L-012

TPM を用いたハイパーバイザの完全性検証手法の実装

Implementation of Hypervisor Integrity Verification with TPM

岡本 拓也† 山口 利恵† 五島 正裕‡ 坂井 修一†
Takuya Okamoto Rie Yamaguchi Masahiro Goshima Shuichi Sakai

1. はじめに

Amazon EC2 に代表される Infrastructure as a Service (IaaS) では，サービスの利用者に対して， Virtual Machine (VM) と呼ばれる仮想的に構築された基盤の提供を行っている．VM は，ハイパーバイザと呼ばれるソフトウェアによって仮想化されたリソースの上で，実行されている．そのため，ハイパーバイザは VM よりも高い権限を持っている．IaaS において，クラウドオペレータは，IaaS 利用者の VM(ユーザ VM)の管理を行うための特権を持った VM(管理 VM) にアクセスし，ユーザ VM の管理を行うことが考えられる．仮に，IaaS プロバイダ自体に悪意が無くても，クラウドオペレータが悪意を持っていた場合，管理 VM からユーザ VM のメモリ領域を見ることで，ユーザ VM の情報が窃取される可能性がある．また，ハイパーバイザ自体が不正に改変されていた場合，ハイパーバイザがユーザ VM の情報を流出させる危険がある．ところが，IaaS の利用者は，ユーザ VM へのアクセスのみが可能であるため，ユーザ VM から情報が漏洩していることに気付くことは困難である．

このような問題は，次の 2 点を満たすことで解決が可能であると考えられる．1 つは，情報漏洩を防止するハイパーバイザを開発することである．もう 1 つは，そのようなハイパーバイザが，不正な改変を受けることなく IaaS 環境で用いられていることを，IaaS の利用者に保証することである．VMCrypt[1]では，1 つ目の条件を満たすため，管理 VM がユーザ VM のメモリにアクセスした時に，メモリの内容をハイパーバイザが暗号化して返すことで，ユーザ VM のメモリからの情報漏洩を防止している．本稿では，2 つ目の条件を満たすため，ハイパーバイザが，不正な改変を受けていない (完全性が保たれている)こと を，IaaS 利用者に保証するプロトコルの実装を行った結果について述べる．

プログラムの完全性の検証は，プログラムに対する電子署名と，プログラムのハッシュ値を用いて 行うことができる．ここでは例として，プログラムの提供者が，提供するプログラムに対し電子署名を行う場合を考える．この時，完全性の検証者は，電子署名をプログラムの提供者の公開鍵で復号した値を，入手したプログラムのハッシュ値と比較する．2 つの値が一致する場合，プログラムの完全性が保たれていることを確かめることができる．プログラムの提供者が生成した電子署名が，改竄されていないと十分に信頼できる場合は，ソフトウェアによって電子署名が生成されていても問題無い．しかし，IaaS において，悪意を持ったクラウドオペレータがいた場合，クラウドオペレータによって電子署名が改竄される可能性がある．そのため，ソフトウェアによって電子署名を発行することは危険であり，改竄が困難であるハードウェアを基点とした電子署名の発行を行う必要がある．

ハードウェアを基点としたプラットフォームの完全性を検証する手法に Trusted Platform Module(TPM)[2][3][4][5] を用いた Trusted Boot[5]があり，本研究では，この手法を用いてハイパーバイザの完全性の検証を行う．TPM は，暗号化演算や電子署名生成の機能，内部に保護された記憶領域などを持った，改竄が困難なセキュリティチップである． TPM を用いた Trusted Boot では，プラットフォームの起動時に，TPM がプラットフォーム全体のハッシュ値を計測し TPM 内部に保存する．そして，このハッシュ値に TPM の電子署名の発行する専用の鍵で電子署名を発行する．ハッシュ値は TPM 内部に保存され，ハッシュ値への電子署名は専用の鍵を用いて行われるため，任意の値に電子署名を生成することは出来ない．そして，ハッシュ値と電子署名を認証サーバに対して送信を行う．認証サーバは，プラットフォームの TPM の公開鍵を用いて電子署名の検証を行い，ハッシュ値が 改竄されていないことを確認する．そして，ハッシュ値と完全性の保たれたプラットフォームのハッシュ値を比較することで，プラットフォームの完全性を検証することが出来る．この時，認証サーバに渡されたプラットフォームの TPM の公開鍵が，本当に TPM の鍵かどうかを認証サーバに対して証明する必要がある．

