修士論文

VM を保護するセキュアプロセッサと
それを用いたアプリケーション認証手法

Secure Processor for Protecting VM
and its Application to Authentication

平成26年2月6日提出

指導教員　坂井 修一 教授

東京大学大学院 情報理工学系研究科

電子情報学専攻

48-126444 山田 剛史

概要

近年，オープンソースソフトウェアの普及やリバースエンジニアリング技術の進展により，アプリケーションや OS に開発者の意図しない改変が加えられる可能性が高まっている．アプリケーションや OS に開発者の意図しない改変が加えられうる状況において，アプリケーション製作者はアプリケーションが動作する環境の安全性及び，インストールされているアプリケーション自身の完全性を検証する必要がある．プラットフォームの完全性を検証する手法として，TPM を用いた Trusted Boot がある．TPM を用いた Trusted Boot では OS を含めたプラットフォーム全体のハッシュ値を取得し認証を行う．TPM を用いた Trusted Boot が保証できるのは OS を含めたプラットフォームが最新の状態であることであるが，今日の OS は複雑化，高度化しており，OS に含まれる全ての脆弱性を排除することは極めて困難である．これに対し，セキュアプロセッサはアプリケーションを OS の影響を受けることなく実行することができる．しかし，セキュアプロセッサを用いた場合，既存の OS に大きな改変を加える必要がある．また，アプリケーションの認証機能を持たないため，不正なアプリケーションがインストールされてしまった場合，セキュアプロセッサだけでは対処できない．そこで，本論文ではセキュアプロセッサを改良し，既存の OS を改変することなく利用できる VM セキュアプロセッサの提案を行う．また，VM セキュアプロセッサを用いてアプリケーション製作者が自身のアプリケーションの完全性を検証することを可能とする手法の提案も併せて行う．

目 次

図 目 次

第1章　はじめに

近年，音楽や書籍等，多くのコンテンツがデジタル化されている [1]．しかし，デジタルコンテンツは従来のコンテンツと異なり，物理的な媒体が存在せず，複製を行うことが容易である．物理的な媒体の場合，媒体のユーザーが正当なコンテンツのユーザーであったが，デジタルコンテンツには物理的な媒体が存在しないため，正当なコンテンツのユーザーから非正規なユーザーにコンテンツデータが流出しないようにする必要がある [2]．

デジタルコンテンツにおいても，コンテンツを扱うアプリケーションや OS が改ざんされていない場合，非正規なユーザーにコンテンツデータが流出することはない．しかし，アプリケーションや OS が改ざんされていた場合，コンテンツデータが流出する可能性がある．リバースエンジニアリング技術の進展や，オープンソースソフトウェアの普及によって，アプリケーションや OS が改ざんされる可能性は高まっている．

このような問題にはソフトウェアの信頼性に基づく DRM 等では十分な対策を行うことが出来ない．

改変が比較的容易なソフトウェアに対し，ハードウェアは製造後の改変が極めて困難である．しかし，古くから用いられている耐タンパパッケージのような手法では，現在の情報化社会において要求されるソフトウェアの拡張性に対応できない．また，耐タンパパッケージにはセキュリティホールが製造後に発見されても，修正を行うことが困難であるという問題点もある．

現在，ハードウェアの信頼性に基づいてプラットフォーム全体の完全性を検証する手法として，TPM を用いた Trusted Boot が提案されている．TPM[9, 10] を用いた Trusted Boot では，プラットフォーム全体のハッシュ値を取得し認証を行っている．しかし，TPM を用いた Trusted Boot で検証が可能であるのは，プラットフォームが最新の状態であることであり，プラットフォームに脆弱性が存在しないことではない．特に，現在の OS はプログラムの規模が非常に大きく，未知の脆弱性が多数存在していると考えられる．よって，TPM を用いた Trusted Boot では OS の未知の脆弱性からアプリケーションを保護することはできない．

セキュアプロセッサは特権プロセスによるユーザープロセスのメモリ領域へのアクセスを制限することにより，アプリケーションの安全性を確保している．しかし，セキュアプロセッサを用いた場合、既存の OS に大きな改変を加える必要がある．また，セキュアプロセッサはアプリケーションの認証を行わないため，不正なアプリケーションを動作させられる可能性がある．

そこで我々はセキュアプロセッサを改良し，既存の OS を改変することなく利用できる VM セキュアプロセッサの提案を行う．また，VM セキュアプロセッサを用いてアプリケーション認証を行うことの提案も併せて行う．本手法はセキュアプロセッサに認証を行うための機能を追加することで，アプリケーション製作者がユーザー端末上における，自身のアプリケーションの完全性を検証することを可能とする．

以後，2 章で既存のコンテンツ保護手法について述べ，3 章で VM セキュアプロセッサの提案を行う．その後，4 章で VM セキュアプロセッサを用いたアプリケーション保護手法について提案を行い，5 章で提案手法に対する攻撃手法とその対策ついて述べた後，6 章で考察を行い，7 章でまとめを行う，

第2章　既存のコンテンツ保護手法

2.1　DRM

ソフトウェアを用いた認証の１つに，デジタル著作権管理 (Digital Rights Management：DRM) 技術 [3, 4] がある．DRM は，デジタルデータとして表現されたコンテンツの著作権を保護する技術である．DRM では，著作権保護のために，配布されたデータの再利用や複製に制限を課す．

例としては，コンテンツを暗号化された状態で配布し，特定のアプリケーションでしか復号化・再生を出来ないようにすることで，第三者による複製や利用を困難にさせる等の方法 [5, 6] がある．

2.1.1　認証方法

一般的に DRM を含め，ソフトウェアを用いた認証は，以下の 2 つの情報を用いて実現される．

- ユーザー ID とパスワード
- PC のプロダクト ID など（PC や OS 固有の値）

ソフトウェアを用いた認証では，前者を利用して利用者の資格情報を確認し，後者を利用して端末を特定している．

ソフトウェアのみで実装できるため，実装が容易であり広く用いられている．

2.1.2　DRM に対する攻撃方法

リバースエンジニアリング

初期の DRM の実装である DVD の CSS では，DVD の再生ソフトに固定の暗号鍵を埋め込むという単純な手法であったため，リバースエンジニアリングにより暗号鍵が流出し，実効性が失われてしまった．このため，現在ではソフトウェア起動後に認証を行い，ネットワークから暗号鍵をダウンロードする実装が多い．しかし，このようにネットワークから暗号鍵をダウンロードする手法であっても，ソフトウェアのみで機能を実現するために，コンテンツを扱うソフトウェアをリバースエンジニアリングして修正を加えることでコンテンツはクラックされてしまう．

不正な OS を用いる

DRM 等の従来のソフトウェアを用いた認証は，ソフトウェアの実行環境である OS の信頼性の上に成立している．したがって，不正な OS を用いた場合，認証に用いる情報を改竄することが可能であり，認証の信頼性を著しく低下させることが可能である．また，OS は特権プロセスで動作しているため，OS は全てのアプリケーションの実行中のデータに対しアクセスを行うことが可能である．このため，不正な OS を用いることで，アプリケーションがコンテンツの復号化を行ったタイミングでアプリケーションのデータにアクセスし，コンテンツを取得することが可能である．

2.1.3 DRM のまとめと問題点

DRM はソフトウェアのみを用いて実装されているため，リバースエンジニアリングにより，クラックされてしまう．

また，DRM 技術においては，復号可能なアプリケーションを限定し，そのアプリケーションの信頼性を高めることでコンテンツの保護を行う．しかし，このことは，コンテンツの保護が復号アプリケーションの信頼性，さらにはアプリケーションの実行環境の信頼性に，依存することを意味する [7].

