修士論文

VMセキュアプロセッサの構成法

Design of a secure processor for virtual machines

平成27年2月5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

48-136439 宮永 瑞紀

概要

近年，クラウド上の仮想サーバの普及に伴い，多くの企業が自社のサービスを外部のクラウドプラットフォームを用いて展開する事例が増加している．クラウドプラットフォームの利用は自社でサーバ筐体を保有する場合に比べ，導入コストやスケーリングの容易さなど様々な面で優れているが，クラウドの管理者がその権限を悪用し顧客の仮想サーバ内の情報を盗む危険性がある．仮想マシンに対してハイパーバイザが強い権限をもつという構造は，アプリケーションに対して OS が強い権限をもつ構造と似ている．アプリケーションがもつ情報は暗号化されない形でメモリやプロセッサのレジスタに保持されており，これらの情報を悪意ある端末管理者や OS から保護するのは困難である．

これらの問題への解決策として，これまで以上に強い権限による強制アクセス制御，すなわち，端末の管理者に与えられるアクセス権限をもプロセッサで制限する手法がいくつか提案されている．しかしプロセッサによる仮想マシン保護の既存手法もクラウド環境ばかりを視野に入れており，ただでさえ普及コストの面でソフトウェアに劣るハードウェアでのアプローチにおいて用途を限定しているのは大きなデメリットとなり得る．

本研究室では幅広い用途において VM を端末管理者から保護するセキュアプロセッサとして，VM セキュアプロセッサを提案している．本研究はこの VM セキュアプロセッサの実装を目指すものであり，本稿ではそのメモリ暗号化機構の実装と評価の結果，VM セキュアプロセッサの性能が十分実用に耐えるであろうと予測できた．

目 次

図 目 次

第1章　はじめに

近年，Amazon EC2 などのクラウド上の仮想サーバの普及に伴い，日本経済新聞社や東京証券取引所をはじめとした多くの企業が自社のサービスを外部のクラウドプラットフォームを用いて展開する事例が増加 [4] している．クラウドプラットフォームの利用は自社でサーバ筐体を保有する場合に比べ，導入コストやスケーリングの容易さなど様々な面で優れている．

その反面，プラットフォーム提供者が仮想マシン管理に用いるハイパーバイザ（仮想マシンモニタ）は一般に強い権限をもっており，そのため仮想マシン（VM）内の情報を閲覧・改ざんすることは不可能ではなく，VM 管理者がその権限を悪用するリスクが指摘されている．顧客企業は自社が保有する個人情報をクラウド上の VM 内に保持しなくてはならないが，個人情報の流出により企業の信用が大きく損なわれる事件が多発する昨今においてこのようなリスクは無視できるものではなく，クラウドプラットフォーム利用の妨げとなることがある．

VM に対してハイパーバイザが強い権限をもつという構造は，アプリケーションに対して OS が強い権限をもつ構造と似ている．OS や管理者権限を付与した別のアプリケーションによって通常のアプリケーションのデータを読み書きするのは容易である．2014 年の調査 [5] では，全世界の PC ソフトの不正コピー率は 43 %にものぼるとされる．こうした不正コピーの多くは，そのアプリケーション自身が正規のライセンスをもってインストールされたものであるかを検証するコードを迂回するような改変をアプリケーションに施すことにより行われている．また，正規のライセンスをもってインストールされた改ざんされていないアプリケーションであっても，そのアプリケーションがもつ情報は暗号化されない形でメモリやプロセッサのレジスタに保持されており，これらの情報を悪意ある端末管理者から保護するのは困難である．

これらの問題への解決策として，これまで以上に強い権限による強制アクセス制御，すなわち，OS や端末の管理者に与えられるアクセス権限をも制限する手法がいくつか提案されている．

端末の管理者がアプリケーションへの攻撃者である場合，管理者の管理下にある OS はその権限をもってアプリケーションのデータを読み書きする可能性がある．このとき，アプリケーションにとって OS は信用できないと言える．OS が信用できない場合においてもアプリケーションの機密性，完全性を高めるための研究として AEGIS [6], Ascend [1], L-MSP [7] などのセキュアプロセッサが提唱されている．セキュアプロセッサではユーザープロセスのメモリやコンテキストなどのリソースをプロセッサが保護・

暗号化することにより，アプリケーションのもつ平文情報を OS からも秘匿することが出来る．プロセッサ内のキャッシュを除いては平文の状態ではデータが保持されないため，ICE(In Circuit Emulator) 等の機材によりメモリバスにプローブをあてて信号を読み取るようなハードウェアタンパにも耐えることが出来る．

しかし，本来プロセスの管理は OS が行っており，そのためこれらのセキュアプロセッサ上で動作させる OS にはプロセスやキャッシュの管理機構に改変が必要なことが多く，Windows などの商業用 OS のようにソースコードの公開されない OS への適用は OS メーカーの協力なくしては困難であるなどの問題も指摘されている．

一方で VM とハイパーバイザや管理 VM との結びつきはアプリケーションと OS の間よりも密接でなく，近年のクラウドや仮想化のプラットフォームの普及に伴い，Overshadow [8] のように OS からアプリケーションを保護するハイパーバイザや，HyperWall [9] や HyperCoffer [2] のようにハイパーバイザからユーザの VM を保護するプロセッサも提案されるようになった．しかしながらハイパーバイザのみによる VM の保護ではメモリの暗号化や Direct Memory Access (DMA) の扱いが困難であり，メモリバスへのアクセスを始めとするハードウェアタンパを防ぐことは出来ず，プロセッサによる VM 保護の既存手法もクラウド環境ばかりを視野に入れており，ただでさえ普及コストの面でソフトウェアに劣るハードウェアでのアプローチにおいて用途を限定しているのは大きなデメリットとなり得る．

本研究室では VM を端末管理者から保護するセキュアプロセッサとして，VM セキュアプロセッサ [10] を提案している．VM セキュアプロセッサではプロセッサによる VM の保護に加え，アプリケーションを保護するためのプラットフォームも提供することにより，クラウド環境から PC やスマートフォンまで，幅広い用途に応用可能となっている．

本研究はこの VM セキュアプロセッサの実装を目指すものであり，本稿ではそのメモリ暗号化機構の実装と評価について述べる．メモリの暗号化はプロセッサの性能低下の最大の要因になるであろうと推測されることから，本稿における性能評価は VM セキュアプロセッサ完成時の性能を予測する上で非常に重要なものとなる．

本稿では，まず 2 章で手法の対象とする環境を設定する．次に 3 章でハードウェアによるアプリケーションや VM の保護に関する既存手法について説明する．そして 4 章で提案手法である VM セキュアプロセッサについて述べ，5 章でその実装について，6 章にて性能評価について述べる．

第2章　想定する環境

まず，本稿における攻撃者等の環境を整理する．

本稿では1章冒頭で述べたような，クラウドプロバイダ（以下，プロバイダと呼ぶ）が IaaS (Infrastructure as a Service) 形式のサービスを提供している場合を想定して記述する．IaaS とは，ハードウェアや仮想化環境をプロバイダがクラウドユーザ（以下，ユーザと呼ぶ）に提供し，その上でユーザ自身の用意した OS を含むソフトウェアを動作させる形態のサービスである．

また，認証と保護における登場人物として主に以下の三者を想定する．

プロセッサメーカ　プロセッサの製造者．本研究では便宜上プロセッサの認証局を兼ねる．

クラウドプロバイダ　IaaS 提供者．物理マシンの管理者で有り，ハードウェアに直接アクセスできる．

クラウドユーザ　IaaS 利用者．データを保護したい者．

この環境において，ユーザの用意した OS やデータを含む仮想マシンに関して，

- 平文が信頼できない者に読み取られない
- データが意味のある改ざんを受けない

ことをユーザに保証するのが，本稿の手法における目的である．

なお，提案手法はクラウド環境下だけでなく，1章で述べたような個人の端末上においても利用可能である．その場合，クラウドユーザに当たるのはコンテンツやソフトウェアの提供者であり，クラウドプロバイダに当たる端末の管理者はその利用者である．

2.1　信頼できるものと信頼できないもの

提案手法における認証に関して，ユーザの直接操作する（すなわちクラウド上にない）コンピュータやプロセッサメーカ，及びそれらの間の通信路は信頼できるものとする．プロセッサメーカは製造部門などプロセッサ構造への改変が可能な人や，プロセッサの秘密鍵を知りうる人，ユーザと通信する部門などを含み，プロセッサ自体が製造中に改ざんされたり，認証に利用されている鍵が流出することはないものとする．

