Towards a Theory of White-Box Security

Amir Herzberg[1], Haya Shulman[1], Amitabh Saxena[2], and Bruno Crispo[3]

[1] Bar Ilan, Ramat-Gan, 52900, Israel
amir.herzberg@gmail.com, haya.shulman@gmail.com
[2] International University in Germany, Bruchsal 76646, Germany
saxena.amitabh@gmail.com
[3] University of Trento, Italy
crispo@disi.unitn.it

Abstract. *Program hardening* for secure execution in remote untrusted environment is an important yet elusive goal of security, with numerous attempts and efforts of the research community to produce secure solutions. Obfuscation is the prevailing practical technique employed to tackle this issue. Unfortunately, no provably secure obfuscation techniques currently exist. Moreover, *Barak et. al.*, showed that not all programs can be obfuscated. Theoretical research exhibits provably secure albeit inefficient constructions, e.g. using tools from encrypted domain.

We present a rigorous approach to software execution in remote environment based on a new white box primitive, the White Box Remote Program Execution (WBRPE), whose security specifications include confidentiality and integrity of both the local and the remote hosts. *WBRPE* can be used for many applications, e.g. grid computing, digital rights management, mobile agents.

We then present a construction of a specific program such that if there exists a secure *WBRPE* for that program, then there is a secure *WBRPE* for *any* program, reducing its security to the underlying *WBRPE* primitive. The security of *WBRPE* construction is established by reduction among two white box primitives and it introduces new techniques of programs manipulation.

1 Introduction

Ensuring secure execution of programs in remote untrusted environment is of high theoretical and practical importance and is required for a wide range of applications, e.g., digital rights management (DRM), grid computing, private information retrieval, mobile agents, network gaming, Voice over IP (VoIP). In remote program execution a program leaves the site of the originator and is transferred to the remote host for execution, defining a *white-box security* model. In particular, the originator loses all control over its software, which is completely exposed to the executing environment, and the entity controlling the execution environment obtains full access to the program, and can observe and manipulate the execution, code and data. This is in contrast to traditional cryptography, which assumes a trusted platform, i.e., a *black-box*, on which secrets, e.g., secret keys, can be stored. In black-box security all the computations are performed on a trusted platform, and the secret keys never leave its boundaries. More importantly, attackers obtain a black-box access to the cryptographic implementation and can only

D. Gritzalis and J. Lopez (Eds.): SEC 2009, IFIP AICT 297, pp. 342–352, 2009.

observe an input/output behaviour, but cannot access the code or data, or observe the execution inside the platform.

In hardware based approach, an additional hardware that constitutes a secure trusted platform, is supplied, e.g., a smartcard or a trusted third party in [1], on which the secret data can be stored and the computations involving it performed. Hardware based approach produces solutions in *black box security* model, in which an attacker cannot access and observe the internals of the hardware, e.g. secret keys inside it, and can only control the input/output behaviour of the system.

Although applications that employ hardware benefit from high security promises, there are disadvantages, such as high cost, vulnerability to side channel attacks, unreliability and inflexibility of the hardware. In addition the security completely depends on the trust relationship with the additional hardware, thus making it inapplicable to many useful scenarios. Furthermore, in practice hardware alone is often not enough, since even hardware based solutions rely on software to accomplish the overall security. Therefore in order to enable a variety of practical applications secure software white-box techniques should be provided.

In addition to practical importance, understanding the level of security that can be attained by employing software only techniques is intriguing on its own, especially due to prevailing belief that it is difficult to provide a reasonable level of security by employing software only approach, let alone a level of security comparable to the one in black box security. In this work, we present a new basic candidate white-box security building block, the *White-Box Remote Program Execution (WBRPE)* for remote program execution in hostile environment, along with definitions and game-based security specifications. We present a construction based on *WBRPE* scheme, and establish its security by reduction.

