修士論文

クラウドフォレンジックに向けた VMセキュアプロセッサの設計と実装

Design and Implementation of VM Secure Processor for Cloud Forensics

平成 28 年 02 月 04 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

48-146462 千田　拓矢

概要

現代ではコンピュータ技術の発展にともない未知の脆弱性や高度なウイルスの報告件数が増加しており，コンピュータ・インシデントを未然に防ぐことは困難を極めている．そこで，コンピュータ・インシデント発生後の迅速な復旧と再発防止のためにデジタルフォレンジックへの関心が高まりつつある．デジタルフォレンジックとはコンピュータをはじめとする電子機器に残された記録を収集・分析し，法的な証拠性を保証する科学的調査手法の総称である．デジタルフォレンジックのなかでも，クラウド環境での調査を想定したものを「クラウドフォレンジック」と呼ぶ．クラウド環境では大量の電子機器が地理的に分散して稼動する．そのため，コンピュータ・インシデントが発生した際は各電子機器に残されたログファイルや電子データをそれぞれ精査しなければならない．電子機器の所有者が同一でないことも多く，複雑な調査権限の手続きが必要な場合もある．このような問題がクラウドフォレンジックのボトルネックとなり，調査のために膨大な時間を要するのが現状である．

本論文ではクラウドフォレンジックのためにログファイルの認証機能を追加した「VM セキュアプロセッサ」の設計と実装について述べる．VM セキュアプロセッサは我々の研究グループが開発している次世代プロセッサであり，クラウド環境における機密情報の保護を目的とした新しいマイクロアーキテクチャである．現状クラウド環境においてログファイル認証はソフトウェアによって行うのが一般的であるが，提案手法ではハードウェアである VM セキュアプロセッサによって行う．これによって従来よりも高速かつ安全なログファイル認証を実現し，クラウドフォレンジックの効率化を目指す．本研究ではログファイル認証に必要なハッシュ値生成モジュールと電子署名生成モジュールを追加実装し，FPGA 上でハッシュ値が出力され，シミュレーション上で正しい電子署名が得られることを確認した．

目 次

図 目 次

表 目 次

第1章　序論

1.1　本研究の背景と目的

これまでコンピュータセキュリティの世界ではコンピュータ・インシデント（以下インシデントと表記）を未然に防ぐことに主眼を置き，さまざまな手法や製品の研究・開発が進められてきた．だが，コンピュータ技術が発展するにつれて未知の脆弱性や高度なウイルスの報告件数は増加の一途をたどり，インシデントを未然に防ぐ「事前対応」の考え方は限界を迎えている．そこで近年では，インシデント発生後にその原因や被害状況を分析し，迅速なシステム復旧と再発防止を目的とした「事後対応」の考え方（インシデントレスポンス）が重視されつつある．国内では 1996 年に JPCERT/CC [1] と呼ばれるインシデントレスポンスの支援を目的とした組織（CSIRT）が結成され，企業や官公庁の内部に CSIRT の設立を促すようになった．2007 年には CSIRT の連合体として日本シーサート協議会が発足した．国際的な連合体としては FIRST が挙げられる．

インシデントレスポンスで重要な役割を果たすのが「デジタルフォレンジック」と呼ばれる技術である [1]．デジタルフォレンジックはサイバー犯罪をはじめとするインシデントが発生した際にコンピュータに残されたログなどの記録を収集し，法的な証拠として有効であることを科学的に保証するための技術である．例えば，高度なマルウェアやクラッカーは攻撃の痕跡を抹消するためにログファイルや関連データの改ざんや隠蔽を行う．そのようなサイバー犯罪の調査において，デジタルフォレンジックはログファイルの改ざん検出や消去されたデータの復元に関して大きく貢献している．また，デジタルフォレンジックの技術は犯罪調査だけではなく，データの復旧や企業での内部不正の調査などにも利用されており，今日の IT 社会では欠かせないものとなっている．今後のモバイル端末やウェアラブル端末などの更なる普及を見据えると，それらを利用した犯罪や不正も増加していくと考えられるため，デジタルフォレンジックの重要性はより一層高まっている．実際，警察庁や FBI の発表した資料 [2] [3] によるとサイバー犯罪の発生件数は年々増加しており，事実を正確に解明するためのデジタルフォレンジック技術の向上は急務となっている．例えば，2012 年に起きた PC 遠隔操作事件ではウイルスに感染した PC の所有者が誤認逮捕され，真犯人を特定するための証拠収集に時間がかかるなど，サイバー犯罪の捜査における問題点が露呈した．このように，誤認逮捕の防止や迅速な犯人特定のためには，PC に残されたログをいかに正確に効率よく収集するかが重要である．

デジタルフォレンジックのなかでも，クラウドコンピューティングでのインシデントレスポンスや調査・分析に焦点を当てたものを特に「クラウドフォレンジック」と呼ぶ．クラウドコンピューティングはスマートフォンやサーバ仮想化の普及に伴い，急激に利用者数を増やしている．クラウド環境をユーザに提供する企業をクラウドプロバイダと呼び，クラウドサービスはプロバイダが提供するものによって SaaS，PaaS，IaaS [4] のように分類される．特に IaaS ではユーザへ VM 全体を貸し出

[1] https://www.jpcert.or.jp/

すため，自由度が高く便利な反面，悪用もされやすいことが指摘されている [5]．VM を悪用するだけでなく，VM の持つ機密情報を狙った攻撃 [6] も存在する．

このように，クラウドサービスでは高いインシデント発生リスクのためにクラウドフォレンジックの需要は高まっているものの，その課題は多い．クラウド環境では多種多様で膨大なコンピュータ機器が地理的に分散して稼動しており，それぞれが複雑に関わりあって構成されている．したがって，証拠となりうるログファイルやデータも分散して存在し，膨大な量となる．このようなログファイルからインシデントに関わるものを選定・検証することには相当の労力を要する．以上のように，膨大なログファイルが分散して存在することがクラウドフォレンジックのボトルネックとなっている．

本研究の目的はクラウド環境におけるログファイルの管理を効率化することであり，そのために我々の研究グループが開発している「VM セキュアプロセッサ」にログファイルの認証機構を追加実装する．VM セキュアプロセッサは VM の機密情報を保護することを目的として開発中のプロセッサであり，VM を扱う IaaS での利用を想定している．現在クラウド環境でのログファイル管理はソフトウェアによるものが主流であるのに対し，ハードウェアによるログファイル管理は稀である．ハードウェアはソフトウェアと比べて製造後に改変が困難なため，生成されるログファイルの改ざんや漏洩のリスクに対して強固であるという利点がある．さらに，TCP/IP や AES 暗号のようにソフトウェアの代わりに専用ハードウェアを用意することで処理速度を向上させた例は数多い．今後クラウドサービスがさらに普及するにつれてログファイルの量も爆発的に増加することが見込まれるため，ハードウェアによるログファイル管理には大きなメリットがあると言えるだろう．本研究ではより高速かつ安全なハードウェアによって処理することでクラウドフォレンジックのボトルネックを解消することを目指す．

1.2 本論文の構成

本論文は本章を含めて～章から構成される．

第２章では，デジタルフォレンジックの基礎知識をまとめ，クラウドフォレンジックの概要と課題について述べる．これにより，本研究の意義と立場を明らかにする．

第３章では，本研究に関連するハードウェアセキュリティの知識をまとめ，第４章ではクラウドフォレンジックに関連する．

第４章では，クラウドフォレンジックに関連する既存手法や技術について述べる．　第５章では，本研究の提案手法で用いる VM セキュアプロセッサについて，具体的にどのような構造なのか，またどのように動作するのかについて述べる．さらに，クラウドフォレンジックの課題に対して VM セキュアプロセッサを用いたログファイル管理システムを提案する．

第６章では，提案手法の実現のために追加実装したモジュールやプログラムの設計と実装についてその詳細を解説する．

第７章では，提案手法がログファイルを効率よく安全に管理出来るかどうか示すために，実装した VM セキュアプロセッサの性能評価と提案手法のセキュリティ評価を行う．

最後に第８章で本研究で提案するログファイル管理システムについて総括する．

第2章　デジタルフォレンジック

本章ではまずデジタルフォレンジックの基本事項をまとめ，クラウドフォレンジックの概要と課題について述べる．最後にそれらを踏まえて，本研究が課題とする脅威モデルを設定し，提案手法のセキュリティ評価への準備とする．

2.1　デジタルフォレンジックの基礎知識

デジタルフォレンジックの定義　デジタルフォレンジックはIT機器を利用した犯罪や不正を捜査する場合において，それに関連する電磁的記録の証拠性を科学的に証明する技術である．しかし，デジタルフォレンジックで用いられる技術はデータ復旧や犯罪以外の捜査など幅広く応用されるようになったため，定義が拡大しつつある．そこで本論文では以下のデジタル・フォレンジック研究会[1]によるデジタルフォレンジックの定義に従う．

> インシデントレスポンス（コンピュータやネットワーク等の資源及び環境の不正使用，サービス妨害行為， データの破壊，意図しない情報の開示等，並びにそれらへ至るための行為（事象）等への対応等を言う.）や法的紛争・訴訟に際し，電磁的記録の証拠保全及び調査・分析を行うとともに，電磁的記録の改ざん・毀損等についての分析・情報収集等を行う一連の科学的調査手法・技術を言います.

デジタルフォレンジックの対象　デジタルフォレンジックの目的はインシデントに関連する電磁的記録の証拠保全及び調査・分析であるため，その対象は電磁的記録媒体かもしくはそのような記録媒体を内蔵するIT機器である．具体的には以下のようなものが挙げられる．

- 記録媒体
  - 磁気ディスク（HDDなど）
  - 光学ディスク
  - フラッシュメモリ
- 記録媒体を含む機器
  - デスクトップ端末
  - モバイル端末（スマートフォン，タブレットなど）

[1] https://digitalforensic.jp/

- 汎用サーバ
- ネットワーク機器（ルータ，スイッチなど）
- その他電子機器（ゲーム機，プリンタ，コピー機，カーナビゲーションシステム，POS 端末など）

今後，IoT(Internet of Things) が普及するに連れて，生活家電や自動車がコンピュータと同等の機能を持つ場合や全く新しい情報機器が開発される場合が想定される．そのような場合でも，インシデントに関連する記録媒体を内蔵する場合は全てデジタルフォレンジックの対象となりうる．

また，デジタルフォレンジックの対象によって表 2.1 のように分類が可能である．

表 2.1: 独立行政法人情報処理推進機構『情報セキュリティ白書 2011』による

種類	対象
ディスクフォレンジック	ハードディスクを中心とした不揮発な記憶媒体
ネットワークフォレンジック	ネットワーク機器，サーバのログなど
電子メールフォレンジック	電子メールを中心とするデータ
Web フォレンジック	Web サイトに関連するデータ
モバイルフォレンジック	携帯電話，携帯端末，スマートフォン等のデータ

2.1.1 デジタルフォレンジックのプロセス

デジタルフォレンジックの基本的なプロセスは以下のとおりである [1]．それぞれの段階について概要と留意点をまとめる．

対象特定　デジタルフォレンジックによる調査では，まず第一に対象となる IT 機器を特定する必要がある．インシデントに直接的に関連のある IT 機器の特定は容易である．しかし，近年ではネットワーク技術の発展が顕著であり，ほとんど全ての IT 機器がインターネットに接続されるようになった．それに伴って，インシデントに間接的に関連のある IT 機器はますます増加しており，それらを隈なく捜索する必要があることに注意しなければならない．

証拠保全　インシデントに関連する IT 機器や記録媒体から，それらが保持している全ての電磁的記録を抽出し，解析用の別の記録媒体に保存する．このとき，対象となる IT 機器をインシデント発生後のそのままの状態に保つことが非常に重要となる．具体例としてインシデント被害にあった PC 内蔵のハードディスクの場合について述べる．ウイルス感染や不正アクセスなどのインシデントの被害が発覚したら，シャットダウンやファイルの削除をしてはならない．これによってインシデントに関する情報が失われてしまう危険性があるためである．その他の IT 機器の場合も同様である．同様に，対象となる IT 機器を直接解析するとインシデントに関連する重要な情報が損なわれる可能性が

あるため，解析用の記録媒体を用意することも必須である．実際の証拠保全作業では図 2.1[2] のような専門ハードウェアが用いられる．

解析　解析用の記録媒体から時系列やアクセス履歴などを利用してインシデントに関連する情報を精査する．この際，隠蔽された情報や消去された情報も復元ソフトなどを利用して網羅的に解析しなければならない．解析で得られる情報は対象となる IT 機器やインシデントの状況によって異なる．具体的には以下のようなものが挙げられる．

- レジストリ情報
- ログデータ
- 電子メール
- 文書ファイル
- 画像データ

解析して得られた情報は改ざんや偽装されている場合があるため，電子署名やハッシュ値を検証することで情報の同一性を証明しなければならない．

結果報告　解析で得られた情報の同一性証明や改ざん・毀損の有無をまとめて，法的に証拠として有効か否かを報告書にまとめる．報告書は法的紛争や訴訟の際に提出される．しかし，報告書には企業機密やプライバシーの侵害に繋がる情報が含まれる場合もあるため，裁判所などの法的機関が不当な開示請求に対する保護命令や閲覧者の制限といった措置が必要とある．

2.1.2　ログファイルの分類

デジタルフォレンジックで解析対象となるログファイルにはさまざまなものがある．ログファイルはアプリケーションや OS などさまざまな場所で生成されるため体系的な分類は難しいが，ここでは主にクラウド環境で生成されるログファイルについて具体例を述べる．

- アプリケーション固有のログ
  - アンチウイルスソフトやマルウェア対策ソフト
  - Apach や IIS などのサーバソフトウェア
- OS に関するログ
  - Windows のイベントログ
  - Unix 系 OS の syslog など
- ネットワーク機器やセキュリティ機器のログ

[2] http://www.ubic.co.jp/products/enterprise.html より引用

図 2.1: Image MASSter Solo4 Forensic Enterprise

- ファイアウォール
- IDS/IPS
- ルータやプロキシサーバ
- 認証サーバ

2.1.3 ログファイルの検証

2.1.1 で述べたようにデジタルフォレンジックの解析では，証拠保全から結果報告までログファイルの情報の同一性を証明することが必要不可欠である．ログファイルの同一性証明にはハッシュ値が利用される．保全されたログファイルの原本と解析結果であるログファイルのハッシュ値が一致していれば，改ざんが無いこと（ハッシュ値の完全性）を証明することが出来る．

また，ログファイルの否認も防止しなければならない．例えばログファイルの内容が不都合なものであったり脅迫を受けていたりする場合，ログファイルの生成者がログファイルの内容を「偽造である」と否定する事態が想定される．このような否認が出来ないような状態を否認不能性と呼び，ログファイルの生成者のなりすましが無いこと（電子署名の真正性）を検証することで確保される．ハッシュ関数と電子署名の概要を以下に示す．

ハッシュ関数　ハッシュ関数はある任意の長さのメッセージを入力として固定長の値（ハッシュ値）を出力する関数である．出力値はメッセージによって異なるため，メッセージの完全性を検証した

い場合はハッシュ値を照合すればよい．このようにハッシュ値によって完全性を検証することでメッセージの「改ざん」を検出することが出来る．代表的なハッシュ関数として，SHA-1SHA-2，MD5などが挙げられる．SHA-1 と MD5 に関しては攻撃手法が報告されているため，現在では SHA-2 の利用が推奨されている．デジタルフォレンジックにおいてはログファイルの同一性証明に利用されている．

電子署名　電子署名は公開鍵暗号を応用したものであり，メッセージの作成者が誰であるか（真正性）を証明するための技術である．メッセージが長い場合は電子署名を生成に時間がかかるため，メッセージのハッシュ値を利用して電子署名を作成するのが一般的である．電子署名は公開鍵暗号の秘密鍵を利用して生成されるため，電子署名を作成できるのは秘密鍵の所有者のみである．したがって，メッセージのハッシュ値に対して電子署名を生成すれば，メッセージの作成者が秘密鍵の所有者本人であることが証明できる．電子署名を検証する場合は，互いにハッシュ演算のアルゴリズムと公開鍵暗号の種類を前もって共有しておかなければならない．代表的なものはあらかじめ方式が定められており，利用される公開鍵暗号方式に倣って RSA 署名や DSA 署名などと呼ばれる．図 2.2 に電子署名の生成と検証の概要図を示す [3]．

また，秘密鍵が漏洩していないという条件下であれば，メッセージの作成者が「メッセージを作成していない」と否認することも防止できる．したがって，そのような条件下では電子署名の真正性を検証することで否認不能性が保証される．

デジタルフォレンジックにおいてはログファイルの否認不能性を証明するために利用されている．

以上のようにログファイルの検証ではハッシュ値による同一性証明と電子署名による否認不能性の確保が要請される．

2.2　クラウドフォレンジック

2.2.1　クラウドフォレンジックの概要

クラウドフォレンジックとはクラウドコンピューティングでのインシデント調査を目的としたデジタルフォレンジック技術である．クラウドコンピューティングの定義でもっとも著名なアメリカ国立標準技術研究所 (NIST: National Institute of Standards and Technology) の定義を以下に引用する．[4]

> Cloud computing is a model for enabling ubiquitous， convenient， on-demand network access to a shared pool of configurable computing resources (e.g.， networks，servers，storage，applications， and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction. This cloud model is composed of five essential characteristics，three service models， and four deployment models.
> 【クラウドコンピューティングは，共用の構成可能なコンピューティングリソース（ネットワーク，サーバー，ストレージ，アプリケーション，サービス）の集積に，どこからで

[3]図は https://upload.wikimedia.org/wikipedia/commons/2/2b/Digital_Signature_diagram.svg を参考にしている．
[4]和訳は独立行政法人　情報処理推進機構の『NIST によるクラウドコンピューティングの定義』から引用

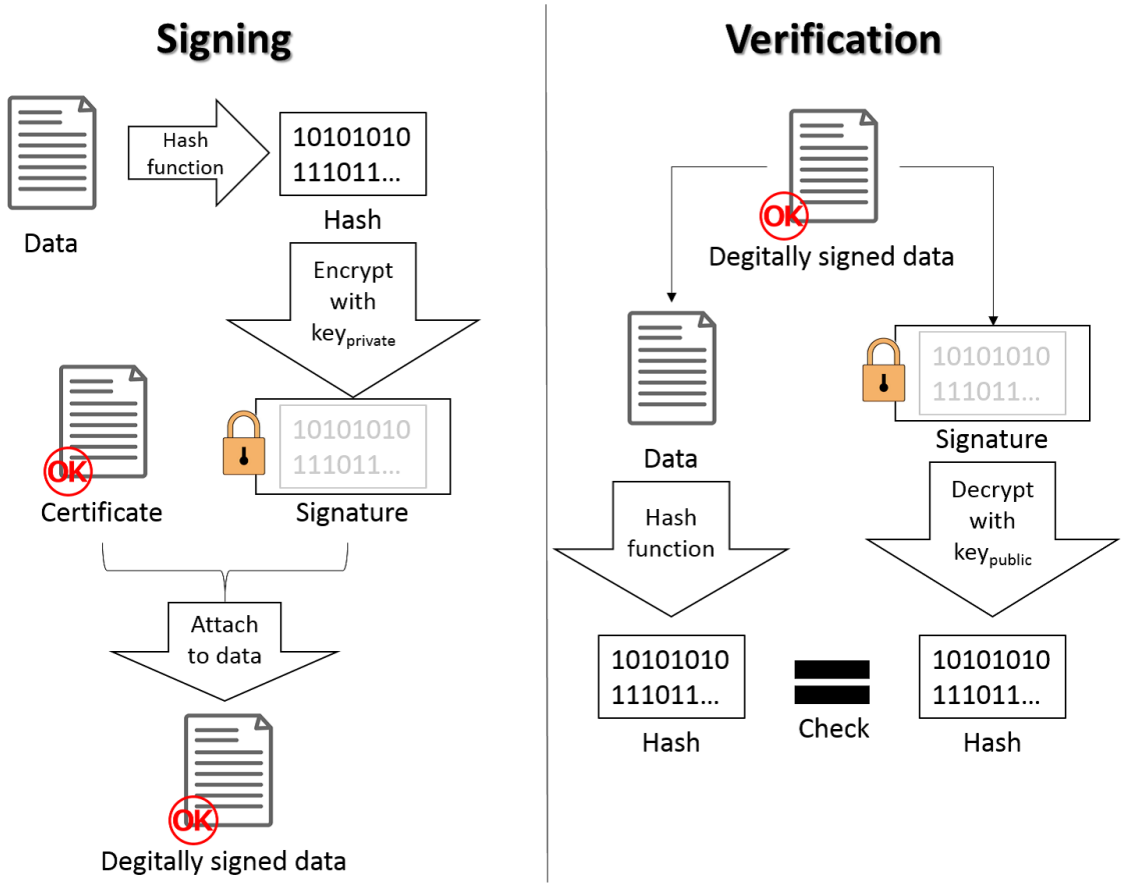


図 2.2: Digital Signature diagram

も，簡便に，必要に応じて，ネットワーク経由でアクセスすることを可能とするモデルであり，最小限の利用手続きまたはサービスプロバイダとのやりとりで速やかに割当てられ提供されるものである．このクラウドモデルは 5 つの基本的な特徴と 3 つのサービスモデル，および 4 つの実装モデルによって構成される.】