従って，実行環境である OS が改竄されていたり，不正な OS を用いられていたりした場合，DRM は無効化されてしまう．

2.2 耐タンパパッケージ

耐タンパパッケージとは古くからある手法で，全ての機能を１つのパッケージに含めてしまうことで物理的なセキュリティを確保すると同時に，ソフトウェアの改竄等を防止する手法である．

例えば，IBM 4758 cryptograph coprosessor は 1997 年に IBM が発表した PCI 接続のセキュリティプロセッサである．図 2.1 に示すように，パッケージ内に Intel i486 プロセッサ，ハードウェア DES 処理チップ，公開鍵チップ，DRAM，乱数生成チップ，秘密鍵を格納するバッテリー付き RAM，システム格納用の FLASH ROM，時刻用のチップを内蔵している．

IBM 4758 cryptograph coprosessor においては，CPU 部分に汎用的なチップである Intel i486 プロセッサーを用いることで，システムのコストを低減している．

2.2.1 耐タンパパッケージのまとめと問題点

耐タンパパッケージはハードウェア・ソフトウェア両方の面から見て，非常に高いセキュリティレベルを提供することが可能である．

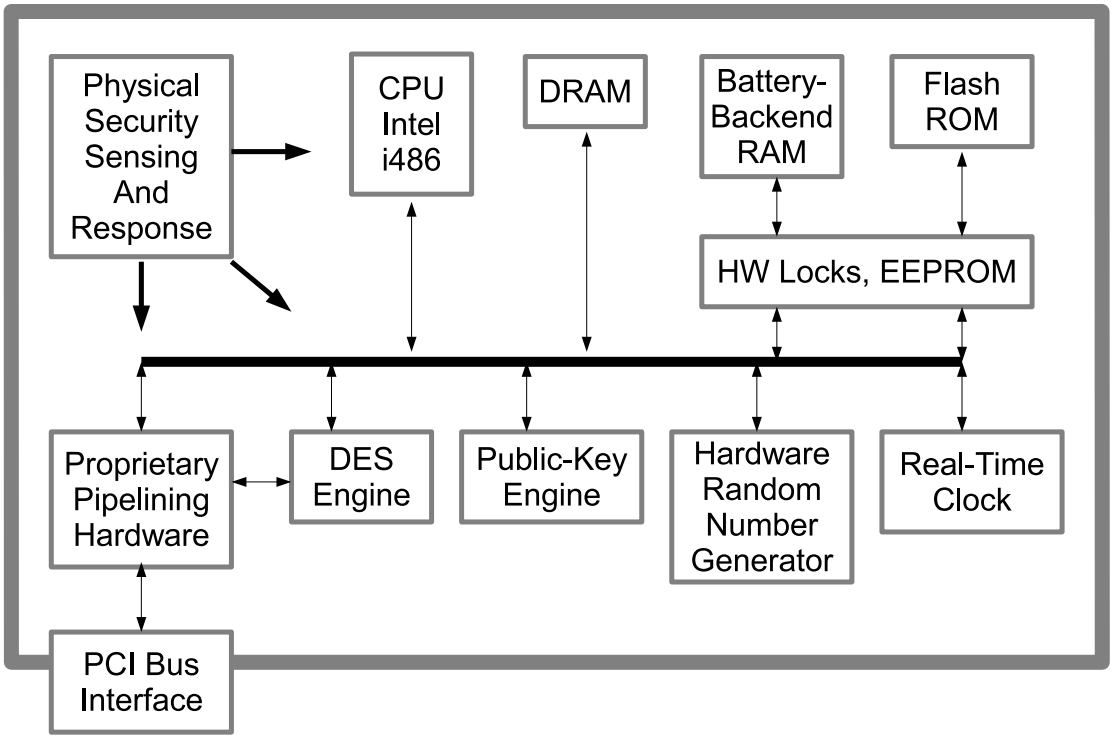

図 2.1: IBM 4758

しかし，目的が限定されがちであり，システムの汎用性が低く，汎用性が低いため高コストにつながりやすい．

また，全ての機能を 1 つのパッケージに含めてしまっているため，故障時やアップグレード時にシステム全体を取り替える必要があり，一旦製品として出荷された後に不具合やセキュリティホールが発見された場合，修正を行うことが極めて困難である．

2.3 TPM を用いた TrustedBoot

本節ではコンテンツ保護に応用可能な，既存のプラットフォーム認証手法として，TPM を用いた Trusted Boot を取り上げる．

2.3.1 TrustedBoot

TPM とは暗号化・復号・電子署名の生成・検証等の機能を持つセキュリティチップのことである．この TPM を用いて，プラットフォームの完全性を検証する手法である Trusted Boot を行うことができる．

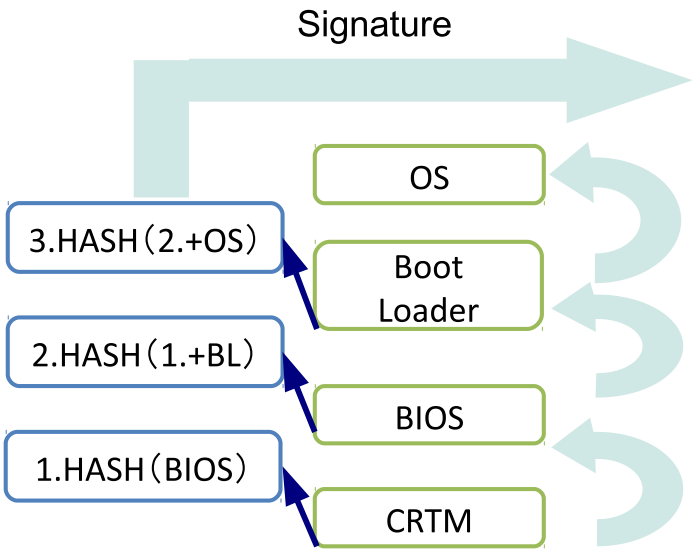

図 2.2: TPM を用いた Trusted Boot

Trusted Boot において TPM はハッシュ値を順に畳み込みながら計測し保存する [14]．Trusted Boot とは，図 2.2 に示すように，起動時に物理的に信頼できる BIOS の CRTM(Core Root of Trust Measurement) から計測を開始し，直前に起動したコンポーネントが起動するコンポーネントのハッシュ値の計測を行う．ハッシュ値の計測は OS を含むディスク全体のハッシュ値を取得するまで繰り返される．各計測結果であるハッシュ値はそのつど PCR(Platform Configuration Register) に畳み込んで保存していく．

最終的なハッシュ値に対し，TPM は電子署名・暗号化を行った上で，認証者に通知する．認証者はハッシュ値により，認証対象のプラットフォームにおいて完全性が保たれているかを検証する．

CRTM

CRTM とは最初に起動するコードのことである．TPM にはコードの計測を行う機能はないため，最初に起動する CRTM は CRTM 自身の計測を行う．CRTM が TPM を利用した認証の信頼性の根幹となるので，CRTM は BIOS 内の書き込み不可能な領域に記録されている．

PCR

PCR は TPM 内の揮発性レジスタである．PCR はブート時に初期化され，特殊な命令によってのみ書き込みが可能である．PCR に物理的な攻撃が行われた場合

であっても，TPM チップにはセンサー入力があるため，物理的な攻撃が行われたことを検出することが可能である．

2.3.2 TPM を用いた TrustedBoot のまとめと問題点

TPM を用いた TrustedBoot では，CRTM を起点にして BIOS やストレージを連鎖的に計測することで，プラットフォームの各コンポーネントのハッシュ値の計測を行う．計測されたハッシュ値に対し，TPM が電子署名を行った上で認証者へ通知を行い，認証者はあらかじめ用意された，正しいプラットフォームより得られるハッシュ値と比較を行うことで，プラットフォームの認証を行う．

