A New Digital Rights Management Solution Based on White-Box Cryptography

Jun Liu[1] and Yupu Hu[2] *,**

Xidian University, Xi'an, China
Email:junl1212@163.com

Abstract. Digital rights management is an important technique to protect digital contents from abuse. Usually it is confronted with severe challenges because of the untrusted environment its application executed in. This condition is formally described as white-box attack model. White-box cryptography aims at protecting software implementation of cryptographic algorithms from white-box attack, hence can be employed to provide security guarantee for digital rights management. Key extraction, code lifting, and illegal distribution are three major threats in digital rights management application, they extremely compromise the benefit of content producer. In this paper, we propose the first solution based on white-box cryptography against the three threats above simultaneously, by implementing traceability of a white-box scheme which has unbreakability and incompressibility. Specifically, We constructively confirm there exists secure white-box compiler which can generate traceable white-box programs, by hiding slight perturbations in the lookup-table based white-box implementation. Security and performance analyses show our solution can be effectively achieved in practice.

1 Introduction

Since the arrival of the digital age, the carrier of audio-visual contents has gradually changed from physical to digital. Although it is more flexible to produce, distribute, and store digital contents, the task of protecting them becomes more tedious. In order to make the rules how digital contents are being managed, a collective set of methods and techniques should be employed, which are referred as digital rights management (DRM) [1]. During decades of development, many standardized DRM solutions and commercial products have came forth. Published specifications include OMA DRM2.1 (Open Mobile Alliance) [2], Marlin (Marlin developer Community) [3], and GY/T 277-2014 (ChinaDRM) [4]. Classic DRM systems include Apple FairPlay system and Microsoft Windows Media DRM system.

* Jun Liu and Yupu Hu are with State Key Laboratory of Integrated Service Networks Lab of Xidian University.

** This work is supported by the National Natural Science Foundations of China (61472309, 61672412), National Key R&D Program of China under Grants No.2017YFB0802002.

It is essential to use cryptography in DRM systems to protect contents and restraint participants′ actions, especially restrict consumption of valuable contents. In a typical DRM scenario, content owner provides encrypted contents and hands it out over public network. While only paying subscribers, or legitimate users, can decrypt and access the original contents. The decryption routine together with content decryption key is embedded in the client DRM application (hardware or software based platform).

At the side of the consumer, the main problem is that client DRM application is executed in an untrusted (probably malicious) open platform, so it is vulnerable to malware, trojan, and spyware. The most usual threat is content decryption key extraction, because that makes it possible to decrypt the protected content without restriction. A second threat is extraction of the whole decryption function from the DRM application, which is denoted as *code lifting* [6,12]. A successful code lifting attack gives an adversary the same advantage as key extraction does, since the decryption routine and the corresponding decryption key are together included in DRM application. Once an attacker isolate and extract the intact decryption code, he can use it in a stand-alone manner, circumventing authentication and rights verification. Another problem is illegal distribution of the client decryption software from legitimate consumers to unauthorized users [6], this makes it possible for an illegal (unregistered) user to decrypt the contents. Such malicious legitimate consumers are called traitors.

1.1 Related Work

To solve above problems, white-box cryptography is a good choice [7]. In fact, DRM is the most common application of white-box cryptography. In 2002, Chow /et al/. first introduced the conception of white-box cryptography and proposed a white-box implementation of AES [19]. Soon they proposed a white-box implementation of DES [8]. These two seminal work started the research on white-box cryptography.

There are many solutions addressing above threats separately by using white-box cryptography, but to our knowledge, there are no solutions addressing them simultaneously. Key extraction and code lifting are often discussed together, since individual key extraction security is meaningless [18]. The initial design in [chowAES] adopted lookup-table based implementation to protect the key. The main idea is to combine key-dependent parts into a series of tables, then encode these tables with random bijections. By this way the key is hided in large-scale lookup-tables, and the mapping which represents the key was obfuscated due to random bijections. Finally, /external encoding/ was introduced to prevent code lifting. Concretely, secret input and output encoding (decoding) were added to the original cipher, then the lookup-table based implementation of the encoded variant made code lifting attack useless. Because the decryption routine indeed implemented the functionality of the encoded cipher rather than the original cipher. The secret encoding can only be canceled out when executing authentication code. Although external encoding is useful to avoid code lifting attack, such solution affects interoperability of DRM platform, hence is undesirable. This

pioneer scheme was broken by Billet /et al/. [BGE attack], and many succedent variants based on it were proposed and broken one after another.

In a recent line of research which aims to study dedicated white-box secure block cipher, some promising schemes began to appear. In 2014, Biryukov /et al./ proposed an ASASA based block cipher [9]. Key extraction security was reduced to the decomposition problem of ASASA. Besides, a new conception of /weak white-box security/ was introduced to quantitatively estimate the security against code lifting by code size. The more the amount of data that an adversary need to extract from the white-box implementation, the more difficult code lifting attack is. One year later this scheme was broken by Minaud /et al./ [10] and Gilbert /et al./ [11]. In 2015, Bogdanov and Isobe [12] proposed a family of white-box secure block cipher SPACE based on Feistel network. Key extraction security relied on key recovery of AES in black-box setting. The difficulty of code lifting was evaluated by /space hardness/, including weak and strong (M, Z)-space hardness. This new conception was indeed a more formalized description of the weak white-box security. In 2016, Bogdanov /et al./ [13] optimized efficiency of space hard block cipher, proposed a new family of block cipher SPNbox based on SPN(substitution-permutation network). This scheme adopted the same principle against key extraction and code lifting as [12]. In the same year, Fouque /et al./ [14] proposed a Feistel-type construction named WhiteBlock, which offers provable security guarantee against code lifting. The main concern was about incompressibility model, which is similar with space hardness security model. Aforementioned schemes in [12–14] remain unbroken, but they displayed different efficiency.

With regard to illegal distribution, the topic of /traitor tracing/ [**?**] can provide useful solution. Many traitor tracing schemes were designed, while few of them can be used in white-box cryptography. The first white-box instance with traceability capability based on Isomorphisms of Polynomials problem (IP problem) was briefly mentioned by Billet and Gilbert [**?**]. Later IP problem was broken by Faugère and Perret [**?**]. Delerablée /et al./ [18] provided a new idea to implement traceability of white-box programs, while no formal analysis was presented and its feasibility was based on some assumptions.