It is important to identify weakest possible primitives for white-box security, e.g., by failed cryptanalysis, which could serve as basic building blocks for provably secure protocols and schemes. More specifically, the security of the scheme would be reduced to the security of the building block that underlies the construction. This is similar in nature to traditional cryptography where few basic, simple building blocks are employed in constructions of cryptographic schemes and primitives.

Security of protocols is established by reduction to the basic building blocks. The motivation is that the cryptanalysis proven standard should be simple and basic, so that it is easy to test its security and the security of the overall construction that uses it. We propose the *WBRPE* as a candidate white-box security building block, which could be employed to develop and analyse well-defined white-box security constructions. Existing practical primitives are proprietary, and their security relies on vague assumptions.

The *WBRPE*, in Figure 1, is comprised of two phases, the generation phase, run by offline trusted third party, and the protocol execution phase, between the local and the remote hosts. The trusted party generates the parameters of the scheme, i.e. the keys which are sent to local host, and the *OVM*, which is transfered to the remote host. The *OVM* emulates a trusted platform, and executes the input programs supplied by the local host in a secure manner. The local host uses the keys to harden programs which it sends to remote host for execution. The *OVM* receives the hardened program, and possibly an input of the remote host. It has the corresponding keys to unharden and execute the

program, and then harden the result of the computation. The remote host returns the hardened result to the local host. We require that the local host learns only the result of the computation, while the remote host learns nothing at all.

1.1 Existing Works

Obfuscation. Is a candidate building block for white box security, which received substantial attention from theoreticians and practitioners. An obfuscator $\mathcal{O}$ is an efficient compiler that transforms a program P into a hardened program $\mathcal{O}(P)$, which pertains the functionality of the original program but is equivalent to black-box access to P, i.e. should be hard to analyse and to reverse engineer.

Obfuscation is the prevailing practical approach to software hardening, and was also investigated by theoreticians. However in both theory and practice, obfuscation exhibited insufficient results. The impossibility result by [2] states that there does not exist a general obfuscator for any program. Although there are some positive results, e.g. [6], these are restricted and do not suffice for practical applications. In addition, experts in practical obfuscation, e.g. [10], cannot say whether obfuscators can protect even simple programs, e.g. to hide intermediate state of programs.

White-Box Cryptography. (A special case of obfuscation) aims at protecting secret data embedded inside software implementations of cryptographic algorithms, by integrating a secret key in the cryptographic algorithms, thus preventing from attacker, which controls the execution environment, and may be a legitimate user, from extracting the keys for use on a different platform.

A number of cryptographic implementations have appeared for symmetric key ciphers such as [15] and [11], that have claimed to be secure in a white-box model. More specifically, the white box AES in [9], and the white-box DES in [8]. So far, proposed white box cryptography solutions were subsequently broken [4,13,16]. The *WBRPE* scheme that we present can be seen as an extension of white-box cryptography.

Mobile Code Cryptography. It is possible to employ theoretical tools from two party computation protocols, to produce provably secure white-box security schemes. The central approaches used to tackle two party computation scenario are secure function evaluation, computing with encrypted data, and encrypted function. One of the earliest techniques for two party computation, due to [17], is via encrypted circuit construction and evaluation. A solution to mobile code, for computing all polynomial time functions efficiently, based on encrypted circuit evaluation, is presented in [5] using tools from two-party computation. However, their solution only provides for privacy of one of the inputs, but not both. As a result if the input of one participant is a program, it may expose the input of the other participant, e.g. if the program is computing an identity function. In [14] they construct a practical implementation of two party-secure function evaluation, thus showing a practical feasibility of encrypted circuits evaluation approach.

1.2 White Box Remote Program Execution (WBRPE)

In this work, we propose the *White Box Remote Program Execution (WBRPE)* scheme, as a candidate white-box security building block. *WBRPE* can be employed to facilitate a variety of applications, see Section 1.4.