しかし，近年 OS の規模が肥大化しており，OS に含まれる未知の脆弱性を完全に排除することは困難になっているが，TPM を用いた Trusted Boot で保証できるのは OS が最新の状態であることであり，OS に脆弱性がないことは保証できない．通常，アプリケーションは実行環境である OS の脆弱性の影響を受ける．従って，TPM を用いた Trusted Boot では OS に含まれる未知の脆弱性からアプリケーションを保護することはできない．

さらに TPM を用いた Trusted Boot においては，認証を行うサーバーにリアルタイムにハッシュ値が通知されているわけではない．そのためプラットフォームが変更された場合，再度認証を行う必要がある．しかし，図 2.3 に示すように，プラットフォームを再度認証する場合，プラットフォームに変更が加えられた後に認証要求を出すため，認証が再度完了するまでにはタイムラグが存在し，この認証のタイムラグの間にプラットフォームに対して改変が行われる可能性がある．

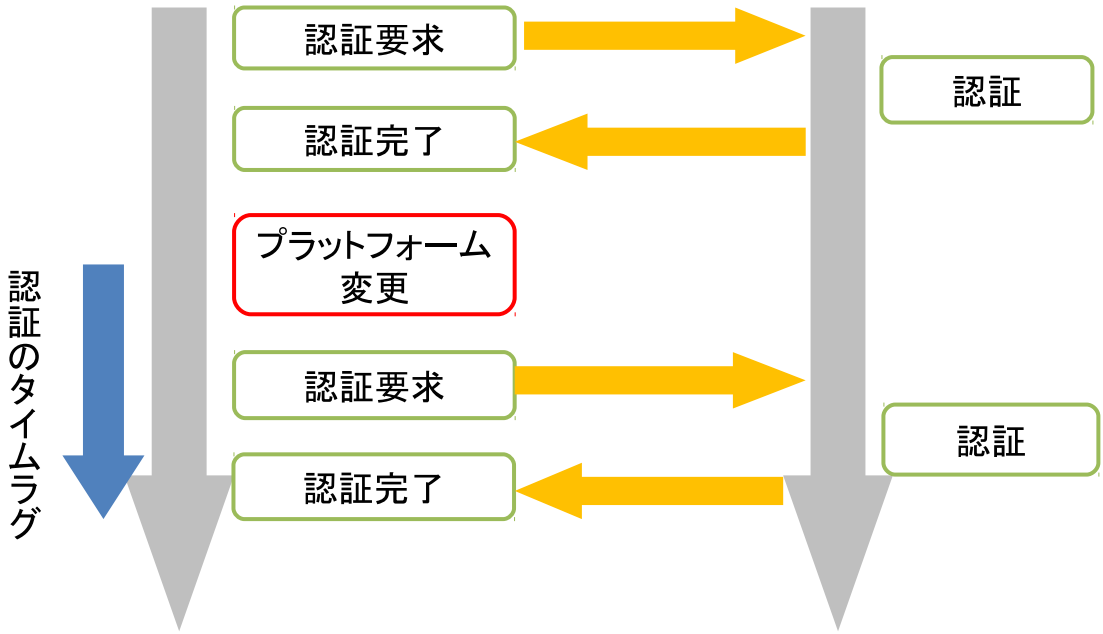

図 2.3: 認証時のタイムラグ

第3章　VMセキュアプロセッサ

本章ではハードウェアを用いた既存のアプリケーション保護手法として，セキュアプロセッサを取り上げ，セキュアプロセッサの問題点に対応したVMセキュアプロセッサの提案を行う．

3.1　セキュアプロセッサ

セキュアプロセッサとは特権プロセスによるユーザープロセスのメモリ領域へのアクセスを制限することにより，信頼できないOSやハードウェアの上でプログラムを安全に動かすことのできるプロセッサである．セキュアプロセッサには秘密鍵および公開鍵が製造時に埋め込まれており，秘密鍵を流出させることは極めて困難である．セキュアプロセッサを用いたシステムでは，プロセッサチップを信頼できるもの，プロセッサチップ以外を信頼できないものとする．主記憶や二次記憶，OSや他のアプリケーションは信頼できず，攻撃者によって改竄されている可能性があるものと考える．セキュアプロセッサではアプリケーションを解析から保護するために，以下の対策を行う [15].

- アプリケーションのコードの秘匿
- 実行中のアプリケーションの秘匿

3.1.1　アプリケーションのコードの秘匿

アプリケーションのコードを秘匿するために，セキュアプロセッサではアプリケーションバイナリの暗号化を行う．アプリケーション作成者は，プロセッサの公開鍵を用いてアプリケーションを暗号化しておき，プロセッサは実行時にプロセッサ内の秘密鍵を利用してアプリケーションを復号して実行する．復号処理はプロセッサ内部で行われるため，アプリケーションの安全性は保たれる．

3.1.2　実行中のアプリケーションの秘匿

実行中のアプリケーションを秘匿するためには，アプリケーション中で用いるレジスタ，キャッシュ，メモリを秘匿する必要がある．

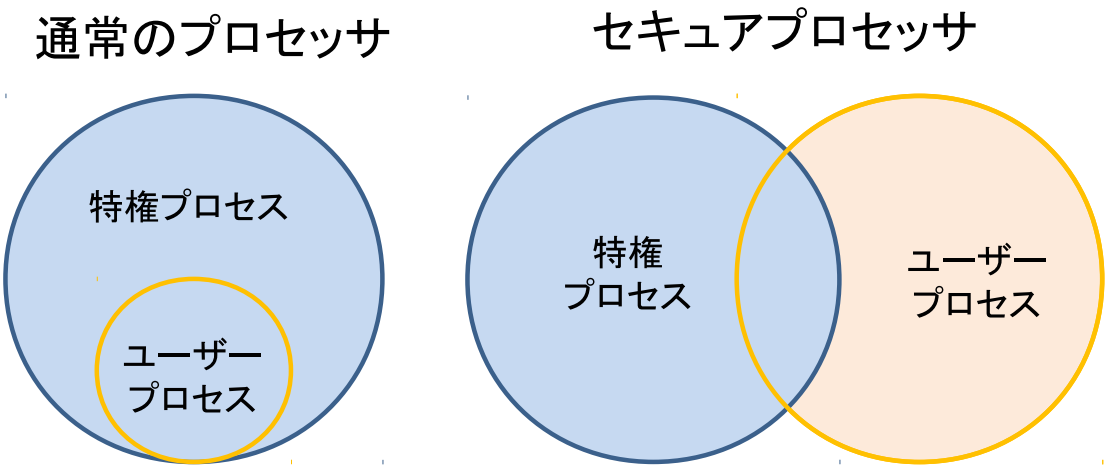

図 3.1: プロセッサによるアクセス制御

レジスタ／キャッシュのアクセス保護

Lie らの提案する XOM[16, 17] では，キャッシュやレジスタにプロセス識別用のタグを追加し，これが一致しないプロセスからの読み込みができないようにしている．このタグにより，特権モードを利用してのキャッシュやレジスタの値の読み取りを防ぐことができる．

メモリの保護

メモリを OS や他のプロセス，ハードウェアを用いたアクセスから秘匿するためメモリの保護を行う．具体的には，図 3.1 に示すように，特権プロセスの権限の及ばないユーザープロセスのメモリ空間を作成し，特権プロセスによるユーザープロセスのデータの盗聴を防止する．

AEGIS　Suh らの提案する AEGIS[18] においてメモリの保護には暗号化を用いる．実行するプロセスごとにランダムに鍵を生成し，プロセッサが管理する．この鍵を用いてアプリケーションのメモリ上のデータの暗号化を行う．プロセスごとに異なる鍵をランダムに生成して利用するので，同一アプリケーションを複数プロセス実行した場合にも，プロセス間で情報が漏洩することはない．