Trusted Cloud Computing Platform (TCCP) [7]では，Trusted Boot を用いて， IaaS 利用者に対して，ハイパーバイザの完全性が保たれていること を保証するプロトコルの提案を行っている．TCCP では，最初に，IaaS プロバイダとは異なる，信頼できる第三者の認証サーバが，ハイパーバイザの完全性の検証を行う．そして，完全性の満たされているハイパーバイザ上でのみ，ユーザ VM の起動を可能にする．このプロトコルによって，IaaS 利用者は，ユーザ VM が起動した時点で，完全性の保たれたハイパーバイザ上で，ユーザ VM が実行されていることを確認することができる．しかし，TCCP では TPM の鍵の扱いが曖昧であり，認証サーバに対する鍵の証明も考慮されていない．

本研究では，TCCP を元にし，TPM の鍵の証明を加えたプロトコルの実装を行った．そして，実装したプログラムを，ハイパーバイザ端末，ユーザ端末，認証サーバ端末で実行した結果に対する評価を行った．今回，ハイパーバイザ端末のプログラムは，管理 VM 上のアプリケーションとして実装を行ったので，このプログラムが不正に改変されていても TPM の計測するハッシュ値には影響がない．そのため，完全性が満たされていない HV でもユーザ VM の起動が行われてしまう危険があり，十分な実装とは言えない．今後，必要最低限の機能をハイパーバ

† 東京大学, The University of Tokyo
‡ 国立情報学研究所, National Institution of Informatics

イザに組み込むことで，IaaS 利用者に対して，確実にハイパーバイザの完全性を保証することが出来る仕組みの考案と実装を行なう必要がある．

以下，2 章では，TPM の概要と TPM を用いた Trusted Boot，TPM の鍵の証明方法を述べる．3 章では，本研究の先行研究である TCCP のプロトコルについて述べる． 4 章では，TCCP を元に本研究で実装したプロトコルを示し，実行した結果に対する評価を行う．

2. TPM

2.1 TPM の概要

TPM[2][3][4][5]とは，Trusted Computing Group(TCG)が策定しているマザーボードに取り付けられるセキュリティチップである． TPM の主な機能として，RSA 暗号の演算，ハッシュ値の計測，電子署名の生成などがある．TPM は内部に記憶領域を持ち，TPM の鍵や TPM の鍵の電子証明書が蓄えられている．TPM 内部は保護されているため，内部を解析して機密情報の取得を行うことや，内部のデータの改竄を行うことは困難になっている．

TPM は，暗号化や電子署名に用いる鍵を生成する機能を持つ．これらの鍵は複数種類あり，それぞれの用途が異なる．以下に，今回用いる鍵の説明を記す．

(1) Endorsement Key (EK)

TPM 製造時に作られ，TPM 内部に保存される，それぞれの TPM に固有の鍵である．EK は，TPM によって生成された鍵が，本当に TPM によって生成された鍵であることを証明する時に用いられる．EK は，特定のデータ構造の復号のみ行う．

(2) Storage Root Key (SRK)

TPM を初期化した際に生成され，TPM 内部に保存される，TPM によって生成された鍵の暗号化を行う鍵である．

(3) Attestation Identity Key (AIK)

TPM によって生成される，TPM によって計測されたハッシュ値，TPM によって生成された鍵に電子署名を行う鍵である．秘密鍵は SRK によって暗号化され，外部に保存し，必要なときに TPM に読み込んで利用する．

(4) Binding Key

Binding Key は一般のデータの暗号化・復号を行うことのできる鍵である．秘密鍵は SRK によって暗号化され，外部に保存し，必要なときに TPM に読み込んで利用する．

(5) Signature Key

AIK は限られたデータに電子署名を行うに対し，Signature Key はあらゆるデータに電子署名を行うことができる．秘密鍵は SRK によって暗号化され，外部に保存し，必要なときに TPM に読み込んで利用する．

本研究では，これらの機能を用いて，プラットフォームの完全性を検証する Trusted Boot[5]を行う．

2.2 Trusted Boot

Trusted Boot では，プラットフォームの TPM が，プラットフォーム全体のハッシュ値を計測し，電子署名を行う．そして，認証サーバがハッシュ値と電子署名を用いて，プラットフォームの完全性の検証を行う．以下では，Trusted Boot の詳細について述べる．

Trusted Boot では，プラットフォームのシステムのブート時に，現在実行されているプログラムが TPM を用いて，次に起動するプログラムのハッシュ値を計測し，TPM 内部の Platform Configuration Register (PCR) に保存していく．この計測を BIOS 内にある書き換えの不可能な Core Root of Trust for Measurement (CRTM) と呼ばれるプログラムから始め，システムのブート時に実行されるプログラムの順番に沿って，BIOS，Boot Loader，ハイパーバイザという順に計測を行っていく．システムのブート途中に不正なプログラムが実行されることや，実行されるプログラムが不正に改変されることがあっても，それらのプログラムのハッシュ値は，プログラムが実行される前に PCR に書き込まれる．そのため，PCR の値は正しくプラットフォームが起動した時の値と異なるため，PCR の値を見ることで，不正を検出するこができる．