In *Remote Program Execution*, programs are sent by a *local host* (a.k.a. the originator) for execution on a *remote host*, and possibly use some data available to the *remote host*. The local and the remote hosts may be with conflicting interests, therefore the 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* should satisfy confidentiality and integrity, 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 a hardening key *hk*, and a program, which we call the *obfuscated virtual machine (OVM)*.
- The hardening key *hk* is used by the hardening procedure to harden, e.g. encrypt and/ or authenticate, the input programs.
- The obfuscated virtual machine *OVM* receives a hardened input program along with input from the remote host. It decodes the hardened program, e.g. decrypts and/ or validates it, and returns the encoded result, e.g., encrypted and/or authenticated, of the program applied to the inputs.
- The unhardening procedure unhardens, e.g. decrypts and validates the result received from the remote host.

1.3 White Box RPE for ALL Programs

The negative result by Barak *et al.* [2], shows that an obfuscator for all programs does not exist, however this result does not imply that there cannot be alternative hardening schemes which would work for any program. In particular, is there a *WBRPE* for all programs? To address this question we present a specific program, denoted *UP* (for

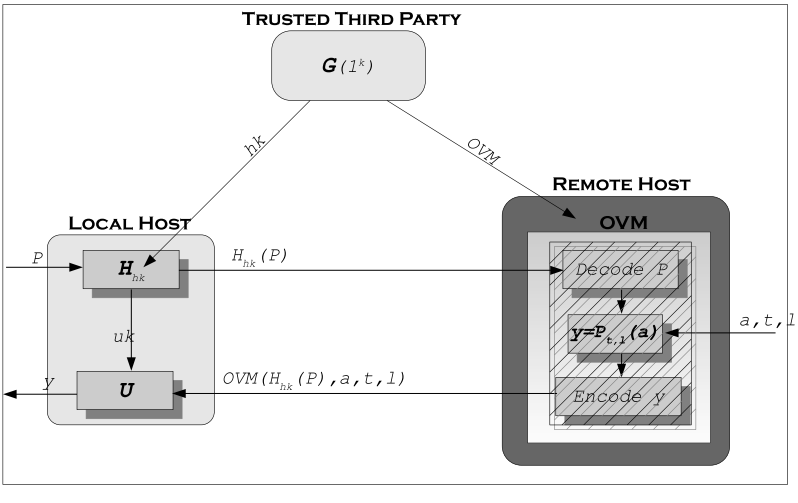

Fig. 1. *WBRPE* scheme

universal program), with parameter K (key). Given a *WBRPE* scheme that works for the family of universal programs $\{UP_K\}$, we present a *'Universal' WBRPE* scheme that works for any program, i.e. provides the security specifications of *WBRPE* for *any* input program.

1.4 Applications of WBRPE

Following are some potential applications of *WBRPE*:

- Mobile agent, that traverses the Web, searching and purchasing goods on behalf of its owner. The agent may include secret data, e.g., secret signing/decryption keys, credit card number, and therefore needs to be protected from a possibly hostile execution platform or from other agents, which e.g., may try to learn the secret information of the agent, or modify its execution.
- In grid computing a large number of users (nodes) donate their idle CPU cycles to perform computation on behalf of the local host. *WBRPE* can ensure that the confidentiality and integrity of the program and input data are not violated.
- In P2P systems, e.g., for VoIP systems (such as Skype), the client code contains secrets (e.g., cryptographic keys and proprietary protocols) that, if leaked to the remote host, would e.g., allow users to make free calls.
- Protection of intellectual property, e.g, music and programs.
- Typically applications based on the setting of online database, e.g. the model of Private Information Retrieval in [7], involve two parties, a server which holds the database and a client who wishes to query the database. The privacy and the integrity of both the local and the remote hosts should be provided. *WBRPE* can be applied directly to map the security requirements of applications based on online databases. In *WBRPE* scheme, the client is the local host and the server is the remote host. The input supplied by the client is a query, and the remote input of the server is a database, and the client wishes to compute the result of its query on the database. The privacy and the integrity of both inputs of the client and the server are preserved, since the server cannot observe the queries submitted by the client, further since the database is queried inside the *OVM* the server cannot observe the process of the computation.