耐ソフトウェアタンパプロセッサ　清水らの提案する耐ソフトウェアタンパプロセッサ [19, 20, 21] においては，図 3.2 に示すように，メモリ内に権限ビットを追加してメモリの保護を行う．耐ソフトウェアタンパプロセッサではメモリ内の情報を暗号化する代わりにビットによる保護を行っている．メモリ内の暗号化を行わないため，ハードウェアタンパに対する耐性はないが，その分高速に動作する．

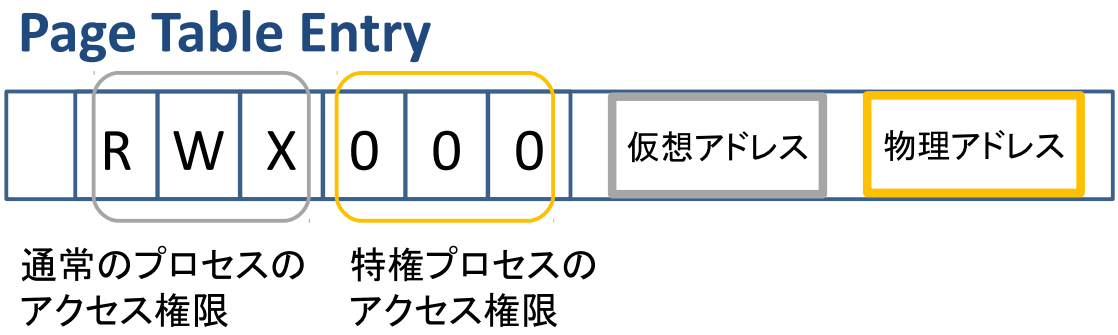

図 3.2: Page Table Entry

3.1.3 セキュアプロセッサのまとめと問題点

セキュアプロセッサはアプリケーションを OS や他のアプリケーションの影響を受けることなく実行することができる．しかし，セキュアプロセッサはプロセスの保護をハードウェアレベルで行うため，セキュアプロセッサに対応した OS が必要である．近年，OS の規模は非常に大きくなっており，セキュアプロセッサに対応した OS を新規に用意するのは非現実的である．また，アプリケーションの認証を行わないため，不正なアプリケーションが実行されたことを検出することはできない．よって，クライアント端末上でデジタルコンテンツを保護することを考えた場合，セキュアプロセッサを用いるだけでは不十分である．

3.2 VM セキュアプロセッサ

本節ではセキュアプロセッサを仮想化環境で動作するように改良した VM セキュアプロセッサの提案を行う．セキュアプロセッサはプロセス単位で情報の保護を行っていたが，VM セキュアプロセッサでは VM 単位で情報の保護を行う．VM セキュアプロセッサは製造時に埋め込まれた秘密鍵から VM 固有の鍵を生成し，各 VM に割り当てられたメモリ領域をハードウェアレベルで透過的に暗号化・復号する．暗号化と復号の処理は透過的に行われるため，既存の OS をそのまま用いることができる．暗号化と復号に用いる鍵はプロセッサチップの内部に埋め込まれているため，外部から取り出すことは極めて困難である．VM セキュアプロセッサを用いたシステムでは，セキュアプロセッサと同様にプロセッサチップのみを信頼するもの，プロセッサチップ以外のものは信頼できないものとする．

3.2.1 VMの保護

VMセキュアプロセッサはVMに割り当てられたメモリ領域を暗号化するため，VMの扱う情報を保護することができる．暗号化はプロセッサ内部のキャッシュからプロセッサ外部のメモリに書き出す際に行われ，復号はメモリからキャッシュに読み込む際に行う．プロセッサのメーカーは製造時に公開鍵，秘密鍵を埋め込み，暗号化・復号には秘密鍵から生成したVM固有の秘密鍵を用いる．通常の仮想化環境で想定される攻撃手段としては，仮想化環境の管理者がVMのメモリを解析し，情報を漏洩させる危険性が指摘されている．しかし，VMセキュアプロセッサを用いたシステムでは仮想化環境の管理者や仮想マシンの物理メモリが信頼できない状況であっても，VMのメモリ領域は暗号化されているため，VMのメモリ領域から情報が漏洩することはなく，安全性は確保される．また，仮想化環境で各VMは互いに異なる鍵で暗号化されているため，他のVMに対して情報が漏洩することはない．

3.2.2 アプリケーションへの応用

VMセキュアプロセッサはVMの保護を行うことができるが，プロセスレベルでの保護機能は持たない．そこで，アプリケーションにVMインターフェイスを搭載し，アプリケーションがVMとして機能するようにする．アプリケーションに対しVMインターフェイスを搭載し，VMとして扱えるようにすることで，VMセキュアプロセッサのメモリの暗号化機能を利用することができる．VMセキュアプロセッサはVMに対し固有のキーを発行しメモリの保護を行うため，ホストOSやホストOS上で動く他のアプリケーション，及びVMインターフェイスを搭載した他のアプリケーションに情報が流出することはない．

3.2.3 VMセキュアプロセッサのまとめ

VMセキュアプロセッサは各VMに割り当てられたメモリ領域を透過的に暗号化・復号する．これにより，既存のOSをそのまま利用しつつ，VMの保護を行うことが可能となる．このように，VMセキュアプロセッサを導入することで，セキュアプロセッサの導入が困難であるという問題点を解決することができた．しかし，VMセキュアプロセッサを用いるのみでは，アプリケーションの認証を行わないため，不正なアプリケーションが実行されたことを検出することができない，というセキュアプロセッサの問題点は解決できない．

第4章　VMセキュアプロセッサを用いたアプリケーション認証手法

本手法は，図 4.1 に示すように，アプリケーション製作者が VM セキュアプロセッサの信頼性の元，外部の認証サーバーを用いて，OS の信頼性に依存せずアプリケーションを認証するものである．アプリケーションの認証を行うため，セキュアプロセッサに対し，メモリ上に展開されたアプリケーションのハッシュ値を直接取得する機能を付加する．また，本手法において，アプリケーションはあらかじめ認証サーバーの公開鍵を内蔵している．

4.1　認証に用いる手法について

4.1.1　共通鍵暗号

本手法では AES[22] 等の共通鍵暗号を用いる．共通鍵暗号においては，あらかじめ情報 m の送信者 S と受信者 R が暗号化に用いる鍵 key を共有しており，S と R 以外に key が流出していないとき，S と R の間で安全に m をやり取りすることができる．以後本稿では m を key を用いて共通鍵暗号化した際の出力を ce_m とするとき，

$$ce_m \leftarrow ckenc_{key}(m) \tag{4.1}$$

と定義する．

4.1.2　公開鍵暗号

本手法では RSA[23] 等の公開鍵暗号を用いる．公開鍵暗号においては，S ・ R ともに公開鍵 pk ，秘密鍵 sk を所持している．S は m を R に送る際，まず自身の秘密鍵 sk_S を用いて m のハッシュ値に対して電子署名を行う．sk_S が外部に流出していないとき， S の公開鍵 pk_S を用いて電子署名の検証を行うことが可能である．検証を行うことにより， m に対する改竄の有無，署名者の正当性の確認を