1.2 Our Contributions

In this paper, we first consider a more complicated scenario of DRM where the three threats of key extraction, code lifting, and illegal distribution are involved simultaneously. We present a solution based on white-box cryptography, which is quite useful to provide security in practice. First of all, we investigate several advanced white-box constructions and choose an efficient white-box block cipher instance as the underlying symmetric key encryption scheme. Then we make its white-box implementation traceable by adding some perturbations. Most importantly of all, we demonstrate that there does exist white-box compiler which can generate traceable white-box programs. Specifically, we first construct a white-box compiler which can add perturbations to a white-box program, next we show such perturbations dose not influence the functionality of the original white-box

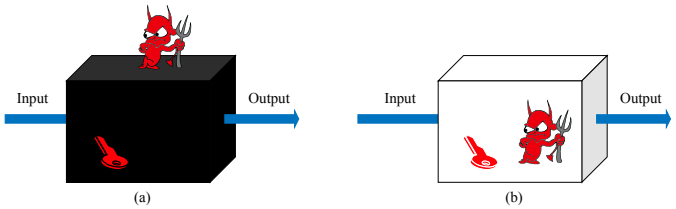

Fig. 1. Attack model. (a) black-box attack model. (b) white-box attack model.

programs. Then we prove the perturbative white-box programs can be made traceable by proving the security of our construction of white-box compiler. Based on the above works, the security and performance of our DRM solution is guaranteed, which means our proposal can be veritable efficiently achieved in practice.

1.3 Structure of This Paper

This paper is organized as follow. In Section 2, we abstractly represent the cryptographic model in DRM application, including the environment under which cryptographic algorithms are executing, adversary model, and security requirements. In Section 3, we describe the proposed DRM solution. In Section 4, we elaborate on the implementation details of our solution, including choosing appropriate symmetric primitive, generating traceable white-box decryption programs, designing tracing scheme. In Section 5, we analyze the security of our solution and give a simple discussion about performance. In section 6, we conclude the paper and put forward some future works.

2 Problem Formation

2.1 DRM Threat Model: White-Box Attack Model

In a DRM scenario, cryptographic algorithms are executed in an untrusted environment. This situation is quite different from the standard cryptographic model, denoted as black-box attack model (see Fig.1(a)), which assumes that the adversary can just observe input-output. DRM threat model is more aligned with white-box attack model (see Fig.1(b)), which is defined in Definition 1.

Definition 1. (white-box attack model, [19]) *A white-box attack model assumes cryptographic algorithms are executed in untrusted context where:*

- *fully privileged attack software shares a host with cryptographic software, having complete access to the implementation of algorithms;*
- *dynamic execution (with instantiated cryptographic keys) can be observed;*
- *internal algorithm details are completely visible and alterable at will.*

We replace white-box attack model by WBAM for short hereafter. White-box cryptography and white-box implementation are two fundamental conceptions for WBAM. The former refers to secure cryptographic schemes under WBAM, the latter represents the implementation of cryptographic primitives under W-BAM.

2.2 DRM Attacker: White-Box Attacker

An attacker in WBAM is called white-box attacker. On the one hand, he can launch static analysis with the help of disassemblers, for instance, IDAPro. On the other hand, he can dynamically analyze the code with the help of debuggers, for instance, OllyDbg. Besides, he can search keys in memory, or derive keys by inserting break-points and whitening subkeys. All in all, compared with black-box attacker who can only get some input-output values, white-box attacker is more matching with the practical situation, hence is more challenging to prevent. Here in DRM, we say a user, whether legal or illegal, is a such white-box attacker.

2.3 DRM Security Requirements: White-Box Security

In light of the above, the security requirements in DRM are as follows:

- **Key extraction security:** It should be hard to derive the key from the decryption white-box implementation in client DRM software. This security notion was also formalized as unbreakability [dlpr13].
- **Code Lifting Security:** To mitigate code lifting, the decryption white-box implementation should provide space-hard property. In other word, it should be infeasible for the attacker to get an equivalent but more compact implementation. Generally, code lifting security is closely related to incompressibility of the decryption implementation [2016yami provable].
- **Traceability:** The decryption implementation should be different for each legitimate user so that it can identify and track the traitor when a legitimate user leak his own decryption software.

3 DRM Solution Based on White-Box Cryptography

3.1 Notations and Preliminaries

The main notations used in this paper are listed in Table 1. Symmetric encryption schemes and their implementations are mainly involved.

Definition 2. (Symmetric Key Encryption) *A Symmetric Key Encryption scheme ε_{sym} is defined as a triple $\varepsilon_{sym} = (G, E, D)$ where $G : \mathcal{K} \to \mathcal{K}$ is the key generation algorithm, $E : \mathcal{M} \times \mathcal{K} \to \mathcal{C}$ is the encryption algorithm, and $D : \mathcal{C} \times \mathcal{K} \to \mathcal{M}$ is the decryption algorithm. For a given key $k \in \mathcal{K}$ and amessagem $\in \mathcal{M}$, $E(m, k) = c$ implies $D(c, k) = m$. $\mathcal{K}, \mathcal{M}, \mathcal{C}$ are key space, plaintext space, ciphertext space respectively. Symmetric means the same k is used in both E and D.*

Table 1. Notations

notation	description
CP	Content Producer
CI	Content Issuer
LI	License Issuer
RO	Rights Object
ε_{sym}	Symmetric Key Encryption
K	Content Key
AES	Advanced Encryption Standard
DES	Data Encryption Standard
KDF	Key Derivation Function
PRP	Pseudo Random Permutation
C_ε	White-Box Compiler

White-box implementation is an obfuscated program of a key-customized instance of the cipher. We use E_k^* to denote the white-box implementation of a encryption algorithm $E(\cdot, k)$, such that for a given message $m \in \mathcal{M}$, the encryption of m by E_k^* equals to the encryption of m by $E(\cdot, k)$, i.e. $E_k^*(m) = E(m, k)$. Similarly, D_k^* denotes white-box implementation of $D(\cdot, k)$ such that for a given $c \in \mathcal{C}, D_k^*(c) = D(c, k)$. Generally, E_k^* includes a large table T derived from k. We say T is an effective large key of k, in other words, k is hiding in T. When executing E_k^*, k is not an input argument, but k should be input to a white-box compiler to get E_k^*.

Definition 3. (White-Box Compiler) *A white-box compiler C_E for an encryption scheme*[1] *E is defined as a publicly known compiling function that takes a key $k \in \mathcal{K}$ as input and outputs E_k^*, namely, $E_k^* = C_E(k)$. Generally, a white-box compiler works as first generating a big lookup table T that hides k, then obfuscating the input-output of T while maintaining its function. Finally, a white-box implementation including T is produced.*

3.2 Solution Model

We adopt a simple DRM architecture [20] based on distributed techniques and mobile code to interpret our solution, the DRM model is illustrated in Fig.2. For ease of explanation, we omit cockamamie procedures and mainly describe some cryptographic procedures here. Let $\varepsilon = (G, E, D)$ is a symmetric encryption scheme.

1. At the side of the producer, CP first chooses a unique content key K_{CEK} for a content, then generates round keys set K by the key generation algorithm G , then encrypts the content message m by black-box encryption $E(\cdot, K)$:

$$c = E(m, K) \tag{1}$$

[1] Without loss of generality, we can also define a white-box compiler C_D for a decryption algorithm D. In this paper, we will continuously use C_D due to the scenario.

