Nested SEV：ネストした仮想化へのAMD SEVの適用

瀧口 和樹[1]　光来 健一[1]

概要：
パブリッククラウドの仮想マシン（VM）上で重要なデータを扱うと，クラウドの内部犯などからVM内の機密情報を盗まれる可能性がある．このリスクを低減するために，AMDプロセッサではSEVと呼ばれるVMのメモリを透過的に暗号化するセキュリティ機構が提供されている．一方，クラウドにおいてVMの中でVMを動作させるネストした仮想化を用いた様々なシステムが提案されているが，ネストした仮想化を用いるシステムにはSEVを適用することができない．本稿では，ネストした仮想化にSEVを組み合わせることを可能にするNested SEVを提案する．SEVの適用方法によって4種類のシステム構成が考えられるため，Nested SEVは透過的SEV，SEVパススルー，SEV仮想化の3つの方式を提供する．透過的SEVは外側のVMに適用されているSEVの機能を用いて内側のVMのすべてのメモリを暗号化する．SEVパススルーは内側のVMに外側のVMのSEVをそのまま適用する．SEV仮想化は内側のVMに専用の仮想SEVを適用する．これらをXen，KVM，BitVisorに実装し，I/O性能を調べる実験を行った．

1. はじめに

近年，ユーザに仮想マシン（VM）を提供するパブリッククラウドが様々な用途に活用されている．それに伴い，クラウド上で重要なデータが扱われるようになっており，クラウドの内部犯などからVM内の機密情報を盗まれるリスクが高まっている．そのため，AMDプロセッサではSecure Encrypted Virtualization (SEV) [1]と呼ばれるVMのセキュリティ機構が提供されている．SEVはVMのメモリを透過的に暗号化し，VMの内部でのみ復号可能にする．そのため，VM外部のハイパーバイザ等によってメモリ内部の機密情報が盗聴されるのを防ぐことができる．Google CloudやMicrosoft Azureなどで SEVを適用したConfidential VM [2,3]が提供されている．

一方，クラウドにおいてネストした仮想化 [4]を用いた様々なシステムが提案されている．ネストした仮想化はVM内でVMを動作させる技術であり，本稿では外側のVMをL1 VM，内側のVMをL2 VMと呼ぶ．例えば，クラウドのVMをホストとして用いることにより仮想クラウドを提供することができる [5,6]．しかし，現状ではネストした仮想化を用いるシステムにはSEVを適用することができない．SEVを有効にしたL1 VM内ではL2 VMを作成することができず，SEVを適用していないL1 VM内にSEVを有効にしたL2 VMを作成することもできない．ネストした仮想化とSEVを併用するシステムも提案されている [7,8]が，SEVを適用して実装することはできていない．

本稿では，ネストした仮想化にSEVを組み合わせることを可能にするNested SEVを提案する．L1 VMとその中のL2 VMにSEVを適用するかどうか，SEVの暗号鍵を同じにするかどうかで4種類のシステム構成が考えられる．L2 VMにSEVを適用するために，Nested SEVは透過的SEV，SEVパススルー，SEV仮想化の3つの方式を提供する．透過的SEVはL1 VMに適用されているSEVの機能を用いてL2 VMのすべてのメモリを同じ鍵で暗号化する．SEVパススルーはL1 VMに適用されているSEVをそのままL2 VMにも適用し，L2 VMが指定したメモリを同じ鍵で暗号化する．SEV仮想化は仮想SEVをL2 VMに適用し，L2 VMが指定したメモリを異なる鍵を用いて暗号化する．

SEVによって用いられるページテーブルエントリ（PTE）のCビットを適切に制御できるようにすることで，SEVを有効にしたL1 VMでXen，KVM，BitVisorを動作させられるようにした．透過的SEVをXenおよびKVMに実装し，SEVに非対応のLinuxをL2 VMで動作させられるようにした．SEVパススルーをXenおよびBitVisorに実装し，SEVを適用したL2 VMにおいてPCIパススルーを可能にした．SEV仮想化をKVMに実装し，L1 VMに適用されるSEVと同等の鍵管理をL2 VMに対しても行えるようにした． これらを用いて，ネストした仮想化とSEVを組み合わせたシステムのI/O性能を測定した．

