Vol.2018-CSEC-83 No.14
2018/12/14

White-Box CryptographyのCode Lifting対策における耐タンパーソフトウェアの機能改変困難性

大石 和臣[1,a)]

概要：White-Box Cryptography は，内部に実装されている秘密の暗号鍵を抽出することが難しい暗号実装である．暗号やセキュリティ機能をソフトウェア実装するときに求められる性質は，秘密情報守秘性と機能改変困難性であり，それらの性質を持つソフトウェアを耐タンパーソフトウェアと呼ぶ．従って，White-Box Cryptography は，耐タンパー暗号ソフトウェアの秘密情報守秘性を実現する手段の一つである．White-Box Cryptography の具体的方式としてテーブルネットワーク実装が活発に研究され，既知の攻撃に対して強い方式が提案されている．White-Box Cryptography における課題の一つは，秘密の暗号鍵を抽出する代わりに実装自体を抽出・コピーして盗用する Code Lifting である．本研究では，Code Lifting 対策を検討する．White-Box Cryptography の特徴を活かした対策としてオンラインで実装を認証するプロトコルを提案し，同時多重実行を検出して対応できることを示す．Code Lifting 対策にとって耐タンパーソフトウェアの機能改変困難性が重要であることを指摘する．

キーワード：White-Box Cryptography, Code Lifting, 耐タンパーソフトウェア, 秘密情報守秘性, 機能改変困難性，自己インテグリティ検証, オンライン検出

Countermeasure for Code Lifting of White-Box Cryptography utilizing Resistance to Modification of Tamper Resistant Software

OISHI, Kazuomi[1,a)]

Abstract: White-Box Cryptography is a cryptography implementation, from which secret cryptographic key embedded within is hard to extract. When cryptography or security functionalities are implemented as software, the software implementation should attain resistance to observation and resistance to modification, and such software is refferred to as *tamper resistant software*. Thus, white-box cryptography is a method to realize resistance to observation of tamper resistant cryptography software. As a concrete scheme of white-box cryptography, research on table network implementation has been actively conducted and table network implementation schemes strong against known attacks are proposed. A problem of white-box cryptography is *code lifting*, which extracts, copies, and uses the implemantation itself instead of the embedded secret key. In this manuscript, we study countermeasure for code lifting. As a countermeasure utilizing nature of white-box cryptography, an online protocol to verify implementation is explained, which can detect simultaneous multi execution and take appropriate action. It is pointed out that resistance to modification of tamper resistant software is important in order to realize effective countermeasure for code lifting.

Keywords: White-Box Cryptography, Code Lifting, Tamper Resistant Software, Resistance to Observation, Resistance to Modification, Self Integrity Verification, On-line Detection

1. はじめに

White-Box Cryptography は，内部に実装されている秘密の暗号鍵を抽出することが難しい暗号実装である．暗号

[1] 静岡理工科大学 コンピュータシステム学科, 〒 437-8555 静岡県袋井市豊沢 2200-2, Shizuoka Institute of Science and Technology, Department of Computer Science, 2200-2 Toyosawa, Fukuroi, Shizuoka 437-8555, Japan
[a)] oishi.kazuomi@sist.ac.jp

やセキュリティ機能をソフトウェア実装するときに求められる性質は秘密情報守秘性と機能改変困難性であり，それらの性質を持つソフトウェアを耐タンパーソフトウェアと呼ぶ．従って，White-Box Cryptography は，耐タンパー暗号ソフトウェアの秘密情報守秘性を実現する手段の一つである．White-Box Cryptography の研究は，共通鍵ブロック暗号の DES と AES を対象とする方式が最初に提案され，それらに対する攻撃と改良が研究された．2018 年において，DES や AES を White-Box Cryptography として実現する方式は全て（全数探索より効率的に鍵を見つける方法によって）破られている．一方，White-Box Cryptography の目標を満たすように設計された独自の共通鍵ブロック暗号がいくつか提案されており，それらは既知の攻撃に対して安全性が証明されている．

White-Box Cryptography の課題の一つは，秘密の暗号鍵を抽出する代わりに実装自体を抽出・コピーして盗用する Code Lifting である．Code Lifting はソフトウェアの不正コピーであり，その防止は容易ではない．本稿では，White-Box Cryptography の Code Lifting 対策について検討する．White-Box Cryptography の特徴を活かした対策としてオンラインで実装を認証するプロトコルを提案し，同時多重実行を検出して対応できることを示す．Code Lifting 対策にとって耐タンパーソフトウェアの機能改変困難性が重要であることを指摘する．