信頼できないものとしては，上記のものを除くあらゆる組織，すなわちプロバイダ，プロセッサの流通業者，プロバイダとユーザー間の通信路などを信用しない．これはハイパーバイザや管理 VM の管理者やハードウェアにアクセスできるものが悪意を持つものである可能性を意味する．また，プロセッサ以外のもの，つまりメモリやネットワークカードなどのデバイスも信用できないものとする．

さらに，用いられているプロセッサが正規のものでない可能性を考慮する．プロセッサも認証されるまでは信用できないものとする．

2.2 考慮しないもの

本稿においては，電子顕微鏡などによるプロセッサ内部の解析や，電力消費量の測定などによる解析，およびユーザーの用意したソフトウェアの脆弱性は考慮しないものとする．

第3章　既存手法

ハイパーバイザのみによって情報を保護する手法では他の VM などによるソフトウェアタンパには耐えることが出来るが，既存のプロセッサを用いるためにプロセッサとメモリ間のバスを暗号化することは出来ない．プロセッサ外のアプリケーションのメモリも暗号化するためには，セキュアプロセッサの利用が必要である．

ここでは，アプリケーションを OS から保護するセキュアプロセッサと，ハイパーバイザから VM を保護するプロセッサの例について述べる．

3.1　一般的なセキュアプロセッサ

セキュアプロセッサでは，ユーザープロセスのメモリやコンテキストなどのリソースを OS よりも強い権限でプロセッサが保護・暗号化することにより，アプリケーションのもつ平文情報の機密性，完全性を向上することが出来る．

多くのセキュアプロセッサはプロセッサ外部のメモリを暗号化するが，プロセッサ内のコンテキストやキャッシュは平文であるためこれらに OS がアクセスできないよう制御しなくてはならない．XOM [11] や L-MSP では一意なプロセスの識別子をキャッシュタグに書き込み，アクセス可能なプロセスをプロセッサが制限している．

実用化への大きな障害として，セキュアプロセッサを既存 OS に用いるには OS の根幹部分の処理に改変が必要となることが挙げられる．OS とユーザープロセスの結びつきは強く，ファイル読み込みやプロセス切り替えなどの処理に関して，それぞれの OS に合わせての対処が必要となる．ソフトウェアであるハイパーバイザを用いる手法では OS に合わせてハイパーバイザを調整することで対処は可能ではあるが，ハードウェアであるセキュアプロセッサにおいて各 OS に合わせた処理をプロセッサ設計時に組み込むのはあまり現実的ではない．そのため，セキュアプロセッサを用いるには OS の根幹に改変が必要と言える．

3.2　Ascend

Ascend [1] は，プロセッサ以外にデータの平文を与えずに計算できるセキュアプロセッサである．

環境としては，ユーザーから預かった機密データをクラウドサービス上などで，クラウドサービスが用意したプログラムを用いて処理することを想定している．計算を

行うためにはサーバーに平文を開示しなくてはならず，それは様々なアプリケーションやハイパーバイザ，あるいはクラウドシステムの管理者から攻撃を受ける可能性があることを意味する．完全準同型暗号等を用いれば計算者に平文を渡さずに計算できるが，計算量は非常に大きくなってしまう [12].

そこで Ascend ではユーザーとプロセッサ間で共通の鍵を共有し，プロセッサ外への出力は暗号化した状態で行うことにより，プロセッサ以外の，OS や周辺のデバイス，またそのアプリケーションにすら平文のデータを渡すことなく計算を行うことができる．

プログラムが信頼できない場合には，一般にはそのプログラムが扱う情報を秘匿するのは困難である．Fletcher らは [1] の中で，次のようなアルゴリズムを例としてあげている．

```
y = M[0]
while (y & 1) == 0 do
  issue a random load or store request
    from/to memory
end while
```

このようなアルゴリズムを用いれば，プログラムの実行時間やメモリアクセスから暗号化されたユーザーデータ M が取得できるとしている．

そこで Ascend では Oblivious RAM(ORAM) [13] を用いてメモリアクセスの時間的・位置的局所性を緩和し，また実行時間をユーザーが指定することによりプログラムを信用しなくても安全にデータを操作することが可能となっている．

具体的な手順は次の通りである．

1. ユーザーと Ascend は秘密の鍵の共有を行う．鍵の共有には Ascend とユーザーの間で公開鍵暗号を用いる．

2. ユーザーは共有した鍵を用いて入力 x の暗号 $encrypt(x)$ と計算に掛ける時間や消費電力 S，そしてサーバーから与えられていないなら入力に対して決定的なアルゴリズム A を送信する．

3. Ascend は S の間入力 $encrypt(x)$ とサーバーからの入力 y に関して A を実行し，暗号化されたメモリイメージ M_s を返す．このときメモリの暗号化を解き，ユーザーに送信するように暗号化された計算結果 M'_s を返すこともできる．

4. ユーザーは M_s から計算 $A(x, y)$ が S 以内に終わったかどうか確認することができる．

図 3.1 に暗号化を行わないプロセッサ（baseline）を 1 とした Ascend で暗号化を行った場合のベンチマーク（SPEC CPU2006 INT [14]）での速度低下比を示す．通常の RAM アクセスの遅延 $50cycle$ に対し Ascend においては ORAM(16GB) へのアクセスの遅延

図 3.1: Ascend における相対的な性能低下量 (from [1])

は $5880cycle$ にもなるが，この結果はインタープリタ型言語と同程度であり，驚く程良いものとされていたが，これほどの性能低下の環境で OS や VM を動作させるのは困難である．

3.3 耐ソフトウェアタンパプロセッサ

上の Ascend で利用されていた ORAM のようにメモリアクセスの時間的・位置的局所性を緩和する場合の保護対象のアプリケーションのメモリアクセス性能は著しく低下する．時間的・位置的局所性の緩和を伴わない通常のメモリ暗号化・復号処理のオーバーヘッドを削減する手法として，当研究室では耐ソフトウェアタンパプロセッサを提案していた．

デバッガなどのプログラムやパッチなどを導入するだけで容易に行えるソフトウェアタンパと比べ，技術上・設備上のハードルの高いハードウェアタンパを行うことの出来るユーザー数ははるかに少なく，そのためソフトウェアタンパのみでも対策できれば大多数のタンパを防ぐことが出来ると考えられる．そこで，耐ソフトウェアタンパプロセッサはこのソフトウェアタンパのみをプロセッサで防ぐことにより，非常に少ないオーバーヘッドでアプリケーションのタンパ性を高めることが出来るとしている．

通常のページテーブルエントリやセグメントディスクリプタにはユーザープロセスからのアクセスを制限するか否かのビットと書き込みを制限するビット，および要求する特権レベル程度しかアクセス制御の項目は存在しない．耐ソフトウェアタンパプロセッサではページテーブルエントリに，所有者，特権プロセス，その他のプロセスそれぞれからの読み込み，書き込み，実行のビットを用意し，これらのビットを TLB，DMA，MMU が参照し，アクセス制御を実施することを提案している．なお，OS がページテーブルエントリを改ざんすることはプロセッサで防ぎ，タンパが容易なストレージへの書き出し時には DMA がメモリの暗号化を行うとしている．また，保護されたメモリ領域と同じ場所にマップされると別のアクセス制御ビットを利用して制御を迂回される恐れがあるためこれを禁止しなくてはならない．

なお，このような特権プロセスからのアクセスの制御ビットをページテーブルエントリに追加する手法として，セキュアプロセッサではなくハイパーバイザを用いる手法 [15] も当研究室では提案している．

3.4 H-SVM

H-SVM [16] はプロセッサが仮想化支援のページングの延長でゲスト VM のメモリの内容をハイパーバイザから隠蔽することにより，ハイパーバイザが信用できない場合にも柔軟なメモリ管理が行える手法である．

一般的な OS ではプロセスが扱うメモリ空間はプロセスごとに仮想化され，原則として異なるプロセスのメモリにはアクセスできない．この仮想化されたアドレスでの参照時にプロセッサによって TLB (Translation Lookaside Buffer) やページテーブルを用いて物理的なアドレスに変換され，メモリ上のデータやを読み書きする．ページテーブルは OS によって管理される仮想アドレスから物理アドレスへの変換表であり，そのエントリは TLB にキャッシュされる．アーキテクチャによっては TLB ミス時のページテーブルの検索もプロセッサが行う．

