult to obtain, and in particular, existing techniques and schemes exhibit various problems for practical applications, such as lack of efficiency, limitations to specific functions, and more. When considering practical applications efficiency considerations are of high importance. However, in homomorphic encryption scheme for polynomial functions the possible number of terms in a function is exponential to the number of inputs, therefore the encrypted function to be transfered will be extremely large. An additional common limitation of both secure distributed computation and encrypted computation schemes is the representation of programs as functions. When a program is represented by a function the encoding size may be exponential, thus increasing both computation and communication complexity. There are also technical drawbacks: in encrypted computation, it is hard to find encryption schemes that can transform arbitrary functions to their encrypted executable version. More importantly, no one has yet discovered an algebraically homomorphic encryption scheme. However, additively homomorphic encryption schemes are known to exist, thus making encrypted computation of polynomials possible.

Other approaches are based on Yao's secure function evaluation protocol Yao [40], and present solutions to protect program's code and data as well as host's data, and it is more powerful in terms of the type of functions it can compute. Some works combine secure circuit evaluation in tandem with oblivious transfer, e.g. Cachin et al. [7], Algesheimer et al. [3], Zhong and Yang [41]. However, there are inherent disadvantages, e.g. Yao's Yao [40] non-interactive protocol, in order to reduce interactiveness, leaks the whole circuit structure. An additional common limitation of both secure distributed computation and encrypted computation schemes is the representation of programs as functions. When a program is represented by a function the encoding size may be exponential, thus increasing both computation and communication complexity.

9 Conclusions and Open Questions

In this work we investigate combiners for *white-box* primitives. More specifically, we present a robust (1,2)-cascade combiner for *WBRPE* scheme and for *WBRPEwV* scheme with privacy, and present constructions, along with reductions to the underlying candidates. We leave the following directions below for further research in robust combiners for white-box security.

As we discussed in Section 8, the existing techniques in white-box security are insufficient when it comes to securing practical applications and in particular no provably secure candidates are known to exist. We therefore employ robust combiners to fortify white-box constructions.

- Our construction is exponential in the number of the candidate primitives, i.e. we generate a program for each candidate and then the programs are executed inside each other, thus multiplying the running time complexity, (cf. robust copy combiner for unforgeability property, which complexity is an additive complexity of the candidates). We believe this to be an optimal construction of a combiner for white-box primitives. An interesting question to consider is whether it is possible to achieve a more efficient combiner construction than the one presented in this paper or to show a lower bound on the complexity time of the combiner. Alternately, an equivalently important result is to prove that the combiner presented here is optimal.
- Another important aspect is to consider a construction of the robust combiner without assuming the correctness of the candidate schemes. More specifically, in our constructions we assume that the underlying candidate schemes are correct. However, not relying on correctness assumption is critical for many applications, and especially important for combined constructions, where one of the goals is to prevent erroneous software implementations and design bugs, or intentionally bogus implementations and trapdoors. Hence, to prove security of a cascade construction given at least one secure and correct candidate poses an interesting challenge.
- In addition, we assume that the validation requirement holds, i.e. that the $\mathcal{OVM}$, prior to execution, always runs the validation program, defined during the generation phase, on the input program. The input program is executed only if validation passes. However, an erroneous $\mathcal{OVM}$ may not run the validation procedure at all. It is a challenge to present a combined construction without relying on the correctness of $\mathcal{OVM}$ and in particular on the validation property of both candidates.

Bibliography

[1] M. Abadi and J. Feigenbaum. Secure circuit evaluation. *Journal of Cryptology*, 2(1):1–12, 1990.

[2] R. Agrawal and R. Srikant. Privacy-preserving data mining. In *Proc. of the ACM SIGMOD Conference on Management of Data*, pages 439–450. ACM Press, May 2000. ISBN 1-581-13218-2. URL `citeseer.ist.psu.edu/agrawal00privacypreserving.html`.

[3] 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.

[4] 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.

[5] M. Bellare and P. Rogaway. The security of triple encryption and a framework for code-based game-playing proofs. *Advances in Cryptology–EUROCRYPT*, 4004:409–426, 2006.

[6] O. Billet, H. Gilbert, and C. Ech-Chatbi. Cryptanalysis of a white box AES implementation. *Selected Areas in Cryptography*, 3357:227–240.

[7] 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`.

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

[9] S. Chow, Y. Gu, H. Johnson, and V.A. Zakharov. An Approach to the Obfuscation of Control-Flow of Sequential Computer Programs. *Information Security: 4th International Conference, ISC 2001, Malaga, Spain, October 1-3, 2001: Proceedings*, 2001.

[10] Stanley Chow, Philip A. Eisen, Harold Johnson, and Paul C. van Oorschot. A white-box DES implementation for DRM applications. In Joan Feigenbaum, editor, *Digital Rights Management Workshop*, volume 2696 of *Lecture Notes in Computer Science*, pages 1–15. Springer, 2002. ISBN 3-540-40410-4.

[11] Stanley Chow, Philip A. Eisen, Harold Johnson, and Paul C. van Oorschot. White-box cryptography and an AES implementation. In *SAC '02: Revised Papers from the 9th Annual International Workshop on Selected Areas in Cryptography*, pages 250–270, London, UK, 2003. Springer-Verlag. ISBN 3-540-00622-2.