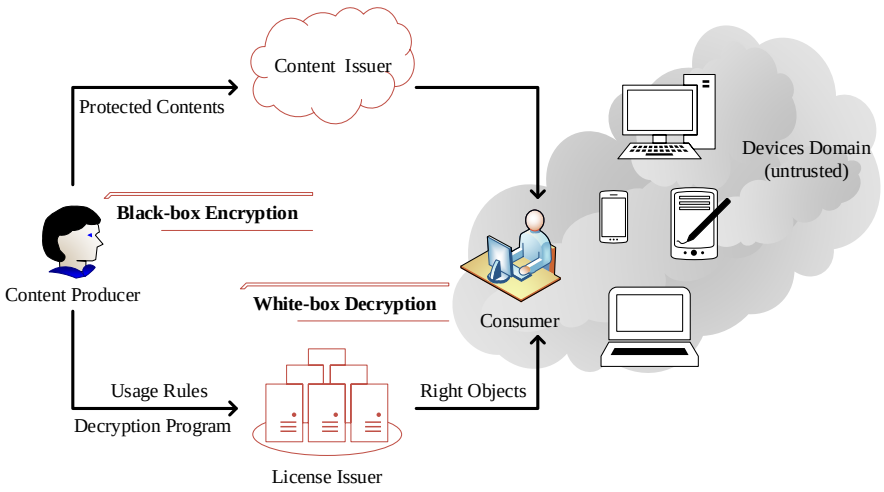

Fig. 2. A simple DRM architecture based on white-box cryptography.

2. CP generates the corresponding decryption program D_K^* by the white-box compiler C_D:

$$D_K^* = C_D(K) \tag{2}$$

3. CP submits c to CI, simultaneously submits mobile code of D_K^* and usage rules of m to LI.
4. When there is a request for m from a consumer, CI will distribute c to him.
5. LI will generate RO including mobile code of D_K^* and corresponding usage rules of m, and then distribute it to the consumer.
6. At the side of the consumer, D_K^* will be loaded into the consumer′s device by DRM agent, then c will be decrypted by white-box decryption:

$$m = D_K^*(c) \tag{3}$$

Finally, the original content (video, music, or others) will be presented in clear in the consumer′s device.

4 Our Techniques: Implementation Details

The solution model in Section 3.2 is quite simple and general, while the implementation details of the encryption algorithm, decryption algorithm as well as the white-box compiler are different, which are key points in the security of the whole model. In this Section we show how to implement the algorithms such that the above model resist the threats we aim to address.

4.1 Symmetric Primitive

First we need to choose an appropriate symmetric key encryption scheme. Since there are no secure white-box implementations of standard cryptographic

algorithms such as AES and DES, some dedicated white-box block ciphers were proposed in recent three years. We investigate three state-of-the-art structures which are unbroken, they are SPACE and its variants [15CCS], SPNbox and its variants [16yami], and WhiteBlock and its variants [16yami2]. Table 2 shows some properties of them. All of them have the same block length $\lambda = 128$ bits and the same key length $l = 128$ bits except SPNbox-24, which has the block length $\lambda = 120$ bits.

Table 2. Comparison of SPACE, SPNbox, and WhiteBlock

Instances	Structure	#Rounds	#Tables	T	T size	Security	Performance
SPACE-8 [?]	Feistel	300	1	8→120	3.84KB	$(T/4, 128)$	300
SPACE-16 [?]	Feistel	128	1	16→112	918KB	$(T/4, 128)$	128
SPACE-24 [?]	Feistel	128	1	24→104	218MB	$(T/4, 128)$	128
SPACE-32 [?]	Feistel	128	1	32→96	51.5GB	$(T/35, 128)$	128
SPNbox-8 [?]	SPN	10	1	8→8	256B	$(T/2^{0.80}, 128)$	160
SPNbox-16 [?]	SPN	10	1	16→16	132KB	$(T/2^{1.61}, 128)$	80
SPNbox-24 [?]	SPN	10	1	24→24	50.3MB	$(T/2^{2.57}, 128)$	50
SPNbox-32 [?]	SPN	10	1	32→32	17.2GB	$(T/2^{3.20}, 128)$	40
WhiteBlock 16 [?]	Feistel	18	4	16→64	2MB	$(T/4, 112)$	19
WhiteBlock 20 [?]	Feistel	23	3	20→64	24MB	$(T/4, 108)$	24
WhiteBlock 24 [?]	Feistel	34	2	24→64	256MB	$(T/4, 104)$	35
WhiteBlock 28 [?]	Feistel	34	2	28→64	4GB	$(T/4, 100)$	35
WhiteBlock 32 [?]	Feistel	34	2	32→64	64GB	$(T/4, 96)$	35

#rounds: the number of round
#tables: the number of lookup tables in the whole implementation
$T(a \to b)$: T is a map representing $\{0,1\}^a \to \{0,1\}^b$
T size: the size of the overall lookup tables
security: (M, Z)-space hardness
performance: number of internal block cipher calls

we choose SPNbox-24 as ε_{sym} for performance and security consideration. On the one hand, SPN-type design is highly parallel compared with Festiel-type block cipher, hence has a better performance. Besides, the memory cost of SPNbox-24 is acceptable in most of the devices, such as desktop, pc, mobile phone, pad, and so on. On the other hand, SPNbox-24 performs similar code lifting security with variants which have the same magnitude in memory. In a nutshell, SPNbox-24 is the most appropriate scheme. Its algorithm representation is given in Algorithm1. There are three layers in a single round, including the nonlinear layer γ, the linear layer θ, the affine layer σ^r. S_{24} is used in the nonlinear layer, and is realized by an internal small SPN-type block cipher of block length 24bits. Please refer to [13] for more details about S_{24}.