NIST の定義によればクラウドサービスのモデルは図 2.3 のように 3 つに分類される．中でも IaaS は他の二つよりもコンピューティングリソースに対するコントロール権が強いため，他の二つのサービスモデルは IaaS のコントロール権を制限したものとみなすことが可能である．したがって，本論文ではクラウドサービスとして IaaS を想定して議論する．IaaS では遠隔からクラウドプロバイダの提供する仮想マシン（VM: Virtual Machine）を利用することができる．クラウドプロバイダは世界各地にデータセンターを保有しており，そこで稼動するサーバやストレージによって VM が構成される．例えば，IaaS は Web アプリケーションやデータベースなどの仮想サーバとして利用されることが多く，従来の物理サーバよりも初期・運用コストやメンテナンスの面で大きなメリットがある．利用する OS やハードウェアの設定も自由であり，利用していない時は VM を止めることも可能であ

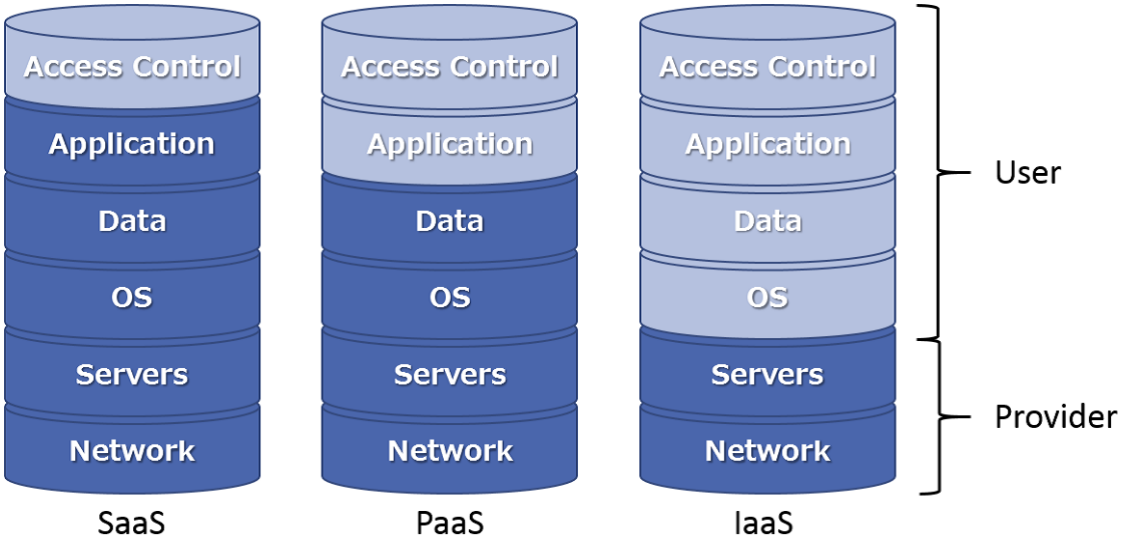


図 2.3: Cloud Service Models

る．IaaS ではこのように利用者に与えられる自由度が高く便利な反面，セキュリティに関する設定や構成を自分で行う必要がある．したがって，VM からの機密情報漏洩や root 権限奪取による VM 悪用というような危険性が指摘されており，そのような場合のログファイルを対象とした証拠調査・収集技術としてクラウドフォレンジックへの期待が高まっている．

2.2.2 クラウドフォレンジックにおける課題

クラウド環境はこれまでのデジタルフォレンジックとは異なり，新たな課題が多くさまざまな議論が交わされている．以下にクラウド特有の課題を 4 点述べる．

保全データサイズの爆発 米国の FBI の RCFL の 2012 年の報告書 [3] によるとフォレンジックで保全したストレージの容量は 4263TB から 5986TB と年に 40 %も増加している．このように保全対象のサイズが巨大化すると保全に膨大な時間が必要となる．クラウド環境では通常よりも多くのログファイルを収集する必要があるため，現実的な時間で保全が終わらず捜査が難航する可能性がある．また，クラウド環境ではネットワークログやプロセスログなど様々なコンポーネントでログを収集している．ログファイルはクラウド上で散在しており，ログファイルの管理・解析は非クラウド環境と比べて複雑なものとなっている．Ruan らはクラウドフォレンジックの重要性と課題をまとめ，クラウドフォレンジックではディスクフォレンジック，ネットワークフォレンジック，モバイルフォレンジックなど様々なフォレンジック技術を複合させる必要があると報告 [7] している．このように保全対象となるデータはサイズ・種類ともに通常よりも多く，保全に膨大な時間を要するという課題がある．

Resource Pooling（リソース共有） 一般的にクラウドプロバイダの保有するデータセンターには膨大な数のコンピューティングリソースが集積されており，不特定多数のユーザ VM が同一の物理ハードウェアを共有している．これを Resource Pooling [4] と呼ぶ．特にメモリやストレージなどの機密データが存在するハードウェアを共有している場合，VMI(Virtual Machine Introspection) [8] と呼ばれる技術で他人の VM の情報へ不正アクセスする攻撃やハイパーバイザを乗っ取るような危険性が指摘されている [5] [9] [10] [11] [12]．図 2.4 は VMI の概要図である．Xen や KVM といったハイパーバイザでは libVMI などのライブラリを利用することが出来る．これによって他人の VM のハードウェア情報を覗き見ることが出来る．Resource Pooling はデータセンター内に止まらず外部の他のデータセンター間で行われる場合もあり，マシンのコントロール権限や所有権の境界が曖昧になる．その結果，フォレンジックの際の調査権限の手続きや証拠収集の煩雑化を引き起こしている．以上のように，Resource Pooling はクラウドフォレンジックを困難にしている課題の一つである．

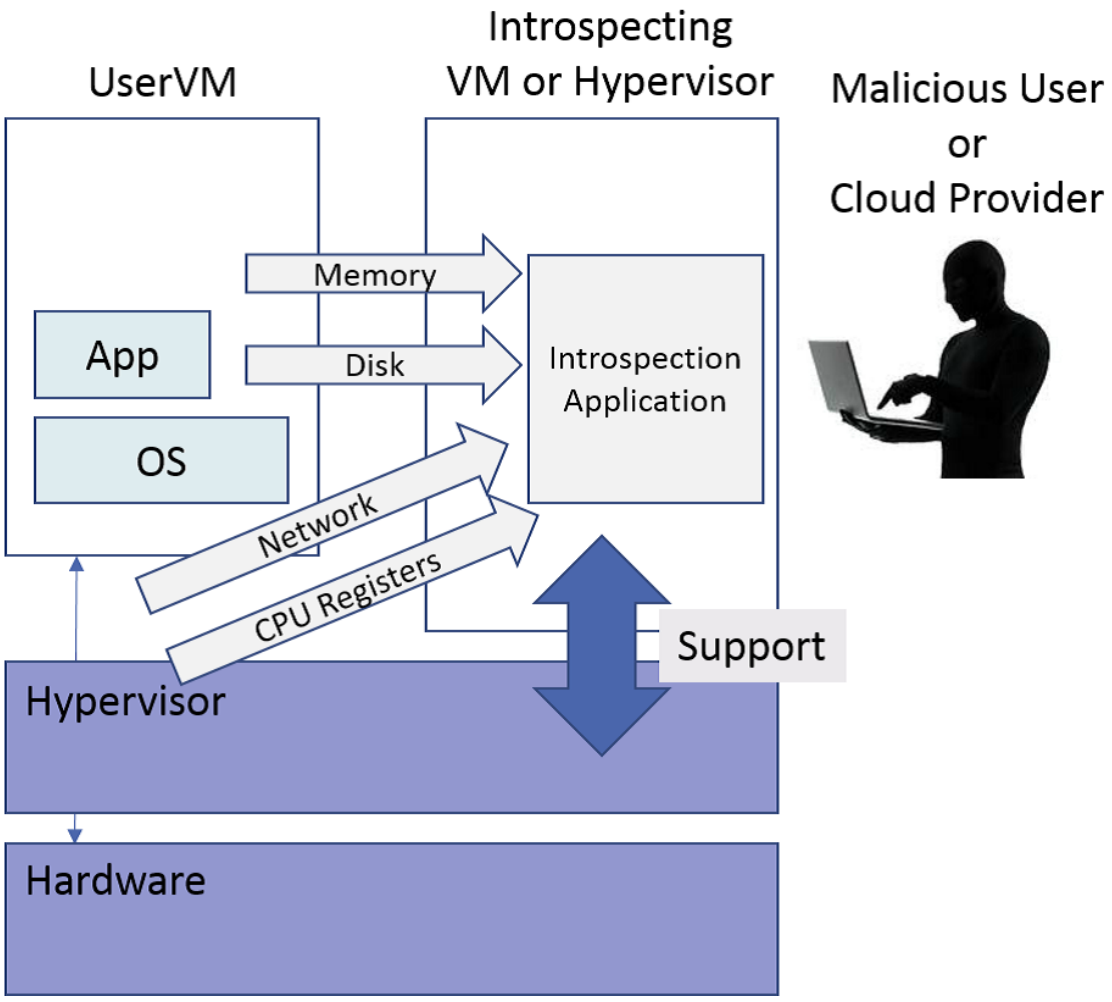


図 2.4: Virtual Machine Introspection

Chain of Custody(証拠保全の一貫性)　高度な攻撃者はログファイルを改ざんすることで不正アクセスの痕跡を隠蔽することが多く，保全したログファイルを証拠として用いるならば完全性・真正性の検証が必須である．NIST(National Institute of Standards and Technology) が作成しているクラウドフォレンジックの課題をまとめた報告書 [13] では，ログファイルの収集から処分までのプロセスを明示しログの原本同一性を保証する「Chain of Custody(証拠保全の一貫性)」の重要性が述べられている．

クラウドサービスプロバイダが信頼できるか　また，昨今ではクラウドサービスを提供するプロバイダ内部の従業員によるデータの持ち出しや機密データの不正利用も問題となっている．Dycstra らの調査 [14] ではクラウドプロバイダが証拠となるデータを提供するのがシンプルで効率よいと報告しているが，クラウドプロバイダを信頼できるかどうかの議論は不可欠である．

以上の問題が今日のクラウドフォレンジックのボトルネックとなっている．

2.3　クラウド環境におけるログファイル管理

2.3.1　クラウドフォレンジックが対象とするログファイル

一般にクラウド環境はデータベース・Web アプリケーション・プロキシといった各種サーバやネットワーク機器などさまざまなコンポーネントで構成され，ログファイルの種類も多岐にわたる．クラウドフォレンジックにおいて特に重要なログファイルの種類を以下に列挙する．

- ファイアウォール
- IDS/IPS
- ネットワークログ
- 各種サーバ OS のイベントログ
- ハイパーバイザのイベントログ

このような種々のログは各コンポーネントで一定量まで蓄積される．この際，各コンポーネント間で時刻が同期されていることが重要である．ログファイルはタイムスタンプとログメッセージの羅列であり，このタイムスタンプが各コンポーネント間で同期されていないと，各メッセージを時系列順に解析することが不可能となってしまう．クラウド環境では NTP サーバを用意することで各コンポーネント間でのタイムスタンプに整合性を持たせるのが一般的である．

2.3.2　syslog サーバによるログファイル管理

各コンポーネントに蓄積されたログは syslog サーバと呼ばれるログファイル管理の専用サーバに集約されて管理されるのが一般的である．各コンポーネントには予め syslog サーバへログファイルを送信するように直接プログラムするか，コンポーネント内でログファイルを管理する syslogd のような常駐型プログラムをインストールしておく．コンポーネント内で一定量に達したログファイルは圧縮された後，syslog サーバに転送される．

syslog　syslog はログメッセージを IP ネットワークで転送するためのクライアント/サーバ型プロトコル [5] であり，Unix 系の OS における標準的なログ出力方法である．syslog は様々なプログラミング言語でライブラリ関数として用意されており，アプリケーションや OS やデバイスから syslog を送信できるようになっている．また，syslog は Unix 系の OS に限らず Windows をはじめとする様々な OS でも利用することが可能であり，コンピュータシステムにおけるログ管理のデファクト・スタンダードとなっている．

syslogd　各アプリケーションや OS の送信するログメッセージは syslogd と呼ばれる常駐型プログラムが一元的に管理を行う．syslogd は Unix 系の OS にデフォルトでインストールされているが，本来はデバッグやトラブル復旧のための記録を目的としたものであったため，管理情報が乏しいことや TCP 通信が出来ないことなど信頼性に課題が多く，その克服に強い要望があった．現在では信頼性を向上させた rsyslog [6] や syslog-ng [7] といった新世代の syslogd が提供されている．

2.4　本研究での脅威モデル

本研究では IaaS 環境でのクラウドフォレンジックを前提として，ログファイルの機密性・完全性・真正性を確保するとともにログファイルの検証を効率化するシステムを提案する．ここでは想定する脅威モデルを定義する．まず第一に保護すべき情報資産と攻撃者を述べ,「誰から何を保護するのか」を明確にする．さらに，クラウド環境の各コンポーネントのシステム構成などを定め，全体的な脅威モデルを図示する．

2.4.1　保護すべき情報資産

クラウドフォレンジックではログファイルの他にも電子メールやドキュメントなどさまざまなデータが対象となるが，本研究ではログファイルに対象を限定する．ログファイルはフォレンジックの際に完全性・真正性が求められるだけでなく，通信記録やアプリケーションデータなどのプライバシーに関わる情報の機密性を確保することも必要である．2.3.2 で述べたようにログファイルは syslog サーバに集約されるのが一般的である．したがって本研究において，保護対象は syslog サーバが保有するクラウド環境全体に関連するログファイルとする．

2.4.2　攻撃者

攻撃者としては以下の 3 者を想定する．

- クラウドプロバイダ
- 悪意を持ったユーザ

[5] http://tools.ietf.org/html/rfc3164
[6] http://www.rsyslog.com/
[7] https://www.balabit.com/network-security/syslog-ng

- フォレンジック調査者

クラウドプロバイダは企業そのものと企業に属する従業員全てを含む．たとえクラウドプロバイダの社員でなくても，外部委託や派遣契約によってクラウドサービスに関わるものは攻撃者となりうるものと仮定する．また，ユーザの中にも高度な技術を持ち金銭目的や政治的な意図を持って他のユーザの機密情報を狙う攻撃者がいるとする．フォレンジック調査者とはユーザや法的機関から依頼を受けてフォレンジック調査を行う企業や組織を意味する．フォレンジック調査者も内部の人間が賄賂や脅迫などによって，不正な機密情報を流出や虚偽の報告をする可能性が想定される．

2.4.3 想定される脅威

ログファイルの機密性に対する脅威として「情報漏洩」，完全性・真正性に対する脅威として「ログファイルの改ざん・捏造・隠蔽」が想定される．具体的な脅威を以下に示す．

オペレータによる情報漏洩　データセンターではクラウドプロバイダの社員（オペレータ）がサーバやソフトウェアの管理を行うが，そのようなオペレータの中には金銭目的や政治的な意図を持ってユーザの個人情報を不正に利用しようとする場合が想定される．このような企業内部の人間の手による情報流出事件は実際に報告されている．また，悪意を持っていなくても USB メモリの紛失や操作ミスから機密情報が流出することも考えられる．

VMI による情報漏洩　VMI は管理 VM やハイパーバイザのアクセス権限を利用してユーザ VM のメモリやネットワーク情報などを監視する技術であり，これを悪用することでユーザ VM の機密情報に不正にアクセスすることが出来る．VMI はオペレータだけでなく，悪意をもったユーザが管理 VM やハイパーバイザの権限を奪取した場合も可能である．

外部からの不正アクセスによる情報漏洩　2014 年には米国 Apple 社のクラウドサービス「iCloud」においてプライベート写真や動画が流出する事件があった．原因はパスワードが推測されアカウントが乗っ取られたことであった．iCloud は強固なセキュリティ対策が施された SaaS であるが，IaaS の場合セキュリティに関する設定は各々が自己責任で行わなければならない．したがって，IaaS 環境では秘密鍵を不正に手に入れたりアクセス権限を奪ったりするという不正アクセスの危険性がより一層高まる．

ログファイル改ざん・捏造・隠蔽　フォレンジック調査者はいつも正しいとは限らない．金銭目的や脅迫を受けている場合，ログファイルの解析の際に一部を改ざんしたり不都合なログファイルを抹消したりする危険性が考えられる．クラウドプロバイダのオペレータも同様である．また，ユーザ自身が犯罪を犯したり不正を働いたりする場合も，ユーザにとって不都合なログファイルを改ざん・捏造・隠蔽を行う可能性も想定される．

以上の脅威を攻撃者ごとにまとめると表 2.2 のようになる．フォレンジック調査者はユーザのログファイルを扱う必要が生じるため，ログファイルの情報漏洩に関しては想定しないものとする．

表 2.2: 想定される脅威のまとめ

攻撃者	機密性	完全性・真正性
クラウドプロバイダ	○	○
悪意を持ったユーザ	○	○
フォレンジック調査者	—	○

2.4.4 クラウド環境のシステム構成

本研究におけるクラウド環境のシステム構成について述べる．クラウド環境としては IaaS を想定し，Xen や Hyper-V といったハイパーバイザ型の仮想化環境で VM が管理されているものとする．ユーザ VM が管理されているサーバは図 2.5 では VMserver である．ユーザはプロバイダから提供されたゲスト OS を選択し，メモリやストレージを設定してユーザ VM を起動させるが，プロセッサは第 5 章で後述する VM セキュアプロセッサが採用されているものと仮定する．その他各種サーバ（syslog サーバや Web サーバ）でも VM セキュアプロセッサが稼動しているものとする．クラウドはファイアウォール，データベースサーバ，認証サーバ，IPS/IDS，Web サーバ，ストレージサーバ，その他ネットワーク機器などさまざまなコンポーネントで構成されるが，各コンポーネントで生成されるログファイルは全て syslog サーバに集約されるものとする．その際，コンポーネントと syslog サーバの通信路は暗号化され，機密性は保たれるものと仮定する．