システムのブートが完了した後，PCR の値と PCR の値に対する電子署名を，プラットフォームとは異なる認証サーバに送信する．この時，認証サーバに対して送信される値は，現在の PCR の値でなければならない．2.1 で説明した TPM の鍵のうち，電子署名を行う鍵は，AIK と Signature Key である．Signature Key は，任意のデータの電子署名の生成にも用いることができるため，PCR の値が偽装される危険があり，AIK を用いる必要がある．しかし，ただ AIK を用いて PCR の値の電子署名を生成しただけでは，その電子署名が現在のプラットフォームの状態を表すことは保証できない．このことは，一度 AIK によって生成した電子署名を保存しておき，プラットフォームを再起動した場合に，その電子署名は現在の PCR の値から生成された電子署名では無い，という例を考えてれば明らかである．

そこで，TPM は，ただ単に AIK を用いて PCR の値に電子署名を行うのではなく，認証サーバが生成した乱数と現在の PCR の値の連結に対して電子署名を行うことで，電子署名が現在のプラットフォームの PCR の値を含んでいることを保証する．認証サーバは，プラットフォームから送られてきた電子署名が，送られてきた PCR の値と認証サーバが生成した乱数の連結に対するものであるか検証することで，PCR の値が現在のプラットフォームの状態を表しているかを確認することができる．

これらを踏まえた，実際に認証サーバがプラットフォームの完全性の検証を行う手順が，図 1 である．図 1 の説明を行う前に，本稿でプロトコルの説明で用いる表記について，表 1 にまとめ，説明を行う．表 1 において，as_pub, as_priv は，公開鍵暗号の鍵ペアを表す．電子署名の評価が二種類あるが，{#A}as_priv は，電子署名がメッセージの一部である場合に用いて，Sig_{AS} は，メッセージ全体に対する AS による電子署名を表すときに用いることとする．

図 1 において，認証サーバ(*AS*) は， プラットフォーム (*Platform*) から完全性の検証を開始するリクエストを受け取ると，乱数 (*nonce*) を生成し，*Platform* に送信する．*Platform* は，AIK を用いて *nonce* と現在の PCR の値 (*pcr_val*) の連結に対する電子署名を生成する．そして，*Platform* は *pcr_val* と電子署名を *AS* に対し送信する．*AS* は，あらかじめ *Platform* から受け取った AIK の公開鍵を用いて，電子書名が *pcr_val* と *AS* の生成した *nonce* の連結

に対するものであるかどうかを検証する．その後，*pcr_val* が正しいプラットフォームのハッシュ値と一致するかを比較し，完全性の検証を行う．

表 1 . 本稿の図で用いる表記

表記	意味
A|B	AとBの連結
{A}as_pub	Aのas_pubによる暗号化
#A	Aのハッシュ値
{#A}as_priv	Aに対する電子署名
SigAS	メッセージ全体に対するASの電子署名

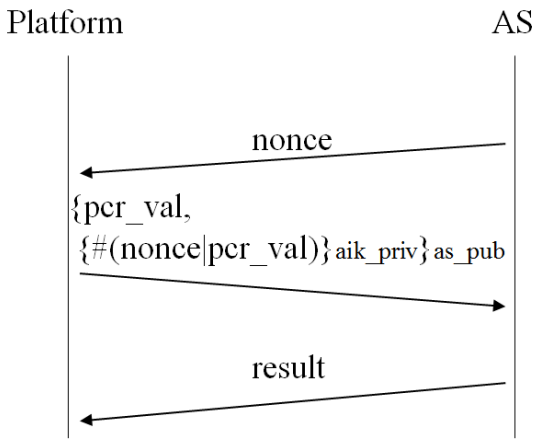

図 1. Trusted Boot

2.3 TPM の鍵の証明

Trusted Boot では，認証サーバは，検証対象のプラットフォームの AIK の公開鍵を用いて，電子署名の検証を行い，プラットフォームから送られてきた PCR の値が現在のプラットフォームの状態を表す値であることを確かめる．この時，認証サーバに対して送信した公開鍵が，TPM によって生成された AIK の公開鍵であることを証明しなければならない．本節では，TPM によって生成される鍵が，1 つの TPM によって生成されたことを照明する方法について述べる．

(1) EK の証明

EK の証明は，TPM に保存されている EK の電子証明書を用いて行われる．EK の電子証明書は，TPM の製造者の，EK の電子証明書を発行する専用の認証局(CA_{EK} とする)によって発行される．認証サーバは，プラットフォームから受け取った電子証明書が，CA_{EK} によって発行されているか検証することで，その電子証明書が EK のものであるかどうかが分かる．