In this model, We assume the execution environment of CP is safe, so differential/linear cryptanalysis and cache timing attack towards black-box implementation are not the main concern of this paper. We mainly deal with the white-box decryption implementation, namely, the white-box implementation of

Algorithm 1 symmetric key encryption scheme ε_{spn}=SPNbox-24=$(G_{spn}, E_{spn}, D_{spn})$

G_{spn}**-Input:** a 128-bit master key K_{CEK}, fixed parameter 504
G_{spn}**-Output:** round keys set K=$(k_0, ..., k_{20})$
$(k_0, ..., k_{20})$=KDF(K_{CEK},504)
E_{spn}**-Input:** a plaintext X^0, round keys set K
E_{spn}**-Output:** a ciphertext X^{10}
1: $X^r = (X_0^r, X_1^r, X_2^r, X_3^r, X_4^r)$. Each X_j^r is a 24-bit binary string, i.e., $X_j^r \in \{0,1\}^{24}$
2: $\gamma : (X_0^r, X_1^r, X_2^r, X_3^r, X_4^r) \mapsto (S_{24}(X_0^r), S_{24}(X_1^r), S_{24}(X_2^r), S_{24}(X_3^r), S_{24}(X_4^r))$ where S_{24} is a bijective K-dependent 24-bit S-box
3: $\theta : X^r \mapsto X^r \cdot M_{24}$. Here $M_{24} = cir(1, 2, 5, 3, 4)$, it is a 5*5 circulant matrix with each element belongs to the finite field GF(2^{24})
4: $\sigma^r : X^r \mapsto X^r \oplus (cr_0^r, cr_1^r, cr_2^r, cr_3^r, cr_4^r)$. $(cr_0^r, cr_1^r, cr_2^r, cr_3^r, cr_4^r)$ are round constants with $cr_i^r = 5 \cdot (r-1) + i + 1, i \in [0, 4]$
5: **for** R = 0 to 9 **do**
6: $X^{R+1} = (\sigma^{R+1} \circ \theta \circ \gamma)(X^R)$
7: **end for**
D_{spn}**-Input:** a ciphertext $Y^0 = X^{10}$, round keys set K
D_{spn}**-Output:** a plaintext $Y^{10} = X^0$
8: **for** R = 0 to 9 **do**
9: $Y^{R+1} = (\gamma^{-1} \circ \theta^{-1} \circ (\sigma^{10-R})^{-1})(Y^R)$. $\gamma^{-1}, \theta^{-1}, (\sigma^{10-R})^{-1}$ are the inverse of $\gamma, \theta, \sigma^{10-R}$
10: **end for**

the decryption algorithm D_{spn} of SPNbox-24. The structure of D_{spn} is illustrated in Figure 3.

4.2 White-Box Decryption Implementation

D_K^* is implemented by repeatedly lookup a big table T that represents S_{24}^{-1}, which is a bijective S-box from 24bits to 24bits. T consists of 2^{24} entries of a 24-bit value each. Let $e \in \{0,1\}^{24}$ denotes an entry, $\langle e \rangle$ denotes the value of e, e.g., e=000001 and $\langle e \rangle$=0A1B2C means the first entry in T has value 0A1B2C. Since SPNbox-24 inherently provides unbreakability and incompressibility, the main technical difficulty comes in making the decryption white-box implementation of SPNbox-24 traceable. Inspired by [18], this can be done by slightly modifying functionality of the white-box implementation, concretely, by adding some perturbations. However, there is no evidence in [18] that the given *perturbations-enabled white-box compiler* (see Definition ?) exists. So we first demonstrate this existence.

Definition 4. ***(perturbations-enabled white-box compiler, [dlpr13])*** *For a symmetric encryption scheme $\varepsilon_{sym} = (G, E, D)$ with key space $\mathcal{K}$, plaintext space $\mathcal{M}$, and ciphertext space $\mathcal{C}$. Let C_D is a white-box compiler with respect to D. C_D takes as input a key $k \in \mathcal{K}$ and an ordered list of dysfunctional ciphertexts $\boldsymbol{c} = (c_1, c_2, ..., c_u)$ with each $c_j \xleftarrow{\$} \mathcal{C}$ (c_j is called a perturbation ciphertext, $\xleftarrow{\$}$*

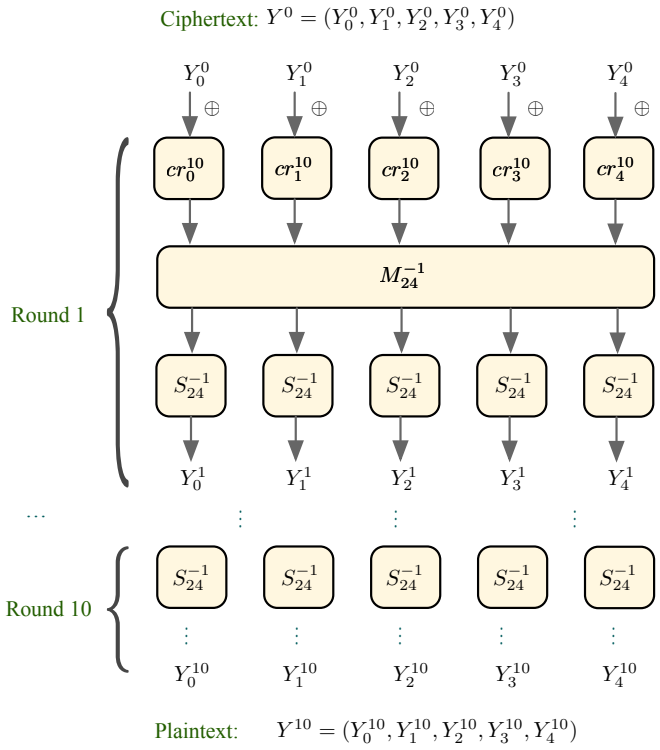

Fig. 3. Decryption algorithm of SPNbox-24.

denotes that the perturbation ciphertext is sampled randomly from ciphertext space, u occupies a negligible fraction of the ciphertext space), and output a program $[D_k^*] = C_D(k, \boldsymbol{c})$. *$C_D$ is said to be perturbations-enabled if both of the following two properties are satisfied:*
(1)$\forall$ *ciphertext* $c \in \boldsymbol{c}$ ($\forall$ *denotes /for any/ hereafter),* $[D_k^*](c) \neq D(k, c)$.
(2)$\forall$ *ciphertext* $c \in \mathcal{C}/\boldsymbol{c}$, $[D_k^*](c) = D(k, c)$.

Theorem 5. *For the symmetric encryption scheme* $\varepsilon_{spn} = (G_{spn}, E_{spn}, D_{spn})$ *as defined in Algorithm 1, there exists a perturbations-enabled white-box compiler* $C_{D_{spn}}$ *(write as* C_D *for short hereafter) with respect to* D_{spn}.

Proof. The proof is constructive. For $\varepsilon_{spn} = (G_{spn}, E_{spn}, D_{spn})$, we construct a white-box compiler C_D. C_D takes as input the key K (K is the output of G_{spn}) and a ciphertext list $\mathbf{c} = (c_1, c_2, ..., c_u)$, and output a perturbated program $[D_K^*]$. C_D works as Algorithm 2.

Then we prove C_D satisfies the two properties in Definition 4.