前提として信頼できるものを以下にまとめる．

- VM セキュアプロセッサ＆プロセッサメーカ
- VMServer や DB サーバ上の syslogd
- syslog サーバ

本研究ではフォレンジックを目的としているため，システムが起動した瞬間は全てのコンポーネントが正常に起動したものとする．たとえ，どこかのコンポーネントがマルウェア感染やハードウェアタンパなどの攻撃を受けたとしてもログファイルの完全性・真正性・機密性が確保できれば本研究の目的は達成される．

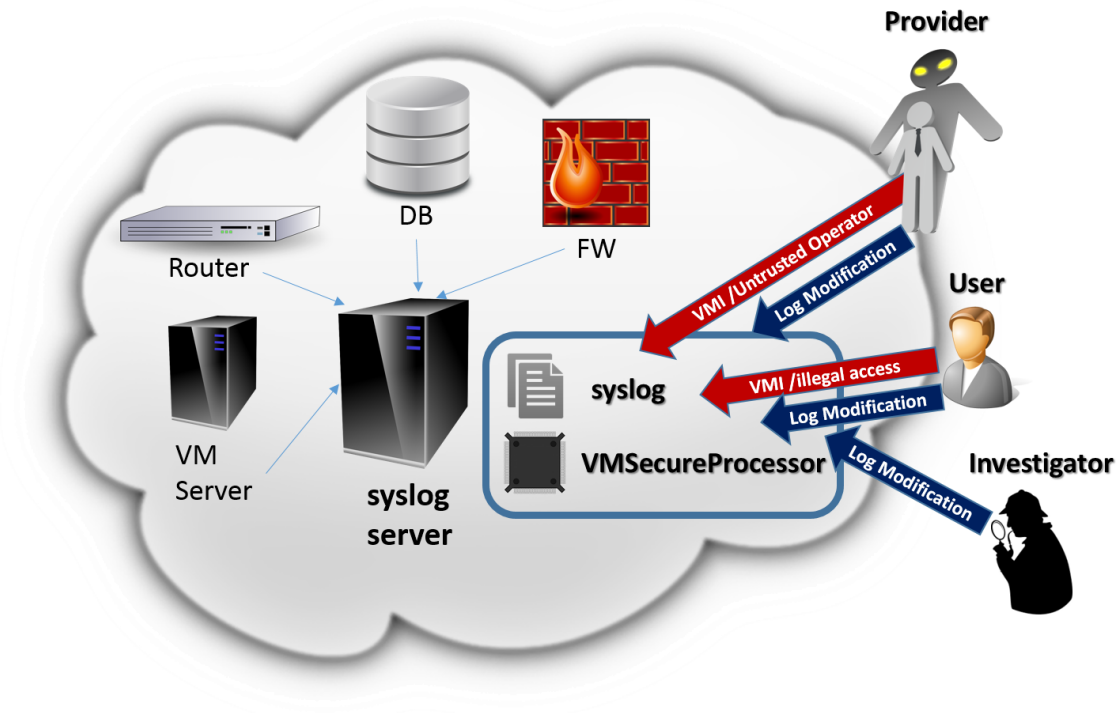


図 2.5: Treat Model

第3章　ハードウェアセキュリティ

本章では第 4 章と本研究の提案手法に関連するハードウェアセキュリティの知識をまとめる．

3.1　TPM

TPM(Trusted Platform Module) は RSA 暗号や SHA-1 の演算やハッシュ値の管理を行うチップであり，マザーボードに直接搭載され CPU と通信しながら動作する．最新版は TPM2.0 である．その仕様は Trusted Computing Group [1] という国際的な業界団体によって策定されている．TPM を用いることで，電子署名の生成・検証やプラットフォームの認証を行うことが出来る．

TPM には内部に PCRs(Platform Configuration Registers) と呼ばれるレジスタを持ち，0 から順番に番号が割り当てられている．PCRs は計測したハッシュ値や特定の数値を保存するために用いられ，TPM(ver1.2) では 24 個のレジスタが用意されている．システム起動時に TPM によって BIOS やブートローダやハイパーバイザのハッシュ値を連鎖的に計測し，PCRs に保管していくことでプラットフォームの認証を行うことができる．

TPM の電子署名生成＆検証プロトコル　TPM でハッシュ値から電子署名を生成する手順は以下のとおりである．

1. 署名の対象となるハッシュ値（H1）を生成する．
2. H1 を生成したハッシュアルゴリズムの識別子 OID(Object Identifier) と H1 を結びつける．この値を D1 とする．
3. D1 を公開鍵のパディングアルゴリズムによってパディングする．この値を P1 とする．
4. TPM 内の秘密鍵を用いて P1 を暗号化し，電子署名を生成する．

この手順は [16] によるもので実際のコードは以下のとおりである．

```
TSS_HHASH hHash;
BYTE *sig, *digest = /*hash value*/;
UINT32 sigLen, digestLen = 20;
/*Create the hash object as type SHA1*/
Tspi_Context_CreateObject(hContext, TSS_OBJECT_TYPE_HASH, TSS_HASH_SHA1, &hHash);
/*Set the digest in the hash object*/
Tspi_Hash_SetHashValue(hHash, digestLen, digest);
/*Sign the hash using hKey, a signing key with signature
*scheme TSS_SS_RSASSAPKCS1V15_SHA1*/
Tspi_Hash_Sign(hHash, hKey, &sigLen, &sig);
```

[1] http://www.trustedcomputinggroup.org/

最新版の TPM2.0 では SHA-256(SHA-2) や楕円曲線暗号が導入されているが，従来の TPM1.2 ではハッシュ関数として SHA-1，公開鍵暗号は RSA のみをサポートしていた．[16] は TPM1.2 の仕様に基づいているためコード内では SHA-1 と RSA が利用されているが，プロトコルとしてはそれぞれ SHA-2 と楕円曲線暗号に置き換えたものとしてみなせる．

TPM の難点　TPM はハードウェアによる強固なセキュリティを提供するが，TPM の難点として内部で利用できる計算資源が非常に小規模であることが挙げられる．クラウド環境で複数の VM を管理するサーバにおいて，TPM を用いて VM を一つ一つ認証することが困難である．VM を認証するためには認証のための情報（公開鍵や署名など）を TPM 内に保管しなければならないが，TPM 内のレジスタは非常に小規模であるため，複数の VM を認証するのは非現実的である．

さらに TPM では署名生成や暗号化を要請するために独自の API が用意されているが，これらの機能をソフトウェアに実装する場合，TPM の内部仕様 [2] を考慮する必要がある．この作業が TPM の普及を妨げているということも課題の一つである．

タンパに関しても課題はある．TPM とプロセッサの間の通信はバスによって行われているため，ロジックアナライザなどで直接バス信号を解析するような攻撃によって機密情報が漏洩してしまう可能性がある．また，TPM がマルウェアによって悪用されてしまい，メモリ上に OS やハイパーバイザからアクセスできない領域を展開するという攻撃 [15] も存在する．

3.2　HSM: Hardware Security Module

HSM は鍵の管理や暗号化・復号を行う専用ハードウェアであり，政府機関や金融機関など機密データを扱うシステムで利用される．HSM は外部デバイスとして既存のシステムに接続して利用するのが特徴であり，USB タイプのものやネットワーク経由のものなど様々な種類が存在する．HSM は多くのメーカから提供されているが，国際標準規格（ ISO/IEC 15408）である「Common Criteria for Information Technology Security Evaluation」や米国連邦標準規格「FIPS140-2」による要求仕様を満たしていることが推奨される．代表的な IaaS である Amazon Web Service(AWS) の提供するサービスの一つである Amazon CloudHSM [3] では米国の SafeNet 社 [4] の LunaSA HSMs を利用している．図 3.2 ユーザは CloudHSM のサーバに SSL で接続することで，AWS で利用する鍵を安全に管理することが出来る．

3.3　セキュアプロセッサ

セキュアプロセッサはアプリケーションのプロセスの持つ機密情報をタンパから保護するプロセッサである．これまでに Ascend [17] や AEGIS [18] [19] [20]，XOM [21] [22]，L-MSP [23] などといったセキュアプロセッサが研究されている．セキュアプロセッサの動作環境では，プロセッサ内部のみを信頼できる領域と想定し，プロセッサ外部（メモリや OS など）は信頼できない領域であると想定している．

[2] エンディアンなど
[3] http://aws.amazon.com/cloudhsm/
[4] http://www.safenet-inc.com/

図 3.1: LunaSA HSM

3.3.1 セキュアプロセッサの機能

プロセッサ外部では機密情報を保護するためにデータに対して暗号化を行う．セキュアプロセッサではプロセッサ内部のキャッシュメモリと外部のメモリ間でキャッシュラインが入れ替わるタイミングで暗号化・復号が行われる．暗号化・復号はキャッシュライン単位で行われる．一方，プロセッサ内部では演算処理を行うためデータは平文でなければならない．したがって，プロセッサ内部の情報はアクセス制御によって保護する．例えば XOM ではレジスタファイルにプロセスの識別子を追加し，異なるプロセスからレジスタへのアクセスを禁止することで実現している．その他にも乱数生成やハッシュ値演算などの機能をもつのが一般的である．

3.3.2 セキュアプロセッサの具体例

セキュアプロセッサの中には固有の機能を有するものも散見されるため，具体例を示す．

AEGIS Suh らの提案する AEGIS ではメモリの完全性検証のためにハッシュツリーと呼ばれる独自のアルゴリズムを用いており，メモリ上の各プロセス領域のハッシュ値をまとめてハッシュ演算を繰り返すことで一つのハッシュ値（ルートハッシュ）に集約している．通常プロセッサ内部に保持できるデータサイズは小さく，複数のハッシュ値をプロセッサ内部に保存することは困難であった．それに対して，AEGIS ではすべてのプロセス領域のハッシュ値がルートハッシュに反映されるため，プロセッサ内部にはルートハッシュのみ保存すればよい．このように AEGIS ではハッシュツリーの頂点であるルートハッシュだけを比較することで，大規模なプログラムに対する効率的で高速な完全性検証を可能にした．

Intel SGX 米国 Intel 社ではプロセッサの命令セットアーキテクチャにプログラムを保護するために 17 の拡張命令を実装した IntelSGX(Software Guard Extensions) [24] を開発しており，一部の製品には既に実装されている．IntelSGX では管理者権限を奪取するようなマルウェアからプロセスを保護することを目的としている．IntelSGX では図 3.3 メモリ上に Enclave と呼ばれる保護領域を展開して，不審なメモリアクセス検出とアドレスマッピングの保護を行い，プロセスを保護している．Intel 社では Intel SGX とともに Intel MPX(Memory Protection Extensions) と Intel SHA(Secure Hash Algorithm

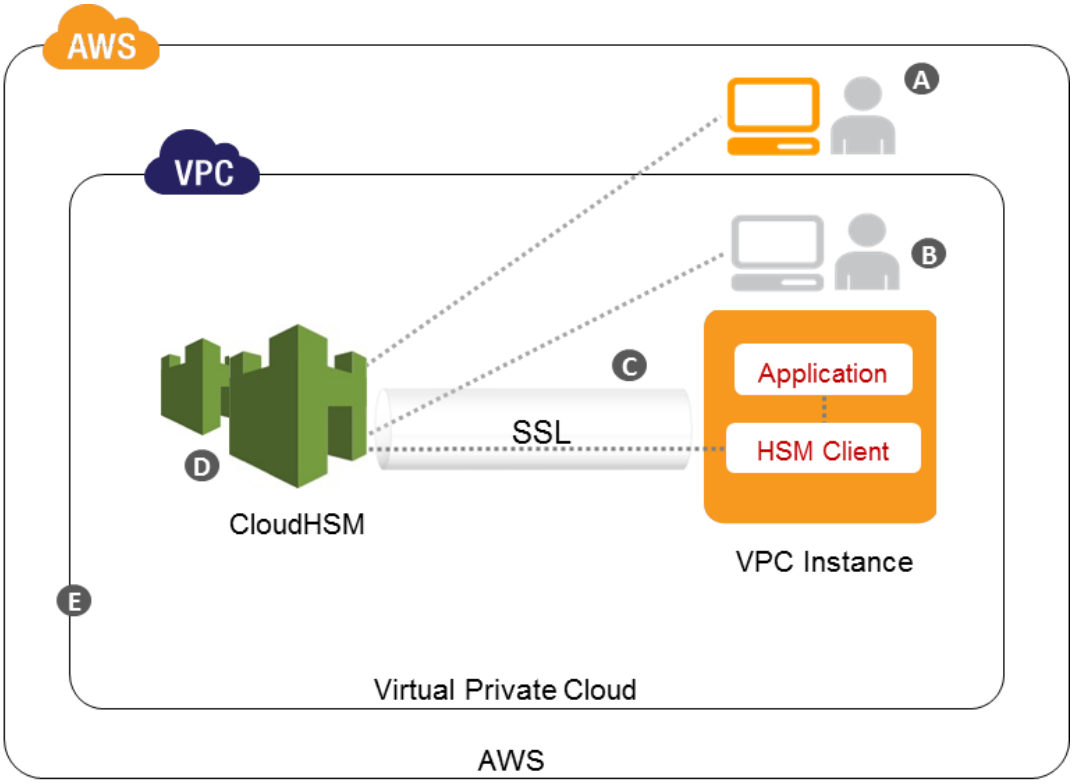


図 3.2: Overview of Amazon CloudHSM

Extensions) の設計も行っている．IntelMPX ではバッファオーバーフローによる範囲外のメモリ参照を防止するために，ハードウェアによってポインタのチェックをサポートしている．IntelSHA では拡張命令を利用することで SHA-1，SHA-2 の演算を高速化する．

3.3.3 セキュアプロセッサの課題

このように様々な研究機関や企業でセキュリティ機能を持つプロセッサが開発されているが，セキュアプロセッサでは OS をセキュアプロセッサ用に改変する必要がある．例えば，プロセッサによってプロセス単位の保護を行う場合，プロセスが切り替わるたびにシステムコールや割り込み処理をプロセッサが検出し，プロセスにあわせて鍵を切り替えるという処理が必要となる．一般的に OS のコードは複雑かつ膨大であり，セキュアプロセッサのために OS を改変することは困難である．また，改変を加えた場合 OS のアップデートを適用することも難しくなる．このように，プロセス管理に関する OS の根幹部分の改変がセキュアプロセッサの実用化への大きな障害となっている．

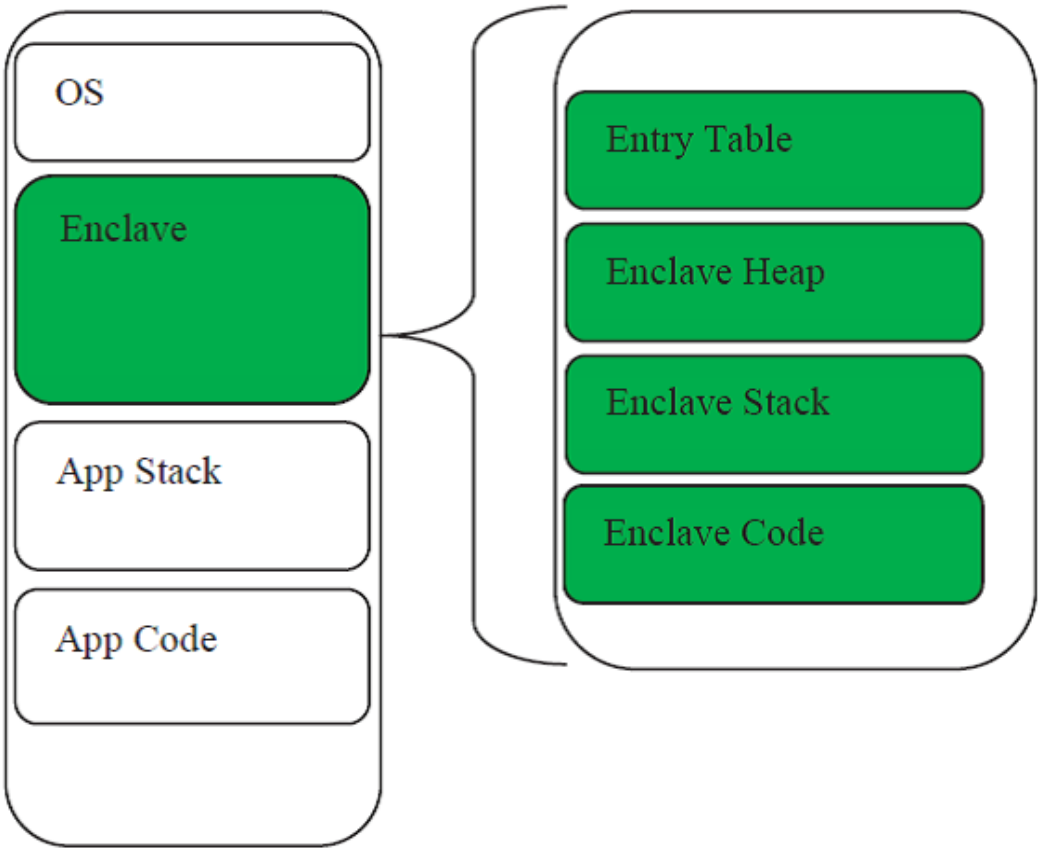


図 3.3: Enclave within Application’s Virtual Address Space

第4章　フォレンジック関連技術

本章では第 2 章の図 2.5 で示したような脅威モデルに対して，その課題の解決を目的とした関連手法や応用可能な基礎技術について説明する．本研究ではログファイルの管理をハードウェアで行うことに主眼を置いているため，ソフトウェアかハードウェアかという軸によって分類する．また，脅威モデルではクラウドプロバイダは信頼できないものとして想定しているが，インシデント調査ではクラウドプロバイダのサポートを受けるのが一般的であるため，参考としてその概要を述べる．

4.1　ソフトウェアを用いる手法

4.1.1　VM 調査・解析専用ソフトウェア

現代クラウドフォレンジックで最も一般的なのが調査・解析に特化したソフトウェアによる手法である．ただし，非クラウド環境で用いるフォレンジックツールに関しては NIST により動作要件の評価が行われているが，クラウド環境で用いる遠隔保全ツールに関してはいまだに明確な動作要件は定義されていない．ここでは現在普及している二つのツールを紹介する．

EnCase Enterprise　(EE)　EnCase は米国ガイダンスソフトウェア社 [1] が開発・販売しているアプリケーションであり，遠隔からストレージのイメージファイルやメモリ情報を保全する．スナップショット機能によりシステムを稼動させたまま解析できるのが大きな特徴である．EE ではまずサーブレットと呼ばれる軽量の常駐型アプリケーションを RDP によってゲスト OS にインストールし，TCP/IP によって調査者と通信する．Boeck らの論文ではログファイルの真正性について述べられていなかったが，現時点ではログファイルの完全性をハッシュ値によって検証している．

Forensic Toolkit (FTK)　Forensic Toolkit は米国 AccessData Corporation [2] が開発・販売しているアプリケーションであり，EnCase Enterprise と同様にエージェントと呼ばれる軽量アプリケーションをクライアントにインストールすることでメモリやハードディスクやネットワークを分析する．FTK ではデータの可視化に注力しており，大規模なログファイルでも効率よく分析できることが特徴である．Dykstra らは FTK Imager LITE を利用してユーザ VM から 100GB の二次記憶データを保全し，1.7GB のシステムメモリ情報が得られることを確認した．

[1] https://www.guidancesoftware.com/
[2] http://accessdata.com/

4.1.2 VMI: Virtual Machine Introspection

VMI は VM のメモリやネットワークの状態を外部の VM やソフトウェアから監視する技術である．クラウド環境における脅威として VMI が挙げられるが，本来は外部から VM を監視する IDS としてマルウェア検出やソフトウェアの異常を見つける技術である．例えば，Livewire [25] や HyperSpector [26] では VMI を利用してユーザ VM 内の rootkit を検出している．仮想化環境を構築するハイパーバイザの Xen や KVM では VMI がライブラリ (libVMI) として提供されている．Dykstra らは libVMI を利用してユーザ VM の物理メモリに，前述の EnCase Servlet などのツールを直接書き込めることを示した．そのほかにも，Symantec 社が VMWare の仮想化環境で VM へウイルス対策コードを書き込めることを示している [27]．Baek らの研究では，ユーザが VMI を行えるようなアプリケーションとして Xen 上に CloudVMI [28] を実装している．