2 White-Box RPE Definitions

A *WBRPE* scheme W is comprised of three efficient algorithms, (G,H,U) for generation, hardening and unhardening, respectively. The generation procedure G generates the obfuscated virtual machine *OVM* and the hardening key hk. The hardening procedure applied on some input program, computes the hardened program, e.g. encryption and/ or authentication of 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 P on remote input a, and returns the (hardened) output $P(a)$. The local host, upon receipt the hardened

output, applies the unhardening procedure with the unhardening key, to obtain the final result of the computation.

Given a Turing machine $P \in \mathbb{TM}$, where $\mathbb{TM}$ is a set of all Turing machines, let $P(a)$ denote a value of the computation of P on a. We introduce a time parameter, to hide the time that it takes each program to execute, and the length parameter to hide the length of the result. 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 $W = \langle G,H,U\rangle$ of PPT algorithms s.t. for all $(hk,OVM) \xleftarrow{R} G(1^k), a \in \{0,1\}^*, P \in \mathbb{TM}$, $OVM \in PPT$, $t,l \in \mathbb{N}$ and $(c,uk) \leftarrow H_{hk}(P)$, holds: $P_{t,l}(a) = U_{uk}(OVM(c,a,t,l))$.*

2.1 Indistinguishability of the Local Inputs Specification

The first security specification we consider is to hide the contents of the input programs from the remote host. To ensure local inputs privacy we employ a variation of the indistinguishability experiment for encryption schemes [12]. We specify the indistinguishability definition w.r.t. a PPT algorithm $A = (A_1,A_2)$, denoting by HO the hardening oracle which the algorithm A obtains access to, during the indistinguishability experiment. The experiment is described in detail in Figure 1, we now give an informal definition. As its first step the experiment generates the keys and the obfuscated virtual machine. Next it invokes the adversarial algorithm with an *OVM* in an input, and provides it with an oracle access to the hardening functionality for its hardening queries. Each application of the hardening procedure generates a hardened program and a one time unhardening key. Eventually the adversary outputs two programs of equal size. The experiment tosses a bit b and one of the programs is subsequently hardened. During the second phase the adversary keeps an oracle access to 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.

In the sequel we introduce a flag $\varphi \in \{PK,SK\}$, and when $\varphi = PK$ we refer to an asymmetric *WBRPE*, while $\varphi = SK$ denotes a symmetric *WBRPE*. When $\varphi = PK$ the adversary receives the public hardening key *hk* in an input and can harden the programs by itself.

Definition 2 (Indistinguishability). *Let $W = (G,H,U)$ be a* WBRPE *scheme and let $A = (A_1,A_2)$ be a pair of PPT algorithms. For $k \in \mathbb{N}$, $r \in \{0,1\}^*$ we define the advantage of the adversary A in the WB-IND-CPA experiment as follows:*

$$\boldsymbol{Adv}^{WB\text{-}IND\text{-}CPA\text{-}\varphi}_{W,A}(k) = 2 * \Pr[\boldsymbol{Exp}^{WB\text{-}IND\text{-}CPA\text{-}\varphi}_{W,A}(k) = 1] - 1$$

Where the probabilities are taken over G, H and A. The experiment $\mathbf{Exp}^{WB\text{-}IND\text{-}CPA\text{-}\varphi}_{W,A}(k)$ is defined in Experiment 1. A *WBRPE* scheme W is *WB-IND-CPA-*φ secure if the advantage function $\mathbf{Adv}^{WB\text{-}IND\text{-}CPA\text{-}\varphi}_{W,A}(\cdot)$ is negligible over all PPT adversarial algorithms A. In private key *WBRPE* there is a secret shared key *hk* between the *OVM* and the local host. This *hk* key is employed by the local host to harden programs and by the

OVM to subsequently unharden them for execution. This implies that there is a unique *OVM* for every local host. In public key *WBRPE* the hardening key *hk* is public, which the attacking algorithm obtains in an input, and there is a corresponding unhardening key embedded inside the *OVM*. Namely, everyone can harden programs and send to *OVM* for execution, and only the *OVM* can unharden the programs, which implies the asymmetry. The obvious advantage of the asymmetric *WBRPE* is in its flexibility, i.e. new hosts can join the system without any effort, e.g. a marketplace scenario where everyone can work with one central remote host and the same *OVM*.