一般的な仮想化環境において VM 上のゲスト OS が物理アドレスとして扱っているものも VM ごとに仮想化されたものであり，そのゲストの物理アドレスはハイパーバイザによってホストのアドレスに変換される．ゲスト VM のアドレスからホストの物理アドレスへと変換するためにはハイパーバイザが生成したシャドウページテーブルやネステッドページテーブルを用いる．プロセスの場合と同様に通常は VM から他の VM のメモリへアクセスすることは出来ないが，ホストは VM のメモリを読み書きすることが出来る．また，ゲストの物理アドレスとホストの物理アドレスとの変換をハイパーバイザが行うため，ハイパーバイザは VM が他の VM のメモリにアクセスできるように設定することが出来る．クラウド環境において悪意のあるクラウド管理者がホスト OS や管理 VM の制御下にあるハイパーバイザのこの権限を悪用した場合や，ハイパーバイザがその脆弱性により他のゲスト VM に乗っ取られた場合には，クラウド上の VM のメモリを自由に読み取り・改ざんすることが出来る．

H-SVM はネステッドページテーブルをプロセッサが管理することにより，ハイパーバイザからのアクセスを防ぐことができる．ハイパーバイザはプロセッサのメモリ管理機能により間接的に VM のメモリ管理を行えるが，プロセッサはページマッピングの変更がある度にそれが有効であるかを確認する．さらにハイパーバイザのみによる VM の保護とは異なり，DMA にもこのアクセス制限を適用することが出来る．

H-SVM ではハイパーバイザとプロセッサの改変が少ないが，VM とハイパーバイザとの間でメモリを共有する場合には，共有したくないデータを共有領域に置かないようゲスト OS 側の改変が必要となる．また，この手法ではメモリの暗号化は行われず，ハードウェアタンパを防ぐことは出来ない．

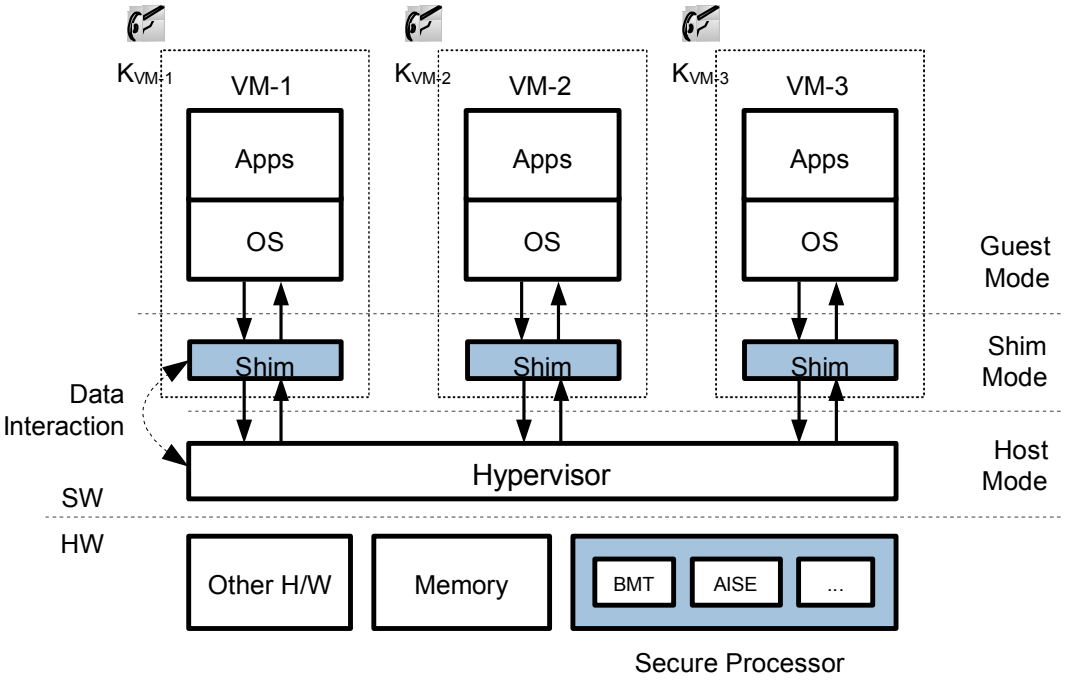

図 3.2: HyperCoffer における構造 (from [2])

H-SVM と同様にプロセッサによってメモリのアクセス制限を行う手法としては HyperWall がある．

3.5 HyperCoffer

HyperCoffer はプロセッサによって VM のメモリを暗号化し，ハイパーバイザから保護する手法である．H-SVM や HyperWall とは異なり，HyperCoffer ではハードウェアタンパにも耐えることが出来る．

また，HyperCoffer では**図 3.2** のようにハイパーバイザと VM の間に VM-Shim という軽量なソフトウェアを挟むことにより，ハイパーコールの実行や透過的なストレージ暗号化・改ざん検出，ネットワークなどの平文での入出力やゲストのページテーブルの変換を可能としながらも，ゲスト OS の改変は不要となっている．

第4章　VMセキュアプロセッサ

VMセキュアプロセッサは，ゲストVMのメモリからの情報漏洩を防止するため仮想化環境に対応したセキュアプロセッサである．VMセキュアプロセッサを用いれば，VMのもつ情報を漏洩しないセキュアなプラットフォームが用いられていること，その上で改変されていないVMが実行されていることを遠隔から検証可能なプラットフォームを構築することが出来る．

以下にVMセキュアプロセッサの構造と，その認証手順を示す．

4.1　VMセキュアプロセッサの構造

ここでは，4章で述べたVMセキュアプロセッサに関して，その機能を実現するための構造と具体的な機能を述べる．

4.1.1　仮想化

VMセキュアプロセッサではVMを端末の管理者から保護するために仮想化を支援する機構をもつ．

特権レベル

OSによってデバイスドライバをどのリングに配置するかなどの違いはあるが，一般的なOSとアーキテクチャにおいてはOSのカーネルとアプリケーションを異なるレベルに配置し特権を管理している．

図4.1は現在広く利用されているIA-32アーキテクチャがもつ保護メカニズムの図である．IA-32は図のように0から3の特権レベルの階層を持ち，数値が小さいほど高い特権となる．図4.1の例ではカーネルなどシステム内の最も重要なコードをレベル0に格納し，重要度が低いアプリケーションのコードは外側のリングで低い特権で動作する．低い特権のレベルにあるコードから高い特権のレベルにあるコードにアクセスするには，ゲートと呼ばれる厳密に制御され保護されたインターフェースを用いなければならないことになっている．十分なアクセス権を持たずに高い特権のメモリにアクセスしようとすると，例外が発生する．

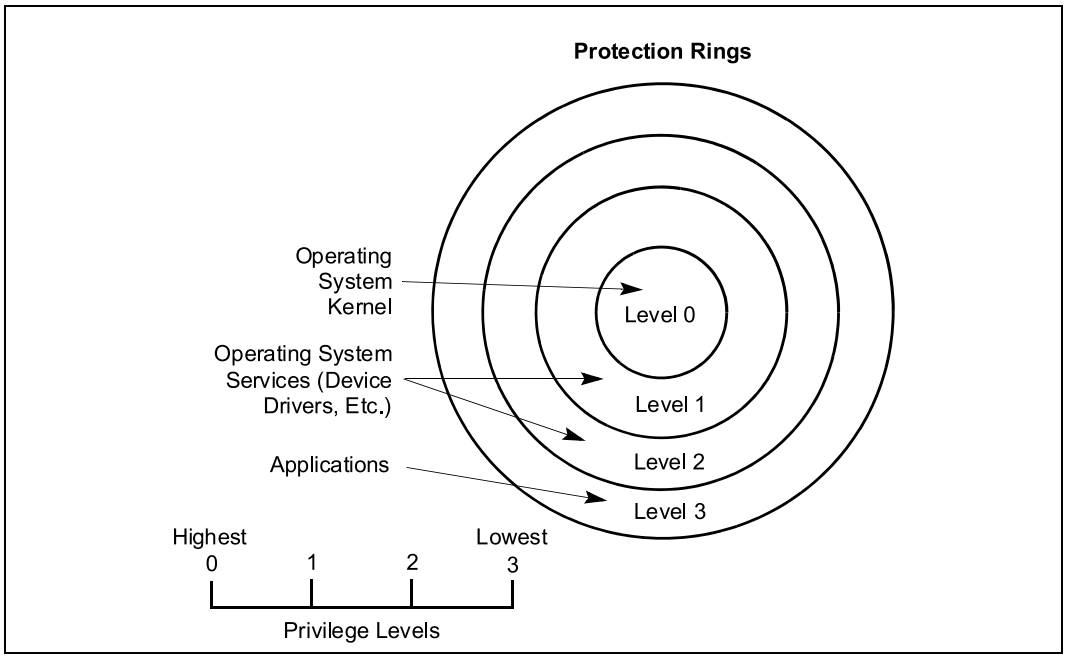

図 4.1: IA-32 におけるプロテクトリング (from [3])