以下，2 で従来の研究についてまとめ，3 で Code Lifting 対策と認証プロトコルについて説明し，考察する．4 でまとめる．

2. 従来の研究

本章では，耐タンパーソフトウェアと White-Box Cryptography の研究について，いままでの流れを概観し得られた結果を述べる．本稿の検討対象である Code Lifting について説明する．

2.1 耐タンパーソフトウェア

耐タンパーソフトウェアは，文献 [1] で Aucsmith により 1996 年に定義された．耐タンパーソフトウェアは，内部の秘密情報やアルゴリズムを不正に読み取ることが困難である性質（秘密情報守秘性）と，内部の情報やアルゴリズムを不正に改変することが困難である性質（機能改変困難性）の 2 つの性質を持つソフトウェアである．

その提案以来，様々な研究が行われている [9], [11], [23], [25], [26]．具体的なプログラムを対象とするアプローチとして Code Obfuscation がある．White-Box Cryptography は秘密鍵を内蔵する暗号プログラムを対象とする Code Obfuscation だとみなすことができる．他には，計算量理論的な数学的アプローチ，具体的な公開鍵暗号およびディジタル署名を対象とするアプローチもあり，否定的および肯定的な成果が得られている．

2.1.1 Code Obfuscation

Code Obfuscation[17] は，オリジナルのプログラムを変換して同じ振る舞いをするが解析が困難なプログラムを得る方法である．難読化は，ソースプログラムを解析することが困難になるように等価変換する方法として提案されたが，Code Obfuscation と同等の考え方とみなすことができる．このアプローチの研究は 1997 年ころから活発かつ網羅的に研究が進み，書籍 [16] にまとめられている．しかし，Code Obfuscation の効果は限定的で，解析に対して数週間の耐性を持つと考えられており，White-Box AES のコンテスト [12], [27], [28] の結果もそれと合致する．

2.1.2 Theoretical, Key, and indistinguishablility Obfuscations

理論的なアプローチとして，計算量理論に基づく実現可能性に関する 2001 年の研究 [3] が有名である．あるプログラム（関数）を実行するとき，その入力と出力は観察されるが，そのプログラム内部については何も理解されることがないようにしたい．そのようにプログラムを変換する Obfuscation は，Virtual Black Box (VBB) property を持つと定義される．任意の関数を obfuscate できる VBB Obfuscator は存在しないことが [3] で証明された．一方，特定の関数（ポイント関数[*1]）を VBB obfuscate できることが証明された [8]．これらの研究は数学的な関数を対象とする obfuscation についての計算量理論的な研究である．

具体的な公開鍵暗号を対象とする研究に関しては，VBB property を緩和した average case VBB property (ACVBP) を持つ obfuscation によって Re-encryption 方式と Encrypted Signature 方式の obfuscation が証明可能安全に実現できることが 2007 年と 2010 年に示された [21], [22]．Encrypted Signature 方式（ES 方式）は，あるメッセージを入力とするとき，そのメッセージに対するディジタル署名を公開鍵暗号で暗号化した暗号文を出力する複合型暗号方式である．文献 [21] の ES 方式の obfuscation は，ES 生成プログラムからディジタル署名生成鍵を抽出することが計算量的に困難であることを保証する．その効率には向上の余地が大きく，特定の方式に依存していたが，具体的なディジタル署名方式のディジタル署名生成鍵に関する obfuscation が安全性の証明付きで実現可能であることを示した点で，ES 方式の obfuscation は画期的な研究成果だと考えられる．

理論的かつ汎用性の高い別の研究成果として，VBB より弱い定義の indistinguishability Obfuscation (iO) と呼ぶ obfuscation について，多項式サイズの全ての回路に対して iO を構成する方法が 2013 年に [19] で提案された．iO は，等価かつほぼ同じサイズの二つの回路を obfuscate したと

[*1] ある特定の入力に対してある値を出力するが他の入力に対しては 0 を出力する関数

きに，それらの obfuscate された回路を識別することが計算量的に困難な obfuscation である．iO を仮定すると様々な暗号方式を構成できるため，暗号の理論的な研究分野において iO に関する研究は大きな流れとなっている．プログラムは論理回路で実現できるので，iO がプログラムの obfuscation に対して与える影響は大きいと思われる．ただし，具体的な応用についてはまだ研究の余地が多く，実際のプログラムから耐タンパーソフトウェアを構成することが可能かについては今後の研究が必要である．