[1] 九州工業大学

以下，2 章では AMD SEV やネストした仮想化について述べる．3 章ではネストした仮想化と SEV の組み合わせおよび，SEV の適用方式について述べる．4 章ではハイパーバイザの SEV への対応および，透過的 SEV，SEV パススルー，SEV 仮想化の実装について述べる．5 章ではネストした仮想化と SEV を組み合わせたシステムを用いて行った実験について述べる．6 章で関連研究に触れ，7 章で本稿をまとめる．

2. 背景

2.1 Secure Memory Encryption (SME)

SME はシステムのメモリの透過的な暗号化を可能にする AMD プロセッサのセキュリティ機能である．暗号化・復号はメモリコントローラで行われ，プロセッサやデバイスがメモリにデータを書き込む時に暗号化され，メモリからデータを読み込む時に復号される．暗号化には 128 ビットの鍵長の AES が用いられ，暗号鍵は AMD Secure Processor (AMD-SP) によって起動時に生成される．SME はコールドブート攻撃やメモリバスの傍受，NVDIMM の盗難などからデータを保護するために用いられる．

SME を用いると，OS がページテーブルエントリ（PTE）の C ビットを 1 に設定した時に，対応する物理ページへのすべてのアクセスが暗号化される．C ビットの位置はプロセッサによって異なっており，PTE に格納されている物理アドレスの最上位ビット（47 ビット目や 51 ビット目）が使われることが多い．Transparent SME (TSME) を用いると C ビットの値に関わらずすべての物理ページが暗号化される．

SME を仮想化支援機構の AMD Virtualization (AMD-V) で用いられるネストしたページテーブル（NPT）と組み合わせることも可能である．NPT は VM の物理アドレスをホストの物理アドレスに変換するためにハイパーバイザによって用いられる．NPT のエントリの C ビットを 1 に設定すると，対応する物理ページに対する VM からのアクセスが暗号化される．

2.2 Secure Encrypted Virtualization (SEV)

SEV は VM のメモリの透過的な暗号化を可能にする AMD EPYC プロセッサのセキュリティ機能である．SEV を有効にした VM は SEV ゲストと呼ばれ，SEV ゲストを動作させるハイパーバイザは SEV ホストと呼ばれる．SME と同様に，VM 内の OS が PTE の C ビットを 1 に設定することによって対応する物理ページを暗号化することができる．一方，SEV の暗号鍵は AMD-SP によって VM ごとに割り当てられ，暗号化された VM のメモリはその VM 内部でのみ復号可能となる．そのため，ハイパーバイザや他の VM，デバイスによるメモリの盗聴を防ぐことができる．ただし，VM が DMA に用いるメモリ領域はハイパーバイザやデバイスからアクセスできるように，C ビットを 0 に設定することで暗号化しないようにする必要がある．

SME とは異なり，SEV では命令フェッチとページテーブルへのアクセスは C ビットの値に関わらず常に暗号化される．これにより，ハイパーバイザによる VM 内のコードやページテーブルの改ざんを防ぐことができる．また，64 ビットモードでなく，ページテーブルエントリを 64 ビットに拡張する PAE が無効の時に行うメモリアクセスも常に暗号化される．SEV を SME と組み合わせることも可能である．VM 内の PTE の C ビットが 0 に設定されている時に，対応する NPT のエントリの C ビットが 1 に設定されていれば，その物理ページは SME によって暗号化される．

SEV は VM のメモリを暗号化するだけであり，レジスタなどの状態は暗号化しない．そのため，VM の外からレジスタを経由して機密データを盗まれたり，レジスタを書き換えて攻撃される可能性がある．このような攻撃を防ぐために，SEV-Encrypted State (SEV-ES) は VM の状態を暗号化する．VM の実行中に I/O アクセスのようなハイパーバイザの関与が必要な動作が行われると，VM に対して例外が発生する．その際に，VM は公開する必要がある状態のみをハイパーバイザとの共有領域に書き込んでから，ハイパーバイザに制御を渡す．