Experiment 1. The indistinguishability $\mathbf{Exp}_{W,A}^{WB\text{-}IND\text{-}CPA\text{-}\varphi}(k)$ and unforgeability $\mathbf{Exp}_{W,A}^{WB\text{-}UNF\text{-}\varphi}(k)$ experiments. Where *HO* is a hardening oracle that the algorithm *A* obtains access to during the course of the experiments.

$\mathbf{Exp}_{W,A}^{WB\text{-}IND\text{-}CPA\text{-}\varphi}(k)$
- $\langle hk, OVM\rangle \leftarrow G(1^k)$
- $(P_0, P_1, s) \leftarrow A_1^{HO_{hk}(\cdot,\varphi)}(1^k, OVM, hk)$
- $b \in \{0,1\}^k$
- $(c_b, uk_b) \leftarrow H_{hk}(P_b)$
- $b' \leftarrow A_2^{HO_{hk}(\cdot,\varphi)}(c_b, s)$
- if $((b = b') \wedge (|P_0| = |P_1|))$ {return 1}
- return 0

$HO_{hk}(P, \varphi)$
- if $(\varphi = PK)$ {return (hk)}
- return $(H_{hk}(P))$

$\mathbf{Exp}_{W,A}^{WB\text{-}UNF\text{-}\varphi}(k)$
- $\langle hk, OVM\rangle \leftarrow G(1^k)$
- $(P, s) \leftarrow A_1^{HO_{hk}(\cdot,\varphi)}(1^k, OVM)$
- $(c, uk) \leftarrow H_{hk}(P)$
- $(\omega, t) \leftarrow A_2^{HO_{hk}(\cdot,\varphi)}(c, s)$
- $y \leftarrow U_{uk}(\omega)$
- if $(y = \perp)$ {return 0}
- if $(\forall a \in \{0,1\}^*, y \neq P_{t,|y|}(a))$ {
  - return 1
- }
- return 0

2.2 Unforgeability Specification

In some scenarios, e.g. shopping mobile agent, a remote host may try to change the result of the programs sent by the originator, e.g. such that instead of looking for the best offer the agent purchases the most expensive item. Our goal is to circumvent adversarial attempts to forge the result output by the scheme. This is captured by the unforgeability specification, based on unforgeability experiment which we present below. The unforgeability experiment applies the generation procedure and obtains hardening key *hk*, and *OVM*. It then invokes the adversary with oracle access to hardening functionality, and with the *OVM* as input. Eventually, the adversary outputs the forgery, i.e. the hardened result of the computation, denoted ω, an input program P, and the unhardening key uk. The experiment applies the unhardening procedure U on ω, P and t, and obtains the result of the computation y. If y is valid, then the experiment checks if it is a forgery for any t and a, and if yes, returns 1, i.e. the adversary successfully generated a forgery, otherwise returns 0, the adversary failed.

In the asymmetric *WBRPE* everyone can harden programs for execution. After recovering the result by applying the unhardening procedure, we cannot know what input

program was hardened to generate the result, and the forgery in this case means that the output is not a result of the computation of the input program on any remote input. Since the adversary has the public hardening key *hk*, it can harden programs by itself. The trivial solution to this issue is to supply the program to the local host as part of the unhardening key *uk*. Local host would then compare the program returned to the program supplied as part of *uk*. However in case of a security specification which requires to keep the input program secret from other remote recipients in this solution we expose the input program, and thus cannot achieve programs privacy from remote recipients. Therefore in asymmetric *WBRPE* a forgery is a generation of a valid result ω such that there does not exist a program P, which could result in $y \leftarrow U_{uk}(\omega)$ on any remote input a, i.e. $\forall a\ y \neq P_{t,|y|}(a)$.