2.2 White-Box Cryptography

White-Box Cryptography は，2002 年に Chow et al. により提案された [13], [14]，秘密鍵を内蔵する暗号の実装方法およびその考え方である．映画のようなディジタルコンテンツをネットワークで配信するビジネスにおいて，コンテンツを平文で配信すると不正コピーされてしまうので，コンテンツ保護が可能なコンテンツ配信システム，DRM (Digital Rights Management) と呼ぶ，が求められる．DRM の基本は，コンテンツを保護するために共通鍵ブロック暗号を用いることとし，秘密鍵を内蔵した復号プログラムを正規のユーザに渡し，暗号化したコンテンツをユーザに配信する方式である．ユーザは，セットトップボックス，PC，スマートフォン，モバイル端末等の機器を用いてコンテンツを再生するが，それらの機器をユーザが自由に改造する可能性があるため，復号プログラムは信頼できるとは限らない環境で動作する．つまり，復号プログラムに加えて実行環境もユーザの支配下に置かれている．この状況においてユーザの攻撃者が復号プログラムを解析・攻撃することを White-Box Attack Context と呼ぶ．White-Box Attack Context において耐性を持つ暗号実装，すなわち，実装内部の秘密鍵を得られないような暗号実装を White-Box Cryptography と呼ぶ．従って，White-Box Cryptography は，耐タンパー暗号ソフトウェアの秘密情報守秘性を実現する手段の一つである．

2.2.1 White-Box DES/AES

White-Box Cryptography の具体的な実現方法として，複数のテーブルを組み合わせた構成で DES や AES を実現するテーブルネットワーク実装が提案された [13], [14]．それに対して，それらの暗号方式の代数的構造やテーブルの特徴等を利用して実装から鍵を効率的に抽出する攻撃手法が提案された [4], [20], [29]．2014 年頃までに，DES や AES のテーブルネットワーク実装方式の全てが，全数探索より効率的に鍵を見つける方法によって破られている [24]．

2.2.2 SPACE, SPNbox, and DCA

前述のテーブルネットワーク実装において使用されるテーブルは DES や AES の S-Box を基にして構成されるが，S-Box の仕様および鍵と S-Box の関係（ラウンド鍵と S-Box が排他的論理和される）は公開されている．そのため，従来のテーブルネットワーク実装の構成では，秘密鍵の抽出を防ぐことはできなかった．そこで，ある鍵を使う AES の暗号化/復号における平文と暗号文のペアを基にしてテーブルを作り，そのテーブルを用いてフェイステルネットワークの構成を採用した独自の共通鍵ブロック暗号 SPACE が 2015 年に提案された [5]．SPACE は，AES の暗号化/復号における平文と暗号文のペアの集合の部分集合をテーブルとして用いるので，AES が安全だと仮定すると，それらのテーブルから鍵を抽出することは困難である．このアイデアを利用して，差分攻撃や線形攻撃等の既存の攻撃に対する安全性を持つように設計された独自の共通鍵ブロック暗号方式が SPACE である．さらに，SPACE より高速な暗号として，AES の要素である暗号化操作（S-Box，MixColumns）等を利用して SP ネットワーク（Substitution-Permutation Network）の構成を採用した独自の共通鍵ブロック暗号 SPNbox が 2016 年に提案された [6]．

また，DES や AES の代数的構造等の知識を必要とする従来の攻撃とは異なり，プログラム解析ツールの Binary Instrumentation を用いてテーブルネットワーク実装から自動的に鍵を抽出できる DCA (Differential Computation Attack) が 2016 年に提案され [7]，DCA に対するソフトウェア保護技術の有効性評価の研究 [2] が発表された．このように White-Box Cryptography の研究は新方式と新攻撃の提案を経て活性化しており，次節で述べる産業界の期待もあるため注目を集めている．