VM のメモリを暗号化しても，VM に割り当てるメモリを古いコピーに置き換えるリプレイ攻撃やメモリデータの破壊は可能である．このような攻撃を防ぐために，SEV-Secure Nested Paging (SEV-SNP) は物理ページとその所有者の対応を管理するテーブルを用いる．アクセスしようとした物理ページの所有者が VM の所有者と一致しなければページフォールトを発生させ，ハイパーバイザによる改ざんから保護する．テーブルには VM の物理アドレスとの対応も記録され，アクセスしようとした VM の物理アドレスと一致しなければページフォールトが発生するため，物理ページのエイリアス作成によるデータ破壊からも保護できる．また，VM が物理ページを検証することにより，VM 内での物理ページの入れ替えを検知できる．SEV-SNP では他にも様々な機能が提供されている．

2.3 ネストした仮想化

ネストした仮想化 [4] は VM の中で VM を動作させるための技術である．ハイパーバイザ上で動作している VM の中で別のハイパーバイザを動作させ，そのハイパーバイザ上でさらに VM を動作させる．ネストした仮想化を用いるシステムは図 1 のように 3 つのレイヤで構成される．従来のハイパーバイザは L0 と呼ばれるレイヤで動作し，L0 ハイパーバイザ（あるいはホストハイパーバイザ）と呼ばれる．その上の VM は L1 と呼ばれるレイヤで動作し，L1 VM（あるいはホスト VM）と呼ばれる．その VM の中で

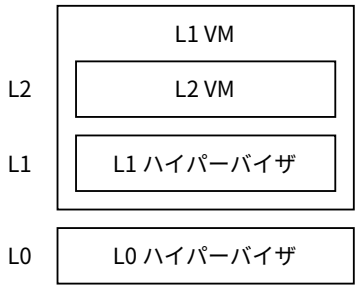


図 1 ネストした仮想化のレイヤ

表 1 ネストした仮想化と SEV の組み合わせ

	L0	L1 VM	L2 VM
構成 1	非暗号化	SEV	SEV（L1 と同じ鍵）
構成 2			SEV（L1 と異なる鍵）
構成 3			非暗号化
構成 4		非暗号化	SEV

動作するハイパーバイザは L1 ハイパーバイザ（あるいはゲストハイパーバイザ）と呼ばれる．その上の VM は L2 と呼ばれるレイヤで動作し，L2 VM（あるいはゲスト VM）と呼ばれる．

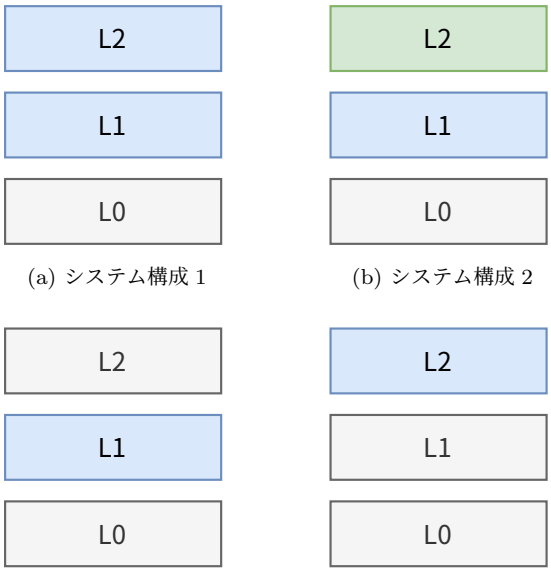
現状では，ネストした仮想化には SEV を適用することができない．VM に SEV を適用するにはその中で動作するシステムソフトウェアに SEV ゲストとしての対応が必要になる．Linux などの多くの OS にはすでに SEV ゲスト対応が含まれているが，ハイパーバイザではまだ対応が行われていない．例えば，KVM 上の L1 VM で SEV を有効にすると，その中で L2 VM を起動することはできない．また，Xen や BitVisor などのタイプ 1 ハイパーバイザは SEV を有効にした L1 VM 内では起動することができない．そのため，ネストした仮想化に SEV を適用することによって構築可能となるシステムについても十分に検討されていない．