仮に，TPM が EK を用いて，TPM によって生成された鍵に対する電子証明書を発行する機能があれば，他の鍵についても電子証明書によって証明を行うことができる．しかし，実際には TPM にそのような機能はなく，別の方法で鍵の証明を行う必要がある．

(2) AIK の証明

AIK の証明方法は，Privacy-Preserving Protocol[6]と呼ばれる提案されているプロトコルによって行われる．2.1 で，EK は特定のデータ構造の復号のみを行うと述べたが，そのデータ構造とは以下のものを含んでいる．

・AIK の公開鍵のハッシュ値 (id_Digest)
・乱数 (random)
・PCR の値 (pcr_value)

AIK の証明は，このデータ構造の暗号化・復号の仕組みを用いて行われる．このデータが EK の公開鍵によって暗号化されている時，EK による復号は次のように行われる．まず，TPM の中に AIK をロードする．TPM はデータの復号を行い，その中に含まれる id_Digest が，ロードされた AIK の公開鍵のハッシュ値と一致し，かつ，現在の PCR の値が pcr_value と一致する場合にのみ，random を返す．PCR の値は必須ではないため，今回は指定していない．

これらを用いた Privacy-Preserving Protocol を図 2 に示す．プラットフォーム(*Platform*)は，EK の電子証明書(*ek_cert*)と AIK の公開鍵 (*aik_pub*) を認証サーバ(*AS*)に対し送信する．*AS* は，(1) の方法により電子証明書が EK のものであるかどうかを検証し，電子証明書から EK の公開鍵(*ek_pub*)を取得する．そして，*AS* は *aik_pub* のハッシュ値を計算し，乱数(*random*)を生成する．そして，これらを *ek_pub* で暗号化して *Platform* に送る．*Platform* は，*AS* から受け取ったデータを復号し，*random* の取得を試みる．この時、実際に *random* が取得できるのは，*AS* に送信した *aik_pub* が，*ek_cert* の保存されている TPM によって生成された AIK の公開鍵であるときのみである．このとき *Platform* が取得した値(*random'*) を *AS* に対して送信する．*AS* は，*Platform* から受け取った *random'* が *AS* の生成した *random* と一致するか比較することで *aik_pub* が，*ek_cert* のある TPM によって生成されたかどうかを検証することができる．

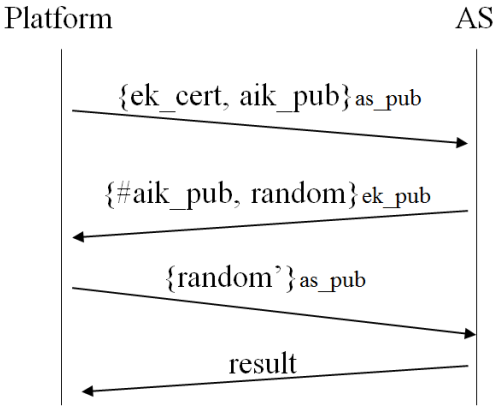

図 2. Privacy-Preserving Protocol

(3) Binding Key, Signature Key の証明

AIK は，AIK を生成した TPM が生成した鍵に対して，電子署名の発行を行う．そのため，Binding Key，Signature Key の証明は，それぞれの鍵に対する電子署名を AIK の公開鍵によって検証することで可能となる．

(1)，(2)，(3)の証明方法によりそれぞれの鍵が 1 つの TPM に存在し，生成されたことが証明される．

3. TCCP

本研究で実装を行ったプロトコルは，Trusted Cloud Computing Platform (TCCP) [7]を元にしている．TCCP では，IaaS プロバイダとは異なる，信頼できる第三者の認証サーバが，IaaS プロバイダのハイパーバイザの完全性の検証を行い，完全性の満たされたハイパーバイザ上でのみ，IaaS 利用者のユーザ VM の起動を可能にするプロトコルの提案を行っている．TCCP では，脅威モデルとして，IaaS のオペレータに悪意があり，管理 VM 上からユーザ VM の情報の窃取や，データの改竄を行うことを想定している．また，認証サーバの公開鍵は安全に入手が可能であり，ハイパーバイザの端末では，プロトコルのサポートをハイパーバイザ自体が行うことで，管理 VM 上からメモリ内容の秘匿を行う，と仮定している．

そのプロトコルは次の二つに分かれている．

・Node management
・Virtual machine management

Node management は，認証サーバが，IaaS プロバイダのハイパーバイザの完全性の検証を行うプロトコルである．ここでは最初に，認証サーバが信頼できるものであるか確かめるために，2.2 に示した Trusted Boot により認証サーバの完全性を，ハイパーバイザが検証している．次に，認証サーバがハイパーバイザの完全性の検証を行う．この時，ハイパーバイザは同時に RSA 暗号の鍵ペアの生成し，認証サーバに公開鍵を送信する．認証サーバは，ハイパーバイザの完全性が保たれている場合，公開鍵の登録を行い，以後のユーザ VM の起動に必要な情報は，登録した公開鍵によって暗号化される．ハイパーバイザの生成した鍵ペアは，ハイパーバイザが再起動した時に失われるように，メモリにのみ保持する．ハイパーバイザは再起動された時，鍵ペアは失われるため，再度 Node management によってハイパーバイザの完全性の検証を認証サーバに依頼する必要がある．