4.1.3 SecLaaS

Zawoad らはクラウドプロバイダが信頼できない場合，ログファイルの真正性だけでなく機密性が保証されないという問題に着目した．この問題に対して，ユーザのログファイルの機密性を保ちながら信頼できるログファイルを取得できるクラウドアーキテクチャ SecLaaS(Secure Logging as a Service) [29] を提案している．SecLaaS は OpenStack [3] を利用して VM のネットワークログを収集するところまで実装済みである．図 4.1 に SecLaaS の概要図を示す．

彼らは，攻撃者としてユーザ・クラウドプロバイダ・フォレンジック調査機関の三者それぞれが悪意を持った場合を想定しており，それぞれが口裏を合わせて結託した場合でも対策可能であると主張している．

SecLaaS では，VM の運用を行う NC(Node Controller) でログを暗号化して，ログストレージサーバに保管する．さらに，ログの生成順序を証明するためにログチェインと呼ばれるハッシュ値を計測して，ログチェインもログストレージサーバに保管する．最後に，一日分のログとログチェインをまとめて，そのハッシュ値と利用した公開鍵などを Web に公開する．フォレンジック調査機関はこれらのサーバに API 経由でリモート接続して，ログファイルを調査することが可能である．SecLaaS はフォレンジック調査機関がクラウドプロバイダに調査を要請する手間を省くことが出来るため，これまでよりも効率のよいログファイル調査を可能にした．

SecLaaS では今後の課題として，ネットワークログ以外のログを現在の実装に統合していくことを挙げている．さらに最終的には OpenStack のモジュールとして公開することを目標としている．

4.2 ハードウェア（TPM）を用いる手法

4.2.1 Boeck らによる手法

Boeck らは現在主流の専用ツールによる手法はソフトウェアの改変によりいずれ無効化されるものとして，信頼性の基点をソフトウェアからハードウェアに移すことを目指した [30]．彼らはログ収集に用いられるアプリケーションを信頼することは遠隔保全ツールの構造的欠陥であると指摘してい

[3] http://www.openstack.org/

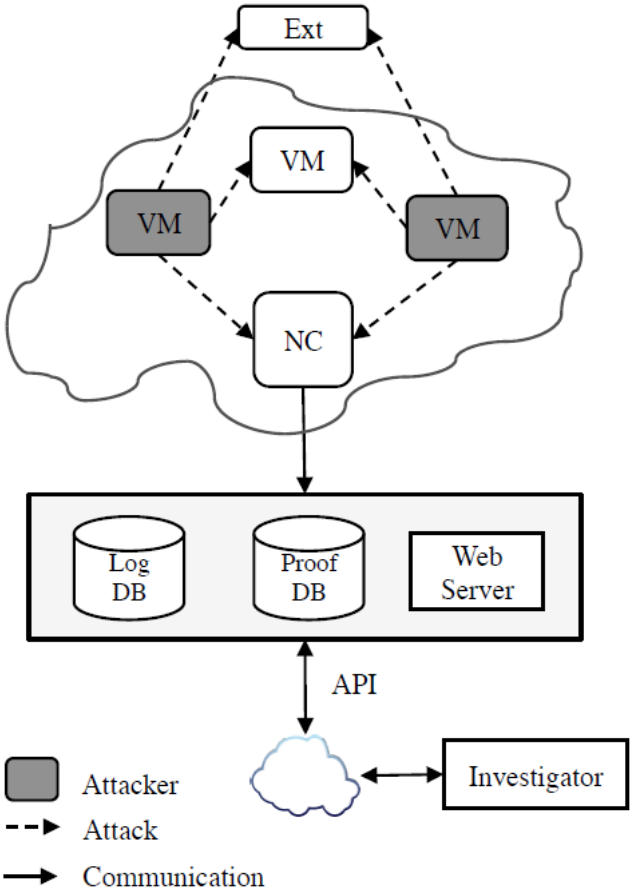


図 4.1: Overview of SecLaaS

る．彼らの手法ではログファイルだけでなくログを生成するアプリケーションが信頼できることを保証できる．

彼らの提案手法では LLC(legitimate logging client) がクライアントマシン上でログを収集し，サーバに送るというモデルを想定している．彼らの論文では「クラウド環境である」とは明記されていないが，IaaS 型のクラウド環境でも同じモデルが想定できる．図 4.2 では「クライアントマシン」がクラウド環境の各コンポーネントに相当し，サーバマシンが syslog サーバである．彼らの手法では LLC の起動に TPM を用いている．TPM によって LLC の真正性を検証し，サーバマシンと公開鍵の交換を行う．ログファイル生成時には TPM による電子署名を作成し，サーバの公開鍵で暗号化してから送信する．以上の手順によりログファイルの機密性と真正性を保証することが出来る．

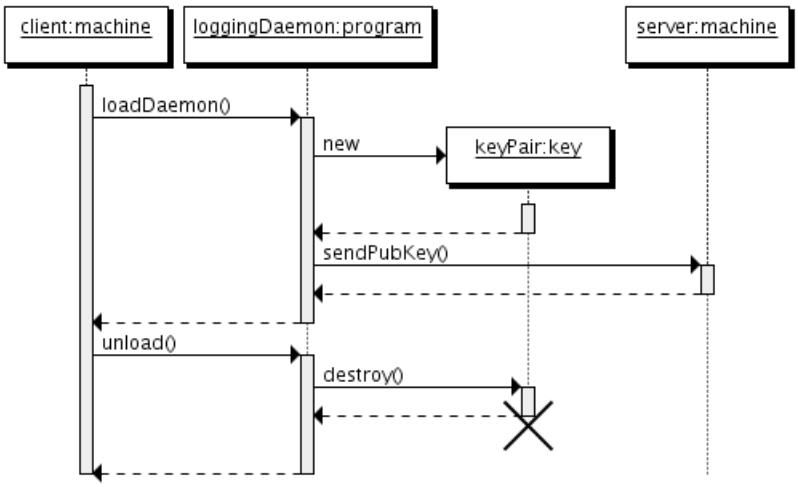


図 4.2: Initialization of Trust Relationship between Client and Server

4.2.2 Liu らによる手法

Liu らはクラウド環境で信頼できるログファイルを収集し，安全に別のクラウド環境へ転送するために，TPM を用いたシステムを実装した [31]．図 7.3 は Liu らのシステムの概要図である．Liu らのシステムではどのコンポーネントにも TPM が導入されていて，徹底的にハードウェアに基づくプラットフォーム認証が行われている．

このシステムで特徴的なのは **Chain-of-Custody Authority(CoCA)** と **Trusted Forensics Service(TFS)** と呼ばれる二つのコンポーネントである．CoCA はデータの受信者と送信者の間で中間者攻撃を防ぐための認証サーバの役割を果たす．TFS は TPM に直接アクセスすることが可能であり，CoCA からの転送命令に応じてデータの認証や鍵の転送を行う．

また，このシステムの収集対象となるデータは VM のスナップショットである．そのため，データ収集にかかる時間は一瞬であるが，スナップショットから不審なファイルを検索するには時間が必要である．また，VM の情報をまるごと保管するわけではないため，スナップショット元の VM に接続できない場合は元の VM のメモリやストレージをまるごと保全する必要があり，保全に要する時間やデータサイズのオーバーヘッドは大きくなる．

彼らのシステムでは VM を稼働させたままデータを転送するために，スナップショットを分割してストリーミングデータとして転送している．また，データは Watermark [32] と呼ばれる手法によって完全性を検証する．Watermark を利用することで VM を稼働させたまま，データの完全性を検証することが可能となる．

彼らのシステムは，例えば「クライアントのクラウド環境からフォレンジック調査機関のクラウド環境へ調査対象のスナップショットを転送して，遠隔からクライアントの VM 内を調査し，証拠となるログを収集する」という利用方法を想定している．

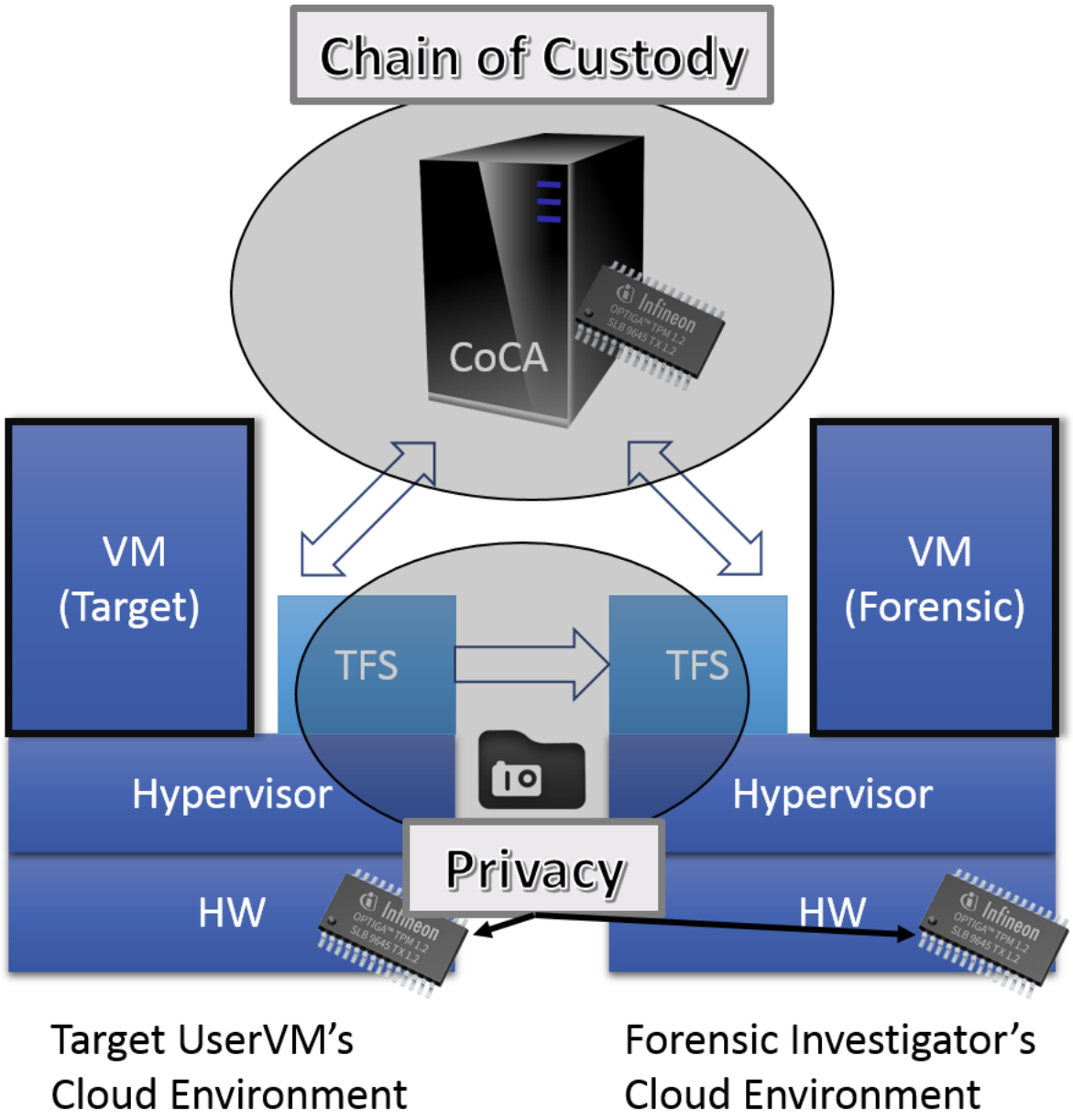


図 4.3: The System Overview with CoCA and TFS

4.3 クラウドプロバイダのサポートを受ける場合

クラウドサービスのプロバイダはクラウド管理のために，VM だけでなく各サーバからイベントログやパケットキャプチャの結果を収集している．法律に基づいて調査する権利が認められれば，クラウドプロバイダから提供されたログファイルをデジタルフォレンジックに利用することができる．

このようなサポートには，クラウドの構造を理解する必要もなく詳細なログファイルが得られるというメリットがある．また，ログファイルのサイズが大きい場合は遠隔から保全すると膨大な時間が必要となるため，クラウドプロバイダが直接機器からログを抽出したほうが効率的だといえる．したがって，法手続きに時間がかかるものの，クラウドプロバイダが信頼できると想定した場合は最善の手法であるといえる．

現実的には，クラウドプロバイダは信頼できない場合が一般的であるため，電子的証拠に対していくつかのリスクが考えられる．クラウドプロバイダの管理が甘い場合は，脆弱性のあるソフトウェアをそのまま利用していたり，オペレータ自体が攻撃者となったりする可能性がある．このような場合，ログファイルの改ざんやプライバシーの侵害といったインシデントが想定される．また，クラウドプロバイダが提示する SLA(Service Level Agreement) によっては，ログファイルの保存期間や開示の条件などが異なる場合もあるので，注意が必要である．

第5章　VMセキュアプロセッサを用いた提案手法

VM セキュアプロセッサは IaaS 環境においてユーザ VM に属する機密情報の保護を目的としたプロセッサであり，我々の研究グループが提案・開発している新しいマイクロアーキテクチャである．本章ではまず VM セキュアプロセッサが想定する脅威モデルを簡単にまとめる．次に VM セキュアプロセッサの構造を説明し，その具体的な機能を述べる．さらに，それらの機能を組み合わせた認証プロトコルを示し，既存のセキュアプロセッサとの比較を行う．最後にクラウドフォレンジックの課題を解決するための VM セキュアプロセッサを用いた提案手法を述べる．

前提として構成する登場人物を定義する．

クラウドユーザ　IaaS の利用者であり，ユーザ VM の正規の所有者．

クラウドプロバイダ　クラウドサービスのプロバイダ企業．サービス開発や顧客情報管理を行う社員だけでなく，データセンターでのサーバ運用やメンテナンスを行う社員も含める．

プロセッサメーカ　VM セキュアプロセッサを製造するメーカ．

ユーザ VM 外部の攻撃者　クラウド内部・外部に関わらず，不正アクセスによってユーザ VM の機密情報を狙う攻撃者．マルウェアや VMI によって不正アクセスを行う場合の他に，VM の通信を傍受した中間者攻撃なども含む．

5.1　VMセキュアプロセッサが想定する脅威モデル

第 2 章と同様に，まず保護すべき情報資産とそれに対する脅威を述べ，全体的な脅威モデルを図示する．

保護すべき情報資産　VM セキュアプロセッサはユーザ VM に属する機密情報の保護を目的とする．ここではユーザ VM のリソースに含まれる情報はすべて機密情報であると定義する．具体的には，プロセッサ内部ではレジスタやキャッシュ，プロセッサ外部ではメモリやストレージ，ネットワークで通信されるデータが保護対象である．以下のような場合は機密性が保てないが，VM セキュアプロセッサの保護対象外とする．

- ユーザが情報を意図的に公開する場合　ex)VM を Web サーバとして利用する場合など

  - ユーザのプライバシーの侵害に当たらないため
- VM の管理のために必要なシステム情報 ex) メモリサイズ，プロセッサコア数，VMCS(Virtual Machine Control Structure) など
  - 攻撃者から見て有意な情報ではないため

攻撃者　攻撃者としては以下を想定する．

- クラウドプロバイダ
- ユーザ VM 外部の攻撃者

想定される脅威　脅威としては以下のようなものが挙げられる．

- ハードウェアタンパ [1]
  - バス解析
  - サイドチャネルアタック
- ソフトウェアタンパ [2]
  - VMI
  - ハイパーバイザの権限悪用
- クラウドプロバイダによる不正行為
- 不正なプロセッサによるなりすまし

5.2　VM セキュアプロセッサの構成

製造時に埋め込まれるルートキー　VM セキュアプロセッサは製造時にプロセッサメーカによって公開鍵と秘密鍵のペア（ルートキー）を埋め込む．これを公開鍵は公開鍵証明書 [3] と共に埋め込まれる．公開鍵証明書はプロセッサメーカーを認証局として，公開鍵がどのプロセッサに所属するのかを証明する役割を果たす．このように VM セキュアプロセッサは公開鍵証明書を用いることで「プロセッサの真正性の証明」を行うことが可能であり，不正なプロセッサによるなりすましなどを防止することが可能である．

[1] ハードウェアによるタンパ
[2] ソフトウェアによるタンパ
[3] 公開鍵の所有者の名前やメールアドレス，公開鍵本体に対して外部の認証局が作成した電子署名

VM キー生成・管理　VM セキュアプロセッサはプロセッサ内部に乱数生成機構を持ち，これを利用することで VM に固有な共通鍵や公開鍵ペアの生成・管理を行う．乱数生成機構では攻撃者に乱数を予測されないように注意を払わなければならない[4]．プロセッサ外部の情報（命令アドレスのばらつきやユーザ入力など）は攻撃者からある程度操作できるため，乱数生成の攪拌源として望ましくない．VM セキュアプロセッサでは現時点で未実装であるが，プロセッサに起因する情報を基に乱数生成する予定である．

メモリ暗号化　VM セキュアプロセッサは従来のセキュアプロセッサと同様にメモリにデータを書き出す際に暗号化を行う．暗号化はキャッシュライン単位で行い，メモリからキャッシュに読み込むときには復号する．図 5.1 にその概要を示す．実装では AES-CTR を用いており，鍵長は 128bit である[5]．

実装ではデータバスのみ暗号化している．その他のアドレスバスや制御バスに関しては VM セキュアプロセッサでは暗号化は不要であると考えられる．セキュアプロセッサではアドレスバスに関しては参照の局所性からプログラムに関する情報が漏洩する危険性があったが，VM セキュアプロセッサでは VM 全体が保護対象であるため前提が異なる．VM の保護を目的とする場合，外部からはどのプロセスがメモリにアクセスしているか不明なため，メモリ参照の局所性からプログラムを推測することは出来ないと考えられる．したがって，アドレスバスやコマンドバスから有意な情報が漏洩することはないとみなし，暗号化は行わない．

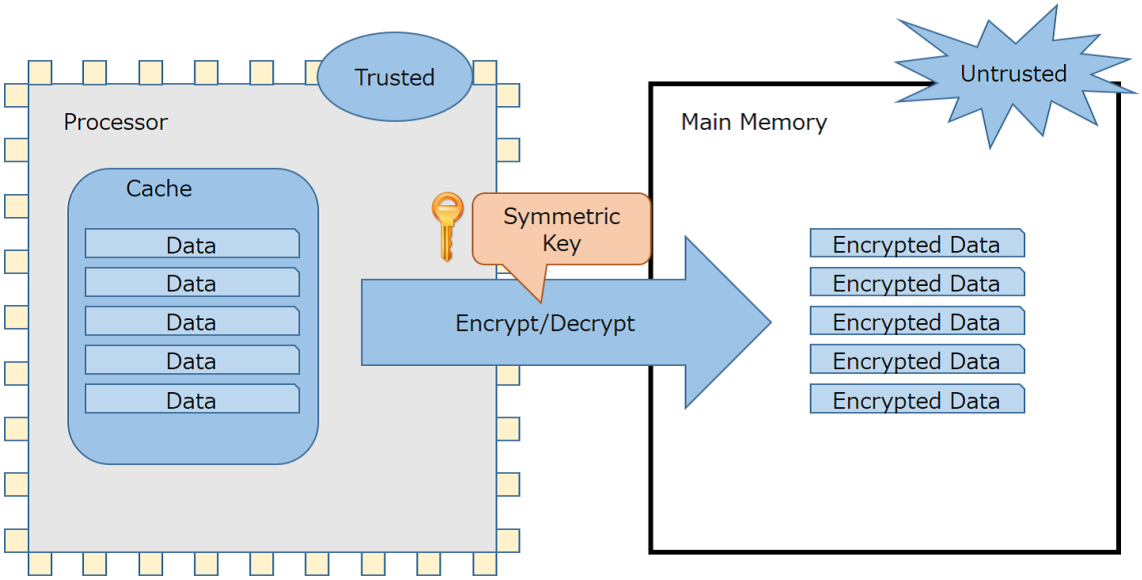


図 5.1: Memory Encryption/Decryption of VM Secure Processor

[4]https://www.ietf.org/rfc/rfc1750.txt
[5]鍵長は変更可能

アクセス制御　プロセッサ内部のキャッシュラインのタグとコンテキスト（レジスタファイルやプログラムカウンタの状態）には VM 固有の ID を付加する．もしハイパーバイザや他の VM からアクセスがあった場合は ID が異なるためアクセスを防止することが出来る．従来のセキュアプロセッサではプロセスの機密情報保護を目的としていたため，レジスタファイルにも識別ビットを付加する必要があった．しかし，VM セキュアプロセッサでは保護対象を VM に拡張しているため，その必要はないことが大きな違いである．