3. Nested SEV

本稿では，ネストした仮想化に SEV を組み合わせることを可能にする Nested SEV を提案する．システム構成として表 1 のような 4 種類の組み合わせが考えられる．L0 で SME を用いるかどうかはシステム内部でアクセスできる範囲に影響を及ぼさないため，ここでは区別しない．

3.1 Nested SEV を用いたシステム構成

システム構成 1 では図 2(a) のように，SEV を使って L1 VM と L2 VM の両方のメモリを同じ鍵で暗号化する．これにより，L0 ハイパーバイザから L1 ハイパーバイザと L2 VM を保護することができる．一方で，L1 ハイパーバイザは L2 VM に自由にアクセスすることができる．例えば，パブリッククラウドが L0 ハイパーバイザを動作させ，ユーザが L1 VM の中に L2 VM を作成することが考えられる．この構成により，ユーザはパブリッククラウド上に仮想プライベートクラウドを構築 [5] し，SEV で保護することができる．また，L1 ハイパーバイザは VM イントロスペクション（VMI）[9] を用いて L2 VM のメモリデータを安全に監視することができる．SEV で保護された別の L1 VM で侵入検知システム（IDS）を動作させ，L1 ハイパーバイザ内のエージェント経由で L2 VM のメモリデータを取得して安全に監視することもできる [7].



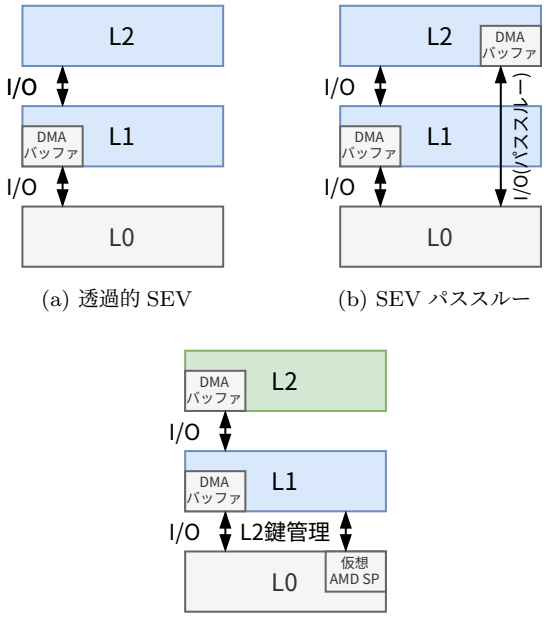
(a) システム構成 1　(b) システム構成 2

(c) システム構成 3　(d) システム構成 4

図 2 Nested SEV を用いたシステム構成

システム構成 2 では，システム構成 1 と同様に L1 VM と L2 VM の両方のメモリを暗号化するが，その際に図 2(b) のように異なる暗号鍵を用いる．これにより，L0 ハイパーバイザから L1 ハイパーバイザを保護しつつ，L1 ハイパーバイザから L2 VM を保護することができる．例えば，パブリッククラウドが L0 ハイパーバイザを動かし，別のクラウドプロバイダが L1 VM に仮想パブリッククラウドを構築して SEV で保護することが考えられる．そのユーザに提供する L2 VM も SEV で保護することができる．また，クラウド内に送り込んだユーザの L1 ハイパーバイザを SEV で保護しつつ，その上でクラウドの L2 VM を安全に動作させることもできる [8]．これにより，ユーザがクラウドサービスの通信を安全に監視することができる．

システム構成 3 では図 2(c) のように，L1 VM にだけ SEV を適用してメモリを暗号化し，L2 VM のメモリは暗号化しない．これにより，L0 ハイパーバイザから L1 ハイパーバイザだけを保護することができる．例えば，パブリッククラウドが L0 ハイパーバイザを動かし，ユーザが L1 VM と L2 VM を使うことが考えられる．この場合，L2 VM のメモリは暗号化されないため，クラウドとユーザの両方が VMI を用いて L2 VM のメモリデータの監視を行うことができる．また，L0 で SME を使って L2 VM のメモ