仮想化支援としては Intel VT-x では VMX no-root モード， AMD-V では Guest モードのようなこの外側に新たに設けられた低い特権レイヤー内のリング 0 から 3 でゲスト VM を動作させることにより，ハイパーバイザはゲスト VM から見てレベル-1 と称されるようなより高い特権を持って動作することが出来る．

ハイパーバイザ（ホスト），ゲスト OS，ゲストアプリケーションの順に高い特権を持つというこのような保護構造は，多少形は異なるものの，ARM など他のアーキテクチャにも見られる．

他のアーキテクチャと同様に VM セキュアプロセッサもモードとしてはハイパーバイザ（ホスト），ゲスト OS，ゲストアプリケーションのモードを持ち，VM はゲストモードで動作することになる．**図 4.2** はその階層構造を示したものである．ただし，VM の保護のためにホストモードのコードですらメモリアクセスなどに関して必ずしもゲストよりも強い権限を得られるとは限らない点は，VM セキュアプロセッサとそれ以外のアーキテクチャとの違いの中でも大きなものである．

アドレス変換とメモリマッピング

VM セキュアプロセッサにおいてもそれぞれの VM は異なるメモリ空間にマッピングされ，VM 間で共有されている部分を除いては他の VM のメモリにアクセスすることは出来ない．これによって，VM セキュアプロセッサは VM のメモリの暗号化を行わなくとも他の VM からのソフトウェアタンパを防ぐことが出来る．また，VM のアドレ

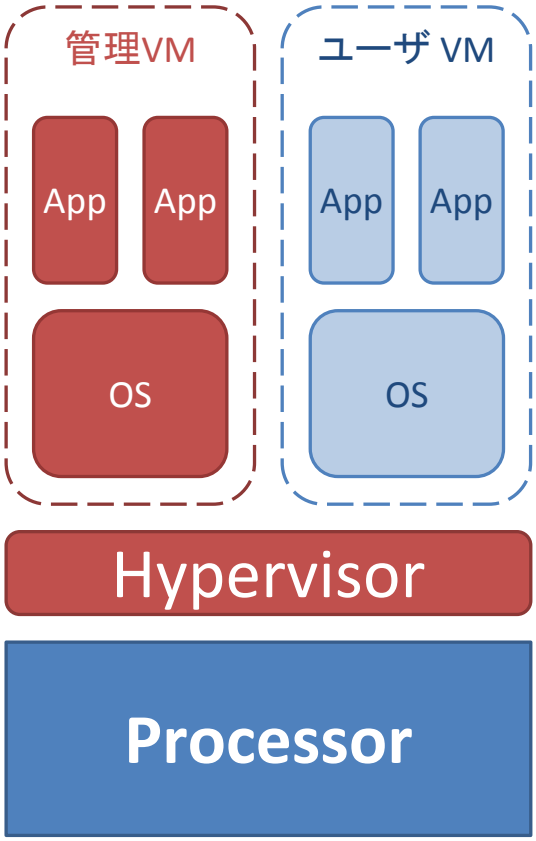

図 4.2: 通常の仮想化環境における VM セキュアプロセッサの階層

ス変換はプロセッサによって行われ，VM の実行には HyperCoffer における VM-Shim のようなソフトウェアのレイヤーを必要としない．

コンテキストスイッチ

実行中のプロセスがシステムコール（スーパーバイザコール）を呼び出した場合やタイマーなどの要因によって割り込みが発生した場合，OS やプロセッサは実行中のプロセスの情報（プログラムカウンタをはじめとするレジスタやフラグなど）を退避し，プロセスを再開する際にそれらの情報を復元する．これらの操作は主に OS のカーネルに実装されたコードによって行われ，OS がプロセスを操作したりプロセスが OS の機能を利用したりする上で重要なものとなっている．このためにプロセスのレジスタを OS から隠蔽するのは OS のカーネルの改変なしには困難であり，従来のセキュアプロセッサの実用化上の大きな障害のひとつとなっていた．

通常の仮想化環境においては VM の切り替え時に VM のコンテキストの退避・復元をハイパーバイザのコードにより行う．割り込み発生時にホストのコードがゲストに

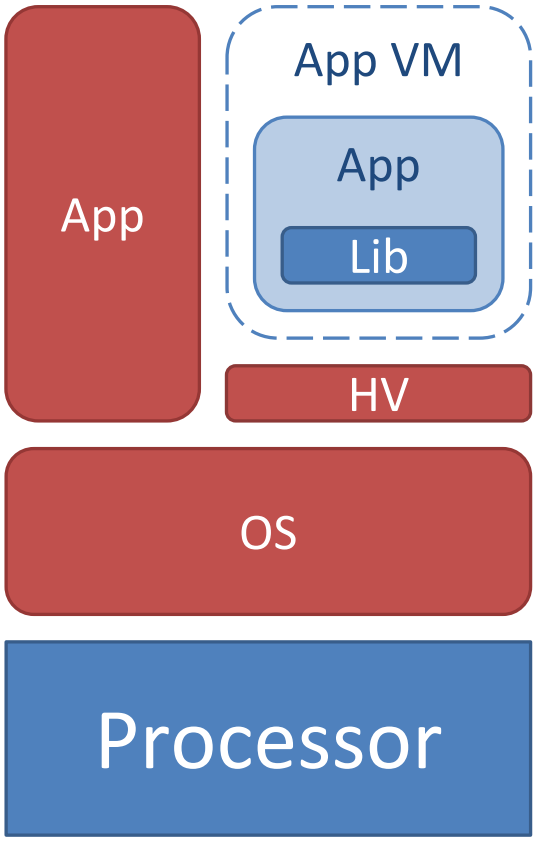

図 4.3: アプリケーションの VM 化環境における VM セキュアプロセッサの階層

優先されて実行されるのを利用しゲスト VM 上のプロセスの情報をゲストの OS から保護する Overshadow などのハイパーバイザも提案されているが，プロセスの管理方法は OS 固有のものとなっており，OS に合わせてハイパーバイザに細かな修正が必要となる．

VM セキュアプロセッサでは VM の切り替え時にはプロセッサがコンテキストの暗号化と復号をハードウェアで行う．暗号化されたコンテキストの管理はハイパーバイザによって行われるが，このときに平文のコンテキストは破棄され，ハイパーバイザが平文のコンテキストを読み書きすることは出来ない．これによって VM セキュアプロセッサでは VM のコンテキストの流出を防ぐことが出来る．ハイパーバイザによるコンテキストの改ざんを防ぐために，VM セキュアプロセッサは VM 再開時にハッシュの比較による改ざん検出を行う．

VM インターフェース

VM セキュアプロセッサではアプリケーションを VM として動作させるためのライブラリを持つ．これによりアプリケーション開発者は暗号化をほとんど意識することなく既存アプリケーションを VM セキュアプロセッサ用に移植し保護することが出来る．**図 4.3** はその実行時の様子を表したものである．UI は通常のアプリケーションとして OS 上で動作するよう実装する．

これは個人の端末上でのコンテンツ保護などの用途に向いた機能である．画面出力や音声を OS に平文に渡すことなく出力することも出来るが，耐 HW タンパ性のためにはディスプレイやヘッドセットなどのデバイスも暗号化通信をサポートするのが望ましい．

オーバーヘッドを無視できるのであれば完全な OS を用いることにより既存アプリケーションの実行バイナリをそのまま利用することも出来る．

4.1.2 アクセス制御

IA-32 などのアーキテクチャにおいて，プロセスは自らの仮想アドレス空間にマッピングされていない領域へはアクセスできないのに加え，ページやセグメントなどのメモリの領域単位で書き込みやユーザーモードでのアクセスを制御するフラグが存在し，実行中のコードが持つ特権が条件を満たさなければ例外が発生して操作を行えない．しかし，OS はこのフラグを変更することで制限を受けることなく任意のアクセスすることが出来る．