仮想化対応　VM のハードウェア制御を完全仮想化[6]するにはプロセッサによる仮想化支援が必要である．図 5.2 はプロテクションリングと呼ばれ，ハードウェアに対する複数のアクセス権限レベルを階層構造で示したものである．従来の非仮想化環境ではもっとも権限の高いリング 0 は OS のカーネルに相当し，直接ハードウェアにアクセスすることが可能であった．仮想化環境では OS の上位にハイパーバイザが位置するので，完全仮想化を実現するにはプロテクションリングの階層構造を拡張する必要がある．具体的には命令セットに仮想化のための拡張命令を追加し，プロセッサがそれを解釈できるようにする．プロセッサはハイパーバイザと VM の権限を判別することが可能となり，ハイパーバイザのリング階層と VM のリング階層が区別される．ハイパーバイザはリング 0 の条件下でコンテキストスイッチやアドレス変換，メモリマッピングを行い，VM の管理を行うことが出来る．また，この間 OS のカーネルからのハードウェアアクセスは制限される．Intel VT-x や AMD-V ではこのような仮想化命令を追加することで，完全仮想化を実現している．現時点の VM セキュアプロセッサは OpenRISC を基に実装されているため，命令セットアーキテクチャ・マイクロアーキテクチャの双方に修正を加える必要がある．

ハッシュ値演算・管理　VM セキュアプロセッサは内部にハッシュ値演算機構を持つ．ハッシュ値は電子署名やデータの完全性検証に利用する．ハッシュ値生成モジュールは本研究で実装した部分の一部であり，アルゴリズムには SHA-3 を採用した．詳細は第 6.2 に後述する．

5.3　認証プロトコル

5.3.1　VM セキュアプロセッサの遠隔認証

VM セキュアプロセッサはプロセッサを信頼し，プロセッサが稼働しているプラットフォームは信頼しない．プラットフォームが信頼出来ない場合はプラットフォーム上で不正なプロセッサによるなりすましの危険性が想定される．これを防止するために VM セキュアプロセッサは遠隔からプロセッサの真正性を検証する機能を有する．遠隔から VM セキュアプロセッサを認証するために，製造時に埋め込まれたルートキーを利用する．ルートキーは公開鍵と秘密鍵のペアであり，公開鍵とともにプロセッサメーカを認証局とした公開鍵証明書が埋め込まれている．具体的な認証プロトコルを以下に示す．

1. ユーザが VM セキュアプロセッサに対して認証要求メッセージを送る

[6]全てのハードウェアの動作を仮想化し，修正することなく VM に提供できる状態

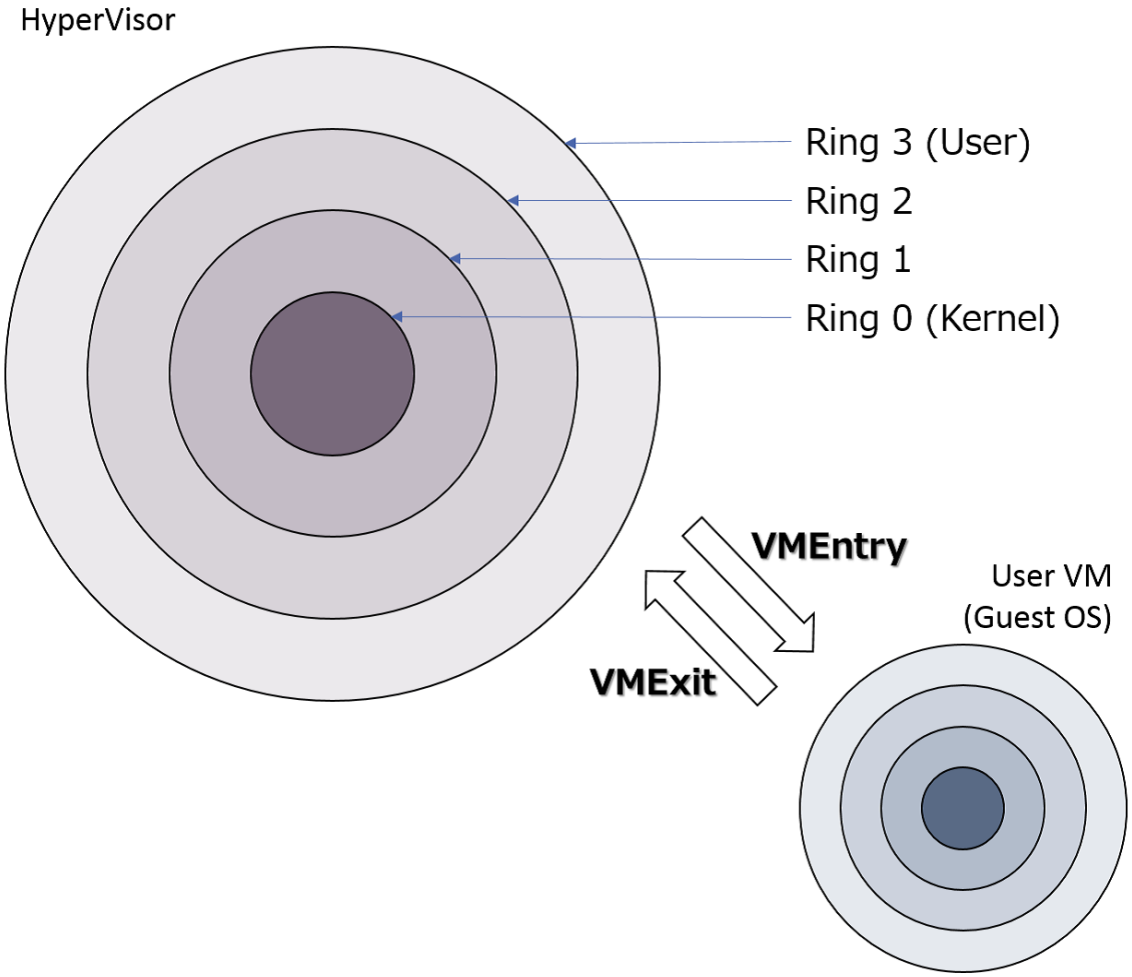


図 5.2: Protection Ring

2. VM セキュアプロセッサはルートキーの秘密鍵を用いて認証要求メッセージの電子署名を生成する
3. VM セキュアプロセッサがルートキーの公開鍵と電子署名をユーザに送信する
4. ユーザは電子署名に使われたルートキーの公開鍵証明書を検証する
5. ルートキーを用いて電子署名を検証する

以上のようなプロトコルによりルートキー公開鍵，電子署名の真正性を検証する．ルートキーの秘密鍵を持つのは正規の VM セキュアプロセッサに限られるため，電子署名を検証することで遠隔からプロセッサの真正性を検証することが可能である．さらにルートキーの真正性も検証することで，中間者攻撃を防止している．

VM セキュアプロセッサの遠隔認証は IaaS で新しく VM を立ち上げた場合や VM を異なるプラットフォームにマイグレーションする場合に行われることを想定している．

5.3.2　アプリケーション認証

VM セキュアプロセッサを用いて VM 内で動作するアプリケーションの完全性・真正性を検証（アプリケーション認証）することが出来る [33]．ここでは前提としてアプリケーションの製作者が認証用の公開鍵ペアと認証サーバを用意しているものとする．認証の概要を図 5.3 に示す．VM セキュアプロセッサはアプリケーション全体のハッシュ値を生成する．アプリケーションには認証サーバの公開鍵が同梱されており，これを用いてハッシュ値を暗号化する．暗号化されたハッシュ値を認証サーバに送信し，復号・ハッシュ値検証を行えばアプリケーションの完全性・真正性を保証することが出来る．

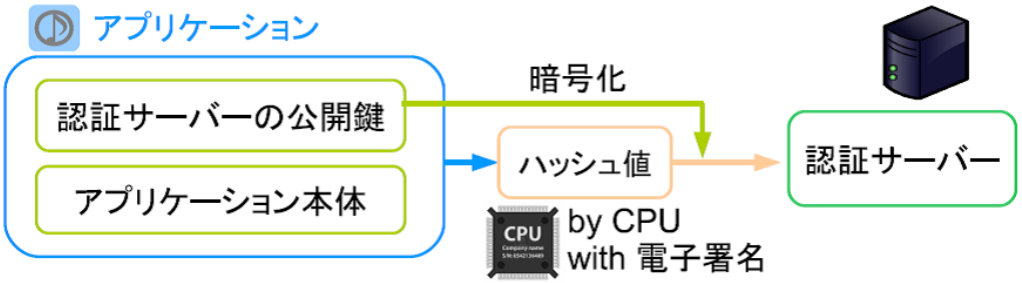


図 5.3: Application Authentication using VM Secure Processor

5.4　セキュアプロセッサとの比較

我々の提案する VM セキュアプロセッサと従来のセキュアプロセッサの比較を表 5.1 に示す．VM セキュアプロセッサと従来のセキュアプロセッサでは保護対象が全く異なることに注意しなければならない．従来のセキュアプロセッサは OS のプロセス管理の根幹部分を改変し，コンテキストスイッチなどに対応しなければならなかった．OS のコードは複雑で膨大なだけでなく，コード自体が公開されていない場合もある．したがって OS を改変するのは困難であり，これが実用上の大きな課題となっていた．一方 VM セキュアプロセッサでは保護対象が VM であるため，OS の内のプロセスを保護対象としない．この場合，VM 内の OS にプロセス管理を任せることで OS 改変は不要となる．ただし，VM が切り替わるときには同じようにメモリやレジスタファイルの状態などをコンテキストスイッチによってストレージに移す必要が生じるので，ハイパーバイザに改変する必要が生じる．

以上のように VM セキュアプロセッサは仮想化に対応しているのが大きな特徴であるが，セキュアプロセッサも仮想化に全く対応していないわけではない．例えばホスト OS 型の仮想化ソフトウェアを稼働させた場合，VM はプロセスとして扱われるため仮想化に対応する可能性はある．しかし，

表 5.1:

	セキュアプロセッサ	VM セキュアプロセッサ
保護対象	プロセス	VM
OS 改変	必要	不要
仮想化対応	△	○

ホスト OS 型仮想化はホスト OS の機能に依存してハードウェアを仮想化するため，改変された OS に対して改めて仮想化ソフトを作りなおす必要が生じる．さらに，ホスト OS 型の仮想化とセキュアプロセッサの組み合わせはオーバーヘッドが大きく，実用的に見て非現実的であると言える．

5.5 本研究での提案手法

本研究ではクラウドフォレンジックの課題であったログファイルの機密性・完全性・真正性の確保と調査の効率化の問題を VM セキュアプロセッサの導入により解決する．VM セキュアプロセッサは開発中であり，提案手法に必要なハッシュ値演算・管理を行う機構（ハッシュ値生成モジュール）と公開鍵によって署名を生成する機構（電子署名生成モジュール）が未実装である．したがって，本研究では提案手法を実現するためにこの 2 つの実装に取り組んだ．

5.5.1 提案手法の概要

図 5.4 に提案手法の概要図を示す．

クラウドサービスではユーザ VM のゲスト OS として Linux が利用されることが多いため，syslogd が OS カーネルの一部として組み込まれていると仮定する．syslogd は各アプリケーションやゲスト OS からのログメッセージを収集しており，ログが蓄積したら syslog サーバへ転送する．VM セキュアプロセッサは VM ごとに固有の共通鍵を持っており，他にも認証に用いる公開鍵などを保持している．

5.5.2 具体的な手順

syslogd 認証

提案手法ではゲスト OS 内の syslogd を VM セキュアプロセッサによって認証する．これには山田らの提案した，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 [33] を用いる．この手法ではプロセッサメーカの認証サーバと VM セキュアプロセッサが通信することで，クラウドプロバイダやハイパーバイザが信頼出来ない場合でもアプリケーションの完全性を検証することが可能である．

提案手法では syslogd を対象に山田らの手法を用いることで，syslogd の完全性を検証する．認証は VM の初回起動時に行い，その後再起動するたびに認証を繰り返す．

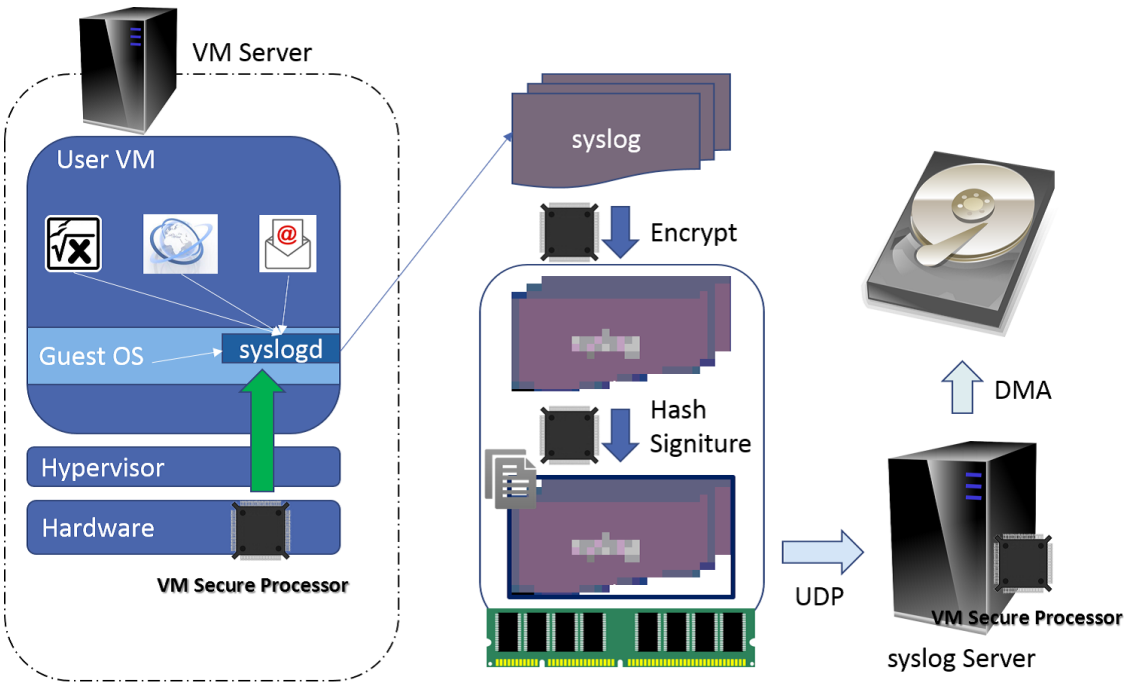


図 5.4: Overview of Proposed Technique

各コンポーネントの相互認証

VM セキュアプロセッサの遠隔認証の機能を用いて，各コンポーネントと syslog サーバ間で相互に認証を行う．これによって，各コンポーネントで VM セキュアプロセッサが稼動していることが保証され，外部の攻撃者による中間者攻撃やなりすましといった不正アクセスを防止することが可能である．

VM セキュアプロセッサによるログ暗号化・署名

syslog は VM セキュアプロセッサ内部のキャッシュ上では平文であり，メモリに書き出す際に VM 固有の共通鍵で暗号化される．syslogd はメモリ上の syslog データに対して署名をつけるように VM セキュアプロセッサへ要求し，VM セキュアプロセッサは暗号化された syslog に対してデジタル署名をつける．

syslog に対して暗号化を行うことで VMI やハイパーバイザの特権による不正アクセスによる情報漏洩を防止することができる．さらに，メモリやデータバスを直接解析するようなハードウェアタンパからも機密情報を保護する．syslog に対してデジタル署名をつけることでログの改ざんを防ぐと同時に，syslogd が起動時からログ生成時まで同一プラットフォーム上で動作していることも保証される．これにより，syslogd を改ざんしてログを隠ぺいするような攻撃も VM セキュアプロセッサ側で検出することが可能である．

ログの生成順序の改ざんに対しては SecLaaS と同様に LogChain を計測していくことで対策する．さらに，VM セキュアプロセッサはログファイルに「ログ署名時のタイムスタンプ」と「ログを生成した syslogd の情報」を平文の証明書として添付する．添付された証明書はストレージ上で VM を識別するために利用する．この 2 つの情報はログファイルの管理に必要なため平文で添付するが，ログファイル自体は暗号化されているためユーザ VM だけが復号することが出来る．

syslog サーバへの転送

ログファイルがある程度蓄積したら，VMServer をはじめとする各サーバから syslog サーバへログファイルを転送する．この際通信路は暗号化されているため機密性は保たれる．また，各コンポーネントでは相互認証によって syslog サーバでも VM セキュアプロセッサが稼動していることが保証される．

DMA によるストレージへの書き出し

syslog サーバに転送されたログファイルは VM セキュアプロセッサが DMA に転送命令を出して，強制的にストレージへ書き出す．ログファイルに対する転送命令は VM セキュアプロセッサのみ許可するものとする．また，ストレージへの書き出しは必ず DMA 経由で行い，例外は認めないようにする．ストレージ上のデータはフォレンジック調査機関から管理できるように，平文の VM の識別 ID とタイムスタンプが添付される．

ストレージ内でのログ管理

ストレージ内では，書き出されるログに添付されている証明書の情報を元にログを整理することが可能である．署名時のタイムスタンプからは**ログの保全日時**が得られ，syslogd の情報からは**ユーザ VM** を特定することができる．これをもとにデータを検索することで，フォレンジック調査機関は「ユーザ」と「日時」を指定して効率よくログを調べることが可能である．ストレージ上のログファイル本体は暗号化されているため，フォレンジック調査機関はユーザへ依頼してログの内容は VM セキュアプロセッサに復号させる必要がある．

以上の手順では暗号化されたログファイルの内容を復元できるのは正規のユーザ VM だけであり，クラウドのように Resource Pooling や外部からの不正アクセス，オペレータの不正があった場合でもログファイルの機密性は保たれる．さらに提案手法ではログ生成からストレージへの転送までを VM セキュアプロセッサがサポートするため，いつでもログファイルの完全性・真正性を確認することが可能であり，証拠保全の一貫性が保たれているといえる．