In the symmetric *WBRPE*, the adversary obtains an oracle access to the hardening procedure. If the adversary did not query the hardening oracle on the program for which the result was generated, then the adversary wins the experiment. The experiment keeps a vector $Q[..]$, with queried programs and the respective unhardening keys output along with the hardening upon each query. 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).

Definition 3 (Unforgeability). *Let $W = (G,H,U)$ be a* WBRPE *scheme and let A be a PPT algorithm. For $k \in \mathbb{N}$, $\varphi \in \{PK, SK\}$ we define the advantage of the adversary A in the unforgeability experiment as follows:*

$$\boldsymbol{Adv}_{W,A}^{WB\text{-}UNF\text{-}\varphi}(k) = \Pr[\boldsymbol{Exp}_{W,A}^{WB\text{-}UNF\text{-}\varphi}(k) = 1]$$

Where $\mathbf{Exp}_{W,A}^{WB\text{-}UNF\text{-}\varphi}(k)$ and the hardening oracle are defined in Experiment 1. A *WBRPE* scheme W is *WB-UNF-*φ secure, if the advantage $\mathbf{Adv}_{W,A}^{WB\text{-}UNF\text{-}\varphi}(\cdot)$ is a negligible function for all PPT adversarial algorithms A.

3 Universal WBRPE

In this section we show that if there exists a *WBRPE* scheme that satisfies the security specifications for a *specific* family of universal programs, UP then there exists a *Universal WBRPE* scheme that satisfies the security specifications for every program. More specifically, we present the construction of the *Universal WBRPE* scheme given a *WBRPE* scheme for a specific universal program UP in Figure 2.

3.1 The Universal Program UP

Let $\Pi = (G_{AE}, AE, VD)$ be a scheme, that performs encryption and authentication, see [3], and decryption and validation of inputs. The universal program UP_K (in Figure 2) is a Turing machine, that is created and instantiated with a secret key K, by the hardening procedure H. When invoked by the obfuscated virtual machine OVM, the universal program UP_K reads a' off the input tape, and parses it to obtain

Algorithm 2. The *Universal WBRPE* scheme $W' = (G', H', U')$, where *createOVM* and *createUP* are macros, each generating a string that encodes a program (*OVM'* and *UP* respectively)

$G'(1^k)$
  $(hk, OVM) \leftarrow G(1^k)$
  $OVM' \leftarrow createOVM(OVM, k)$
  return $\langle hk, OVM' \rangle$

$U'_{uk'}(\omega)$
  $y \leftarrow U_{uk'}(\omega)$
  return y

$createOVM'(\mathbf{OVM})$
  return [`read (c,a,t,l)`
    $(c_{UP}, c_P) \leftarrow$ `c`
    `a'` $\leftarrow$ `(a,t,l,`c_P`)`
    `t'=p(t)+3`
    `l'=l+|P|+|t|+|K|`
    return $\mathbf{OVM}(c_{UP}$`,a',t',l')`]

$H'_{hk}(P)$
$K \leftarrow G_{AE}(1^k)$
  $c_P \leftarrow AE_K(P)$
  $UP_K \leftarrow createUP(K)$
  $(c_{UP}, uk) \leftarrow H_{hk}(UP_K)$
  $c \leftarrow \langle c_{UP}, c_P \rangle$
  $uk' \leftarrow \langle uk, K, P \rangle$
  return (c, uk')

$createUP(\mathbf{K})$
  return [`read a'`
    `(a,t,l,`c_P`)` $\leftarrow$ `a'`
    `P` $\leftarrow \mathrm{VD}_{\mathbf{K}}(c_P)$
    `y` $\leftarrow P_{t,l}$`(a)`
    return `y`]

(a, t, l, c_P), i.e. the remote input, the number of steps of program's execution, the length of the output and the encrypted program. UP decrypts and validates c_P using the key K. The UP then runs P on a for t steps and truncates the output y' to l bits. Finally, UP writes $y' = \langle y, P, t, K \rangle$ on the output tape and halts. The parameters (P, t, K) are output to allow the unhardening procedure U' to verify that the result of the computation is authentic. The output y' of UP is encoded, i.e. encrypted and/ or authenticated, by the OVM (the encoded value returned by the OVM is denoted ω).