2.2.3 White-Box Cryptography の応用

White-Box Cryptography が必要とされている現実の製品やサービスとして，DRM に加えて，CAS (Conditional Access System) と HCE (Host-based Card Emulation) があげられる [6], [28]．CAS は限定受信システムと呼ばれ，ディジタル放送のコンテンツ管理方法のひとつである．特定のユーザだけが放送を視聴できるようにアクセス制御する仕組みであり，日本では B-CAS が普及している．HCE はスマートフォンに搭載された NFC (Near Field Communication) を使って非接触型決済を実現する機能，いわゆるお財布ケータイの機能である．HCE ではない非接触型決済の実現方法ではセキュア・エレメントと呼ばれるハードウェアが必要である．セキュア・エレメントは暗号の鍵を安全に保管し，暗号処理や決済処理を安全に実行できる IC カード（耐タンパーモジュール）に相当する[*2]．HCE は，セキュア・エレメントを必要とせず，スマートフォンの CPU がその機能を代わりに実行する仕組みである．その場合，専用のハードウェアの代わりにソフトウェアで IC カード相当の耐タンパー性を保証することが必要であり，暗号鍵の秘密情報守秘性を保つ White-Box Cryptography

[*2] セキュア・エレメントはスマートフォン本体に内蔵するか，SIM カードあるいは SD カードに搭載する．

が注目され必要とされている．

2.3 Code Lifting

Code Lifting は White-Box Cryptography および Obfuscation に対する汎用的で有効な攻撃である．Code Lifting は，実装から秘密の鍵を抽出する代わりに実装自体あるいは実装内の本質的な部分を抽出し，それをコピーして盗用する攻撃である．Code Lifting はソフトウェアの不正コピーであり，秘密情報守秘性とは別の仕組みにより防ぐ必要があるが，White-Box Cryprography あるいは Obfuscation の実装だけで防止を実現することは容易ではない．

2.3.1 Incompressibility/Space-Hardness

Code Lifting への耐性を評価する定量的な指標として Incompressibility[18] あるいは Space-Hardness[5], [6] が提案されている．これらは，Code Lifting が行われるとき，ある条件で暗号を破るための攻撃に必要な White-Box Cryptography 実装のコードサイズとその成功確率で耐性を見積もる．ある攻撃の成功確率をみたすために必要なサイズが大きいほど攻撃者は多くのコードをコピーしなければならないので耐性が強いと評価する．直観的には，復号プログラムのテーブルネットワーク実装の場合，暗号文から平文を計算するために必要なテーブルが多ければ多いほど耐性は強くなる．標準実装の DES や AES の場合，鍵さえ抽出できればそれ以外の処理は公開されているため，Incompressibility/Space-hardness は小さい．つまり，暗号化/復号するときに使うテーブルの総サイズが大きいほど Incompressibility/Space-hardness の耐性は強くなると思われるが，この指標は White-Box Cryptography が使われる環境との整合性に問題がある．

例えば，White-Box Cryptography が DRM に使用される場合，ユーザは PC やスマートフォンなどの機器を使ってコンテンツを視聴する．このとき，機器のメモリーは有限であるためプログラムのサイズは小さいほど使い勝手は良いが，Incompressibility/Space-hardness の指標はそれに逆行する．従って，Incompressibility/Space-hardness の評価が高い White-Box Cryptography 方式が優れていると判断することは必ずしも適切ではない．

3. Code Lifting 対策の検討

Code Lifting 対策について検討し，耐タンパーソフトウェアの機能改変困難性を利用する多重利用検出方法を説明する．

3.1 Code Lifting による不正使用

Code Lifting の結果，同一の White-Box Cryptography 実装が複数存在する状況を想定する．それらはコピーであるため，同じ鍵を内蔵する．

プログラムの不正コピーを防ぐ方法はいくつかある．例えば，プログラムが特定の機器でのみ実行できる仕組みが考えられる．これは node-locking[14], [24] と呼ばれ，機器のハードウェアのユニークなビット列[*3]とプログラムが関連付けられており，関連付けられたビット列との組み合わせでなければプログラムは正常実行しないように構成される．PC アプリケーションの不正コピー対策のために PC のパラレルポートや USB ポートにドングルと呼ぶ専用ハードウェアを接続しなければアプリケーションが実行できない仕組みが node-locking の例である．特別なハードウェアを必要とすることが欠点である．

現在，広く使用されている不正コピー対策は，オンラインのライセンス認証である．例えば，Microsoft の Windows OS において Windows XP 以降はユニークなプロダクトキーを入力してアクティベーションを行わないと OS を継続的に使用することができない．Windows OS をインストールした PC がインターネット経由で Microsoft が管理するサーバと通信して，正規プロダクトキーが認証される．サーバが一元的に管理することによって同じプロダクトキーの多重使用は防がれる．