At the very beginning, $T(S_{24}^{-1})$ is implemented by 20-round AES round transformations. If we assume AES is a pseudo random permutation (PRP), then S_{24}^{-1} is a PRP. Hence the input values of T are uniformly distributed, i.e., during each

Algorithm 2 A perturbations-enabled white-box compiler C_D

Input: $K, \boldsymbol{c}$

Output: $[D_K^*]$

1: for each $e \in \{0,1\}^{24}$, calculate $S_{24}^{-1}(e)$ with K, let $\langle e \rangle$ equals to the output.
2: add all of the $\langle e \rangle$ into a table T in the order of e.
3: **for** $i = 1$ to u **do**
4: run $D_{spn}(c_i)$, mark the input of table accesses in the last round as $e_i^{46}, e_i^{47}, e_i^{48}, e_i^{49}, e_i^{50}$ (because D_{spn} has 10 rounds with each round 5 table accesses, we use superscript 46-50)
5: update T: let $\langle e_i^{46} \rangle = \langle e_i^{50} \rangle$, $\langle e_i^{47} \rangle = \langle e_i^{46} \rangle$, $\langle e_i^{48} \rangle = \langle e_i^{47} \rangle$, $\langle e_i^{49} \rangle = \langle e_i^{48} \rangle$, $\langle e_i^{50} \rangle = \langle e_i^{49} \rangle$.
6: **end for**
7: generate the white-box program $[D_K^*]$ with T, $[D_K^*]$ is a perturbated white-box implementation of D_{spn}
8: return $[D_K^*]$

table access, a uniformly distributed entry is lookup. In other words, table accesses are independent events.
(1) $\forall i \in [1, u]$, run $D_{spn}(c_i)$ executes 50 table accesses, so run $D_{spn}(\boldsymbol{c})$ executes $50u$ table accesses in total. According to lemma 2 in [2016practical], for $50u$ table accesses, the expected value of the number of used entries in T is $2^{24} \cdot [1 - (1 - \frac{1}{2^{24}})^{50u}]$. Let A denotes this variable, we get $50u - 1 < A < 50u$, since A is an integer, let $A = 50u$. So, $50u$ table accesses requires $50u$ entries. This implies in each table access, a distinct entry is lookup.
Finally, we get two immediate observations. On the one hand, the entries e_i^{46},e_i^{47},e_i^{48},e_i^{49},e_i^{50} are not lookup in preceding 9 rounds when running $D_{spn}(c_i)$. On the other hand, the entries $e_i^{46}, e_i^{47}, e_i^{48}, e_i^{49}, e_i^{50}$ are not lookup when running $D_{spn}(c_1)$,...,$D_{spn}(c_{i-1})$, $D_{spn}(c_{i+1})$,...,$D_{spn}(c_u)$. So when run $[D_K^*](c_i)$, the output must be $m_i' = \langle e_i^{50} \rangle || \langle e_i^{46} \rangle || \langle e_i^{47} \rangle || \langle e_i^{48} \rangle || \langle e_i^{49} \rangle ||$, while when run $D_{spn}(K, c_i)$, the output is $m_i = \langle e_i^{46} \rangle || \langle e_i^{47} \rangle || \langle e_i^{48} \rangle || \langle e_i^{49} \rangle || \langle e_i^{50} \rangle ||$. Because T is a PRP, these 5 values are different. So we get $m_i' \neq m_i$, i.e., $[D_K^*](c_i) \neq D_{spn}(K, c_i)$.
(2)After updating T, there are $5u$ entries are modified, we call them *dysfunctional entry*. $\forall c \in \mathcal{C}/\mathbf{c}$, run $[D_K^*](c)$ executes 50 table accesses, requires 50 entries. If each of these entries are not dysfunctional entry, then c will be correctly decrypted with probability 1. Otherwise, c may be correctly or wrongly decrypted. So, the probability that c will be correctly decrypted is evaluated by

$$p \geq \left(\frac{2^{24} - 5u}{2^{24}} \right)^{50}. \tag{4}$$

When u is appropriate (for instance, $u < 10$), p is very close to 1. We can say $[D_K^*](c) = D_{spn}(K, c)$. □

Now that we construct perturbations-enabled white-box compiler which can add some perturbations to a white-box program, such that the perturbative white-box program can identify a consumer. We need to verify such perturbations

can not be corrected by an adversary (a consumer). This means that it is hard to recover the original output of the perturbation ciphertexts. This requirement is formalized with the security notion of Perturbation-Value Hiding (PVH).

Definition 6. ***(Perturbation-Value Hiding, [DLPR13])*** *Let C_D is a perturbations-enabled white-box compiler as defined in Definition 4, let $\mathcal{A}$ be an adversary. Let*

$$\mathrm{Succ}^{\mathrm{PVH}}_{\mathcal{A},C_D} \stackrel{\mathrm{def}}{=} \Pr\left[\begin{array}{ll} m \stackrel{\$}{\leftarrow} \mathcal{M}; c = E(K,m); \boldsymbol{c} = (c); & \\ \quad [D^*_K] = C_D(K, \boldsymbol{c}); & : \tilde{m} = m \\ \quad \tilde{m} \leftarrow \mathcal{A}^{\mathcal{O}}(c, [D^*_K]) & \end{array}\right]$$

denotes the probability of $\mathcal{A}$ winning the PVH game as below. Here $\mathcal{O}$ is a recompiling oracle $\mathcal{O}(\cdot) \stackrel{\mathrm{def}}{=} C_D(K,(c,\cdot))$ that takes as input a list of perturbation ciphertexts containing c and return a perturbated program. C_D is said to be (τ,ϵ)-secure in the sense of PVH if $\mathcal{A}$ running in time at most τ implies $\mathrm{Succ}^{\mathrm{PVH}}_{\mathcal{A},C_D} \leq \epsilon$.

PVH game, [DLPR13]:

1. The challenger selects $m \stackrel{\$}{\leftarrow} \mathcal{M}$ and calculates $c = E(K,m)$, let $\boldsymbol{c}$=(c).
2. The challenger compile $[D^*_K] = C_D(K, \mathbf{c})$, and sends $c, [D^*_K]$ to the adversary $\mathcal{A}$.
3. $\mathcal{A}$ runs on input $(c, [D^*_K])$.
4. $\mathcal{A}$ return some message $\tilde{m}$.
5. $\mathcal{A}$ wins if $\tilde{m} = m$.