Virtual machine management は，Node management によって認証サーバによって完全性が保たれていると判断されたハイパーバイザ上でのみ，IaaS 利用者のユーザ VM の起動を可能にするプロトコルである．IaaS 利用者は，ユーザ VM のディスクイメージを暗号化するための，共通鍵を生成する．そして，ディスクイメージを暗号化し，共通鍵を認証サーバの公開鍵で暗号化し，ハイパーバイザに対して送信する．ハイパーバイザはそのままでは，ユーザ VM の起動を行うことができないため，認証サーバに対し暗号化された共通鍵を送信する．認証サーバは，ハイパーバイザが完全性の満たされたものであれば，共通鍵を復号し，Node management で登録された，ハイパーバイザの生成した公開鍵で暗号化し，ハイパーバイザに送る．ハイパーバイザは，完全性が満たされている場合にのみ共通鍵を復号し，取得することができる．そして，ユーザ VM のディスクイメージを復号し，起動を行う．IaaS の利用者は，ユーザ VM に接続することが出来た場合，ユーザ VM が完全性の保たれたハイパーバイザ上で実行されていると判断できる．

TCCP では，ユーザ VM を起動したまま別のハイパーバイザ上に移すマイグレーションを安全に行うプロトコルの提案も行っているが，本論文ではマイグレーションについて考慮していないため省略する．

4. 実装・評価・検証

4.1 実装プロトコル

本研究では，TCCP を元にしたプロトコルの実装を行った．そのため，実装プロトコルを TCCP 同様に，Node management と Virtual machine management に分けて述べる．本研究で実装を行ったプロトコルと，TCCP のプロトコルとの差異は以下のとおりである．

TCCP では，2.3 で示した TPM の鍵の証明が考慮されていない．TPM の鍵の証明は，ハイパーバイザと認証サーバが通信を始める際に最初に行われるべきである．そのため本研究では，Node management の最初に，認証サーバに対して，ハイパーバイザが TPM の鍵の証明を行う手続きを加えた．また，TCCP では，認証サーバの成りすましを防ぐために，ハイパーバイザが認証サーバの完全性を検証している．しかし，この手続きを行なわなくても，誤ってユーザ VM が起動される危険はない．なぜなら，IaaS の利用者が認証サーバの公開鍵を安全に入手できるという TCCP における仮定から，仮に認証サーバが成りすまされていても，Virtual machine management において，成りすました認証サーバは，ユーザ VM のディスクイメージを暗号化している共通鍵の復号を行うことができないからである．そのため，本研究の Node management においては，ハイパーバイザによる認証サーバの完全性の検証は行わない．Virtual machine management では，TCCP で提案されていたプロトコルと差異はない．以下に，本研究で実装したプロトコルの説明を行う．

・Node management (図 3)

図 3 では，既にハイパーバイザ(*HV*)が起動し，プラットフォーム全体のハッシュ値が PCR に保存されているものとする．Node management では，最初に *HV* が認証サーバ(*AS*)に対して TPM の鍵の証明を行う(①～④)．そして，その後，*AS* が *HV* の完全性をが保たれているかどうかを検証する(④～⑥)．

① *HV* は，AIK，Signature Key，Binding Key の生成を行う．そして，EK の電子証明書 (*ek_cert*) ，AIK の公開鍵 (*aik_pub*) ， Signature Key の公開鍵 (*sign_pub*) ，Binding Key の公開鍵 (*bind_pub*) を *AS* の公開鍵 (*as_pub*) で暗号化し，*AS* に送信する．

② *AS* は，2.3 に示した TPM の鍵の証明方法によって，*ek_cert* が EK の電子証明書であるかどうかを検証する．*ek_cert* が EK の電子証明書であった場合，乱数 (*nonce_aik*)と *aik_pub* のハッシュ値を生成し，*ek_cert* から取得した EK の公開鍵(*ek_pub*)で暗号化し *HV* に送信する．

③ *HV* は，*ek_pub* で暗号化されたデータを復号し，*nonce_aik* の取得を試みる．このとき，*nonce_aik* の取得が行えるのは，2.3 で示したように *ek_cert* のある TPM によって *aik_pub* が生成された場合のみである．このとき，*HV* が，*ek_pub* で暗号化されたデータの復号によって得た乱数を *nonce_aik'*とする．*HV* は，*sign_pub*，*bind_pub* に対する電子署名を *HV* の AIK を用いて生成し，*nonce_aik'*と共に *as_pub* で暗号化し