White-Box Cryptography および Obfuscation の Code Lifting 対策にもオンライン認証による多重使用の検出が適用できる．この対策の欠点はネットワーク接続とサーバを必要とすることであるが，White-Box Cryptography のユースケースである DRM ではネットワーク接続とサーバが存在する環境が前提であるため必ずしも欠点とはならない．CAS や HCE の場合は双方向にリアルタイムの通信ができるとは限らない．

3.2 対策のアイデア

安全な White-Box Cryptography あるいは Obfuscation として，共通鍵ブロック暗号方式（SPACE，SPNbox）と Encrypted Signature 方式が知られている．従って，共通鍵ブロック暗号に基づくメッセージ認証コード (MAC) あるいはディジタル署名を用いる認証機能を実装できる．

White-Box Cryptography 実装あるいは obfuscation 実装毎にユニークな鍵（暗号化鍵/復号鍵あるいはディジタル署名生成鍵）を内蔵することにより，実装を識別できる．

アイデアは，同時多重実行の検出である．複数の実装が同時刻あるいは同じ時間帯に例えば DRM コンテンツを視聴することを検出するために，コンテンツを視聴する時間帯に適切なタイミングで認証プロトコルを適宜実行する．

認証プロトコルを実装するとき，プロトコルの各ステップにおける処理が改変されてはならない．例えば，メッセージ認証コードを作成できない実装が，作成できる実装からメッセージ認証コードを横流しをしてもらい，あたかも自分で作成したかのようにプロトコルを実行できてはな

[*3] ユニークなビット列として PUF (Physically Unclonable Function) の出力を用いる仕組みも考えられる．

らない．そのためには，プロトコルを実装するプログラムからそれに含まれる機能を抽出することや改変することができない性質，つまり機能改変困難性が求められる．従って，White-Box Cryptography あるいは Obfuscation による認証プロトコルの実装は機能改変困難性を持たなければならない．具体的なプログラムを対象として秘密情報守秘性と機能改変困難性を実現できる汎用的な手法として自己破壊的耐タンパーソフトウェアが知られている [25], [26] ため，それを用いて機能改変困難性を付与することができる．

3.3 多重実行検出方法

例として，ユーザに配られる DRM 用の再生プログラムを以下のように構成し，運用する場合を説明する．コンテンツはストリーミングを仮定する．

コンテンツ・プロバイダ: コンテンツとユーザを管理し，暗号化したコンテンツをネットワークを通して正規ユーザへ配信する．

DRM 再生プログラム: ネットワークから受信した暗号化コンテンツを復号して再生する，認証サーバと認証プロトコルを実行する．

認証サーバ: コンテンツ・プロバイダが安全に管理・運営するサーバであり，正規ユーザに配布した DRM 再生プログラムをライセンスに応じて管理する．

正規ユーザ: コンテンツ・プロバイダとライセンス契約を交わし，DRM 再生プログラムを受け取り，それを自分の機器で動作させ，コンテンツを視聴する．

不正ユーザ: コンテンツ・プロバイダとライセンス契約を交わさず，なんらかの手段で正規ユーザの DRM 再生プログラムを入手し，それを自分の機器で動作させ，コンテンツを不正視聴しようと試みる．

3.3.1 DRM 再生プログラムの作成と配布

DRM 再生プログラムは，正規ユーザそれぞれに異なるプログラムとなるようにコンテンツ・プロバイダによって次のように作成・配布される．

(1) 安全な White-Box Cryptography 実装あるいは Obfuscation 実装を用意する．例えば，SPNbox のコンテンツ復号プログラムと SPNbox のメッセージ認証コード生成プログラムを作成する．それぞれ異なる鍵を用い，ユーザが異なれば鍵も異なるものとする．

(2) 正規ユーザに ID を割り当て，その ID と上記コンテンツ復号プログラムとメッセージ認証コード生成プログラムを含み，後述の認証プロトコルを実装した DRM 再生プログラムを作成する．

(3) 前記プログラムに対して機能改変困難性を付与する耐タンパー化を適用する．例えば，自己破壊的耐タンパーソフトウェア作成方法で耐タンパー化する．

(4) 耐タンパー化された DRM 再生プログラムを正規ユーザに配布する．

3.3.2 認証サーバ

コンテンツ・プロバイダは後述の認証プロトコルを実行するプログラムおよびユーザ，ID，ライセンス，暗号化鍵，メッセージ認証コード生成鍵，ステータスを管理するデータベースを認証サーバに実装する．