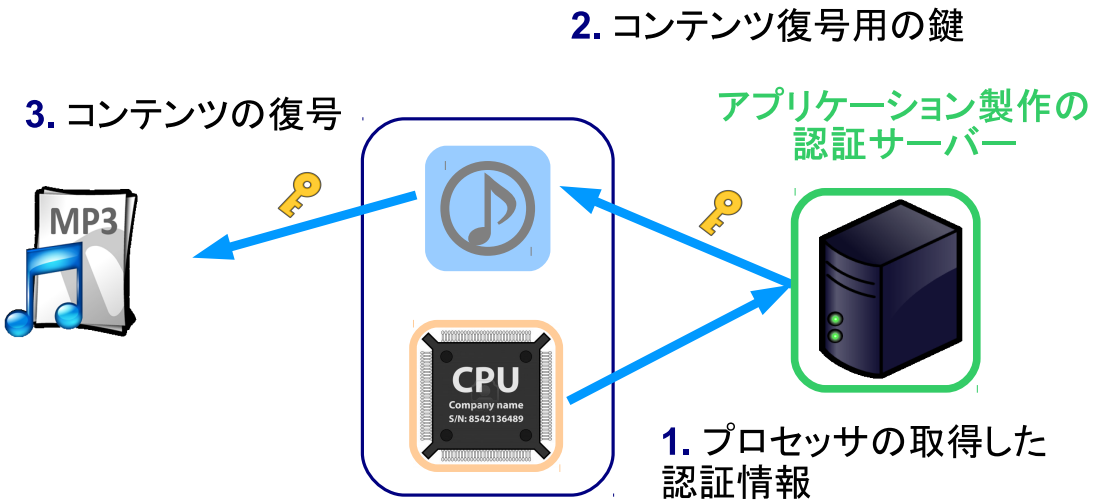

図 4.1: 認証の概要

行うことができる．以後本稿では m に対して sk_S を用いて電子署名を付加したデータを sd_m とするとき，

$$sd_m \leftarrow sig_{sk_S}(m) \tag{4.2}$$

と定義する．

次に， S は R の公開鍵 pk_R を用いて m の暗号化を行う．ここで，公開鍵を用いた暗号化では， pk_R を用いて暗号化を行うと，R の秘密鍵 sk_R を知る者以外は復号を行うことが極めて難しい．以後本稿では pk_R を用いて m の暗号化を行った際の出力を pe_m とするとき，

$$pe_m \leftarrow pkenc_{pk_R}(m) \tag{4.3}$$

と定義する．

4.1.3 ハッシュ関数

本手法では SHA[24] 等の一方向性のハッシュ関数を用いる．ハッシュ関数は入力データに対して，ハッシュ値を出力する．一方向性ハッシュ関数においては特定のハッシュ値を得られる入力データを見付けることが困難である．以後本稿では m に対するハッシュ値を hd_m とするとき，

$$hd_m \leftarrow hash(m) \tag{4.4}$$

と定義する．

4.1.4 データの結合

複数のデータをまとめて扱う場合，データの結合が必要である．以後本稿では m_1 と m_2 を結合した値を m_{12} とするとき，

$$m_{12} \leftarrow m_1 | m_2 \tag{4.5}$$

と定義する．

4.2 認証に用いる鍵について

4.2.1 公開鍵ペアのプロセッサへの埋込

本手法で用いる VM セキュアプロセッサ P には製造時に公開鍵 pk_P，秘密鍵 sk_P を埋め込んでおく．ここで，pk_P には P の製造者の秘密鍵 pk_M にて電子署名を行っておく．プロセッサの製造後の改変は極めて困難であり，製造時に pk_P，sk_P を埋め込んでおくことで，P が正規の本手法に対応したプロセッサであるか確認を行うことが可能である．

4.2.2 アプリケーション内蔵の公開鍵

一般的に外部のサーバーの公開鍵は OS に内蔵されている認証局の公開鍵を利用して検証する．しかし，OS を信頼しない場合， アプリケーション A は OS に内蔵されている認証局の公開鍵も信用することができず，A はネットワークから取得した認証サーバー X の公開鍵 pk_X を検証できない．このため，アプリケーション開発者は， pk_X をあらかじめアプリケーションに入れておく．あらかじめ内蔵しておくことで， A は正しい公開鍵 pk_X を用いて認証情報の暗号化を行うことができる．

4.2.3 ワンタイムキー

本手法では認証のたびに数十文字程度のランダムな文字列を鍵として生成し利用する．認証のたびに生成するため，鍵を生成した時点で生成者以外に鍵が流出していないことが保証されている．以後本稿ではこのような鍵のことをワンタイムキー otk と呼ぶ．

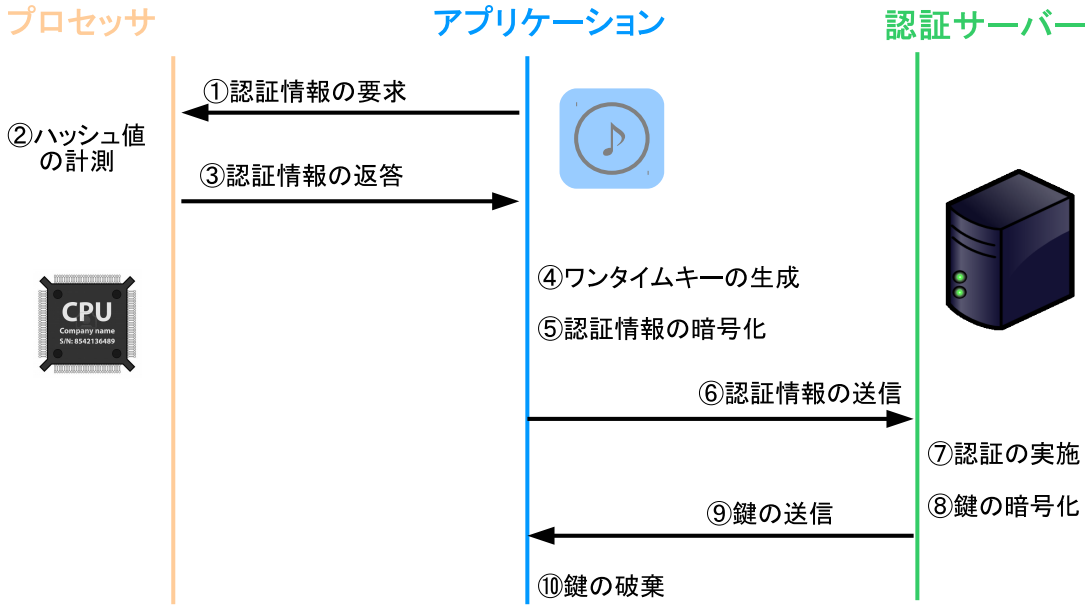

図 4.2: 認証の流れ

4.3 認証の流れ

認証の流れを図 4.2 に示す．なお，図 4.2 における認証手順の番号は以下における番号と一致している．

1. アプリケーション A は起動時に A のハッシュ値 $hash(d)$ をプロセッサ P に対して要求する．

$$hd_d \leftarrow hash(d) \tag{4.6}$$

d は A のメモリ上のデータであり， d には 4.2.2 に示すアプリケーション内蔵の公開鍵 pk_X のデータも含まれる．ここで用いるハッシュ関数は 4.1.3 に示すハッシュ関数である．

2. P は hd_d に P の秘密鍵 sk_P を用いて電子書名を行い， hd_d に電子署名を付加したデータ $sig_{sk_P}(hash(d))$ を A に返す．

$$sd_d \leftarrow sig_{sk_P}(hash(d)) \tag{4.7}$$

ここで用いる電子書名は 4.1.2 に示す電子署名である．また，4.2.1 に示すように P の秘密鍵 sk_P，公開鍵 pk_P は P 製造時にプロセッサに埋め込まれている．