VM セキュアプロセッサにおいては，前述のようにゲスト VM と管理 VM は原則としてホストの物理アドレス空間上で異なる領域にマッピングされ，そのため管理 VM はゲスト VM のメモリにアクセスすることは出来ない．ハイパーバイザはホストの物理アドレス空間上で動作するためにロード・ストア命令などで各 VM がマッピングされているメモリを指定できるが，VM のメモリが暗号化されていればハイパーバイザは VM のメモリの平文での読み書きを行うことは出来ず，加えてホストからのメモリの読み書きに関するフラグが不許可としてセットされていた場合にはホストモードで動作しているハイパーバイザですら VM のメモリにアクセスすることは出来ない．ゲストのカーネルモードのコードは自身の VM に割り当てられたメモリに対するホストモードのコードからのアクセス制御フラグを変更することが出来る．

VM セキュアプロセッサ内部においては，キャッシュやコンテキストは VM ごとに管理され，他の VM やホストのコードから見ると全て 0 か無効化された状態に見える．これによりプロセッサ内部で保護された VM の情報が読み書きされることはなく，VM セキュアプロセッサは VM のメモリの暗号化を行わなくともホストからの攻撃を含むソフトウェアタンパを防ぐことが出来る．

4.1.3 暗号化

VM セキュアプロセッサでは鍵をプロセッサ内部に保管し，暗号化，復号処理もプロセッサ内で行う．これによりその VM 自身を除いてはストレージ内の平文データは有意に読み書きすることが出来ず，またメモリ上の平文のデータはハードウェアタンパをもってしても有意に読み書き出来ない．これにより，VM セキュアプロセッサはハードウェアタンパとソフトウェアタンパの両方を防ぐことが出来る．

暗号化されたデータは管理 VM やハイパーバイザに渡され，それぞれの物理デバイスを制御する通常のドライバによって処理される．

以下に，それぞれの項目の暗号化について詳細に述べる．

メモリの暗号化

VM のメモリは VM セキュアプロセッサ内部で透過的に暗号化・復号され，VM 外部と情報の授受を行わない限り，VM 自身はこれを意識する必要が無い．プロセッサ内部ではキャッシュラインのタグやコンテキストに VM 固有の ID を付加し，実行中の VM の ID と一致しない場合のアクセスを禁じる．

ただし，メモリと外部との情報の授受を行う場合にはこれを考慮した制御が行われなくてはならない．外部との通信は DMA によるデータ転送，他の VM とのメモリ共有，プロセッサを経たデバイスなどとの通信（PIO）が主である．これらの制御は VM セキュアプロセッサ上で動作する OS のためのドライバとして提供される．

なお，保護対象の VM のメモリは暗号化するが，メモリとの間のアドレスバス，コマンドバスは暗号化しない．Ascend では ORAM の利用によりアドレスやアクセスタイミングも隠蔽していたが，その代償として性能低下も大きく，またこれらのバスの情報から有意なデータを盗むのは困難と考えられるためである．

また，ハイパーバイザの機能を最低限とするならハイパーバイザがゲスト VM 上のメモリの内容を読み書きする必要はほとんどなく，そのためハイパーバイザがゲスト VM のメモリの平文を読み書きできない設計で有る限りにおいて，ハイパーバイザ自身の情報や，外部から読み込んだ暗号化されていないデータを一時保管するのみのハイパーバイザのメモリは暗号化する必要がなく，管理 VM から隠蔽する必要はない．ハイパーバイザは暗号化された状態でのゲスト VM のメモリさえ読み書きできれば，ゲスト VM のメモリのスワップアウトやサスペンドが可能である．

さらに，VM の設定によってはメモリの暗号化を行わないことも出来る．この場合は耐ソフトウェアタンパプロセッサのようにアクセス制御のみを行えば，性能をほとんど落とさずにソフトウェアタンパのみを防ぐことが出来る．

ストレージの暗号化

VM のストレージは VM セキュアプロセッサによって透過的に暗号化．復号され，VM 自身はこれを意識する必要はない．

VM のストレージは VM からの読み込み指示があった際にファイルとしてホスト OS または管理 VM によって暗号文の状態でメモリに読み込まれ，プロセッサによって復号，対象の VM のメモリ上にメモリの鍵で暗号化して保存される．書き込みの際にはこれらの手続きが逆の順に行われる．

その他のデバイス

遠隔での VM の操作内容を保護するためには，ディスプレイ出力やキーボード入力といったものも暗号化して VM に接続されなくてはならない．VM セキュアプロセッサではこれらも標準的なドライバで制御できる形で VM に提供し，ユーザはプロセッサによって暗号化された仮想コンソールを通して VM を操作することが出来る．また，重要なデバイスはプロセッサが仮想的なデバイスとして VM に提供しているため，IO バッファとして VM とハイパーバイザでメモリを共有する必要がほとんどなく，ゲスト VM のメモリアクセスをユーザが細かく設定したりゲスト OS を改変したりしなくてよい．ただし，特殊な暗号化デバイスが必要な場合には専用のドライバを用意するか，あるいは HyperCoffer の VM-Shim のような軽量なコードを OS 外に付加する必要がある．

VM の管理情報の暗号化

ここまでに挙げられた VM のストレージやメモリの鍵，コンテキスト，メモリマッピングの全てを VM セキュアプロセッサ内部に保管することは出来ない．そのため，VM セキュアプロセッサではこれらの情報をプロセッサや VM の鍵で暗号化した状態でプロセッサ外部に出力する．暗号化された VM の管理情報はホスト OS や管理 VM，ハイパーバイザによって保管される．

4.1.4 認証

VM セキュアプロセッサは認証のためにいくつかの機能を持つ．これらを用いることにより VM は自身の完全性とプラットフォームの真正性を検証することが出来る．

認証対象

あるメモリ上のデータの有意な改ざんと漏洩を防ぐとき，そこへ間接的にでもアクセスすることの出来る全てのコードについてそのコードが保護対象のデータを不正に

改ざんしたり外部に漏洩したりしないものであること，さらにそのコードが不正に書き換えられないことを確認できれば，データを確実に保護することが出来る．

保護したいデータが VM 内にあり，OS など VM 内のソフトウェアやプロセッサがそのデータを不正に改ざんしたり外部に漏洩したりしないものであると仮定すると，保護したいデータにアクセスできるソフトウェアを VM 内のものに制限するとき，そのソフトウェアへの有意な改ざんを防げばよいことになる．

そこで VM セキュアプロセッサでは，VM のメモリやストレージ，プロセッサ自体の保護・認証を提供するが，それらへのアクセス権のないハイパーバイザなどは認証する必要がないと言える．

公開鍵と秘密鍵

VM セキュアプロセッサは製造時に埋め込まれた公開鍵と秘密鍵のペアをもつ．公開鍵にはプロセッサメーカーによる署名がなされ，いつでもプロセッサから取得することが出来る．付与された署名はプロセッサメーカーから署名に対応する公開鍵を取得し，検証することが出来る．プロセッサが外部との通信に用いる公開鍵・秘密鍵のペアは，プロセッサの認証にも用いる．公開鍵にはプロセッサメーカーによる電子署名が付加される．この電子署名により，この公開鍵が VM セキュアプロセッサのうちの 1 つのものであることがメーカーによって保証される．すなわちこの鍵ペアによって保護される通信の相手が VM セキュアプロセッサであることを保証できる．この鍵ペアを VM セキュアプロセッサ同士が互いに認証することにより，VM セキュアプロセッサ間で VM を保護したままのマイグレーションも可能である．これらの鍵ペアは VM セキュアプロセッサとの通信や，プロセッサが計測したハッシュへの署名に用いる．

ハッシュの計測

VM セキュアプロセッサは，指定されたメモリやストレージ，ハイパーバイザなどとハッシュを計測することが出来る．プロセッサによって署名されたハッシュ値は署名を要求したプロセスや VM のメモリ空間上の指定アドレスに，プロセッサによって書き込まれる．このハッシュ値はユーザとプロセッサの通信において送信することも可能である．