AS に対して送信する．

④ *AS* は *nonce_aik'* が *AS* の生成した *nonce_aik* と一致するか比較することで，*aik_pub* が，*ek_cert* のある TPM と同じ TPM によって生成されたかどうかを検証することができる．そして，*bind_pub*，*sign_pub* に対する電子署名を，*aik_pub* を用いて検証することで，*HV* から受け取った鍵がすべてひとつの TPM によるものであるかどうかを確かめることができる．*AS* は，次に，*HV* の完全性を検証するために，2.2 に示した手順に則り，乱数 (*nonce_quote*) を生成し，*HV* に対して送信する．

⑤ *HV* は，*HV* の完全性が保たれている場合に，以降の通信の暗号化に用いる共通鍵(*ot_key*)を生成する．そして，TPM の機能を用いて *nonce_quote* と PCR の値(hash)の連結に対する電子署名を生成し，*ot_key* と，*hash* と共に *AS* に送信する．

⑥ *AS* は，*aik_pub* を用いて，*HV* から受け取った電子署名の検証を行う．そして，*AS* は，*hash* が期待されたハイパーバイザの状態を表す値と一致していた場合，*HV* の完全性が保たれていることが分かる．この場合，*AS* は *ot_key* の登録を行い，*HV* に対して完全性検証成功の返答をする．

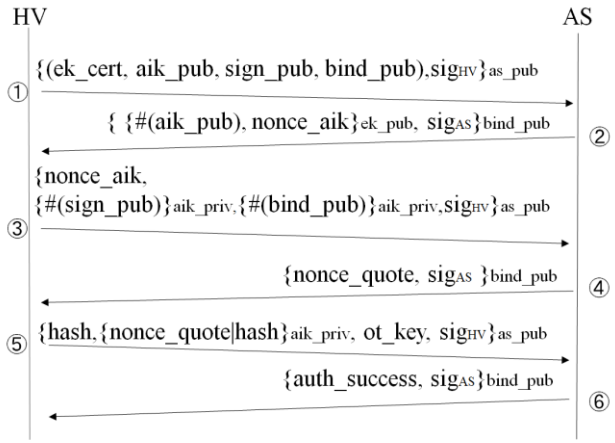

図 3. Node management

・Virtual machine management (図 4)

Virtual machine management においては，TCCP で提案されていたプロトコルと差異はない．Virtual machine management では，IaaS の利用者が生成した鍵によって，ユーザ VM の暗号化を行う．そして，完全性の保たれたハイパーバイザのみがユーザ VM の復号鍵を取得し，ユーザ VM の起動を行なうことができる．

① IaaS の利用者(*User*) はユーザ VM のディスクイメージ (*vm.img*) を暗号化するための共通鍵 (*vmkey*) と乱数 (*random*) を生成する．そして，*vm.img* と *#vm.img* を *vm_key* で暗号化し，*vm_key* と *random* を認証サーバ (*AS*)の公開鍵(*as_pubkey*)で暗号化して，IaaS プロバイダのハイパーバイザ(*HV*) に送信する．

② *HV* は，*vm.img* を復号するための *vm_key* が *as_pub* によって暗号化されているので，ユーザ VM の起動を行うことができない．そのため，*HV* は *AS* に対して暗号化された *vm_key* を送信する．

③ *AS* は，*HV* が完全性の満たされたハイパーバイザであれば，*vm_key* の復号を行い，Node management で登録した *ot_key* で暗号化し *HV* に送信する．

④ *HV* は，*ot_key* を有している場合，*vm_key* を復号することができる．そして，*vm_key* で *vm.img* の復号を行いユーザ VM の起動を行う．この時，*vm.img* が改竄されていないことを*#vm.img* を用いて検証する，*HV* は *User* に対し，ユーザ VM の接続に必要な情報 (*connect_info*)と *random* を送信する．

⑤ *User* は，*random* と最初に生成した値を比較し，*vmkey* が *HV* によって復号されたかどうかを検証する．*HV* から受け取った *conncet_info* を元にユーザ VM に接続を行う．ユーザ VM に接続が可能であることは，ユーザ VM を実行しているハイパーバイザの完全性が満たされていることを意味する．

