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 †

† 東京大学情報理工学系研究科
113-8656 東京都文京区本郷 7-3-1
{yamada,rie.yamaguchi,goshima,sakai}@mtl.t.u-tokyo.ac.jp

あらまし 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．このような状況において，アプリケーション製作者は自身のアプリケーションの完全性を検証する必要がある．TPM を用いた Trusted Boot 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，プラットフォームの完全性を検証するものである．しかし，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の認証機能を持たないが，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．そこで我々は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を提案する．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．本稿では，先に提案した手法に対しプロトコルの整理を行い，信頼性の定式化を行った．

Application Authentication Using Secure Processor and Formalizing Proofs of Security

Tsuyoshi Yamada†　　Rie Shigetomi Yamaguchi†　　Masahiro Goshima†
Shuichi Sakai†

†Graduate School of Information Science and Technology, The University of Tokyo
7-3-1, Hongo, Bunkyoku, Tokyo, JAPAN
{yamada,rie.yamaguchi,goshima,sakai}@mtl.t.u-tokyo.ac.jp

Abstract These days, open source software and reverse engineering technology are developed, so applications and OS can be modified unexpectedly. An application publisher needs to verify the application is correct by authentication. Trusted Boot with TPM calculates the hash value of the entire platform including OS and if the hash value is correct, the whole platform can be reliable. However, OS is too complex to extinguish vulnerability, if there is vulnerability in the OS, also affect the application. In contrast, a secure processor can execute an application without interference from OS but it cannot validate the application itself. We propose a system that authenticates an application using a secure processor with adding a function to calculate the hash value of an application on the memory. In our protocol, the application publisher is able to verify the application. In this paper, we formalize proofs of security from the past proposal.

1　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．

ここで，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．

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[3, 4] 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，アプリケーションのコード及び実行中のプロセスを秘匿することで，プロセスのメモリ上のデータに対する他のプロセスからの干渉を防ぎ，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用いて，OSからの干渉を受けることなく外部のサーバーがアプリケーションを認証するための機能を付加することを提案する．

そこで我々は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を提案する．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．

本稿では，先に提案した手法 [5] に対しプロトコルの整理を行い，信頼性の定式化を行った．

以後，2章で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について述べ，3章で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う．4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，5章でまとめを行う，

2　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

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，TPMを用いたTrusted Bootとセキュアプロセッサを取り上げる．

2.1　TPMを用いたTrusted Boot

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．

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 [6]．Trusted Bootとは，図1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

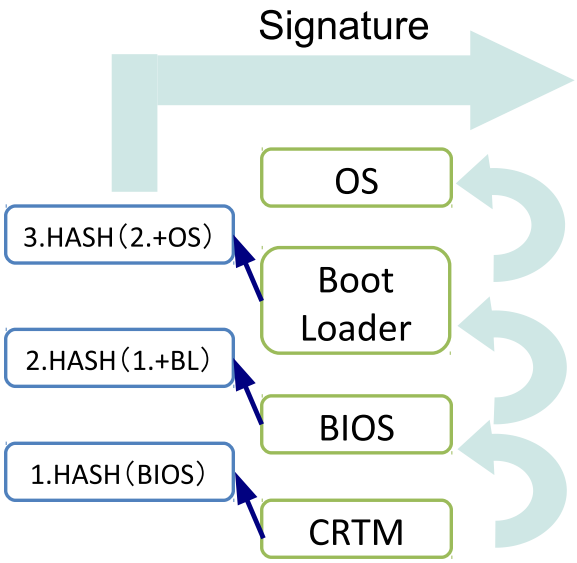

図 1: TPM を用いた Trusted Boot

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．

2.1.1 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．

2.1.2 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．

2.1.3 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 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．

2.2 セキュアプロセッサ

セキュア・プロセッサとは暗号化されたプログラムをプロセッサ内部で復号し，実行することで，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．暗号化のための秘密鍵および公開鍵は製造時に埋め込まれていて，秘密鍵を流出させることはきわめて困難である．セキュア・プロセッサでは，プロセッサチップを信頼できる

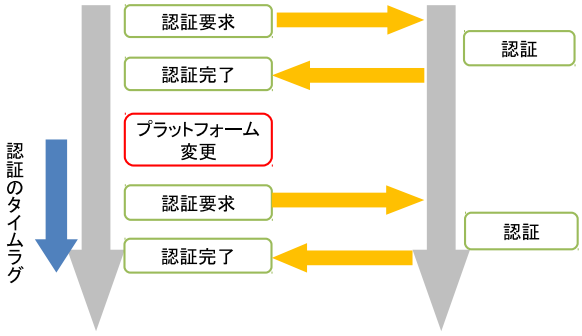

図 2: 認証時のタイムラグ

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 [7].