3.3.3 プロトコル

正規ユーザは配布された DRM 再生プログラムを自分の機器にインストールして，DRM 再生プログラムを実行してコンテンツを視聴する．実行された DRM 再生プログラムは以下のプロトコルを認証サーバと実行する．

step-1 耐タンパー化された DRM 再生プログラムは，そのユーザが視聴したいコンテンツを指定されて実行される．最初に，コンテンツ視聴要求として，ユーザに指定されたコンテンツ名，ID を認証サーバに送る．

step-2 認証サーバは，コンテンツ視聴要求を受け取り，ID をキーとしてデータベースを検索し，その ID のユーザのライセンス条件およびステータスを取得する．ステータスはユーザの視聴状況の記録である．

step-3 認証サーバは，ライセンスおよびステータスが許可するコンテンツ視聴要求ならば，ID，コンテンツ名，コンテンツ視聴要求の送信元アドレス（例，IPv6 アドレス）をステータスに記録し，その ID のユーザに配布した DRM 再生プログラムが復号できるように暗号化したコンテンツを作成する．視聴開始時刻をステータスに記録する．

step-4 認証サーバは，認証要求 Req を DRM 再生プログラムに送信する．ID，コンテンツ名 $name$，時刻 $time_{Req}$，送信元アドレス $address$，乱数 $nonce$ 等を連接したメッセージ

$$M_{Req} = ID||name||time_{Req}||address||nonce||etc$$

を作成し，それをその ID のユーザに配布した DRM 再生プログラムが復号できるように暗号化した暗号文を認証要求 Req とする．

step-5 DRM 再生プログラムは，受け取った認証要求 Req を復号して M_{Req} を取り出し，M_{Req} に時刻 $time_{Rep}$ を挿入したメッセージ

$$M_{Rep} = ID||name||time_{Req}||time_{Rep}||address||nonce||etc$$

に対するメッセージ認証コード MAC を生成し，認証サーバへ M_{Rep} と MAC を送る[*4]．

step-6 認証サーバは，受け取った M_{Rep} と MAC が認証要求 Req に対応する正しいメッセージ認証コード，つまりその ID のユーザに配布した DRM 再生プログラムが M_{Req} に時刻 $time_{Rep}$ を挿入したメッセージに対

[*4] この通信を暗号化する実装を DRM 再生プログラムに含めておき，M_{Rep} と MAC の暗号文を認証サーバ送る方式も本プロトコルのひとつとする．

して作成したメッセージ認証コードであることを認証する．認証された場合は，暗号化したコンテンツをストリーミングで配信し，M_{Rep} と MAC をステータスに記録し，他の送信元アドレスから M_{Rep} と近い時刻の M'_{Rep} とそれに対する MAC' を受け取って認証された記録がステータスに記録されていないことを確認する．認証されない場合あるいは他の送信元アドレスから M'_{Rep} と MAC' を受け取って認証された記録がステータスに記録されている場合は次の step 以降の処理を中断し，ステータスに処理の中断を記録し，その ID をブラックリストに入れる．

step-7 DRM 再生プログラムは，暗号化したコンテンツを受信し，復号し，再生する．

step-8 認証サーバと DRM 再生プログラムは，コンテンツ再生中に適切なタイミングで step-4 から step-7 を繰り返す．

step-10 コンテンツの視聴が終了したら，認証サーバは視聴終了時刻をステータスに記録する．

コンテンツ・プロバイダは，ブラックリストに記録された ID のユーザに対して適切な対応を取る．例えば，そのユーザに落ち度は無いが何らかの方法で Code Lifting されてしまった場合は新しい DRM 再生プログラムを作成して配布し，以前の DRM 再生プログラムからのコンテンツ視聴要求は受け付けないようにする．そのユーザが不正ユーザに意図的に Code Lifting して DRM 再生プログラムを渡したことが判明した場合はライセンス契約を解除する等である．

3.4 考察

Code Lifting 対策としてオンライン認証を採用した多重実行検出方法では，Code Lifting された複数の実装が存在している場合に同じ時間帯に複数の実装が認証プロトコルを実行しなければ多重実行として検出されない．これは，同一の実装が複数存在していても，異なる時間帯にコンテンツ再生が行われるならば正当なコンテンツ視聴だとみなせるからである．前記多重実行検出方法を用いれば，ライセンスに応じて同時に複数の実装の実行を許容する DRM システムを構成することも可能である．