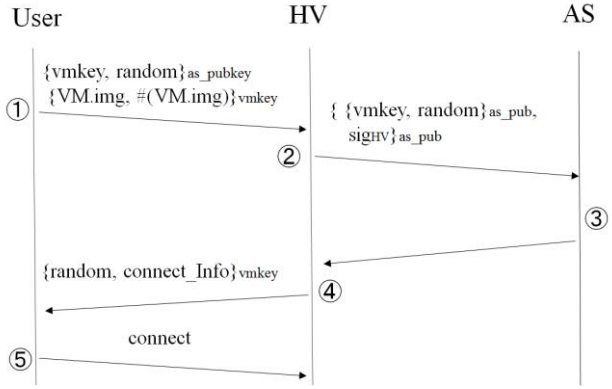

図 4. Virtual machine management

4.2 実行環境

今回実装したプロトコルを，ユーザ VM の起動リクエストを送信する IaaS 利用者の端末，ハイパーバイザ端末，認証サーバ端末で実行した．それぞれのプログラムの実行環境を表 2 に示す．

TCCP では，ハイパーバイザ端末での処理は，ハイパーバイザに機能を追加することで実現すると仮定していたが，今回は，管理 VM 上で実行されるプログラムの実装を行った．Trusted Boot では，ブートローダが，ハイパーバイザのプログラムなどのハッシュ値を計測する機能を備える必要がある．このブートローダは，オープンソースで公開されている TrustedGRUB-1.1.5[8]利用した．また，Linux 上で TPM を扱うライブラリとして TrouSerS-0.3.11.2[9]を利用した．共通鍵暗号を行うライブラリとして，OpenSSL 1.0.1 を用いた．認証サーバが完全性の保たれたハイパーバイザのハッシュ値を管理するデータベースには，SQLite 3.7.9 を用いた．

4.3 評価

・実行時間の評価

プロトコル別の実行時間を図 5 に，また Node management，Virtual machine management の実行時間の内訳をそれぞれ図 6，図 7 に示す．Node management は，3 分ほどかかっておりその時間の大半が RSA 復号(RSA Decryption)に費やされている．実際に復号したデータは 10KByte ほどである．RSA 復号は TPM の鍵を用いて行われている為，TPM 内部のメモリ容量がごく少量に限られていることや，TPM とマザーボードを接続する LPC バスの伝送速度の問題によって，OpenSSL など通常の RSA 復号よりも時間が

表 2. プログラムの実行環境

	IaaS利用者	ハイパーバイザ	認証サーバ
PC	Let's note CF-W8	Let's note CF-W8	Let's note CF-W5
CPU 周波数	Core 2Duo 1.60GHz	Core 2Duo 1.60GHz	Core Duo 1.06GHz
メモリ	4GB	2GB	1GB
OS	Ubuntu 12.04 LTS	Ubuntu 12.04 LTS (管理VMのOS)	Ubuntu 12.04 LTS
ハイパーバイザ		Xen 4.0.1	
TPM	Infineon TPM 1.2		
TrouSerS	TrouSerS-0.3.11.2		
OpenSSL	OpenSSL 1.0.1		
Trusted GRUB		TrustedGRUB-1.1.5	
SQLite			SQLite 3.7.9

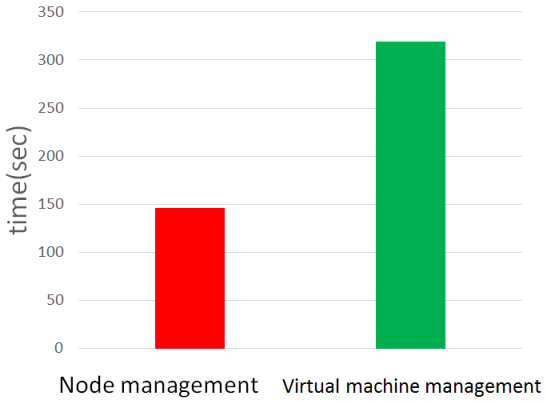

図 5. プロトコル別の実行時間

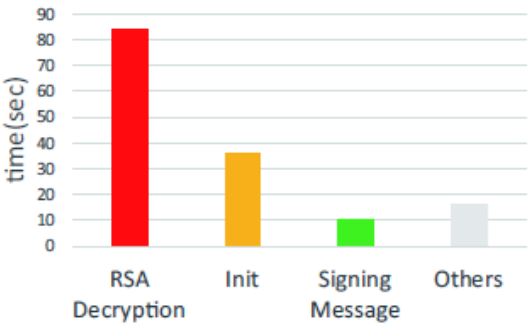

図 6. Node management の実行時間内訳

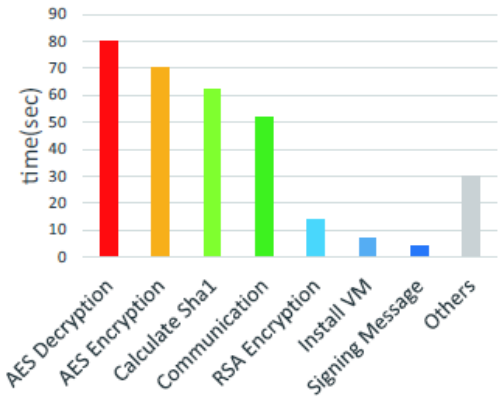

図 7. Virtual machine management の実行時間内訳

多くかかってしまっていると考えられる．しかし，Node management はハイパーバイザの起動時に一度行えば良いため，頻繁に再起動を行わない IaaS の環境では，運用に支障をきたす要因にはならないと考えられる．