- 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

2.2.1 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．

2.2.2 実行中のプログラムの秘匿

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．レジスタとキャッシュはプロセッサ内部にあるため，アクセス保護をすることで秘匿を行う．メモリはプロセッサ外部にあるため，暗号化を行うことで秘匿を行う．

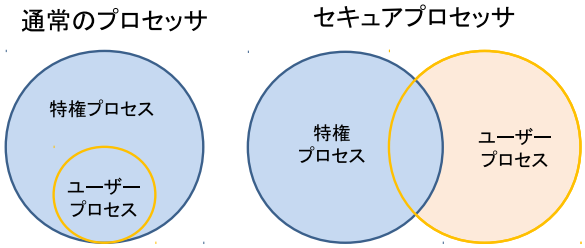

図 3: プロセッサによるアクセス制御

レジスタ／キャッシュのアクセス保護　Lie らの提案する XOM[8, 9] 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 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．

Suh らの提案する AEGIS[10] 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．

また，清水らの提案する耐ソフトウェアタンパプロセッサ [11, 12, 13] においては，図 4 に示すように，ページテーブルエントリ内に権限ビットを追加してメモリの保護を行う．

2.2.3 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，アプリケー

図 4: Page Table Entry

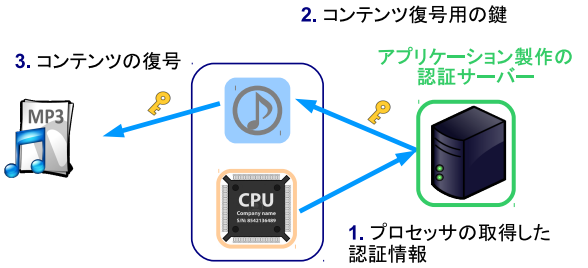

図 5: 認証の概要

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 提案手法

本手法は，図5に示すように，アプリケーション製作者が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．

3.1 認証に用いる手法について

3.1.1 共通鍵暗号

本手法では AES 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 $ckenc_{key}(m)$ とする．

3.1.2 公開鍵暗号

本手法では RSA 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 $sig_{sk_S}(m)$ とする．

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 $pkenc_{pk_R}(m)$ とする．

3.1.3 ハッシュ関数

本手法では SHA 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 $hash(m)$ とする．

3.1.4 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

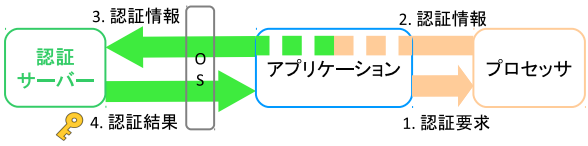

図 6: 認証の流れ

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．

3.2 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．

3.3 認証の流れ

認証の流れを図 6 に示す．

1. 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．ここで d は A のメモリ上のデータであり，d には pk_X のデータも含まれる．これは図 6 の (1) にあたる．
2. P は $hash(d)$ に P の秘密鍵 sk_P を用いて電子書名を行い，$hash(d)$ に電子署名を付加したデータ $sig_{sk_P}(hash(d))$ を A に返す．電子署名とはデータの正当性を保証するための署名情報である．電子署名を検証することで，データの作成者及び，データの完全性を確認することができる．また，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．これは図 6 の (2) にあたる．
3. A は認証に用いるワンタイムキー otk を生成する．
4. A は $sig_{sk_P}(hash(d))$ 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 $pkenc_{pk_X}(otk, pk_P, sig_{sk_P}(hash(d)))$ を X に対し送信する．これは図 6 の (3) にあたる．
5. X は X の秘密鍵 sk_X を用いて $pkenc_{pk_X}(otk, pk_P, sig_{sk_P}(hash(d)))$ 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 $sig_{sk_P}(hash(d))$ 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. $sig_{sk_P}(hash(d))$ が正当なものであった場合，X は $hash(d)$ 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．$hash(d)$ 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．これは図 6 の (4) にあたる．
8. A は otk を用いて $ckenc_{otk}(t)$ 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 本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．

4.1 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 $pkenc_{pk_X}(otk, pk_P, hash(d))$ 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，前章第 1 節 2 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 $sig(hash(d))$ 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．

4.2 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. 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．

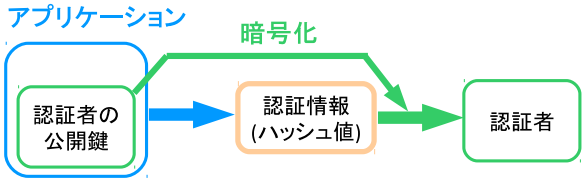

図 7: アプリケーション内蔵の公開鍵による暗号化

1. については，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，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，前章第 1 節 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 $pkenc_{pk_X}(otk, pk_P, hash(d))$ が得られない．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，前章 2 節に示すように 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 $pkenc_{pk_X}(otk, pk_P, hash(d))$ を復号できない．
3. については次節であわせて述べる．

4.3 認証結果の保護

認証結果 t については，t が認証時に作成した otk を共通鍵として暗号化されており，前章第 1 節 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が，前章第 1 節 4 に示す鍵を用いているため，共通鍵の流出は発生していない．従って，認証が行われた A 以外は t を復号できない．つまり，攻撃者が認証情報 $pkenc_{pk_X}(otk, pk_P, hash(d))$ 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 t を復号できないため，攻撃として成立しない．

5 おわりに

5.1 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，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．

5.2 今後の課題

今後の課題としては，複数のアプリケーション間で権限情報をやり取りできるような仕組みについての整備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] Trusted Computing Group. TCG Specification Architecture Overview.

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

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

[6] 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.

[7] 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.

[8] 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.

[9] 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.

[10] 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.

[11] 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.

[12] 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.

[13] 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.