上に述べた認証プロトコルでは Code Lifting 行為そのものを防ぐことはできないが，ユーザに落ち度が無く Code Lifting された場合は，その実装を無効化することが可能である．

CAS や HCE の場合は，双方向のリアルタイム通信ができるとは限らないため，本稿で述べた多重実行検出方法はそのままでは応用できない．CAS の場合は，コンテンツ・プロバイダがユーザに放送するシステムなので，適切なタイミングで実装を更新する方法が Code Lifting 対策のひとつとして考えられる．HCE の場合は，ユーザのスマートフォンから決済システム提供者へ決済データが送られるので，決済システム提供者が決済の多重使用を検出することが可能である．決済の多重使用を検出方法は電子キャッシュ [10] 等の既存の研究が参考になる．

Incompressibility/Space-Hardness の指標に関しては，前記 DRM 再生プログラムにおいて活用された機能改変困難性を用いる以下のような改善案が考えられる．機能改変困難性を付与できる耐タンパーソフトウェア作成方法を利用すれば，コンパクトな White-Box Cryptography 実装を含むプログラム全体を耐タンパー化できる．この場合，そのプログラムから White-Box Cryptography 実装だけを抽出しようとしても機能改変困難性のため抽出は困難である．この考え方を応用すれば，ユーザの機器に存在するライブラリや OS カーネル等と不可分な構成の耐タンパーソフトウェア群を構成することも可能である．このとき，Code Lifiting の対象は White-Box Cryptography 実装を含む耐タンパーソフトウェア群となるため，コピーすべきサイズは増える．従って，Incompressibility/Space-Hardness の評価が高い White-Box Cryptography 実装ではなくても Incompressibility/Space-Hardness の耐性を強めることが可能なので，機能改変困難性は Code Lifitng への対策としても有効な重要な性質である．

4. まとめ

White-Box Cryptography の Code Lifting 対策を検討した．White-Box Cryptography の特徴を活かした対策としてオンラインで実装を認証するプロトコルを提案し，同時多重実行を検出して対応できることを示した．

考察で述べたように，Incompressibility/Space-Hardness の評価が高い White-Box Cryptography 実装ではなくても Incompressibility/Space-Hardness の耐性を強めることが可能なので，機能改変困難性は Code Lifitng への対策としても有効な重要な性質である．

参考文献

[1] D. Aucsmith, “Tamper resistant software: an implementation,” Proceedings of the First International Workshop on Information Hiding, Springer-Verlag, LNCS vol.1174, pp.317–333, 1996.
[2] S. Banik, A. Bogdanov, T. Isobe, M. B. Jepsen, “Analysis of software countermeasures for whitebox encryption,” Fast Software Encryption 2017, 2017.
[3] B. Barak, O. Goldreich, R. Impagliazzo, S. Rudich, A. Sahai, S. Vadhan, and K. Yang, “On the (im)possibility of obfuscating programs” Advances in Cryptology - Crypto 2001, Springer, LNCS vol.2139, pp.1–18, 2001.
[4] O. Billet, H. Gilbert, C. Ech-Chatbi, “Cryptanalysis of a white box AES implementation,” In Selected Areas in Cryptography 2004 (SAC 2004), pp. 227-240, 2004.
[5] A. Bogdanov, T. Isobe, “White-box Cryptography Revisited: Space-Hard Ciphers,” Proceedings of the 22nd ACM