乱数と鍵の生成

VM セキュアプロセッサはプロセッサ内部に暗号生成機構と暗号化鍵の生成機構をもち，これらは認証時や VM の暗号化鍵生成時に用いられる．ハードウェアでの乱数発生機能は Trusted Platform Module や Intel の Ivy Bridge プロセッサなどの既に広く普及しているセキュリティチップやプロセッサにも搭載されているように，暗号鍵の生成などで重要なものとなっている．

リアルタイムクロック

VM セキュアプロセッサは実装によってはリアルタイムクロックをもつ．これにより，プロセッサに電力が供給され続ける限り，認証後一定期間のうちは再認証を省くようなことも出来る．

4.2 プロセッサの認証

VM セキュアプロセッサは認証に関して，いくつかの手順を提供している．

なお，ここでは便宜上 1 つの秘密鍵・公開鍵ペアで暗号化と署名が出来るとしているが，実際には用途によって個別の鍵ペアを用意する可能性がある．また，公開鍵による暗号化は共通鍵によるデータの暗号化の後に公開鍵で共通鍵を暗号化したものを添付するものとする．

4.2.1 プロセッサの認証

前述の通りプロセッサ固有の秘密鍵 sk_{cpu}・公開鍵 pk_{cpu} ペアのうち，公開鍵 pk_{cpu} にはプロセッサメーカの署名がなされており，この署名はプロセッサメーカの公開鍵によって検証できる．これによってこの秘密鍵を用いた通信・署名は正規の VM セキュアプロセッサの特定の個体によってなされたものであることが確認できる．

プロセッサ公開鍵 pk_{cpu} によって暗号化されたストレージをプロセッサストレージとする．プロセッサストレージの平文を読み書きできるのは VM セキュアプロセッサのこの個体のみであり，認証情報などをここに保管する．

このプロセッサストレージはクラウド環境よりもむしろ PC やスマートフォンといった個人のデバイス上での活用が有効である．アプリケーションをプロセッサストレージにインストールすればその平文が端末管理者に読み書きされることはなく，インストールに使用されたプロセッサでのみ実行可能である．このときインストーラを後述のプロバイダストレージなどで構成すればアプリケーションの秘密を保護したまますべての VM セキュアプロセッサで実行可能なインストーラが作成可能である．さらに前述のプロセッサ内のリアルタイムクロックと組み合わせれば，近年のオンラインレンタルビデオのような視聴期限付きのデジタルコンテンツに関しても，端末の時間を変更するなどして不正に期限を延長されることなくオフラインでの復号制限が可能である．

.

4.2.2 プロバイダストレージ

すべての正規の VM セキュアプロセッサは共通の共通暗号鍵 key_{comm} を持つ．

VM セキュアプロセッサは秘密鍵 sk_{prob}・公開鍵 pk_{prob} ペアを生成し，

$Enc(key_{comm}, sk_{prob})$　生成した秘密鍵 sk_{prob} を VM セキュアプロセッサ共通鍵で暗号化したもの．すべての正規の VM セキュアプロセッサが復号し秘密鍵 sk_{prob} を得ることが出来る．

$Sign(sk_{cpu}, pk_{prob})$　生成した公開鍵 pk_{prob} を VM セキュアプロセッサ固有の秘密鍵 sk_{cpu} で署名したもの．VM セキュアプロセッサの公開鍵 pk_{cpu} で検証することが出来る．

pk_{cpu}　VM セキュアプロセッサ固有の公開鍵．ここでは省略しているが，上記のメーカによる署名も含む．

pk_{prob}　生成した公開鍵．

の 4 つを出力する．これらをプロバイダキーとする．

以下に，クラウド環境における，正規の VM セキュアプロセッサ上でのみ実行可能な VM イメージの作成と実行の手順を示す．

1. プロバイダはプロバイダキーのうち暗号化した秘密鍵 $Enc(key_{comm}, sk_{prob})$ を除いた 3 つをユーザに公開する．このときユーザは先程の手順によりこのプロバイダキーが正規の VM セキュアプロセッサによって発行されたものであることを検証できる．
2. ユーザは自らの VM のストレージをこの公開鍵 pk_{prob} によって暗号化してプロバイダに送信する．
3. プロバイダは VM 実行時にプロバイダキーとこの暗号化ストレージを VM セキュアプロセッサに読み込む．
4. VM セキュアプロセッサはこのプロバイダキーが正規のプロセッサによって発行されたものであることを検証した上で，秘密鍵 sk_{prob} を復号し，それをもとにストレージを復号，実行する．

このようなストレージは同一のプロバイダキーを与えられた正規の VM セキュアプロセッサであればどの個体によっても読み書き可能であるが，プロバイダなどの他者にはファイル全体の置き換えを除いて読み書きは出来ない．このようなストレージをプロバイダストレージとする．

プロバイダストレージを用いることによりユーザは VM を正規のプロセッサ上でのみ動作させることが出来，またプロバイダはゲスト VM の平文に触れることなくクラウドのスケーリングやマイグレーションを行うことが出来る．

また，自ら生成したプロバイダキーを構成する 4 つすべてをそれらを用いて暗号化したプロバイダストレージとともに配布することにより，このプロバイダストレージはすべての正規の VM セキュアプロセッサで読み書き可能となる．

ただし，プロバイダストレージから読み取った内容をユーザの意思に依らず別の VM やプロバイダストレージに出力することがないようプロバイダストレージの取り扱いには注意が必要である．

第5章　VMセキュアプロセッサの実装

本研究では，VM セキュアプロセッサの実装の一部として，メモリの暗号化部分の実装と評価を行った．ここでは，その実装について説明する．

5.1　プロセッサ

本稿では実装の基本となるプロセッサとして，オープンソースの RISC プロセッサである OpenRISC [17] を用いた．今回用いたのはその実装の一つである OpenRISC 1200 である．OpenRISC 1200 はハーバードアーキテクチャの 32 ビットスカラプロセッサで，近年の x86 などの CISC のスーパースカラプロセッサに比べれば非常に単純な制御構造となっている．OpenRISC コアに FPGA のピンアサインや各種制御モジュールを追加し，FPGA に書き込んで動作できるものとして ORPSoC が公開されており，本稿の実装は ORPSoCv2 をベースとしている．

図 5.1 は OpenRISC を VM セキュアプロセッサとする場合に追加・改変する箇所を示したものである．本稿ではこの図における Encryption Unit を実装した．これにより，メモリへ読み書きされるデータはすべて暗号化されるようになった．

なお，完全な VM セキュアプロセッサでは暗号化部は MMU と連携し，VM ごと，書き込みごとに鍵を変更するが，今回の実装では MMU と鍵の管理は対象外であり，そのため暗号化の鍵は固定となっている．

5.2　FPGA 評価ボード

本稿の実装では OpenRISC を動作させる環境として Digilent 社の FPGA 評価ボードである Atlys を用いた．Atlys は FPGA として Xilinx Spartan-6 LX45 を，外部の揮発性メモリとして DDR2 メモリを搭載している．Spartan-6 LX45 はあまり大きな FPGA ではないが，6 章に示すように，OpenRISC を実装するだけであれば十分な面積を持つ．

ORPSoCv2 の Atlys 用の実装では DDR2 制御の IP コアの前にキャッシュが実装されており，これが事実上の L2 キャッシュとなっている．起動後に利用するか否かをレジスタで設定する L1I, L1D 両キャッシュとは異なり，このキャッシュはプロセッサ動作中は常時利用される．L2 キャッシュと DDR2 制御モジュール間のデータバスは，ORPSoCv2 内で唯一すべてのデータが通過する 128bit 幅の通信路である．

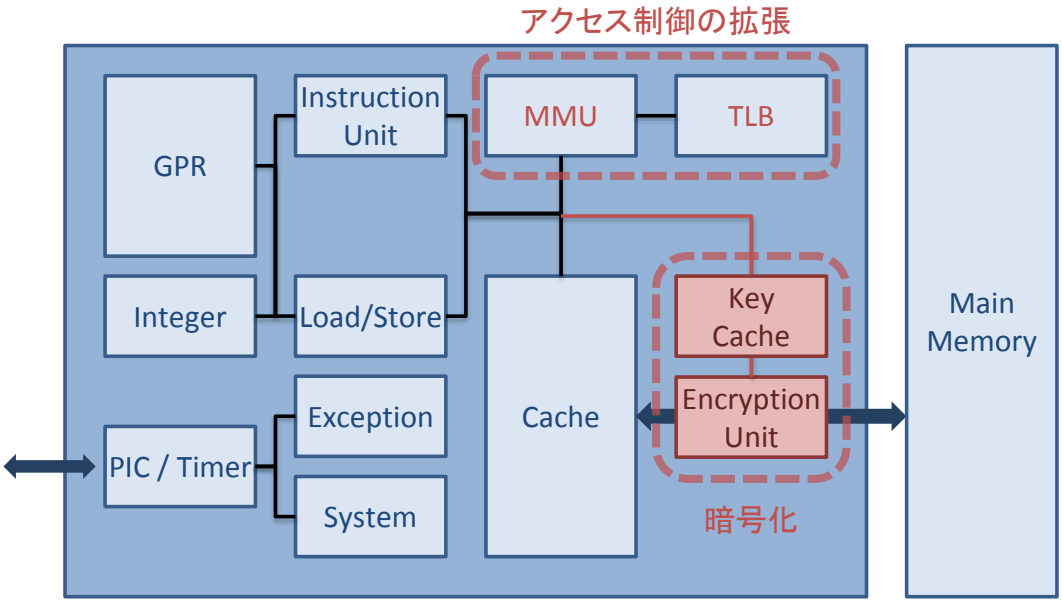

図 5.1: OpenRISC への機能拡張

5.3 暗号化方式

本稿でメモリの暗号化方式としては AES-CTR を用いた.