Theorem 7. *Let C_D is the white-box compiler constructed in Algorithm 2, then $\mathrm{Succ}^{\mathrm{PVH}}_{\mathcal{A},C_D}$ is upper bounded by $\frac{1}{2^{120}}$.*

Proof. $\forall$ PPT adversary $\mathcal{A}$ (PPT means $\mathcal{A}$ has time $O(\lambda^c)$, λ is a security parameter, c is a constant), $\mathcal{A}$ can call recompiling oracle polynomial times, and generate polynomial perturbated programs $[D^*_K]'$. Each $[D^*_K]'$ perturbates u ciphertexts containing c, and each $[D^*_K]'$ has $5u$ dysfunctional entries containing the dysfunctional entries of c in its table T. Because S^{-1}_{24} is a PRP, i.e., the former T generated in step 2 of Algorithm 2 is a PRP. Naturally, the new T after step 5 is still a PRP. So it is infeasible to extract K from T. Besides, since each $[D^*_K]'$ perturbates c, the values of dysfunctional entry of c in each $[D^*_K]'$ is the same, it is infeasible to recover them. Finally, when $\mathcal{A}$ randomly correct a dysfunctional entry of c, the successes probability is less than $\frac{1}{2^{24}}$. So $\mathcal{A}$ can randomly correct the output of $[D^*_K](c)$ with the successes probability of less than $\left(\frac{1}{2^{24}}\right)^5$, i.e., he can return a correct message with the probability of less than $\frac{1}{2^{120}}$. So we get $\Pr[\tilde{m} = m] \leq \frac{1}{2^{120}}$. In a word, C_D is $(O(\lambda^c), \frac{1}{2^{120}})$-secure in the sense of PVH. □

It is shown in [DLPR13] that perturbated programs can be made traceable assuming the white-box compiler is secure under PVH. Now Our evaluation shows that C_D is secure under PVH, so $[D^*_K]$ can be made traceable. So far, we construct secure white-box compiler that can generate traceable white-box programs. Now we construct tracing scheme $\mathcal{T}$, which includes a setup algorithm $\mathcal{T}.setup$ and a tracing algorithm $\mathcal{T}.trace$.

– $\mathcal{T}.setup$. Input (N, K), output $([D_K^*]^i, TK)$.
Suppose there are N consumers, for each user $u_i, i \in [1, N]$, choose $\boldsymbol{c}_i$ as the tracing ciphertext set of u_i, let $\boldsymbol{c}_i \neq \boldsymbol{c}_w$ when $i \neq w (w \in [1, N])$, then call $[D_K^*]^i = C_D(K, \boldsymbol{c}_i)$. Finally, calculate tracing outputs set $\boldsymbol{m}_i = [D_K^*]^i(\boldsymbol{c}_i)$. TK is the secret tracing key, return $TK = [(\boldsymbol{c}_1, \boldsymbol{m}_1), (\boldsymbol{c}_2, \boldsymbol{m}_2), ...(\boldsymbol{c}_N, \boldsymbol{m}_N)]$. Distributing $[D_K^*]^i$ to u_i, each u_i will get a unique decryption white-box implementation that can identify him.
– $\mathcal{T}.trace$. Input(Q, TK), output (i).
When a leaked decryption function Q is detected, for $i = 1$ to N, run $Q(\boldsymbol{c}_i) = \tilde{\boldsymbol{m}}_i$, check whether $\tilde{\boldsymbol{m}}_i = \boldsymbol{m}_i$, if yes then return index i is the leaker.

$\mathcal{T}.setup$ needs CP to generate N white-box decryption implementations, which are different but have the same structure. In next section we will discuss the security of them.

5 Security and Performance

Now we show our solution satisfy the DRM security requirements in Section 2. Because key extraction security and code lifting security are related to the large table T, and the white-box decryption implementations at the side of the consumer have the same structure of T (they are all PRP), it is enough to discuss one of the white-box decryption implementations. Without loss of generality, we discuss a representational implementation $[D_K^*]$ generated in Algorithm 2.

5.1 Key Extraction Security

Key extraction security requires it is hard to derive the key K from $[D_K^*]$. Because $[D_K^*]$ is produced by C_D, we can first demonstrate it is hard to derive the key from C_D, then key extraction security of $[D_K^*]$ is direct. This is formalized by the security notion of unbreakability (UBK).

Definition 8. ***(Unbreakability, [DLPR13])*** *Let C_D is a white-box compiler as defined in Definition 4, let $\mathcal{A}$ be an adversary. Let*

$$\mathrm{Succ}_{\mathcal{A}, C_D}^{\mathrm{UBK-CCA}} \stackrel{\mathrm{def}}{=} \Pr\left[\begin{array}{c} c_i \stackrel{\$}{\leftarrow} \mathcal{C}, i \in [1, u]; \boldsymbol{c} = (c_1, ..., c_u); \\ k \stackrel{\$}{\leftarrow} \mathcal{K}; [D_k^*] = C_D(k, \boldsymbol{c}); \qquad : \tilde{k} = k \\ \tilde{k} \leftarrow \mathcal{A}^{\mathcal{O}}([D_k^*]) \end{array}\right]$$

denotes the probability of $\mathcal{A}$ winning the UBK game as below. Here CCA represents chosen ciphertext attack, $\mathcal{A}$ is a decryption oracle $\mathcal{O}(\cdot) \stackrel{\mathrm{def}}{=} D(k, \cdot)$. C_D is said to be (τ, ϵ)-secure in the sense of UBK-CCA if for any adversary $\mathcal{A}$ running in time at most τ implies $\mathrm{Succ}_{\mathcal{A}, C_D}^{\mathrm{UBK-CCA}} \leq \epsilon$.

UBK game [dlpr13]:

1. The challenger generates a key $k \stackrel{\$}{\leftarrow} \mathcal{K}$ and $c_i \stackrel{\$}{\leftarrow} \mathcal{C}, i \in [1, u]$, let $\mathbf{c} = (c_1, ..., c_u)$.

2. The challenger runs $[D_k^*] = C_D(k, \mathbf{c})$, then sends $[D_k^*]$ to the adversary $\mathcal{A}$.
3. $\mathcal{A}$ is run on input $[D_k^*]$.
4. $\mathcal{A}$ returns a guess $\tilde{k} \in \mathcal{K}$.
5. $\mathcal{A}$ succeeds if $\tilde{k} = k$ holds.

Theorem 9. *Let C_D is a perturbations-enabled white-box compiler, then C_D is secure under UBK-CCA.*

Proof. For any $k \in \mathcal{A}$, S_{24}^{-1} is a PRP. According to the proof of Theorem 7, the new T after step 5 of Algorithm 2 is still a PRP. In other words, T is indistinguishable with a random permutation in 24 bits. Because it is infeasible to recover the key from a random permutation, it is infeasible to recover k from T. That means the successes probability of extracting k from T is negligible: $\mathrm{Succ}_{\mathcal{A},T}^{\mathrm{UBK-CCA}} \leq \epsilon$ where ϵ is a negligible function of security parameter $\lambda \in \mathbb{IN}$. According to [2016practical], extracting k from $[D_k^*]$ can be reduced to recovering k from T in $[D_k^*]$ in black-box model, i.e., $\mathrm{Succ}_{\mathcal{A},C_D}^{\mathrm{UBK-CCA}} \leq \mathrm{Succ}_{\mathcal{A},T}^{\mathrm{UBK-CCA}}$. So we get $\mathrm{Succ}_{\mathcal{A},C_D}^{\mathrm{UBK-CCA}} \leq \epsilon$ where ϵ is a negligible function of λ. Now we can say C_D is secure under UBK-CCA. □

5.2 Code Lifting Security

There are some similar description of code lifting security, such as (λ, δ) - incompressibility [dlpr13], weak white-box [14yami], (M, Z)-space hardness [15ccs] [16practical], as well as weak and strong incompressibility [16provable]. They all reflect incompressibility, which has the goal of preventing an adversary producing a more compact implementation.Here we adopt a more formal security notion coined ENC-TCOM(c.f. [2016provable]) to analyze incompressibility of $[D_K^*]$. With consideration of the scenario, we need to discuss DEC-TCOM. Because in $\varepsilon_{spn} = (G_{spn}, E_{spn}, D_{spn})$, E_{spn} and D_{spn} are deterministic encryption and decryption being inverse of each other, we can exchange roles without loss of generality and get the definition of DEC-TCOM as below. Moreover, because $[D_K^*]$ is a white-box decryption program with traceability, the security game should be sightly modified in order to fit the scenario.