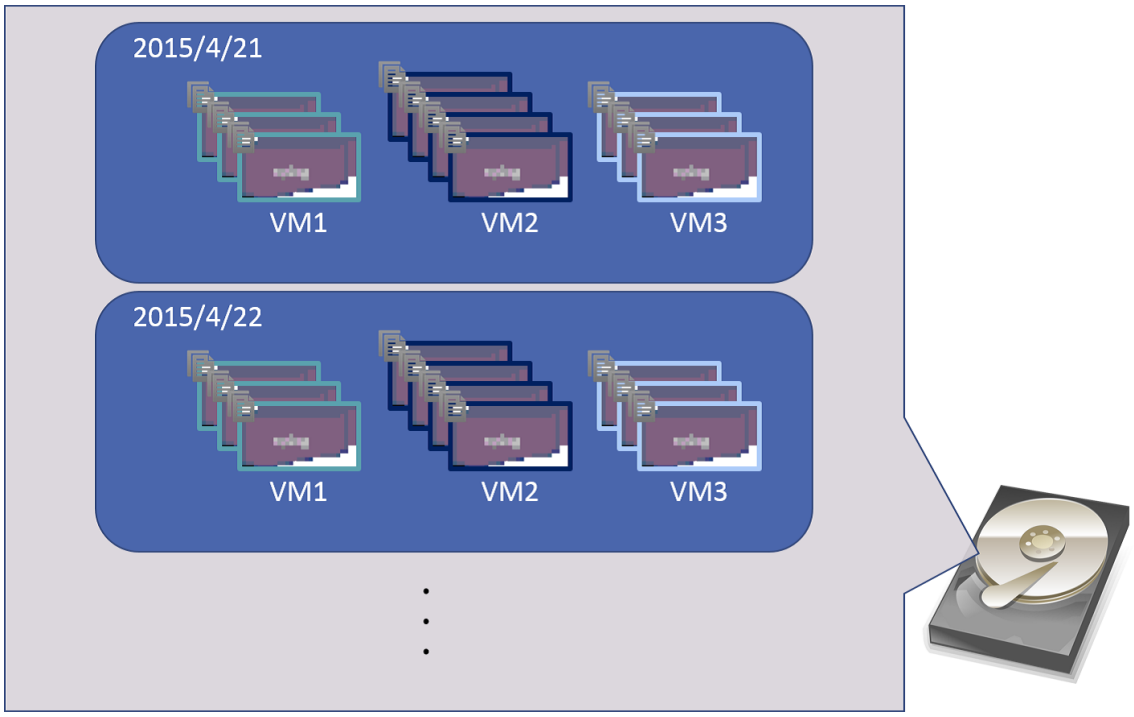


図 5.5: Log Management in External Storage

第6章　設計と実装

6.1　設計

6.1.1　VM セキュアプロセッサの設計

VM セキュアプロセッサは OpenRISC プロジェクト [1] によるプロセッサコア「OpenRISC1200」をベースに実装されている．OpenRISC プロジェクトの目的はフリーでオープンソースのコンピューティング基盤を作ることであり，その一環として RISC-CPU アーキテクチャの開発を行っている．OpenRISC による命令セットアーキテクチャは OpenRISC1000 アーキテクチャと呼ばれ，それをもとに Verilog ハードウェア記述言語で実装された CPU が OpenRISC1200 である．図 6.1 は VM セキュアプロセッサの構造の概要図である．VM セキュアプロセッサではメモリ暗号化に関して Encryption Unit・Key Cache 部分が追加され，アクセス制御に関して MMU と TLB が拡張される．それ以外は OpenRISC1200 を用いたものである．現時点ではメモリ暗号化モジュール（Encryption Unit）は実装されており [34]，電子署名生成モジュール（Signature Unit）はモジュールとして完成している．Key Cache に関しては今後実装予定である．本研究ではハッシュ値生成モジュール（Hash Unit）の実装を行うとともに，Signature Unit を VM セキュアプロセッサに組み込み電子署名を生成できるように実装した．

6.1.2　電子署名生成に必要なモジュール

ログファイルに対する電子署名生成のために，ハッシュ値生成モジュールと電子署名生成モジュールの組み込みが必要である．本研究ではハッシュアルゴリズムには暗号学的強度とハードウェア実装コストを考慮し，SHA-3（Keccak）を採用した．電子署名生成モジュールでは公開鍵暗号として楕円曲線暗号を用いた ECDSA を採用している．以下，SHA-3 と ECDSA について述べる．

Keccak　Keccak は G. Bertoni らによって設計された暗号学的ハッシュ関数であり，NIST によって公募された次世代ハッシュ関数 SHA-3 に採用されたものである．Keccak はスポンジ構造という内部構造を持つのが特徴で，スポンジ構造は入力によってその状態を変化させ，ハッシュ値を出力する．Keccak は 224bit,256bit,384bit,512bit の四通りの出力ハッシュ値のサイズを設定することが可能であり，出力ハッシュ長を用いて Keccak-512 などと呼ばれる．出力ハッシュ長によって設定するパラメータが異なるが，本研究では Keccak-512 を想定して実装しているため，以下 Keccak-512 のパラメータを用いて説明する．Keccak は「パディング処理 (padding)」と「ブロック置換処理（f-permutation）」の二つによって構成される．入力データは一定の比率で分割されてハッシュ関数の入力とされる．本

[1] http://openrisc.io/

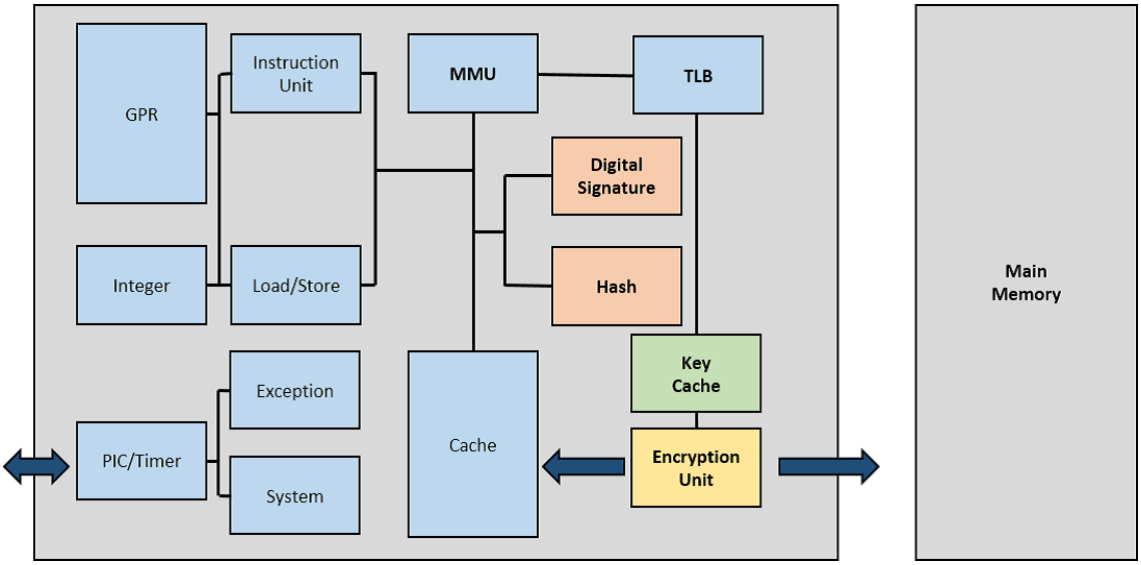


図 6.1: Overview of VM Secure Processor

研究での実装では 32bit ずつに分割されている．Keccak-512 ではブロック置換の前に入力データを 576bit に拡張する必要があるため，パディング処理によって 576bit に拡張する．パディング処理後にブロック置換処理が行われる．ブロック置換では図 6.2 で表されるブロック (state) に対して θ ρ χ π ι の 5 つの処理を繰り返す．NIST による標準規格 FIPS202 の公式文書 [35] では図 6.3，6.4，6.5，6.6 を用いて説明されている． ι は図説されていないが，state と定数の XOR 演算を行っている．ブロックの大きさは

$$(row, column, lane) = (5, 5, 2^l)$$

によって定義される．row と column は固定値であるのに対し，lane は可変値である．Keccak は lane の大きさを変えることでブロック置換処理を軽量化することができる．lane の長さを CPU の l ワードに対応させることでブロックへのマッピング処理の負担が軽減されるため，64bit 環境では l=6，32bit 環境では l=5 のように対応する．Keccak のソフトウェア実装では実行環境を想定して，コーディングする必要がある．

SHA-3　2000 年代には MD5 や SHA-1 といったハッシュ関数が一般的に用いられていたが，その攻撃方法の理論的確立や実証が続々と発表され，そのような既存のハッシュ関数とは構造的に異なるハッシュ関数の設計が急務となった．そこで NIST が次世代暗号学的ハッシュ関数（SHA-3）として新たなるハッシュ関数を一般から公募し，従来よりも強固で高速な次世代ハッシュ関数を求めた．2012 年 10 月 2 日 NIST によって行われたコンペティションを勝ち抜いて Keccak は SHA-3 として選定され，2015 年 8 月 5 日には FIPS202 [2] の公表によって正式に SHA-3 として採用された．コンペティションには Keccak の他にも BLAKE [3] や Skein [4] といった強固なハッシュ関数が最終候補として残っていた

[2]米国 NIST が発行している標準規格．Federal Information Processing Standard
[3]https://131002.net/blake/
[4]http://www.skein-hash.info/

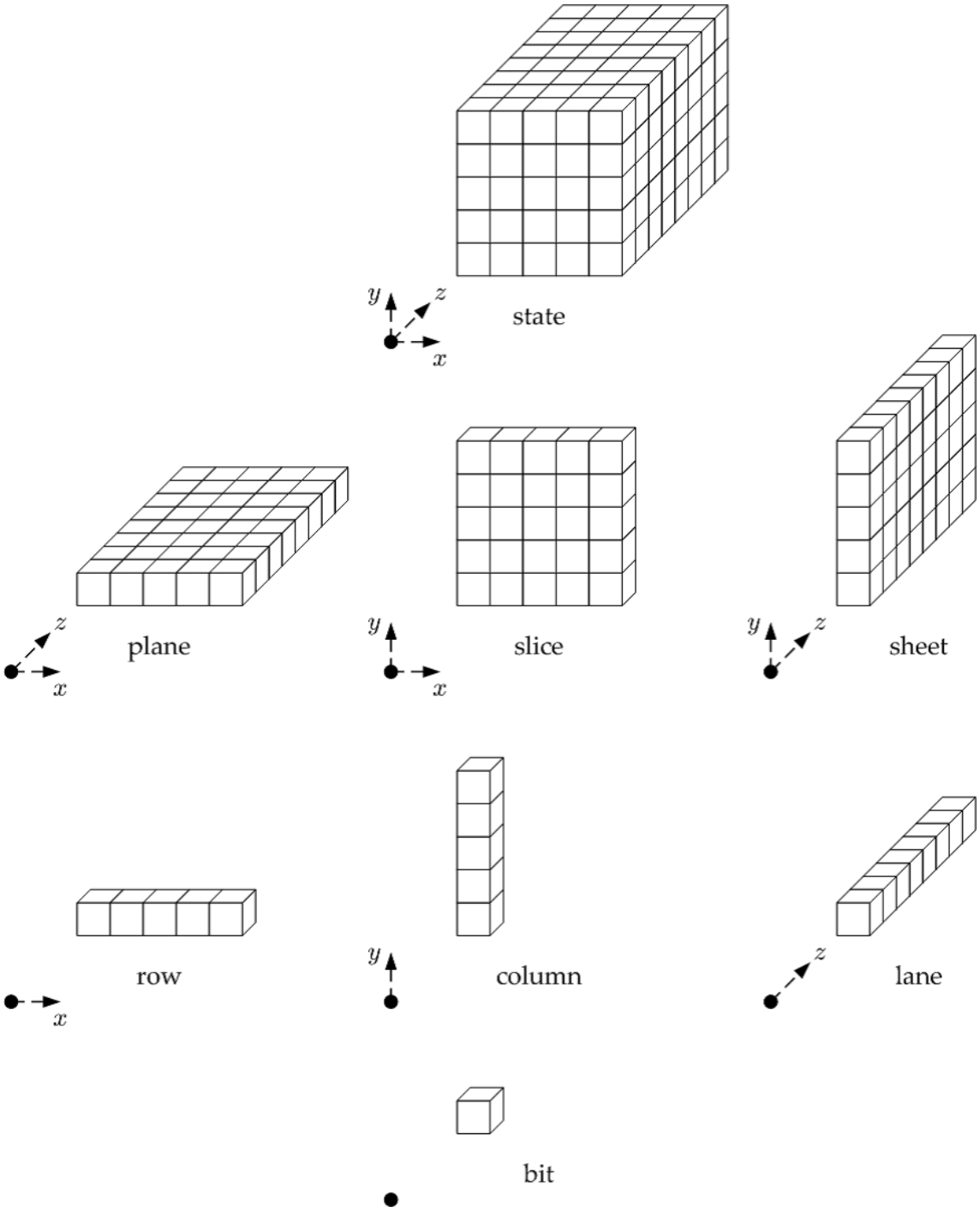


図 6.2: Keccak-f-Pieces Of State

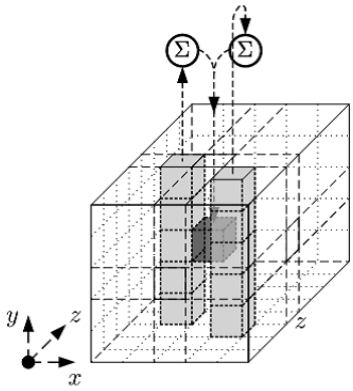


図 6.3: Step θ

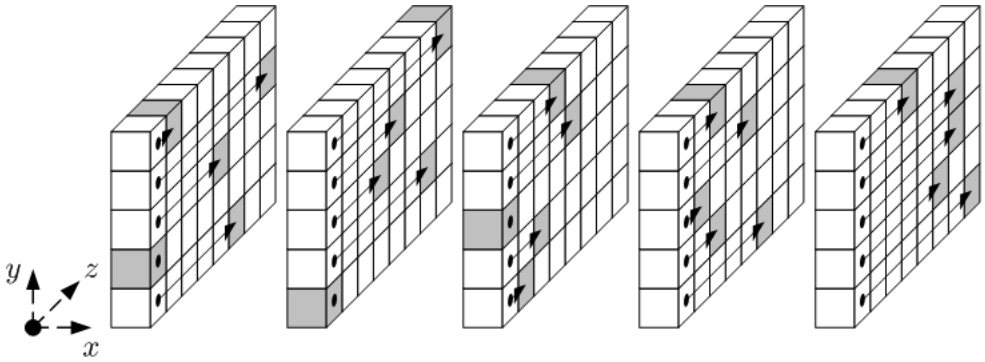


図 6.4: Step ρ

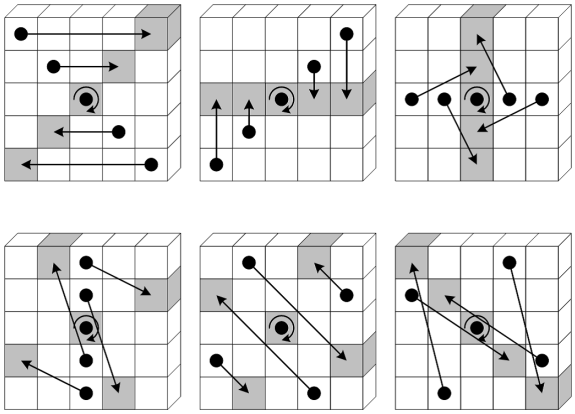


図 6.5: Step π

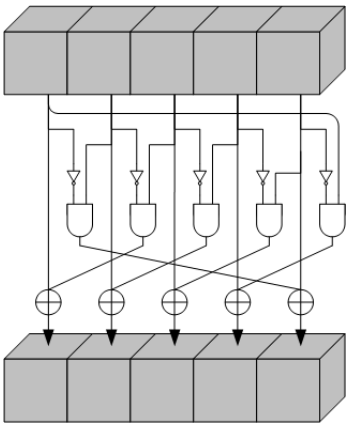


図 6.6: Step χ

が，Keccak は暗号学的な強度だけでなくその処理速度やハードウェア実装コストの低さなどが評価された．そのような多くのメリットの根本にあるのは，スポンジ構造という単純かつ柔軟なアイディアである．スポンジ構造を応用することで Keccak を暗号化関数として実装したもの [5] や木構造を利用したハッシュの高速演算なども存在する．このように，Keccak はハッシュ関数にとどまることなく今後とも更なる普及が見込まれる．

ECDSA　楕円曲線暗号は楕円曲線上の離散対数問題を利用した暗号方式の総称であり，これを利用した電子署名法式を ECDSA（Elliptic Curve Digital Signature Algorithm）と呼ぶ．ECDSA は FIPS[6] によって標準規格として認められており，さまざまなプログラミング言語でライブラリとして利用することが出来る．楕円曲線暗号は他の公開鍵暗号よりも短い鍵長で同程度の安全性を確保することが出来るのが特徴であり，パラメータ設定によってハードウェア実装を簡略化できるため，計算リソースの限られた IC カードや組み込み機器を中心に利用されている．

6.1.3　電子署名生成手順

VM セキュアプロセッサによって電子署名が生成される手順を図 6.7 に示す．入力データは具体的にログファイルのことを示す．ログファイルの大きさは任意（N ビット）とする．ハッシュ値生成モジュールには入力データ N ビットを 32 ビットずつ分割して入力していく．したがって約（N/32）回の作業を繰り返す必要がある．最後のデータが入力されてパディングが終了してから 23 サイクル後にハッシュ値（512 ビット）が得られる．これを電子署名生成モジュールに入力し，約 42100 サイクル後に 233 ビットの値のペア (r,s) が得られる．これが得られる電子署名の値である．

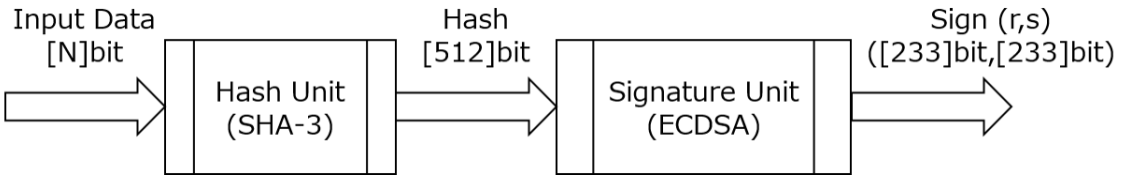


図 6.7: Procedure of Generating Electronic Signature

6.1.4　電子署名要求のための拡張命令

VM セキュアプロセッサに電子署名作成を支持するために，電子署名要求のための拡張命令を用意する．OpenRISC の命令セットである OR1000 には l.cust1 から l.cust8 までの 8 つの拡張命令が用意されている．OR1200 の実装上では既に使用例として l.cust5 が用意されていたため，これを書き換えることで電子署名の要求命令を実装する．l.cust5 のフォーマットを図 6.8 に示す．

D は write のターゲットレジスタ rD，A・B は read のソースレジスタ rA・rB を表し，L は即値（cust5_limm），K は l.cust5 固有のオペコード（cust5_op）を表す．cust5_op の値を変えることで細か

[5] http://ketje.noekeon.org/
[6] http://csrc.nist.gov/publications/fips/fips186-3/fips_186-3.pdf

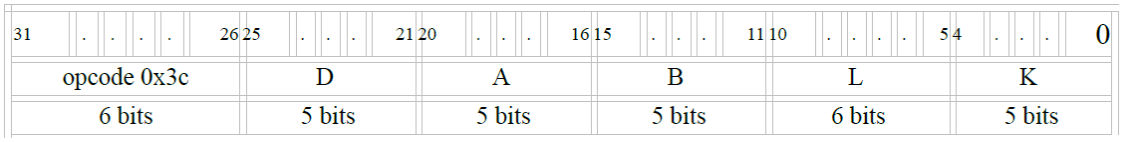


図 6.8: Instruction Format of l.cust5

い動作拡張が可能であり，本研究でもこれを応用する．cust5_op によって指定する動作モードを以下に示す．

- START
- PROGRESS
- FINISH
- HASH-STORE
- SIGN-STORE

START モードはハッシュ値生成モジュールに入力ファイルを送り込むためのモードである．ここでハッシュ値生成モジュールにリセット信号が送られ，最初の 32bit が入力される．PROGRESS モードは入力されている値が途中であることを HashUnit に伝えるモードで，FINISH モードによって入力が完了したことをハッシュ値生成モジュールに伝える．その後，HASH-STORE モードによってハッシュ値が，SIGN-STORE モードによって電子署名の値が DMA 経由で外部ストレージに書き出される．