AES-CTR はカウンターモードとも呼ばれ，**図 5.2** のように，メモリアドレスから生成されるカウンタと書き込みごとに一意な値（Nonce）とを組み合わせたものを固定の AES 鍵で暗号化し，それを入力（平文・暗号文）を排他的論理和で掛け合わせたものを出力（暗号文・平文）とする，128bit 幅のブロック暗号を元にしたストリーム暗号化方式である.

すなわち，これは AES を擬似乱数発生器とするワンタイムパッド方式の暗号であると言える.

AES-CTR は AES の他のモードと比較して，出力が直前の値に依存しないためにランダムアクセスが容易であり並列化も可能である，AES の復号機構を暗号化機構と別に用意する必要がなく回路面積の点で優れるなど，プロセッサがメモリ暗号化に用いやすい特徴を持つ.

なお，AES-CTR では Nonce を書き込みごとに変えることを必須としているが，現時点で VM セキュアプロセッサの鍵の管理機構はまだ実装されておらず，そのため Nonce が固定となっており，差分攻撃により容易に平文を推定できる状態となっている.

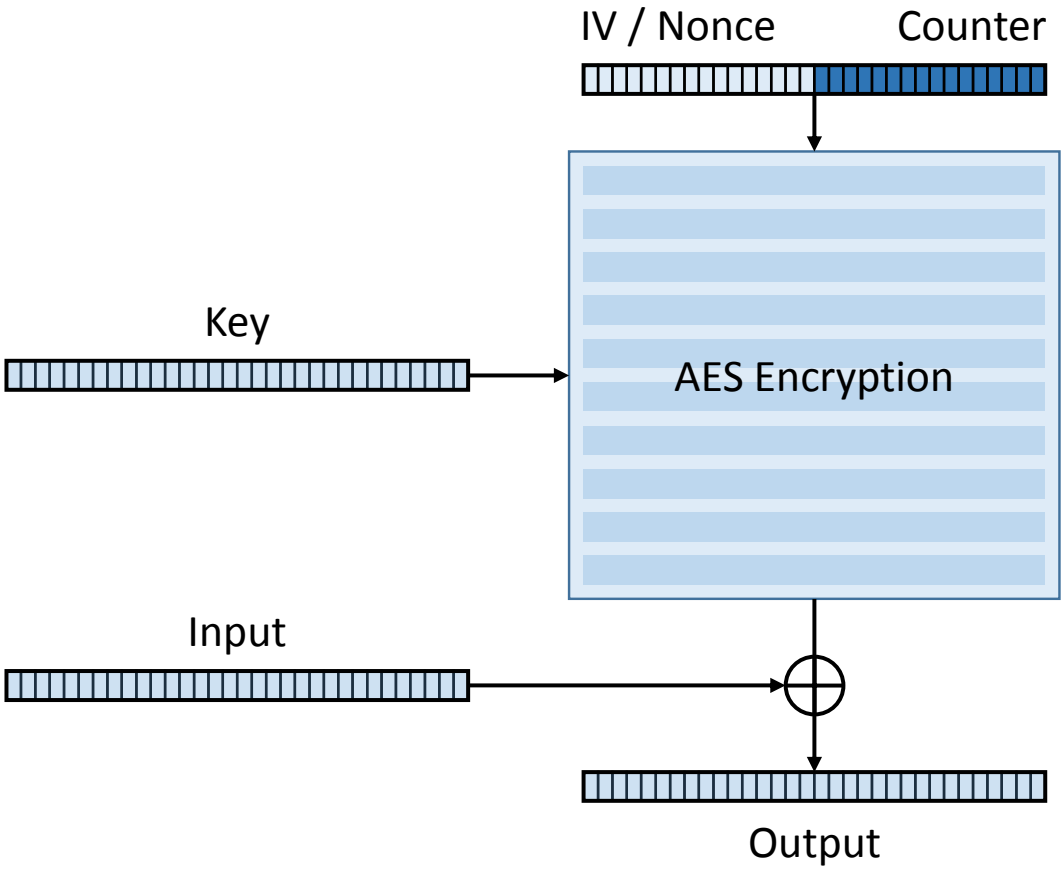

図 5.2: AES-CTR での暗号化

第6章　評価

VM セキュアプロセッサにおいて，プログラムの実行性能に大きく影響を与えるのはメモリの暗号化に伴う遅延である．

ここでは，メモリ暗号化の遅延を考慮した場合の性能の見積もりと，実際に実装を行ったメモリ暗号化に関して行った評価について述べる．

6.1　一般的なプロセッサにおける性能への影響

一般的なプロセッサを VM セキュアプロセッサとした場合の性能低下に関して，プロセッサシミュレータである鬼斬弐を用いて評価を行った．

鬼斬弐は当研究室で提案しているプロセッサシミュレータで，分岐予測やアウトオブオーダ実行を採用しており，OpenRISC よりも高度なプロセッサにおける命令実行の流れをシミュレートすることが出来る．

表 6.1 は，今回使用したパラメータのリストである．メインメモリへのアクセス時間として鬼斬弐の標準である 200 サイクル（Base）と，今回の実装で利用した AES-CTR による遅延 21 サイクルを足したものを比較した．

ベンチマークプログラムとしては SPEC CPU2006 の INT と FP の全 29 本，入力データセットには ref を用い，最初の 1G 命令をスキップした後の 100M 命令において IPC を測定した．

その測定結果として， Base における IPC を 1 としてメモリアクセス遅延を加えた場合の相対 IPC を**図 6.1** に示す．メモリアクセスの遅延が増加したことによりワーキングセットの大きな，すなわちメモリアクセスの多く発生するベンチマークにおいて最大 6.2% の性能低下が見られるが，全体では平均で 0.5% の性能低下となった．これは通常の利用においてはほとんど無視できる値であるが，耐ソフトウェアタンパプロセッサのように，暗号化を行わずアクセス制御のみを行う動作モードを VM セキュアプロセッサに実装する意義もあると言えるだろう．

6.2　OpenRISC における性能への影響

次に，今回実装したメモリ暗号化機構の性能評価として，FPGA 上での使用 LUT 数と FPGA 上の OpenRISC における Linux の起動時間の測定を行った．その結果を**表 6.2** に示す．

表 6.1: 鬼斬弐におけるプロセッサの構成

ISA	Alpha21164A
pipeline stages	Fetch:3, Rename:2, Dispatch:2, Issue:4
fetch width	4
issue width	Int:2, FP:2, Mem:2
instruction window	Int:32, FP:16, Mem:16
branch predictor	8KB g-share
BTB	2K entries, 4way
RAS	8 entries
L1C	32KB, 4way, 3cycles, 64B/line
L2C	4MB, 8way, 15cycles, 128B/line
main memory	200cycles or 221cycles

表 6.2: OpenRISC における暗号化機構の有無による違い

	Base	+ Memory Encryption
L2C writeback + read	151cycle	193cycle
logic LUT	12,037 (44%)	20,558 (75%)
memory LUT	456	589
route-thru LUT	49	42
Linux boot from reset	16.54sec	17.08sec
Linux boot from display out	5.88sec	6.42sec