Definition 10. ***(Weak Incompressibility, DEC-TCOM)*** *Let C_D is a perturbations-enabled white-box compiler, $[D_k^*]$ is a decryption scheme produced by C_D. T is the large table which hides k in $[D_k^*]$. Let $\mathcal{A}$ be an adversary. Let*

$$\mathrm{Succ}_{\mathcal{A},[D_k^*]} \stackrel{\mathrm{def}}{=} \Pr\left[\begin{array}{ll} c_i \xleftarrow{\$} \mathcal{C}, i \in [1,u]; \boldsymbol{c} = (c_1, ..., c_u); & \\ k \xleftarrow{\$} \mathcal{K}; [D_k^*] = C_D(k, \boldsymbol{c}); & : [D_k^*](c) = \tilde{m} \\ c \xleftarrow{\$} \mathcal{C}, \tilde{m} \leftarrow \mathcal{A}^T(c) & \end{array}\right]$$

denotes the probability of success of $\mathcal{A}$ playing the incompressibility game as below. Here $\mathcal{A}^T$ represents that $\mathcal{A}$ can adaptively query some (limited) entries of T and get the values of them. $[D_k^*]$ is said to be τ-secure for (s, λ, δ)-incompressibility if A running in time at most τ implies $\mathrm{Succ}_{\mathcal{A},[D_k^*]} \leq \delta$.

Incompressibility Game [16yami]:

1. The challenger generates a key $k \xleftarrow{\$} \mathcal{K}$ and $c_i \xleftarrow{\$} \mathcal{C}, i \in [1,u]$, let $\mathbf{c} = (c_1, ..., c_u)$.
2. The challenger runs $[D_k^*] = C_D(k, \mathbf{c})$, store T generated in step 5 of Algorithm 2. T is indeed a function which represents a mapping: $\{0,1\}^{24} \leftarrow \{0,1\}^{24}$, k is hidden in this mapping.
3. The adversary $\mathcal{A}$ adaptively chooses $q_i \in \{0,1\}^{24}, i \in [0,s)$, and receives $T(q_i) = \langle q_i \rangle$ from the challenger. s is the the number of entries in T the adversary can adaptively query.

At this point the adversary is tasked with trying to decrypt a random message:

4. The challenger chooses $c \in \mathcal{C}$ uniformly at random, sends c to the adversary.
5. The adversary returns a guess $\tilde{m} \in \mathcal{M}$.
6. The adversary wins if $\tilde{m} = [D_k^*](c)$ holds.

Theorem 11. *Let $[D_K^*]$ is the white-box decryption program generated in Algorithm 2, then* $\mathrm{Succ}_{\mathcal{A},[D_K^*]}$ *is upper bounded by* $\frac{N}{2^{120}} + \frac{2^{120}-N}{2^{120}} \cdot \left(\frac{s}{2^{24}}\right)^{50}$ *with* $N = \frac{log_e\left(1-\frac{s}{2^{24}}\right)}{50}$, $e = \frac{2^{24}-1}{2^{24}}$.

Proof. According to [2016practical,Theorem 3], SPNbox-24 has weak ACS-$\left(M, -\log_2\left(\frac{N}{2^{120}} + \frac{2^{120}-N}{2^{120}} \cdot \left(\frac{M}{T}\right)^{50}\right)\right)$-space hardness. This means when given code size is less than M, it is infeasible to decrypt any random ciphertext with probability more than $\frac{N}{2^{120}} + \frac{2^{120}-N}{2^{120}} \cdot \left(\frac{M}{T}\right)^{50}$. So in the above game, when an adversary adaptively store code with size M, i.e., he can get $s = 2^{24} \cdot \frac{M}{T}$ entries, then $\mathrm{Succ}_{\mathcal{A},[D_K^*]}$ is less than $\frac{N}{2^{120}} + \frac{2^{120}-N}{2^{120}} \cdot \left(\frac{s}{2^{24}}\right)^{50}$. In other words, $\forall$ PPT adversary $\mathcal{A}$, $[D_K^*]$ is $O(\lambda^c)$-secure for $\left(s, \lambda, \frac{N}{2^{120}} + \frac{2^{120}-N}{2^{120}} \cdot \left(\frac{s}{2^{24}}\right)^{50}\right)$-incompressibility. □

5.3 Traceability

It is shown in Secion 4.2 that $[D_K^*]^1, ..., [D_K^*]^N$ is traceable, now we show $\mathcal{T}.trace$ will return the correct leaker. When Q is detected, there exists $i \in [1,N]$ such that $Q = [D_K^*]^i$, so when run $Q(\boldsymbol{c}_i)$ for all $i \in [1,N]$, there exists i such that $\tilde{\boldsymbol{m}}_i = \boldsymbol{m}_i$ with the probability of 1. Hence $\mathcal{T}.trace$ will return the correct leaker with the probability of 1.

Compared with the example tracing scheme in [18], our scheme only needs constant size perturbations, namely, the number of perturbation ciphertext (u) is independent with user number N. Because if the number of tracing ciphertexts is linearly or sublinearly grow with N, for instance, $O(N)$ or $O(logN)$, then the properties of perturbations-enabled white-box compiler may be compromised.

5.4 Performance Analysis

At the side of the producer, the performance mainly depends on the black-box encryption implementation of SPNbox-24 and the white-box compiler, it is shown in [13] that SPNbox family block cipher can provide an optimized high-performance software implementation(encryption and decryption, black-box and

white-box). The running time of a white-box compiler mainly rely on generating the big table, in SPNbox-24, there are 2^{24} entries in the table. So for each 24-bit entry $e \in (0,1)^{24}$, the white-box compiler need to calculate the output of e by calling 20-round AES round transformation. Since AES has fast software implementation in contemporary processors, the white-box compiler can efficiently execute. Suppose the white-box compiler can generates a white-box program in time $O(\tau)$, then it will cost $O(\tau)$ to generate N traceable white-box programs, assuming the producer supports parallel processes. At the side of the consumer, only some entries in the table are modified, this dose not influence the decryption performance of original white-box implementation.