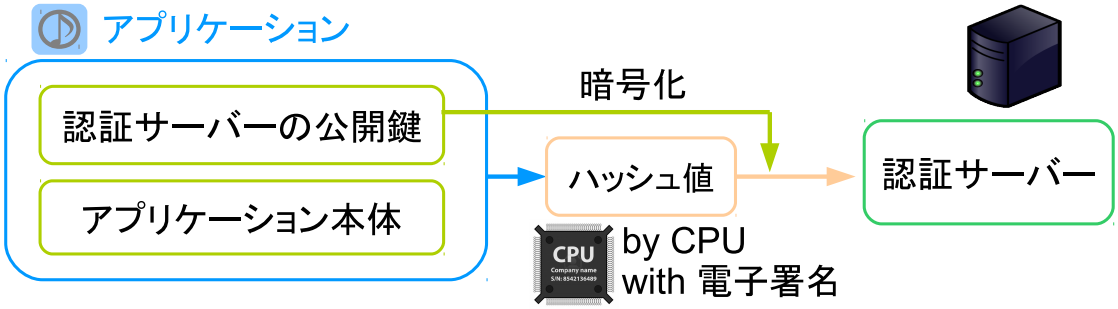

図 4.3: アプリケーション内蔵の公開鍵による暗号化

3. A は認証に用いるワンタイムキー otk を生成する．ここで用いるワンタイムキーは 4.2.3 に示すものである．

4. 図 4.3 に示すように，A は sd_d 及び pk_P そして otk をあらかじめアプリケーションに内蔵されている pk_X で暗号化し，暗号化したデータ $pkenc_{pk_X}(otk|pk_P|sd_d)$ を X に対し送信する．

$$pe_d \leftarrow pkenc_{pk_X}(otk|pk_P|sd_d) \tag{4.8}$$

この暗号化には 4.1.2 に示す暗号化手法を用いる．

5. X は X の秘密鍵 sk_X を用いて pe_d を復号する．

6. X は P の製造者の公開鍵 pk_M を用いて pk_P の正当性の検証を行い，P が本手法に対応した正当なプロセッサであることを確認する．P が本手法に対応した，正当なプロセッサであった場合 pk_P を用いて sd_d が P を用いて取得されたハッシュ値であるか否かの検証を行う．

7. sd_d が正当なものであった場合，X は hd_d がアプリケーションが完全性を保っている場合に得られるハッシュ値と一致しているか検証する．hd_d が正当なものであった場合，X は認証結果 t を otk で暗号化した $ckenc_{otk}(t)$ を A に対して送信する．

$$ce_t \leftarrow ckenc_{otk}(t) \tag{4.9}$$

この暗号化には 4.1.1 に示す暗号化手法を用いる．

8. A は otk を用いて ce_t を復号し，t を取得する．アプリケーションは t を用いてコンテンツの復号を行う．

9. A は終了時に t の破棄を行う．

4.4　アプリケーションのアップデートについて

本認証手法においては，起動時にアプリケーションのバイナリのハッシュ値を測定し，認証を行っている．そのため，アプリケーションのアップデートを行う際は認証サーバー側でアップデートされたバイナリのハッシュ値を登録する必要がある．逆に，認証サーバーにアップデートされたバイナリのハッシュ値が登録されていれば，どのような方法でアプリケーションのアップデートを行っても構わない．

第5章　本認証手法に対する攻撃とその対策

本手法では以下の3つの点において安全性を確保する必要がある．

1. 認証情報の取得

   対象となるアプリケーションの認証情報を本提案手法に対応した正規のプロセッサが取得する．

2. 認証情報の漏洩防止

   認証情報が正しい認証サーバーにのみ解読可能である．

3. 認証結果の保護

   認証結果は認証対象となったアプリケーションのみが知ることができる．

本章では以上3点について順に説明していく．

5.1　認証情報の取得

認証情報の取得に関する攻撃には以下の2つが考えられる．

1. プロセッサに対し認証対象とは異なるアプリケーションのハッシュ値を取得させる．

2. 不正なプロセッサを利用する．

前者に関して，P は実行されているアプリケーション A を把握しているので，このような攻撃は成立しない．

後者については，認証情報 pe_d は $pkenc_{pk_X}(otk|pk_P|sd_d)$ としてプロセッサメーカーの秘密鍵で電子署名された pk_P も送信しているため，4.2.1 に示すように X は pk_P を pk_M を利用して検証可能であり，検証を行った pk_P を用いて sd_d を取得したのが P であることが確認可能である．よって，このような攻撃は成立しない．

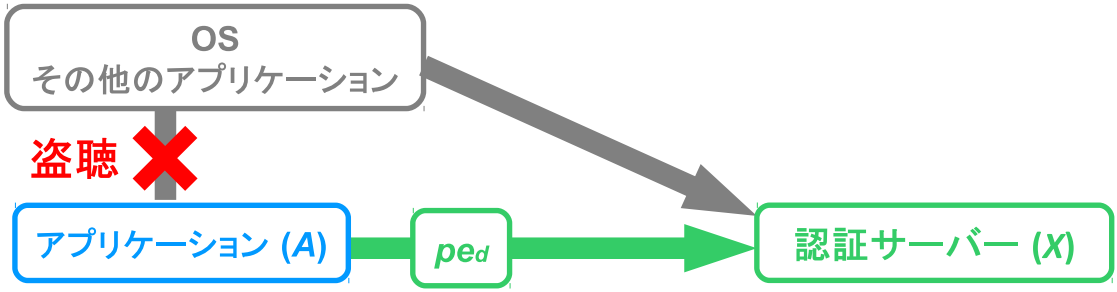

図 5.1: OS，その他のアプリケーションによる認証情報の盗聴

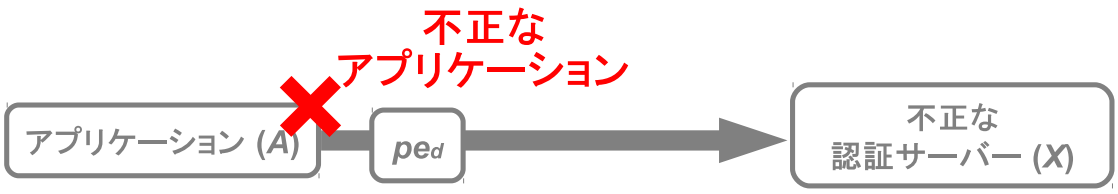

図 5.2: アプリケーションに悪意がある場合

5.2 認証情報の漏洩防止

認証情報を漏洩させる攻撃については，以下の 3 つが考えられる．

1. 図 5.1 に示すように，他のアプリケーションや OS が認証情報を盗聴する．
2. アプリケーションに対し，不正な認証サーバーに認証情報を送信させる．
3. 認証情報を盗聴し，認証情報を認証サーバーに送信する．
4. 認証結果を盗聴する．

1. については，VM セキュアプロセッサによりメモリ上のデータが保護されるので発生しない．
2. については，図 5.2 に示すように， A が自発的に不正な認証サーバーに情報を送信する場合，4.1.3 に示すようにハッシュ値が一方向性を持つため A から正しい認証情報 pe_d つまり $pkenc_{pk_X}(otk|pk_P|sd_d)$ が得られない．図 5.3 に示すように，ネットワークの改変等により， A に認証サーバーを偽装した場合には，4.2.2 に示すように A が X の公開鍵 pk_X を内蔵しており，これを用いて暗号化するため，不正な認証サーバーは pe_d を復号できない．
3. については次節であわせて述べる．