まず，鬼斬弐でのシミュレーションと比べ L2 キャッシュミス時の遅延が少なく，追加した暗号化機構による遅延が相対的に大きくなっている．これは FPGA で実装された OpenRISC の動作周波数に対して DDR2 メモリが非常に高速であるためだと考えられる．

次に， Linux の起動時間としてはリセットからシェル起動可能までの時間と，画面出力を開始してからシェル起動可能までの時間を計測した．前者はフラッシュROM からメインメモリへのカーネルイメージの転送を含んでおり，後者の方がベンチマークとしてよりパラメータの違いを反映した計測となっている．Linux の起動はメモリアクセスが多く，先の鬼斬弐での例のように遅延の挿入の影響を受けやすいプログラムと言える．そのような状況下で， L2 ミス時の遅延が最大 28 % 増加しているにも関わらず性能低下が 9% に抑えられているのは，通常用途においては実用の範囲内であると言えるだろう．

一方で， FPGA 上での面積に関しては大幅な増加が見られる．この増加分のほとんどは AES の各ステージの演算によるものである．これは OpenRISC や Spartan-6 FPGA があまり大きくないために相対的に増加の比率が大きくなってしまったと考え

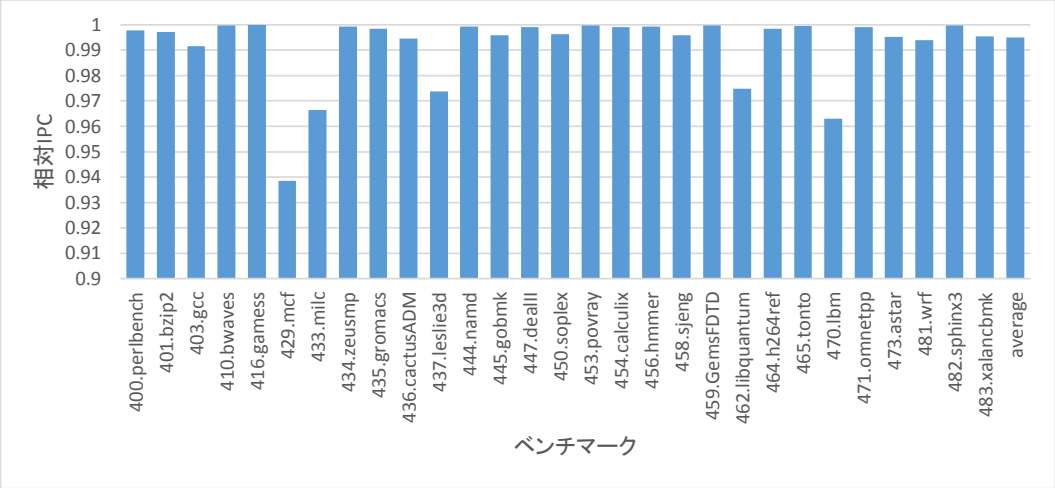

図 6.1: 暗号化機構の追加による性能低下

られる．特に面積増加以上に残り面積の大幅な減少は問題であり，現在の構成のまま VM セキュアプロセッサの実装を続けていけば FPGA 面積の制約は大きな障害となりかねない．VM セキュアプロセッサ実装の基盤とするプロセッサと FPGA について，今一度検討する必要があるだろう．

なお，今回の実装では OpenRISC の既存の構造をほとんど改変しないよう暗号化機構を実装したが，面積効率または実行速度を優先する実装も可能である．

第7章　おわりに

本稿ではVMを保護対象としたセキュアプロセッサであるVMセキュアプロセッサのもつ具体的な機能に関して述べ，そのメモリ暗号化機構の実装と性能評価を行った．その結果，VMセキュアプロセッサの性能は十分実用に耐えるものと予想されたが，面積評価においてがFPGAの大きさに限界が見られた．

今後は仮想化支援や鍵の管理などVMセキュアプロセッサの実装を進めるが，FPGAと実装の基本となるプロセッサの組み合わせを再度検討する必要もあるだろう．また，性能だけでなくセキュリティ面での評価も今後の課題である．

参考文献

[1] Christopher W. Fletcher, Marten v. Dijk, and Srinivas Devadas. A secure processor architecture for encrypted computation on untrusted programs. pp. 3–8, 2012.

[2] Yubin Xia, Yutao Liu, and Haibo Chen. Architecture support for guest-transparent vm protection from untrusted hypervisor and physical attacks. In *High Performance Computer Architecture (HPCA2013), 2013 IEEE 19th International Symposium on*, pp. 246–257, Feb 2013.

[3] Intel Corporation. *Intel 64 and IA-32 Architectures Software Developer's Manual*, pp. 144, 2013.

[4] Amazon Web Services. 国内のお客様の導入事例 Powered by AWS クラウド. 2014.

[5] BSA The Software Alliance. コンプライアンス・ギャップ bsa グローバルソフトウェア調査 2014 年 6 月. pp.9, 2014.

[6] G. E. 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.

[7] 橋本幹生, 春木洋美, 川端健. オープンソース os と共存可能なセキュリティプロセッサ技術, Vol. 60, No. 6, pp. 44–47, 2005.

[8] Xiaoxin Chen, Tal Garfinkel, E. C. Lewis, Pratap Subrahmanyam, Carl A. Waldspurger, Dan Boneh, Jeffrey Dwoskin, and Dan R. Ports. Overshadow: a virtualization-based approach to retrofitting protection in commodity operating systems. In *ACM SIGOPS Operating Systems Review*, Vol. 42, pp. 2–13. ACM, 2008.

[9] Jakub Szefer and Ruby B. Lee. Architectural support for hypervisor-secure virtualization. *SIGPLAN Not.*, Vol. 47, No. 4, pp. 437–450, March 2012.

[10] 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. *2014* **年 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.

[11] David Lie, Chandramohan A. Thekkath, and Mark Horowitz. Implementing an untrusted operating system on trusted hardware. In *ACM SIGOPS Operating Systems Review*, Vol. 37, pp. 178–192. ACM, 2003.

[12] Craig Gentry, Shai Halevi, and NigelP Smart. Homomorphic evaluation of the aes circuit. Vol. 7417, pp. 850–867, 2012.

[13] Oded Goldreich and Rafail Ostrovsky. Software protection and simulation on oblivious rams. *J. ACM*, Vol. 43, No. 3, pp. 431–473, may 1996.

[14] John L. Henning. Spec cpu2006 benchmark descriptions. *SIGARCH Comput. Archit. News*, Vol. 34, No. 4, pp. 1–17, sep 2006.

[15] Koki Murakami, Tsuyoshi Yamada, Rie S. Yamaguchi, Masahiro Goshima, and Shuichi Sakai. A cloud architecture for protecting guest's information from malicious operators with memory management. In *Proceedings of the 4th ACM conference on Data and application security and privacy*, pp. 155–158. ACM, 2014.

[16] S. Jin, J. Ahn, J. Seol, S. Cha, J. Huh, and S. Maeng. H-SVM: Hardware-assisted Secure Virtual Machines under a Vulnerable Hypervisor. *Computers, IEEE Transactions on*, Vol. PP, No. 99, pp. 1–1, 2015.

[17] Damjan Lampret, Chen-Min Chen, Marko Mlinar, Johan Rydberg, Matan Ziv-Av, Chris Ziomkowski, Greg McGary, Bob Gardner, Rohit Mathur, and Maria Bolado. Openrisc 1000 architecture manual. *Description of assembler mnemonics and other for OR1200*, pp. 1–340, 2003.

発表文献

主著論文

1. 宮永 瑞紀, 山田 剛史, 山口 利恵, 五島 正裕, 坂井 修一:
   VM セキュアプロセッサの構成,
   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

1. 千田 拓矢, 坂井 修一, 五島 正裕, 山口 利恵, 宮永瑞紀:
   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-010 (2014).

採録済み・発表予定

1. 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).

謝辞

本修士論文は，筆者が東京大学 大学院 情報理工学系研究科 電子情報学専攻在学中に坂井・五島研究室および坂井研究室において行った研究をまとめたものです.

本研究を進めるにあたって，坂井修一教授には，不自由なく研究できる環境の構築から研究発表の推敲に至るまで，相談会などを通じ様々な御指導を頂きました.

国立情報学研究所の五島正裕教授には，研究方針をはじめ多くの相談に乗って頂き，御指導頂きました.

山口利恵特任准教授には，この分野における研究方法などを御指導頂きました.

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

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

なお，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す.