SIGSAC Conference on Computer and Communications Security (CCS'15), ACM, pp. 1058–1069, 2015.
[6] A. Bogdanov, T. Isobe, E. Tischhauser, "Towards practical whitebox cryptography: optimizing efficiency and space hardness," ASIACRYPT 2016, Part I, LNCS, vol. 10031, pp. 126–158, 2016.
[7] J. W. Bos, C. Hubain, W. Michiels, P. Teuwen, "Differential computation analysis: hiding your white-box designs is not enough," Cryptographic Hardware and Embedded Systems – CHES 2016, Springer, pp. 215–236, 2016.
[8] R. Canetti and R. R. Dakdouk, "Obfuscating point functions with multibit output," Advances in Cryptology - EUROCRYPT 2008, Lecture Notes in Computer Science, vol.4965, Springer, pp. 489-508, 2008.
[9] H. Chang and M. J. Atallah, "Protecting software codes by guards," Revised Papers from the ACM CCS-8 Workshop on Security and Privacy in Digital Rights Management, Springer-Verlag, LNCS vol.2320, pp.160–175, 2002.
[10] D. Chaum, A. Fiat, M. Naor, "Untraceable electronic cash," Advances in Cryptology — CRYPTO' 88, pp. 319–327, 1990.
[11] Y. Chen, R. Venkatesan, M. Cary, R. Pang, S. Sinha, and M. H. Jakubowski, "Oblivious hashing: a stealthy software integrity verification primitive," Information Hiding 2002, Lecture Notes in Computer Science, vol.2578, Springer, pp.400–414, 2003.
[12] CHES 2017 Capture the Flag Challenge — The WhibOx Contest An ECRYPT White-Box Cryptography Competition, `https://whibox.cr.yp.to/` is redirected to `https://whibox-contest.github.io/` accessed Nov. 19, 2018.
[13] S. Chow, P. Eisen, H. Johnson, P.C. van Oorschot, "White-box cryptography and an AES implementation," In 9th Annual Workshop on Selected Areas in Cryptography (SAC 2002), Springer, LNCS 2595, pp.250-270, 2003.
[14] S. Chow, P. Eisen, H. Johnson, P.C. van Oorschot, "A white-box DES implementation for DRM applications," In Proceedings of 2nd ACM Workshop on Digital Rights Management (DRM 2002), Springer, LNCS 2696, pp. 1–15, 2003
[15] F. B. Cohen, "Operating system protection through program evolution," Computers & Security, vol.12, No.6, pp. 565–584, 1993.
[16] C. Collberg, J. Nagra, Surreptitious Software: Obfuscation, Watermarking, and Tamperproofing for Software Protection, 1st edition, Addison-Wesley Professional, 2009.
[17] C. Collberg, C. Thomborson, and D. Low, "A taxonomy of obfuscating transformations," Technical Report # 148, Department of Computer Science, The University of Auckland, July 1997.
[18] P.-A. Fouque, P. Karpman, P. Kirchner, B. Minaud, "Efficient and Provable White-Box Primitives," ASIACRYPT 2016, Part I, LNCS, vol. 10031, pp. 159–188, 2016, and Cryptology ePrint Archive: Report 2016/642, 2016.
[19] S. Garg, C. Gentry, S Halevi, M. Raykova, A. Sahai, B. Waters, "Candidate indistinguishability obfuscation and Functional encryption for all circuits," FOCS '13 Proceedings of the 2013 IEEE 54th Annual Symposium on Foundations of Computer Science, pp. 40–49, 2013.
[20] L. Goubin, J-M. Masereel, M. Quisquater, "Cryptanalysis of White Box DES Implementation," Cryptography ePrint Archive, 2007/035.
[21] S. Hada, "Secure obfuscation for encrypted signatures," Advances in Cryptology - EUROCRYPT 2010, Lecture Notes in Computer Science, vol.6110, Springer, pp.92–112, 2010.
[22] S. Hohenberger, G. N. Rothblum, A. Shelat, and V. Vaikuntanathan, "Securely obfuscating re-encryption, " Theory of Cryptography, 4th Theory of Cryptography Conference, TCC 2007, Lecture Notes in Computer Science, vol.4392, Springer, pp.233–252, 2007.
[23] B. Horne, L. R. Matheson, C. Sheehan, and R. E. Tarjan, "Dynamic self-checking techniques for improved tamper resistance," Revised Papers from the ACM CCS-8 Workshop on Security and Privacy in Digital Rights Management, Springer-Verlag, LNCS vol.2320, pp.141–159, 2002.
[24] Y. De Mulder, "White-box cryptography analysis of white-box AES implementations," Doctoral Dissertation, KU LEUVEN, Feb. 2014.
[25] 大石和臣，松本勉，"自己破壊的タンパー応答を発生する耐タンパーソフトウェア," 電子情報通信学会論文誌. A, J94-A(3), pp.192-205, 2011.
[26] K. Oishi and T. Matsumoto, "Self destructive tamper response for software protection," ASIACCS'11, pp.490-496, 2011.
[27] M. Rivain, J. Wnag, CHES 2017 Challenge, `https://ches.2017.rump.cr.yp.to/` accessed Nov. 19, 2018.
[28] WhibOx 2016, `https://www.cryptoexperts.com/whibox2016/`, accessed Nov. 19, 2018.
[29] B. Wyseur, W. Michiels, P. Gorisson, B. Preneel, "Cryptanalysis of White-Box DES Implementation with Arbitrary External Encodings," SAC2007, 2007.