Virtual machine management は，全体で 5 分程度かかっている．この時間の大半を，2GByte ほどあるユーザ VM の暗号化(AES Encryption)，復号(AES Decryption)，ハッシュ値計算(Calculate Hash)，ユーザ端末からハイパーバイザ端末への転送(Communication)が占めている(図 7)．AES による暗号化復号，ハッシュ値の計算は OpenSSL を用いており，今回実験に用いた端末の性能を考えると特別異常な値とは言えない．

・安全性の評価

今回，ハイパーバイザ端末で行なわれる処理は，HV の管理 VM 上のプログラムが行なった．そのため，実行したプログラムまで含めた完全性の検証を行うことができない．その結果，不正に改変されたハイパーバイザが，認証サーバによって完全性が満たされているとされる危険がある．また，Virtual machine management においては，ユーザ VM の復号に必要な vmkey を，プログラムの使用するメモリを覗くことで取得される危険がある．

このような危険は，今回のプロトコルを管理 VM 上のアプリケーションではなく，ハイパーバイザ自体がサポートすることで解決が可能であると考えられる．しかし，その実現には，TPM，ネットワークのドライバをハイパーバイザに組み込む必要があり，非常に膨大なプログラムになり，現実的な解決方法とは言えない．そのため，今後，必要最低限の機能をハイパーバイザに組み込むことで，IaaS 利用者に対して，確実にハイパーバイザの完全性を保証することが出来る仕組みを考える必要がある．

5. まとめ

IaaS においては，クラウド利用者に対して，ユーザ VM を動かすハイパーバイザの完全性が保たれていることが，保証されていなければならない．本研究では，このことの必要性を確認した後，先行研究である TCCP で提案されているプロトコルに，TPM の鍵の証明を加えたプロトコルの実装を行った．今回の実装では，ハイパーバイザ端末で実行されるプログラムを管理 VM 上のアプリケーションとして実行していた．そのため，完全性が満たされていない HV でもユーザ VM の起動が行われてしまう危険があり，十分な実装とはならなかった．今後，必要最低限の機能をハイパーバイザに対して組み込むことで，今回生じた問題を解決が出来る仕組みの考案と実装を行う必要がある．

謝辞

本論文の研究の一部は，公益財団法人セコム科学技術振興財団の助成を受けたものである。

参考文献

[1] 田所 秀和，光来 健一，千葉 滋，“IaaS 環境における VM のメモリ暗号化による情報漏洩の防止”，情報処理学会研究報告. 計算機アーキテクチャ研究会報告，一般社団法人情報処理学会，2011

[2] Trusted Computing Group, "TCG Architecture Overview, Version 1.4", https://www.trustedcomputinggroup.org/files/resource_files/AC652DE1-1D09-3519-ADA026A0C05CFAC2/TCG_1_4_Architecture_Overview.pdf, 2007.
[3] Trusted Computing Group,“ Trusted Platform Module (TPM) Summary", https://www.trustedcomputinggroup.org/files/resource_files/4B55C6B9-1D09-3519-AD916F3031BCB586/Trusted%20Platform%20Module%20Summary_04292008.pdf, 2011.
[4] Trusted Computing Group, "Trusted Platform Module Library Part 1: Architecture", http://www.trustedcomputinggroup.org/files/resource_files/1556DBFF-1A4B-B294-D085B90B831DA92E/TPM%20Rev%202.0%20Part%201%20-%20Architecture%2000.99.pdf, 2013.
[5] David Challenger, Kent Yoder, Ryan Catherman, Leendert Van Doorn, "A Practical Guide to Trusted Computing", IBM PRESS, 2009.
[6] Trusted Computing Group, "TCG Infrastructure Working Group A CMC Profile for AIK Certificate Enrollment", https://www.trustedcomputinggroup.org/files/resource_files/738DF0BB-1A4B-B294-D0AF6AF9CC023163/IWG_CMC_Profile_Cert_Enrollment_v1_r7.pdf, p33-38, 2011.
[7] Nuno Santos, Krishna P. Gummadi, Rodrigo Rodrigues, "Towards Trusted Cloud Computing", Proceedings of the USENIX HotCloud ‘09, 2009.
[8] Sirrix AG security technologies, “TrustedGRUB”, http://www.sirrix.com/content/pages/trustedgrub_en.htm
[9] IBM, “TrouSerS”, http://trousers.sourceforge.net/