6 Conclusion

In our work, we put three severe threats of DRM together, they are key extraction, code lifting, and illegal distribution. We have proposed an efficient solution based on white-box cryptography, we construct white-box compiler that can generate traceable white-box programs, in addition, we prove our construction is secure. This result set the stage for more traceable white-box implementations. However, there are still many to do in the future, because the actual situation is always more complex than we consider. Two possible directions of this work are as below.

- It is interesting to investigate whether the performance of SPNbox family block cipher can be improved. Or, whether there exists possibility to design new white-box secure ciphers as our building block.
- A constraint associated with our solution lies in we cannot give absolute answer whether our tracing scheme resist collusion. Namely, if there exist a coalition $U = (u_1, u_2, ..., u_t)$ with $t \in [1, N]$, $\forall u_a \in U, u_a$ provides some code segments of his own $[D_K^*]^i$(espically, u_a should provide some entries of his own T), then U can construct a new lookup table T^* to produce a new decryption software PD. We can intuitionally observe it is hard for U to produce a perfect PD(perfect means PD can correctly decrypt $c_{content}$ with the probability of 1), since T^* cannot include all of the entries in $\boldsymbol{e}_f$ with probability 1 . But once a perfect PD is generated, we do not know whether there exists a new $\mathcal{T}$ can trace at least one traitor in U. So we might consider one more threat within illegal distribution in the presence of collusion. Till date, many traitor tracing systems based on PLBE towards fully or t collusion-resistant were proposed, but the ciphertext size is $O(N)$, $O(\sqrt{N})$ [21], or $O(logN)$ [22]. Schemes with constant ciphertext size are yet to be provided. But our solution needs the number of tracing ciphertexts (u) be independent with user number N, because the function of the traceable white-box program is influenced by u. So we would like to study and design collusion-resistant tracing scheme with constant ciphertext size. Or we can find new achieved secure white-box compiler that can help us implementing collusion-resistant tracing scheme.

References

1. Kuo, C. -C. J., Kalker, T., Zhou, W.: Digital rights management. IEEE Signal Process. Mag. 21(2), 11-14 (2004)
2. OMA digital rights management v2.1 (2010). www.openmobilealliance.org-/release/DRM/V2_1_1-20100406-A/
3. Marlin Specifications. www.marlin-community.com/technology#Marlin_Specifications
4. Technical specification of digital rights management for internet television (2014).
5. De Mulder, Y.: White-box cryptography: analysis of white-box AES implementations. Ph.D. thesis, Katholieke Universiteit Leuven (2014)
6. Wyseur, B.: White-box cryptography. Ph.D. thesis, Katholieke Universiteit Leuven (2009)
7. Gilbert, H.: On White-Box Cryptography. invited talk, Fast Software Encryption 2016 (2016). http://fse.rub.de/slides/wbc_fse2016_hg_2pp.pdf
8. Chow, S., Eisen, P., Johnson, H., 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. Biryukov, A., Bouillaguet, C., Khovratovich, D.: Cryptographic schemes based on the ASASA structure: black-box, white-box, and public-key (Extended Abstract). In: Sarkar, P., Iwata, T. (eds.) ASIACRYPT 2014. LNCS, vol. 8873, pp. 63-84. Springer, Heidelberg (2014).
10. Minaud, B., Derbez, P., Fouque, P.A., and Karpman, P.: Key-recovery attacks on ASASA. In: Iwata, T., Cheon, J.H. (eds.) ASIACRYPT 2015, LNCS, vol. 9453, pp. 3-27. Springer, Heidelberg (2015).
11. Gilbert, H., Plut, J., Treger, J.: Key-recovery attack on the ASASA cryptosystem with expanding S-boxes. In: Gennaro, R., Robshaw, M. (eds.) CRYPTO 2015. LNCS, vol. 9215, pp. 475?490. Springer, Heidelberg (2015).
12. Bogdanov, A., Isobe, T.: White-box cryptography revisited: space-hard ciphers. In: Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, pp. 1058-1069. ACM (2015)
13. Bogdanov, A., Isobe, T., Tischhauser, E.: Towards practical whitebox cryptography: optimizing efficiency and space hardness. In: Cheon, J.H., Takagi, T. (eds.) ASIACRYPT 2016. LNCS, vol. 10031, pp. 126-158. Springer, Heidelberg (2016).
14. Fouque, P.A., Karpman, P., Kirchner, P., Minaud, B.: Efficient and provable white-box primitives. In: Cheon, J.H., Takagi, T. (eds.) ASIACRYPT 2016. LNCS, vol. 10031, pp. 159-188. Springer, Heidelberg (2016).
15. Chor, B., Fiat, A., and Naor, M.: Tracing traitors. In: Desmedt, Y.G. (ed.) CRYPTO 1994. LNCS, vol. 839, pp. 257-270. Springer, Heidelberg (1994).
16. Billet, O., Gilbert, H.: A traceable block cipher. In: Laih, C.S. (ed.) ASIACRYPT 2003. LNCS, vol. 2894, pp. 331-346. Springer, Heidelberg (2003).
17. Faugere, J.C., Perret, L.: Polynomial equivalence problems: algorithmic and theoretical aspects. In: Vaudenay, S. (ed.) EUROCRYPT 2006. LNCS, vol. 4004, pp. 30-47. Springer, Heidelberg (2006).
18. Delerablée, C., Lepoint, T., Paillier, P., Rivain, M.: White-box security notions for symmetric encryption schemes. In: Lange, T., Lauter, K., Lisoněk, P. (eds.) SAC 2013. LNCS, vol. 8282, pp. 247-264, Springer, Heidelberg (2014).
19. Chow, S., Eisen, P., Johnson, H., Oorschot, P.C.: White-box cryptography and an AES implementation. In: Nyberg, K., Heys, H. (eds.) SAC 2002. LNCS, vol. 2595, pp. 250-270. Springer, Heidelberg (2003).

20. Ghită, S.V., Patriciu, V.V., Bica, I.: A new DRM architecture based on mobile code and white-box encryption. In: 2012 9th International Conference on Communications, pp. 303-306. IEEE (2012)
21. Boneh, D., Sahai, A., Waters, B.: Fully collusion resistant traitor tracing with short ciphertexts and private keys. In: Vaudenay, S. (ed.) EUROCRYPT 2006. LNCS, vol. 4004, pp. 573-592. Springer, Heidelberg (2006)
22. Boneh, D., Zhandry, M.: Multiparty key exchange, efficient traitor tracing, and more from indistinguishability obfuscation. In: Garay, J.A., Gennaro, R. (eds.) CRYPTO 2014. LNCS, vol. 8616, pp. 480-499. Springer, Heidelberg (2014)