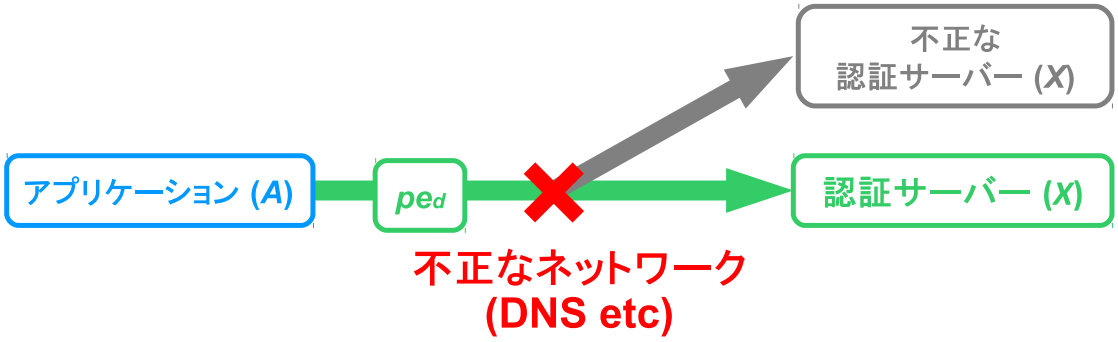

図 5.3: 不正なネットワーク環境化にある場合

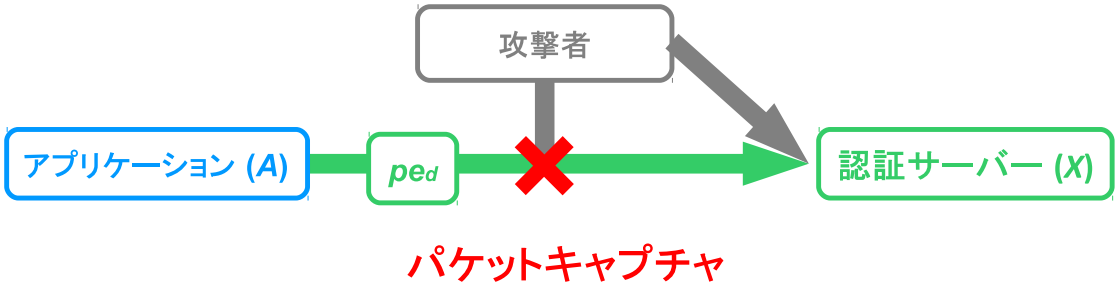

図 5.4: 認証情報を送信中に盗聴された場合

5.3 認証結果の保護

認証結果 ce_t については，認証時開始に作成した otk を共通鍵として暗号化されており，4.1.1 に示すように，共通鍵暗号は共通鍵が流出していない限り復号が困難である．ここで otk には 4.2.3 に示す鍵を用いているため，鍵を生成する以前に共通鍵 otk の流出は発生していない．また，鍵の生成後に otk を扱うのは A と X のみであるが，認証が成功した結果として ce_t を得られるためには，A が正規のアプリケーションである必要があり，A が正規のアプリケーションである場合，A から otk が流出しないことを保証可能である．従って，X が認証者であることと合わせて，認証が行われた A 以外は ce_t を復号できないことが保証可能である．つまり，攻撃者が図 5.4，図 5.5 に示すように，認証情報 pe_d をネットワークの途中で盗聴し，X に送信した場合であっても，盗聴者は ce_t を復号できないため，攻撃として成立しない．

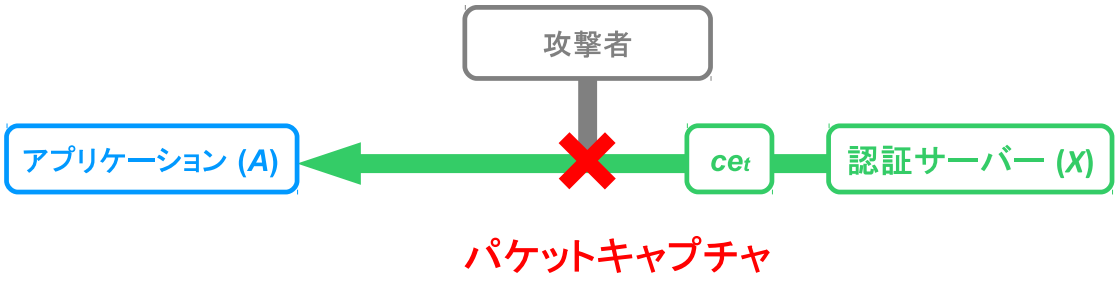

図 5.5: 認証結果を受信中に盗聴された場合

第6章　考察

6.1　過去に提案を行った手法について

まず，セキュアプロセッサを用いてアプリケーション認証を行う手法について提案を行った [11]．その後手法の整理を行い [12]，信頼性の定式化を行った [13]．そして本稿では，これらセキュアプロセッサを用いた手法に対し，既存の OS の改変を必要としない VM セキュアプロセッサの提案を行い，VM セキュアプロセッサを用いたアプリケーション認証手法の提案を行った．

6.2　TPM を用いた Trusted Boot と提案手法の比較

TPM を用いた Trusted Boot と提案手法の比較を行う．図 6.1 左側に示すように，TPM を用いた Trusted Boot は，BIOS，Boot Loader 等の各コンポーネントを順に認証する．順に認証することで，全てのコンポーネントが完全性を保っていることを保証する．

一方で，提案手法では図 6.1 右側に示すように，VM セキュアプロセッサにメモリ上のデータのハッシュ値を取得する機能を付加することにより，プロセッサが直接アプリケーションの認証を行う．本提案手法において BIOS や Boot Loader，OS 等のコンポーネントは認証を行わず，これらのコンポーネントはアプリケーションの安全性に影響を与えない．

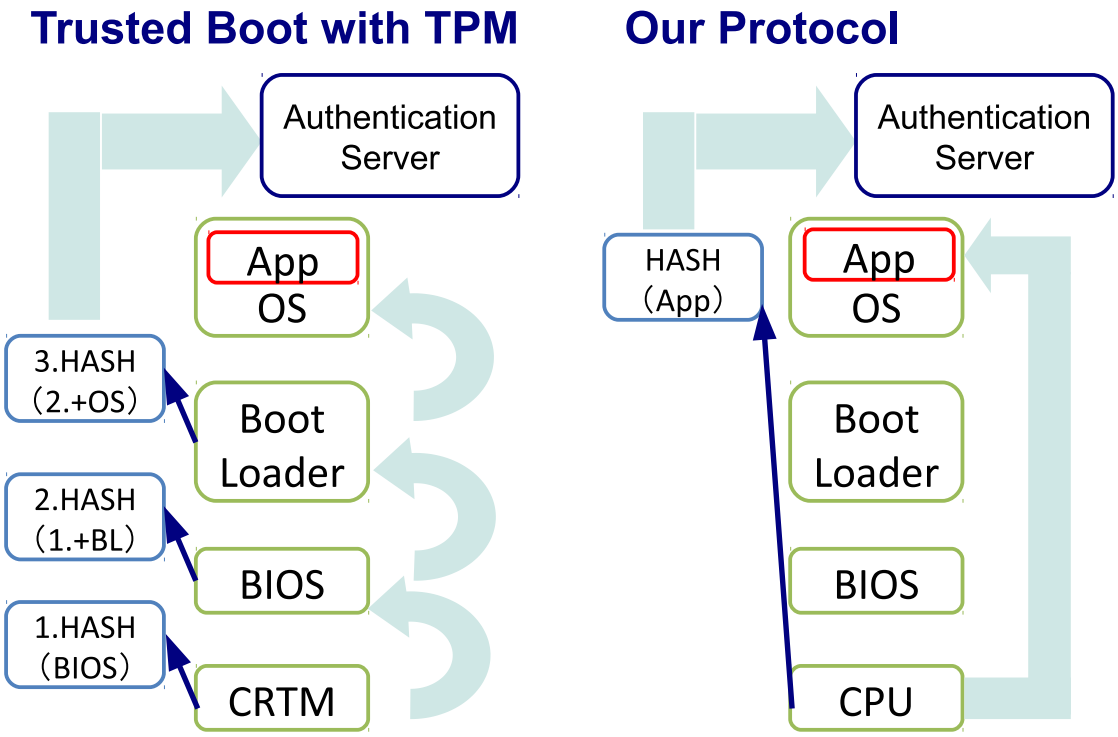

図 6.1: TPM を用いた Trusted Boot と提案手法の比較

第7章　おわりに

7.1　まとめ

本研究はアプリケーション製作者がクライアント側のアプリケーションをOSの信頼性に依存することなく，外部のサーバーから認証し，アプリケーションの完全性を検証するためのものである．セキュアプロセッサには製造時に秘密鍵，公開鍵が埋め込まれており，認証情報の取得者を保証することができる．また，VMセキュアプロセッサを用いることで認証情報がクライアント内で，OSやその他のアプリケーションに漏洩することを防ぐことができる．これらを利用することで，OSの信頼性に依存することなくアプリケーションの完全性を検証できる．

7.2　今後の課題

今後の課題としては，アプリケーションに搭載するVMインターフェイスの開発を行うことがあげられる．

参考文献

[1] 川原崎雅敏. ブロードバンドコンテンツ流通のプラットフォーム recent trends in broadband contents sharing platform.