The macro *createUP*, in Figure 2, given a secret key K, generates and returns the Turing machine UP_K, represented as a string. The secret key K, is instantiated during the generation and is concatenated to the constant parts of the string.

3.2 The Generation Procedure

The generation procedure G' of the *Universal WBRPE* scheme W' applies G of the specific *WBRPE* W and obtains the the hardening key hk, and the OVM. It applies the $createOVM'$ function on the OVM of the specific *WBRPE* scheme to generate the OVM' of the *Universal WBRPE* scheme W' and returns the tuple $\langle hk, OVM' \rangle$. See Figure 2. The $createOVM'$ function generates the OVM' Turing machine encoded in a string. The OVM' reads (c, a, t, l) of the input tape and generates an input for the OVM Turing machine. The OVM decodes c_{UP} and runs the universal program on input a', for t' steps and writes an l' bit output on its output tape, where t' comprised of the number of steps performed by UP, the number of steps the input program P is executed and of the number of steps it takes the virtual machine to execute P, i.e. bounded by some polynomial $p(\cdot)$ in t. The output length l' is the length of UP's output, which is the tuple $\langle y, P, t, K \rangle$.

3.3 The Hardening Procedure

The input to the hardening procedure H' of the *Universal WBRPE* scheme W' is a program P supplied by the local host. The universal hardening procedure first applies the generation procedure of the authenticated encryption scheme, e.g. in [3], obtains the secret key K and then encrypts the input program P using K, which results in c_P. Next, it generates the universal program, given the secret key K, and hardens it using H to obtain the pair c_{UP} and uk, subsequently returning the ordered pairs $\langle c_{UP}, c_P \rangle$ and $\langle uk, K \rangle$. Details in Figure 2.

We employ authenticated encryption in order to prevent forgery of the input programs, and to ensure that the input program P of the *Universal WBRPE* was not modified on transit, and replaced with some other input program P'.

3.4 The Unhardening Procedure

The unhardening procedure receives an ω and optional $[P,t]$ in an input, and applies the unhardening procedure U of the specific *WBRPE* scheme W on ω. Obtains the tuple $(y, \tilde{P}, \tilde{t}, \tilde{K})$. It then checks if the P,t parameters were supplied, if not it simply returns y, otherwise the validation of the input is also performed. U' verifies that the pair (P,t) supplied by the adversarial algorithm and the pair $(\tilde{P}, \tilde{t})$ output from the universal program UP_K are identical, and that the secret key K from uk equals to the secret key $\tilde{K}$ from the output of UP. This is critical in order to verify that the result of the computation is authentic and not a forgery. If the result is authentic, U is applied on the universal program UP_K, t' and ω, such that UP_K and t' are generated from the input parameters supplied to U'. These steps are performed in order to validate the result ω, i.e. that it is an authentic computation the universal program after a t' steps execution. The universal unhardening procedure returns y as its output. See the details of the implementation in Algorithm 2.

Theorem 1. *Let* $\phi \in \{WB\text{-}IND\text{-}CPA\text{-}\varphi, (WB\text{-}UNF\text{-}\varphi \ \& \ WB\text{-}IND\text{-}CPA)\}$ *and let* $\Pi = (G_{AE}, AE, VD)$ *is an IND-CPA secure authenticated encryption scheme. If* $W=(G,H,U)$ *is a* ϕ *secure* WBRPE *scheme for the universal program UP, then* $W' = Univ(W)$ *is a* ϕ *secure WBRPE scheme for every program.*

We prove the theorem for each value of ϕ, in full version of the paper.

Acknowledgements. We thank Yoram Ofek, Jasvir Nagra, and Christian S. Collberg for useful discussions and helpful comments. This work was supported by funds from the European Commission (contract N 021186-2 for the RE-TRUST project). Part of this research was supported by the NATO Collaborative Linkage Grant n. 982332.