6.1.5 電子署名要求のためのプログラム

VM セキュアプロセッサへ l.cust5 を発行するためのプログラムを用意する．このプログラムは syslogd と syslog サーバに組み込まれ，VM セキュアプロセッサの外部アプリケーション認証機能によって認証を行う．このプログラムから l.cust5 の動作モードを操作し，VM セキュアプロセッサ内の処理を制御する．提案手法ではこのプログラムがログファイルに電子署名とタイムスタンプを付けて，VM セキュアプロセッサが透過的にログファイルを暗号化してから DMA によって自動的に外部ストレージへ書き出す．

6.2 実装

本研究で実装したものを以下に示す．

- ハッシュ値生成モジュール（SHA-3）
- l.cust5 命令発行プログラム

ハッシュ値生成モジュールは OpenRISC の CPU コア内部でハッシュ値を算出するモジュールである．ハッシュアルゴリズムは SHA-3 であり，512bit のハッシュ値を生成する．l.cust5 命令発行プログラムは VM セキュアプロセッサに電子署名を生成するように要求するための命令を発行するプログラムである．

6.2.1 開発環境

本研究で使用したソフトウェアは表 6.1 のとおりである．本研究では OpenRISC を動作させる環境として Digilent 社の提供する FPGA 評価ボード「Atlys」を用いた．Atlys の基本性能を表 6.1 に示す．Atlys は FPGA として Xilinx 社の Spartan-6 を搭載しており，表 6.3 は回路面積に関するリソースである．ソフトウェアのコンパイル環境としては OpenRISC プロセッサの開発用に提供されている Ubuntu の VirtualBox 用の VM イメージ [7] を利用した．この VM にはコンパイラやシミュレータ，基本的なドキュメントなどが含まれており，開発者は VM を起動するだけで OpenRISC の開発が可能である．本研究でも Linux や l.cust5 命令発行プログラムのコンパイルのために利用した．

表 6.1: 開発に使用したソフトウェア

用途	ソフトウェア
HDL シミュレーション	Modelsim
論理合成・配置配線	Xilinx ISE Design Suite 14.7
コンパイル環境	OpenRISC_Ubuntu_2011-11-28
コンパイラ	or32-linux-gcc
評価ボード接続	Digilent Adept software

表 6.2: FPGA 評価ボード Atlys の基本性能

FPGA	Xilinx Spartan-6 LX45 FPGA, 324-pin BGA package
Memory	128Mbyte DDR2 with 16-bit wide data
IO	USB-UART and USB-HID port etc.
clock	100MHz CMOS oscillator

6.2.2 ハッシュ値生成モジュールの実装

SHA-3 単体の verilogHDL 実装は Opencores によって Keccak プロジェクトとして配布されている [8] ため，本研究ではそれを VM セキュアプロセッサに組み込んだ．VM セキュアプロセッサは OR1200 を

[7] http://opencores.org/or1k/Ubuntu_VirtualBox-image_updates_and_information
[8] http://opencores.org/project,sha3

表 6.3: Spartan-6 XC6SLX45 の基本性能

Slices	6,822
Logic Cells	43,661
CLB Flip-Flops	54,576

ベースとしており，５段の整数型パイプライン処理が可能である．図 6.9 にその概要図を示す．l.cust5 は IF ステージでメモリから読み込まれ，ID ステージで図 6.8 のフォーマットのようにオペコードやレジスタアドレスなどが解釈される．RR ステージではレジスタファイルから指定されたアドレスの値を読み出す．EX ステージの動作は cust5_op によって異なる．cust5_op で指定する５つのモードのうち「START」「PROGRESS」「FINISH」モードでは，EX ステージで SHA-3 の本体である Keccak モジュールに 32bit の入力を送り込み，WB ステージでは何も行わない.「HASH_STORE」モードでは EX ステージで任意のタイミングでハッシュ値を選択し，WB ステージで指定されたレジスタに結果を書き込む．ハッシュ値生成では EX ステージにおいて以下の三つのモジュールが動作する．

keccak　keccak はハッシュ値演算を行う本体モジュールであり，32bit の入力データを繰り返し受け付けたあと，23 40 サイクルで 512bit のハッシュ値を出力する．下位モジュールとして padder と f_permutation を含み，それぞれ「パディング処理」と「ブロック置換」を行う．実装では動作周波数が低い FPGA 用の実装を用いているが，より高い動作周波数用の実装も配布されている．

keccak_ctrl　keccak_ctrl は ID ステージで命令を解釈したあと，l.cust5 のモードによって keccak モジュールへの入力を制御するモジュールである．このモジュールを用意することで keccak 本体を出来るだけ改変せずに VM セキュアプロセッサへ導入することが出来た．最後の入力が終わり，ハッシュ値の生成が完了したら，keccak_devide に 512bit のハッシュ値を入力している．

keccak_devide　SHA-3 では 512bit のハッシュ値が生成されるが，l.cust5 で扱うために 32bit ずつに分割しなければならない．本研究では 512bit を 32bit × 16 で構成されるハッシュ値専用のレジスタファイルを用意し，l.cust5 の cust5_limm で番号を指定することによってメモリへストアできるように実装した．ハッシュ値を連続で生成する場合，keccak_devide は keccak 本体のハッシュ演算が終了するまで値を保持し続けるため，ストアの終了を待つことなく次回の入力を行うことが出来る．

上記の三つのモジュール以外にも or1200_alu や or1200_ctrl，or1200_wbmux などといった OR1200 の既存のモジュールを改変し，keccak モジュールに対応させている．

6.2.3　l.cust5 命令発行プログラムの実装

l.cust5 命令発行プログラムは Ubuntu VM 上で C 言語を用いて実装した．C 言語ではプログラム中にインラインアセンブラを用いてアセンブリをそのまま記述することが可能であり，プログラムの拡張やデバッグを柔軟に行うことが出来た．実装では l.cust5 命令によって，設計した５つのモードを

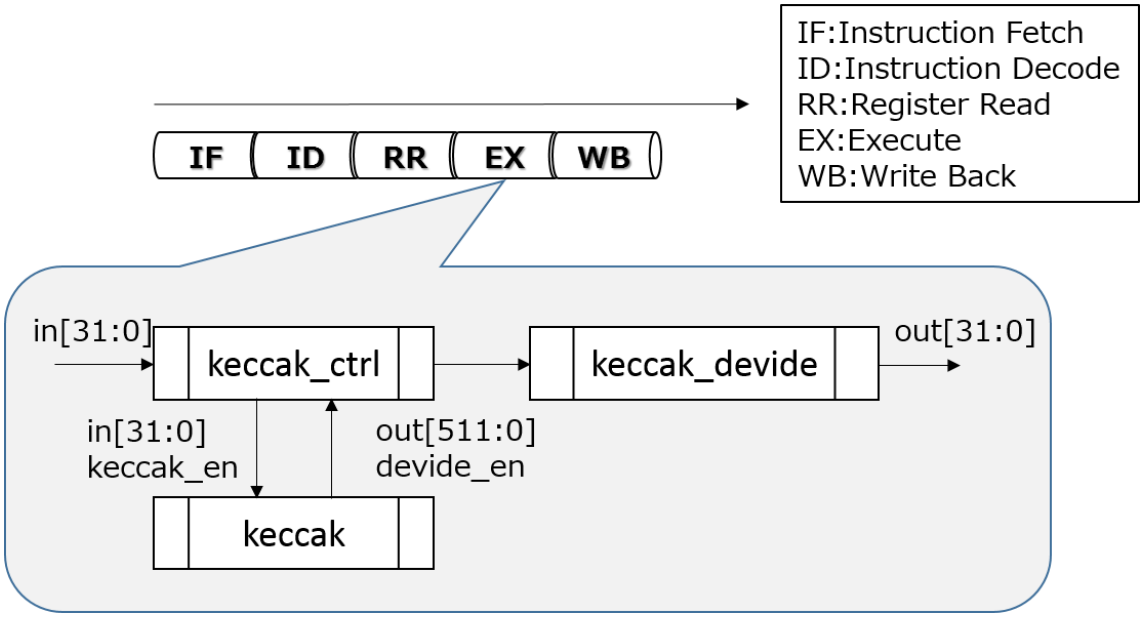


図 6.9: Pipeline Processing in OpenRISC1200

指定することが可能であるが，タイムスタンプや DMA との連携は未実装である．具体例として実装したプログラムでインラインアセンブラが利用されているコードの一部分を以下に示す．

```
__asm__(
        "l.lwz \%0,0(\%1)\n\t"//LOAD FROM input_addr to tmp
        "l.cust5 \%2,\%2,\%2,0,0\n\t"   //reset
        "l.cust5 \%2,\%0,\%3,0,4\n\t"   //start
        :"=r"(tmp)
        :"r"(input_addr),"r"(dummy1),"r"(dummy2)
        :
              ); printf("START:\%d\t \%08lx\n",i,tmp);
```

コードの 4 行目と 5 行目では%で指定するレジスタと C のプログラム内の変数の対応を記述している．このようにインラインアセンブラを用いれば，C の変数とアセンブリのレジスタを直接結びつけることが可能であり，今後の拡張にも有用である．

tftp による通信　Atlys ボードの FPGA 上で動作する Linux へクロスコンパイルした任意のプログラムを転送するために tftp（Trivial File Transfer Protocol）を利用した．tftp は UDP を用いて特定のコンピュータ間でファイルを転送するためのプロトコルであり，認証機能が無く，軽量なのが特徴である．tftp クライアントプログラムは本研究で利用した Linux に組み込まれていたため，tftp サーバを立てるだけで通信することが出来た．本研究でクロスコンパイルしたカスタム命令発行プログラムは tftp によって転送している．

第7章　評価

7.1　性能評価

7.1.1　ハッシュ値生成時間

本研究の既存研究として TPM を用いた手法が挙げられるが，提案手法では TPM と異なりプロセッサの内部でハッシュ値を生成するため，TPM との通信や TPM 特有の API を経由する必要がない．したがって，TPM よりも効率よくハッシュ値を生成することが可能であると考えられる．ここではハッシュ値の生成時間を評価することによって，TPM と VM セキュアプロセッサのどちらが効率よくハッシュ値を生成できるか比較する．

評価方法　ハッシュ値の生成時間を計測するためにカスタム命令発行プログラムのインタインアセンブラの処理にかかる時間を 10 回計測し，その平均値を算出する．インラインアセンブラの処理には l.cust5 の START モードから HASH-STORE モードが終了するまでとし，l.cust5 命令だけでなく入力データやハッシュ値のロードストア命令も含まれている．入力データは 1KByte から 10MByte までを計測する．実際にログファイルのハッシュ値を取ることを考えると 100MByte まで取るべきであるが，開発環境の制限により 100MByte 以上は計測が困難であったため 10MByte を上限とする．

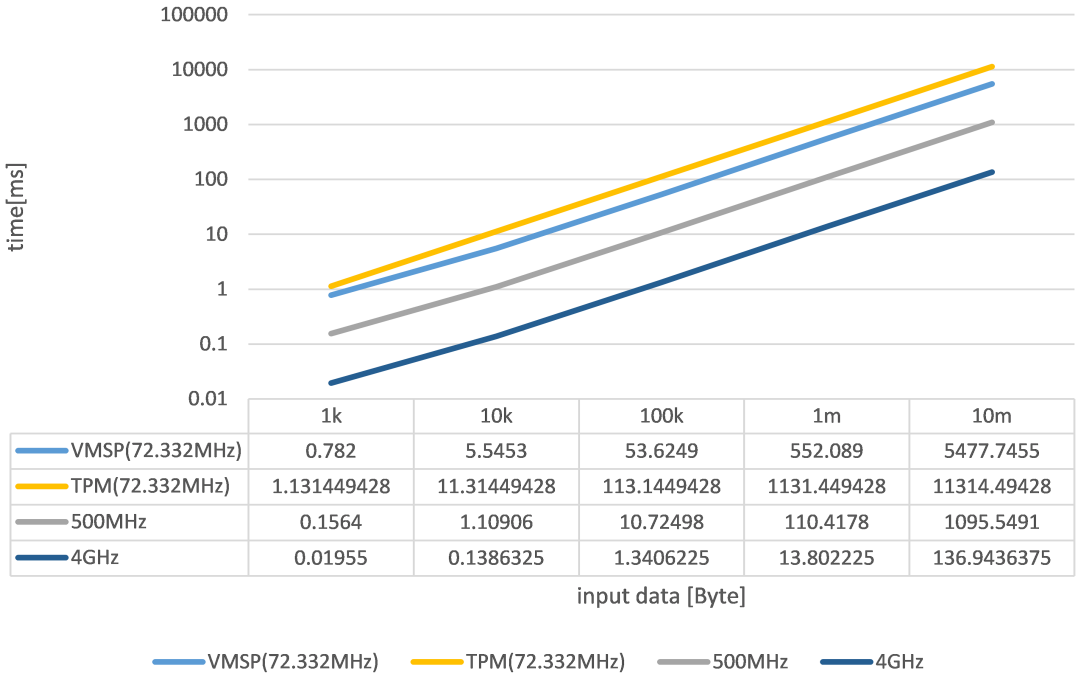
TPM の評価は今後行う予定であるため，本論文では STMicroelectronics 社の製品 [1] の数値を元に評価を行った．実装した VM セキュアプロセッサの最大動作周波数は 72.332MHz であったため，TPM でハッシュを生成した場合同じ 72.332MHz ではどのくらいの時間がかかるか推定した．ただし，TPM は現時点で SHA-3 に対応していないため，推測した生成時間は SHA-1 の場合である．VM セキュアプロセッサと同様に 1KByte から 10MByte までの生成時間を推測した．さらに，本研究で使用した Spartan-6 は最大 500MHz の動作周波数に対応する．プロセッサのロジックやタイミング制約を改善することで動作周波数が向上すると考えられるため，500MHz でハッシュ値を生成した場合の推測値も比較対象とする．

評価結果　実験によって得られた結果を図 7.1 に示す．縦軸は生成時間で横軸は入力データのサイズである．また，500MHz と TPM（72.332MHz）の値は推測値であることに注意しなければならない．動作周波数が 72.332MHz の場合は，ほぼ同程度の時間でハッシュ値が生成されることがわかる．グラフを見ると，FPGA の動作周波数を 500MHz まで高めれば，VM セキュアプロセッサのほうが高速にハッシュ値を算出できることがわかる．また，VM セキュアプロセッサは将来的に FPGA ではなくサーバ用途の CPU として動作することを想定している．実際に広く利用されているサーバ用途

[1] http://www.st.com/st-web-ui/static/active/en/resource/technical/document/data_brief/DM00037936.pdf

の CPU は 2GHz から 4GHz [2] のものが多く，基板上の発振器よりも高い周波数で動作する．それに対して TPM は外部モジュールであり，現時点では発振器の周波数に依存する．

以上のようにプロセッサの動作周波数をさらに上げれば，より速くハッシュ値を生成することも可能であるため，効率性では VM セキュアプロセッサが優位であるといえる．



	1k	10k	100k	1m	10m
VMSP(72.332MHz)	0.782	5.5453	53.6249	552.089	5477.7455
TPM(72.332MHz)	1.131449428	11.31449428	113.1449428	1131.449428	11314.49428
500MHz	0.1564	1.10906	10.72498	110.4178	1095.5491
4GHz	0.01955	0.1386325	1.3406225	13.802225	136.9436375

図 7.1: The Required Time for Hash Value Calculation

7.1.2 回路面積

本研究ではハードウェア実装コストの低いハッシュ関数として SHA-3（Keccak）を採用している．そこでハードウェア実装コストを評価するために回路面積の比較を行った．

評価方法　FPGA は論理ブロックの配列やブロック RAM，配線用チャネルなどで構成されている．論理ブロックは VerilogHDL や VHDL といったハードウェア記述言語によって再構成可能であり，Xilinx 社では CLB（コンフィギャラブル・ロジック・ブロック）と呼ばれている．今回用いた Xilinx 社の FPGA「Spartan 6」では，CLB は 4 つのスライス（LUT や FF で構成されるブロック単位）によって成り立つ．スライスとは FPGA の最小構成単位であり，大きくロジック部分と記憶部分に分けられる．ロジック部分は LUT で構成され，記憶部分はレジスタで構成される．したがって，実装した

[2] http://ark.intel.com/ja/products/family/78585/Intel-Xeon-Processor-E7-v3-Family#@Server

回路のスライスの LUT とレジスタの使用率を求めれば，FPGA のおおよその回路面積を評価することが出来る．

本論文の回路面積評価では以下のパラメータを用いた．

- Number of Slice LUTs
- Number of Slice Registers
- Number of Occpupied Slices

数値は ISE の論理合成ログから得ることが出来る．スライスは LUT・レジスタどちらも使う場合だけでなく，LUT 部分だけ用いる場合やレジスタ部分だけ用いる場合もある．したがって,（Number of Slice LUTs + Number of Slice Registers）ではなく，Number of Occupied Slices が全体として利用したスライスの数となる．

また，比較対象は以下のとおりである．

- Bace
- +AES
- +SHA-3
- +(AES,SHA-3)
- +(SHA-3,ECDSA)

「Bace」は OR1200 のアーキテクチャのみを論理合成・配置配線した場合である.「+AES」は OR1200 に AES によるメモリ暗号化機能を追加した場合であり,「+SHA-3」は OR1200 に今回実装した SHA-3 モジュールを追加実装した場合である.「+(AES,SHA-3)」はその双方を同時に論理合成・配置配線した場合である.「（＋ SHA-3，ECDSA）」は SHA-3 と ECDSA を連携させた実装である．

7.1.3 評価結果

評価結果を図 7.2 に示す．グラフでは各パラメータごとの全体に対する面積使用率を示している．グラフを見ると SHA-3 の使用したスライスに関してハードウェア実装コストが 10%程度であることを示している．また,「+AES」「+(AES,SHA3)」の LUT 使用率は 10%程度差があるのに対して，全体として合成したときのスライス使用率はほぼ同程度となっている．AES と SHA-3 を同時に合成することで，AES のスライス中の利用していない LUTs 部分とレジスタ部分を SHA-3 で埋めることが可能となり，全体として面積効率が向上したことがわかる．一方で,「+(SHA-3,ECDSA)」ではほぼ全てのスライスを使用していることがわかる．電子署名を生成するためには SHA-3 と ECDSA が必要であるが，現在の開発環境では AES も含めて論理合成することは出来ないことが確認された．