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

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

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

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

[16] Gertner, Ishai, Kushilevitz, and Malkin. Protecting data privacy in private information retrieval schemes. *JCSS: Journal of Computer and System Sciences*, 60, 2000.

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

[18] S. Goldwasser and Y.T. Kalai. On the impossibility of obfuscation with auxiliary input. *FOCS*, 5:553–562, 2005.

[19] S. Goldwasser and G.N. Rothblum. On Best-Possible Obfuscation. *LECTURE NOTES IN COMPUTER SCIENCE*, 4392:194, 2007.

[20] L. Goubin, J.M. Masereel, and M. Quisquater. Cryptanalysis of white box DES implementations. *Proceedings of the 14th Annual Workshop on Selected Areas in Cryptography*, 2007.

[21] D. Harnik, J. Kilian, M. Naor, O. Reingold, and A. Rosen. On Robust Combiners for Oblivious Transfer and other Primitives. In *Advances in Cryptology – Eurocrypt 2005*, volume 5, pages 96–113. Springer, 2005.

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

[23] Amir Herzberg, Haya Shulman, Amitabh Saxena, and Bruno Crispo. Towards a theory of white-box security. Cryptology ePrint Archive, Report 2008/087, 2008. `http://eprint.iacr.org/`.

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

[25] NM Karnik and AR Tripathi. Design issues in mobile agent programming systems. *Concurrency, IEEE [see also IEEE Parallel & Distributed Technology]*, 6(3):52–61, 1998.

[26] P. Kotzanikolaou, M. Burmester, and V. Chrissikopoulos. Secure Transactions with Mobile Agents in Hostile Environments. *Information Security and Privacy: 5th Australasian Conference, ACISP'2000, Brisbane, Australia, July 10-12, 2000: Proceedings*, 2000.

[27] H. Lee, J. Alves-Foss, and S. Harrison. The use of encrypted functions for mobile agent security. *System Sciences, 2004. Proceedings of the 37th Annual Hawaii International Conference on*, pages 297–306, 2004.

[28] Qiong Liu and Reihaneh Safavi-naini. Digital rights management for content distribution, January 01 2003. URL `http://citeseer.ist.psu.edu/560657.html;http://www.itacs.uow.edu.au/research/smicl/publications/aisw2003.pdf`.

[29] B. Lynn, M. Prabhakaran, and A. Sahai. Positive Results and Techniques for Obfuscation. *Advances in cryptology-EUROCRYPT 2004: International Conference on the Theory and Applications of Cryptographic Techniques, Interlaken, Switzerland, May 2-6, 2004: Proceedings*, 2004.

[30] Remo Meier, Bartosz Przydatek, and Jürg Wullschleger. Robuster combiners for oblivious transfer. In Salil P. Vadhan, editor, *proc. of 4th Theory of Cryptography Conference, TCC 2007*, volume 4392 of *Lecture Notes in Computer Science*, pages 404–418. Springer, February 2007. ISBN 3-540-70935-5. URL `http://dx.doi.org/10.1007/978-3-540-70936-7_22`.

[31] T. Ogiso, Y. Sakabe, M. Soshi, and A. Miyaji. Software Tamper Resistance Based on the Difficulty of Interprocedural Analysis. *3rd Workshop on Information Security Applications (WISA 2002), Korea, August*, 2002.

[32] Tomas S and Christian F. Tschudin. Protecting mobile agents against malicious hosts, November 27 1997. URL `http://citeseer.ist.psu.edu/326522.html;http://www.icsi.berkeley.edu/~sander/publications/MA-protect.ps`.

[33] T. Sander and C.F. Tschudin. On software protection via function hiding. *Information Hiding*, pages 111–123, 1998.

[34] Tomas Sander and Christian F. Tschudin. Towards mobile cryptography, June 08 1998. URL `http://citeseer.ist.psu.edu/330605.html;http://www.icsi.berkeley.edu/~tschudin/ps/ieee-sp98.ps.gz`.

[35] Tomas Sander, Adam Young, and Moti Yung. Non-interactive cryptocomputing for NC 1. In *IEEE Symposium on Foundations of Computer Science*, pages 554–567, 1999. URL `citeseer.ist.psu.edu/sander99noninteractive.html`.

[36] D.E.S.E. Standard. National Bureau of Standards (US). *Federal Information Processing Standards Publication*, 46.

[37] Verykios, Bertino, Fovino, Provenza, Saygin, and Theodoridis. State of the art in privacy preserving data mining. *SIGMODREC: ACM SIGMOD Record*, 33, 2004.

[38] C. Wang. *A Security Architecture for Survivability Mechanisms*. PhD thesis, University of Virginia.

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

[40] A.C. Yao. How to generate and exchange secrets. *Proceedings of the 27th IEEE Symposium on Foundations of Computer Science*, pages 162–167, 1986.

[41] S. Zhong and Y. Yang. Verifiable distributed oblivious transfer and mobile agent security, 2003. URL `citeseer.ist.psu.edu/zhong03verifiable.html`.