[2] 横田侑樹, 塩谷亮太, 五島正裕, 坂井修一. 情報漏洩防止プラットフォーム. 電子情報通信学, 信学技報, vol. 109, no. 237, pp. 7-12, 2009.

[3] P. A. Jamkhedkar and G. L. Heileman. Drm as a layered system, DRM '04:Proceedings of the 4th ACM workshop on Digital rights management, New York, NY, USA, ACM Press, pp. 11-21 2004.

[4] W. Ku and C.-H. Chi. Survey on the technological aspects of digital rights management, ISC, pp. 391-403 2004.

[5] Q. Liu, R. Safavi-Naini and N. P. Sheppard. Digital rights management for content distribution, CRPITS'03: Proceedings of the Australasian information security workshop conference on ACSW frontiers 2003, Darlinghaust, Australia, Australian Computer Society, Inc., pp. 49-58 2003.

[6] M. L. Smith. Digital rights managements protecting the digital media value chain, MUM '04: Proceedings of the 3rd international conference on Mobile and ubiquitous multimedia, New York, NY, USA, ACM Press, pp. 187-191 2004.

[7] T. Hauser and C. Wenz. Drm unter attack: Weaknesses in exsting systems, Digital Rights Management, pp. 206-223 2003.

[8] IBM. IBM PCI Cryptographic Coprocessor http://www-03.ibm.com/security/cryptocards/pcicc/overview.shtml

[9] Trusted Computing Group. TCG Specification Architecture Overview.

[10] Trusted Computing Group. TPM Specification Version 1.2 Revision 103.

[11] 山田 剛史, 五島 正裕, 坂井 修一: 信頼できない OS 上でアプリケーション認証を行うシステム, 電子情報通信学会技術報告 CPSY2012-11, Vol. 112, No. 173 pp. 13-18（2012).

[12] Tsuyoshi Yamada, Naruki Kurata, Rie Shigetomi Yamaguchi, Masahiro Goshima, Shuichi Sakai. Minimal Additional Function to Secure Processor for Application Authentication. WEWoRC 2013.

[13] 山田剛史，山口利恵，五島正裕，坂井修一．セキュアプロセッサを用いたアプリケーション認証手法の提案と信頼性の定式化. CSS 2013.

[14] 宗藤誠治, 中村めぐみ, 八木豊志樹, Nguyen Anh Quynh, 須崎有康. Knoppix を利用した trusted computing 技術の体験.

[15] 橋本幹生, 春木洋美. 敵対的な OS からソフトウェアを保護するプロセッサアーキテクチャ. 情報処理学会論文誌 コンピューティングシステム, Vol.45, No.3, March 2004.

[16] Dan Boneh, David Lie, Pat Lincoln, Lohn Mitchell, and Mark Mitchell. Hardware support for tamper-resistatnt and copy-resistant software. Technical report, Stanford University Computer Science, 1999.

[17] David Lie, Chandramohan A. Thekkath, and Mark Horowitz. Implementing an untrusted operating system on trusted hardware. In Proceedings of ACM Symposium on Operating Systems Principles, 2003.

[18] G. Edward Suh, Dwaine Clarke, Blaise Gassend, Marten van Dijk, and Srinivas Devadas. AEGIS: Architecture for tamperevident and tamper-resistant processing. In International Conference on Supercomputing, 2003.

[19] 清水一人, 入江英嗣, 五島正裕, 坂井修一. 耐ソフトウェアタンパ・プロセッサ. 情報処理学会研究報告 2007 no.17, pp. 239–244, 2007.

[20] 文栄光, 塩谷亮太, 五島正裕, 坂井修一. 情報漏洩防止のためのプラットフォーム認証. 電子情報通信学会, CPSY2009-29, vol.109, no.237, pp. 13–18, 2009.

[21] 早川薫，塩谷亮太，五島正裕，坂井修一，プラットフォーム部分認証．電子情報通信学会, CPSY2011-12, vol.111, no.163，pp.19–24，2011.

[22] National Institute of Standards and Technology. Federal Information Processing Standards Publication 197 ADVANCED ENCRYPTION STANDARD (AES), November 26, 2001.

[23] Rivest, R., Shamir, A., and Adleman, L. A method for obtaining digital signatures and public-key cryptosystems. Comm. ACM 21, 2(Feb. 1978), 120-126.

[24] National Institute of Standards and Technology. Federal Information Processing Standards Publication 180-4 SECURE HASH STANDARD (SHS), March, 2012.

発表文献

主著論文

1. VM を保護するセキュアプロセッサとそれを用いたアプリケーション認証手法
   山田剛史, 山口利恵, 五島正裕, 坂井修一
   SCIS 2014(2014).

2. セキュアプロセッサを用いたアプリケーション認証手法の提案と信頼性の定式化
   山田剛史, 山口利恵, 五島正裕, 坂井修一
   CSS 2013(2013).

3. Tsuyoshi Yamada, Naruki Kurata, Rie Shigetomi Yamaguchi, Masahiro Goshima, Shuichi Sakai
   Minimal Additional Function to Secure Processor for Application Authentication
   WEWoRC 2013(2013).

4. 山田 剛史, 五島 正裕, 坂井 修一
   信頼できない OS 上でアプリケーション認証を行うシステム
   電子情報通信学会技術報告 CPSY2012-11(2012).

5. 山田 剛史，早川 薫，都井 紘，五島 正裕，坂井 修一
   情報漏洩防止プロセッサ
   情報処理学会 第 74 回全国大会 (2012).

共著論文

1. A Cloud Architecture for Protecting Guest's Information from Malicious Operators with Memory Management
   Koki Murakami, Tsuyoshi Yamada, Rie Yamaguchi, Masahiro Goshima and Shuichi Sakai
   CODASPY 2014(2014).

2. 顧客情報を不正な管理者による窃盗・改変から保護するクラウドアーキテクチャ
   村上航規, 山田剛史, 山口利恵, 五島正裕, 坂井修一
   SCIS 2014(2014).

3. 不正な管理者によるゲスト情報の窃盗・改変を防止するクラウドアーキテクチャ
   村上航規, 山田剛史, 山口利恵, 五島正裕, 坂井修一
   CSS 2013(2013).

謝辞

本研究を進めるにあたり，坂井修一教授には，相談会等を通じ，様々な御指導を頂きました．

五島正裕准教授には研究テーマの決定から論文の添削まで，本研究に関する多くの相談に乗って頂き，御指導頂きました．

山口利恵特任准教授には，論文の添削を行っていただいたり，海外で学会発表を行うにあたり多くのアドバイスをいただきました．

倉田成己氏には，研究内容や特に英語での論文執筆について様々な助言を頂きました．

事務補佐員の八木原晴水さん，長谷部環さんには，研究を行う上での事務などでお世話になりました．

また，坂井・五島研究室の皆様にも，論文執筆や研究室での生活のサポートなどで大変お世話になりました．心より感謝いたします．