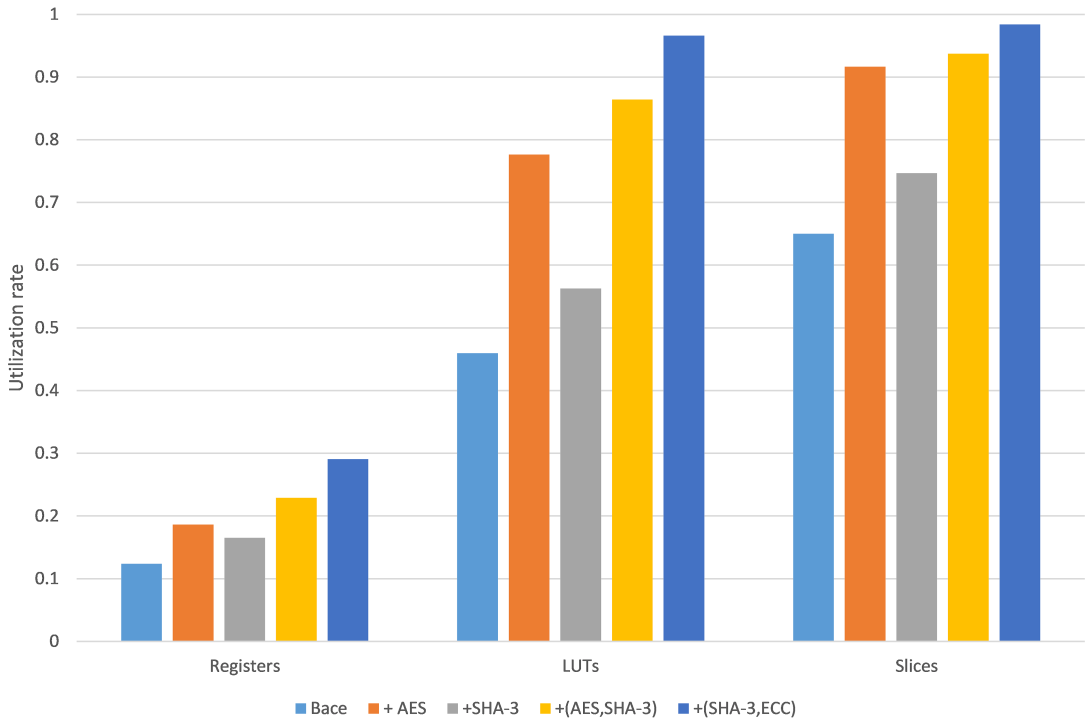


図 7.2: Evalation of Circuit Area

7.2 セキュリティ評価

提案手法は保護対象であるログファイルの機密性・完全性・真正性をハードウェアによって確保しているため，ソフトウェアよりも各脅威に対して強固であると考えられる．ここでは各脅威に対して定性的な評価を行う．

7.2.1 比較対象について

第 4 章で述べた既存手法のうち以下のものを比較対象とする．

- VM 調査・解析専用ソフトウェア
- VMI
- SecLaaS
- SecLaaS+TPM

VM 調査・解析専用ソフトウェアと VMI はログファイルの機密性に関して考慮していないため，ログファイルの完全性・真正性のみ比較対象とする．TPM を利用する手法は既に紹介したとおりさ

まざまな手法が存在するが，どれも提案手法の想定する脅威モデルとは異なっている．したがって，SecLaaS の各サーバでソフトウェアの代わりに TPM がハッシュ値・電子署名を生成した場合を考える．

7.2.2 比較

各脅威に対してそれぞれの手法で対策が可能であるかどうかを考察した結果を表 7.1 に示す．VM 調査・解析専用ソフトウェアの中にはログファイルの完全性を検証できるものも存在するが，電子署名によって真正性を保証できるものは無いため△とした．また，SecLaaS では不正なオペレータが TPM にアクセスして偽のログファイルに電子署名を付加する危険性が否定できないため「不正なオペレータ」と「改ざん・捏造・隠蔽」の欄は△とした．

提案手法では VM セキュアプロセッサのメモリ暗号化・アクセス保護の機能によってログファイルの機密性が確保できる．TPM にも暗号化の機能はあるが，ロードストアにあわせてデータを暗号化することは不可能であるため，機密性に関しては VM セキュアプロセッサの方がより強固である．VM セキュアプロセッサをクラウド環境に導入した場合，各コンポーネントで暗号化するため，通信路でも漏洩の可能性は無く，ストレージ内の管理の際もログファイルの機密性が損なわれることはない．さらに，本研究で実装したハッシュ値・電子署名生成機能を用いてログファイルの完全性・真正性を検証することが可能である．図 7.3 のように，既存手法である SecLaaS や TPM を用いた手法では証拠保全の一貫性の起点が NodeController や TPM であった．それに対して，提案手法ではプロセッサでログが生成された時点から一貫して完全性・真正性を保証している．

表 7.1: 各手法のセキュリティ評価

	機密性			完全性・真正性
	オペレータの不正	VMI	不正アクセス	改ざん・捏造・隠蔽
VM 調査・解析専用ソフトウェア	—	—	—	△
VMI	—	—	—	×
SecLaaS	△	△	○	○
SecLaaS ＋ TPM	△	○	○	△
提案手法	○	◎	◎	◎

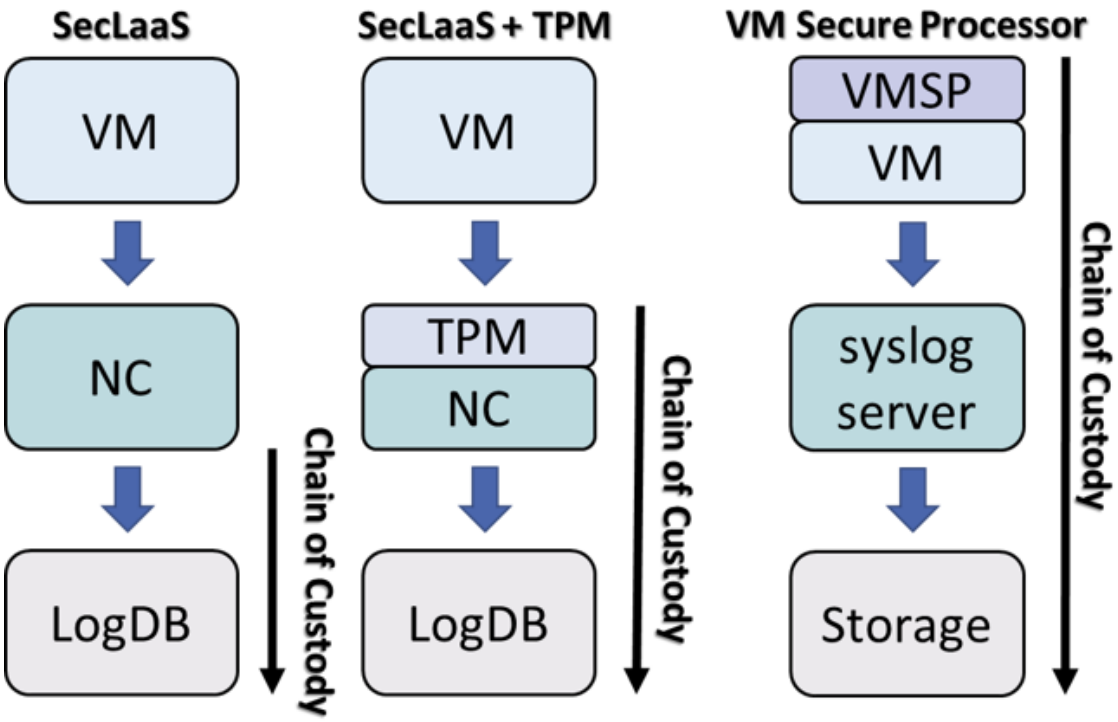


図 7.3: Comparison of “Chain of Custody”

第8章　結論

8.1　本研究のまとめ

本研究ではクラウドフォレンジックの課題であった「強固で効率的なログファイル管理」のためにVM セキュアプロセッサを用いたログファイル管理システムを提案した．VM セキュアプロセッサは現在開発中であり，本研究ではログファイルの認証に必要なハッシュ値生成モジュールと電子署名生成モジュールの実装を進めた．ハッシュ値生成モジュールに関しては FPGA 向けに合成して，実機上でハッシュ値が生成されることを確認した．また，回路面積に関して，SHA-3 モジュールとしては約 10 %の面積コストで抑えられることを確認し，ハードウェア実装コストの低さを示した．電子署名生成モジュールに関してはハッシュ値生成モジュールと連携させて，シミュレーション上で正しい電子署名の値が生成されることを確認した．セキュリティ評価では VM セキュアプロセッサの機能を活かすことで、ログファイルの機密性・完全性・真正性の全てが確保されることを示した．最後に提案したログファイル管理システムがクラウドフォレンジックの課題に対する解になりうるかどうか，得られた結論をそれぞれ述べて本研究のまとめとする．

保全データの爆発　本研究では推測値による性能評価を行い，動作周波数を高めれば TPM と同等，もしくはそれ以上の処理速度が出せるという結論が得られた．しかし，クラウドで扱うデータ量はより一層増加すると考えられるため，クラウドフォレンジックの効率化を進めるためにはさらなる処理速度の向上が求められる．

Resource Pooling（リソース共有）　VM セキュアプロセッサは VM の機密性を保護するために設計されているため，クラウド環境のように計算リソースが共有された状況だとしても，ログファイルをはじめとする機密データは保護される．

Chain of Custody（証拠保全の一貫性）　提案手法ではクラウドの各コンポーネントに VM セキュアプロセッサを導入されるため，ログの生成時から syslog サーバへの転送，外部ストレージへの保管まで一貫してログファイルの完全性・真正性が確保される．

クラウドプロバイダが信頼できるか　VM セキュアプロセッサは元々クラウドプロバイダが信頼できない場合を想定して設計されているため，クラウドプロバイダが信頼できなくてもログファイルの機密性・完全性・真正性は保証される．

8.2 今後の課題

本研究の今後の課題として以下の三点が挙げられる．

- ソフトウェアでハッシュ値を生成した場合との速度比較
- TPM でハッシュ値を生成した場合との速度比較
- 回路面積の大きい FPGA への移行

SHA-3 の開発者はソフトウェア実装をまとめたパッケージを Web [1] で配布している．本研究ではそのパッケージ内のプログラムをクロスコンパイルして FPGA 上で動作させる予定だったが，プログラムのパラメータは CPU のアーキテクチャに依存するため，公平な速度比較を行うことが出来なかった．また，現時点で TPM がサポートするハッシュ関数は SHA-1 と SHA-2 のみであるため，本研究との正確な性能比較を行うためには TPM が SHA-3 をサポートするまで待たなければならない．第 7 章で述べたように，メモリ暗号化と電子署名生成を実装したときの回路面積は現在用意している FPGA に収まりきらない．VM セキュアプロセッサは開発中であり，今後さまざまなモジュールが追加実装されることが予想されるため，回路面積の大きい FPGA への移行は必須となるだろう．

[1] http://keccak.noekeon.org/files.html

参考文献

[1] Karen Kent, Suzanne Chevalier, Timothy Grance, and Hung Dang. Sp 800-86. guide to integrating forensic techniques into incident response. 2006.

[2] 警察庁. 平成 27 年上半期のサイバー空間をめぐる脅威の情勢について. Technical report, 2015.

[3] RCFL. The rcfl program's annual report for fiscal year 2012. Technical report, Annual report, Regional Computer Forensics Laboratory, 2012.

[4] Peter Mell and Tim Grance. The nist definition of cloud computing. 2011.

[5] J Archer, A Boehme, D Cullinane, P Kurtz, N Puhlmann, and J Reavis. Top threats to cloud computing, version 1.0, 2010.

[6] Thomas Ristenpart, Eran Tromer, Hovav Shacham, and Stefan Savage. Hey, you, get off of my cloud: exploring information leakage in third-party compute clouds. In *Proceedings of the 16th ACM conference on Computer and communications security*, pp. 199–212. ACM, 2009.

[7] Keyun Ruan, Joe Carthy, Tahar Kechadi, and Ibrahim Baggili. Cloud forensics definitions and critical criteria for cloud forensic capability: An overview of survey results. *Digital Investigation*, Vol. 10, No. 1, pp. 34–43, 2013.

[8] Kara Nance, Matt Bishop, and Brian Hay. Virtual machine introspection: Observation or interference? *IEEE Security & Privacy*, No. 5, pp. 32–37, 2008.

[9] Samuel T King and Peter M Chen. Subvirt: Implementing malware with virtual machines. In *Security and Privacy, 2006 IEEE Symposium on*, pp. 14–pp. IEEE, 2006.

[10] Joanna Rutkowska. Introducing blue pill. *The official blog of the invisiblethings. org*, Vol. 22, , 2006.

[11] Joanna Rutkowska and Alexander Tereshkin. Bluepilling the xen hypervisor. *Black Hat USA*, 2008.

[12] Diego Perez-Botero, Jakub Szefer, and Ruby B Lee. Characterizing hypervisor vulnerabilities in cloud computing servers. In *Proceedings of the 2013 international workshop on Security in cloud computing*, pp. 3–10. ACM, 2013.

[13] Draft NISTIR. 8006 nist cloud computing forensic science challenges.

[14] Josiah Dykstra and Alan T Sherman. Acquiring forensic evidence from infrastructure-as-a-service cloud computing: Exploring and evaluating tools, trust, and techniques. *Digital Investigation*, Vol. 9, pp. S90–S98, 2012.

[15] Alan M Dunn, Owen S Hofmann, Brent Waters, and Emmett Witchel. Cloaking malware with the trusted platform module. In *USENIX Security Symposium*, 2011.

[16] David Challener, Kent Yoder, Ryan Catherman, David Safford, and Leendert Van Doorn. *A practical guide to trusted computing*. Pearson Education, 2007.

[17] Christopher W Fletcher, Marten van Dijk, and Srinivas Devadas. A secure processor architecture for encrypted computation on untrusted programs. In *Proceedings of the seventh ACM workshop on Scalable trusted computing*, pp. 3–8. ACM, 2012.

[18] G Edward Suh, Dwaine Clarke, Blaise Gassend, Marten Van Dijk, and Srinivas Devadas. Aegis: architecture for tamper-evident and tamper-resistant processing. In *Proceedings of the 17th annual international conference on Supercomputing*, pp. 160–171. ACM, 2003.

[19] G Edward Suh, Charles W O'Donnell, Ishan Sachdev, and Srinivas Devadas. Design and implementation of the aegis single-chip secure processor using physical random functions. In *ACM SIGARCH Computer Architecture News*, Vol. 33, pp. 25–36. IEEE Computer Society, 2005.

[20] G Edward Suh, Charles W O'Donnell, and Srinivas Devadas. Aegis: A single-chip secure processor. *Design & Test of Computers, IEEE*, Vol. 24, No. 6, pp. 570–580, 2007.

[21] Dan Boneh, David Lie, Pat Lincoln, John Mitchell, and Mark Mitchell. *Hardware support for tamper-resistant and copy-resistant software*. Stanford University, 2001.

[22] David Lie, Chandramohan Thekkath, Mark Mitchell, Patrick Lincoln, Dan Boneh, John Mitchell, and Mark Horowitz. Architectural support for copy and tamper resistant software. *ACM SIGPLAN Notices*, Vol. 35, No. 11, pp. 168–177, 2000.

[23] 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, 2004.

[24] Frank McKeen, Ilya Alexandrovich, Alex Berenzon, Carlos V Rozas, Hisham Shafi, Vedvyas Shanbhogue, and Uday R Savagaonkar. Innovative instructions and software model for isolated execution. In *Proceedings of the 2nd International Workshop on Hardware and Architectural Support for Security and Privacy*, pp. 1–1. ACM, 2013.

[25] Tal Garfinkel and Mendel Rosenblum. A virtual machine introspection based architecture for intrusion detection. *In Proc. Network and Distributed Systems Security Symposium*, 2003.

[26] Kenichi Kourai and Shigeru Chiba. Hyperspector: Virtual distributed monitoring environments for secure intrusion detection. *Proc. of the 1st ACM/USENIX International Conference on Virtual Execution Environments*, 2005.

[27] Chiueh T Conover M. Code injection from the hypervisor: Removing the need for in-guest agents, 2008.

[28] Hyun wook Baek, Abhinav Srivastava, and Jacobus Van der Merwe. Cloudvmi: Virtual machine introspection as a cloud service. In *Proceedings of the 2014 IEEE International Conference on Cloud Engineering*, IC2E '14, pp. 153–158, Washington, DC, USA, 2014. IEEE Computer Society.

[29] Shams Zawoad, Amit Kumar Dutta, and Ragib Hasan. Seclaas: secure logging-as-a-service for cloud forensics. In *Proceedings of the 8th ACM SIGSAC symposium on Information, computer and communications security*, pp. 219–230. ACM, 2013.

[30] Benjamin Bock, David Huemer, and A Min Tjoa. Towards more trustable log files for digital forensics by means of "trusted computing". In *Advanced Information Networking and Applications (AINA), 2010 24th IEEE International Conference on*, pp. 1020–1027. IEEE, 2010.

[31] Anyi Liu, Jigang Liu, and Tetsutaro Uehara. Secure streaming forensic data transmission for trusted cloud. In *Proceedings of the 2nd international workshop on Security and forensics in communication systems*, pp. 3–10. ACM, 2014.

[32] Huiping Guo, Yingjiu Li, Anyi Liu, and Sushil Jajodia. A fragile watermarking scheme for detecting malicious modifications of database relations. *Information Sciences*, Vol. 176, No. 10, pp. 1350–1378, 2006.

[33] 山田剛史. 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. PhD thesis, 2014.

[34] 宮永瑞紀. VM セキュアプロセッサの構成法. PhD thesis, 2015.

[35] Penny Pritzker and Patrick D Gallagher. Sha-3 standard: Permutation-based hash and extendable-output functions'. *Information Tech Laboratory National Institute of Standards and Technology*, pp. 1–35, 2014.

[36] Davide Balzarotti, Salvatore J Stolfo, and Marco Cova. *Research in Attacks, Intrusions and Defenses: 15th International Symposium, RAID 2012, Amsterdam, The Netherlands, September 12-14, 2012, Proceedings*, Vol. 7462. Springer, 2012.

著者発表論文

主著論文

[1] 千田拓矢, 谷合廣紀, 宮永瑞紀, 入江英嗣, 坂井修一: クラウドフォレンジックに向けた VM セキュアプロセッサの設計と実装, 組込み技術とネットワークに関するワークショップ ETNET 2016, (発表予定).

[2] 千田拓矢, 宮永瑞紀, 山口利恵, 五島正裕, 坂井修一: VM セキュアプロセッサの提案, 情報科学技術フォーラム講演論文集, No. 13, L-011 (2014).

共著論文

[3] 岡本拓也, 千田拓矢, 宮永瑞紀, 入江英嗣, 坂井修一: Virtual Machine Introspection におけるパフォーマンス・リソース効率を両立したメモリ監視方法の提案, 信学技報, vol. 115, no. 243, CPSY2015-57, pp. 59-61 (2015).

[4] 嶋紘之，宮永瑞紀，千田拓矢，岡本拓也，五島正裕，坂井修一: VM セキュアプロセッサにおける暗号化方式の検討, 情報処理学会第 77 回全国大会, 5X-06 (2014).

[5] 山田剛史, 千田拓矢, 山口利恵, 五島正裕, 坂井修一: 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, 暗号と情報セキュリティシンポジウム予稿集, CD-ROM, 3F3-3, 2014.

謝辞

本研究を作成するにあたり，多くの方々にお世話になりました．
坂井修一教授には研究の方針や論文の構成についてのアドバイスをいただき，大変勉強になりました．心から感謝申し上げます．
入江英嗣准教授には研究内容の技術的なアドバイスを頂いたり，研究を進めるためのツールを紹介してもらったり大変お世話になりました．
国立情報科学研究所の五島正裕教授には研究テーマやその内容について丁寧なご指導を賜りました．ここに感謝の意を表します．
山口利恵特任准教授には提案手法の安全性や評価について多くの的確なアドバイスをいただきました．非常に感謝しております．
研究室の秘書の八木原晴水さんには研究を行ううえでの事務や研究室の備品管理などでお世話になりました．ありがとうございました．
村上航規氏，山田剛史氏，宮永瑞紀氏には研究のテーマ決定をはじめ，コンピュータセキュリティの理論，論文執筆について様々なアドバイスをいただきました．日々の議論を通じて研究を深めることができました．心から感謝いたします．
その他にも，坂井・入江研究室の皆様には研究や論文の章立て・構成・執筆，研究室での生活サポートなど様々な面でご協力いただき，大変お世話になりました．
本論文の研究は一部，公益財団法人セコム科学技術振興財団の助成によります．