References

1. Algesheimer, J., Cachin, C., Camenisch, J., Karjoth, G.: Cryptographic security for mobile code. In: SP 2001: Proceedings of the 2001 IEEE Symposium on Security and Privacy, p. 2. IEEE Computer Society, Washington (2001)

2. Barak, B., Goldreich, O., Impagliazzo, R., Rudich, S., Sahai, A., Vadhan, S., Yang, K.: On the (Im)possibility of obfuscating programs. In: Kilian, J. (ed.) CRYPTO 2001. LNCS, vol. 2139, p. 1. Springer, Heidelberg (2001)
3. Bellare, C., Pointcheval, D., Rogaway, P.: Authenticated Encryption: Relations among notions and analysis of the generic composition paradigm (2000)
4. Billet, O., Gilbert, H., Ech-Chatbi, C.: Cryptanalysis of a white box AES implementation. In: Handschuh, H., Hasan, M.A. (eds.) SAC 2004. LNCS, vol. 3357, pp. 227–240. Springer, Heidelberg (2004)
5. Cachin, C., Camenisch, J., Kilian, J., Muller, J.: One-round secure computation and secure autonomous mobile agents. In: Welzl, E., Montanari, U., Rolim, J.D.P. (eds.) ICALP 2000. LNCS, vol. 1853, pp. 512–523. Springer, Heidelberg (2000), citeseer.ist.psu.edu/article/cachin00oneround.html
6. Canetti, R.: Towards Realizing Random Oracles: Hash Functions that Hide All Partial Information. LNCS, pp. 455–469. Springer, Heidelberg (1997)
7. Chor, B., Goldreich, O., Kushilevitz, E., Sudan, M.: Private Information Retrieval. Journal of the ACM 45(6), 965–982 (1998)
8. Chow, S., Eisen, P.A., Johnson, H., van Oorschot, P.C.: A white-box DES implementation for DRM applications. In: Feigenbaum, J. (ed.) DRM 2002. LNCS, vol. 2696, pp. 1–15. Springer, Heidelberg (2003)
9. Chow, S., Eisen, P.A., Johnson, H., van Oorschot, P.C.: White-box cryptography and an AES implementation. In: Nyberg, K., Heys, H.M. (eds.) SAC 2002. LNCS, vol. 2595, pp. 250–270. Springer, Heidelberg (2003)
10. Collberg, C., Thomborson, C.: Watermarking, tamper-proofing, and obfuscation-tools for software protection. IEEE Transactions on Software Engineering 28(8), 735–746 (2002)
11. Daemen, J., Rijmen, V.: The Design of Rijndael: AES–the Advanced Encryption Standard. Springer, Heidelberg (2002)
12. Goldreich, O.: Foundations of Cryptography. Basic Applications, vol. 2. Cambridge University Press, Cambridge (2004)
13. Goubin, L., Masereel, J., Quisquater, M.: Cryptanalysis of a white box AES implementation. In: Handschuh, H., Hasan, M.A. (eds.) SAC 2004. LNCS, vol. 3357, pp. 227–240. Springer, Heidelberg (2004)
14. Malkhi, D., Nisan, N., Pinkas, B., Sella, Y.: Fairplay: a secure two-party computation system. In: Proceedings of the 13th USENIX Security Symposium, pp. 287–302 (2004)
15. United States. National Bureau of Standards: Data Encryption Standard, Federal Information Processing Standards publication, vol. 46. U.S. National Bureau of Standards, pub-NBS:adr (1977)
16. Wyseur, B., Michiels, W., Gorissen, P., Preneel, B.: Cryptanalysis of White-Box DES Implementations with Arbitrary External Encodings. In: Adams, C., Miri, A., Wiener, M. (eds.) SAC 2007. LNCS, vol. 4876, pp. 264–277. Springer, Heidelberg (2007)
17. Yao, A.C.: Protocols for secure computations. In: Proc. 23rd IEEE Symp. on Foundations of Comp. Science, pp. 160–164. IEEE, Chicago (1982)