(a) 透過的 SEV　(b) SEV パススルー

(c) SEV 仮想化

図 3　Nested SEV の適用方式

リを暗号化しなければ L2 VM の性能がよくなるため，機密データを扱わない処理だけを L2 VM に行わせることも考えられる．

システム構成 4 では逆に，図 2(d) のように L1 VM のメモリは暗号化せず，L2 VM にだけ SEV を適用してメモリを暗号化する．これにより，L0 ハイパーバイザと L1 ハイパーバイザの両方から L2 VM を保護することができる．例えば，パブリッククラウドが L0 ハイパーバイザ上で動作する通常の L1 VM を提供し，ユーザが L2 VM を使って安全に Unikernel [10] を動かすことが考えられる．Unikernel はライブラリ OS を使って構築された，VM 内で動作するアプリケーションのシステムイメージである．アプリケーションごとに SEV の異なる鍵でメモリを暗号化することにより，より強力な分離が可能となる．システム構成 2 でも同様にアプリケーションの分離が可能であるが，この構成では L1 VM の管理をクラウド側で行うことができる．

3.2　Nested SEV の適用方式

L2 VM に SEV を適用するために，Nested SEV は図 3 の 3 つの方式を提供する．

- **透過的 SEV**
  SEV の機能の一つである Virtual Transparent Encryption (VTE) を用いて，L2 VM のメモリを透過的に暗号化する．VTE は L2 VM のページテーブルエントリの C ビットの値に関わらずすべてのメモリ領域を暗号化する．L2 VM のメモリを暗号化するために用いられる鍵は L1 VM と同じである．L2 VM が AMD-V を使って動作しており，L1 ハイパーバイザによって提供される仮想デバイスのみを利用する場合にこの方式を用いることができる．L2 VM のメモリが透過的に暗号化されるため，L2 OS の SEV ゲスト対応は不要である．この方式はシステム構成 1 を実現するために用いることができる．
- **SEV パススルー**
  L1 VM に適用されている SEV をそのまま L2 VM にも適用する．L2 VM のメモリを暗号化するために用いられる鍵は L1 VM と同じである．透過的 SEV と異なり，L2 OS がページテーブルの C ビットを 0 に設定することで任意のメモリ領域の暗号化を解除できる．そのため，L0 ハイパーバイザによって提供される仮想 PCI デバイスへのパススルーアクセスを行うことができる．また，この方式は L2 VM が AMD-V を使って動作していない場合でも用いることができる．L2 OS がページテーブルの C ビットを適切に設定してメモリ暗号化を制御する必要があるため，L2 OS の SEV ゲスト対応が必要になる．この方式もシステム構成 1 を実現するために用いることができる．
- **SEV 仮想化**
  SEV を仮想化した**仮想 SEV** を L1 ハイパーバイザに提供し，仮想 SEV を L2 VM に適用する．そのため，L1 ハイパーバイザは SEV ゲストでありつつ，L0 ハイパーバイザと同様に SEV ホストとして動作する．透過的 SEV や SEV パススルーとは異なり，仮想 SEV は L1 VM のメモリを暗号化するために用いられる鍵とは異なる鍵を使って L2 VM のメモリを暗号化する．L2 VM が AMD-V を使って動作している場合に用いることができ，PCI パススルーを行うこともできる．L2 OS がページテーブルの C ビットを適切に設定する必要があるため，L2 OS の SEV ゲスト対応が必要になる．システム構成 2，4 を実現するために用いることができる．

4.　実装

L0 ハイパーバイザとして KVM，L1 ハイパーバイザとして Xen，KVM，BitVisor，L2 OS として Linux を用いて Nested SEV を実装した．L1 VM に SEV を適用するシステム構成を対象とした．これらのハイパーバイザに対する透過的 SEV，SEV パススルー，SEV 仮想化の実装状況は表 2 の通りである．現在のところ，SEV-ES，SEV-SNP には対応していない．

Xen には最初に起動してハードウェアの制御を行い，他の VM を管理する特権を持った VM である Dom0 と，通常の VM である DomU が存在する．VM には AMD-V を用いずに準仮想化を行う PV，AMD-V を用いる HVM，AMD-V と準仮想化の両方を用いる PVH の 3 種類がある．Dom0 としては PV と PVH が利用できるが，KVM 上で

表 2 実装状況

ハイパーバイザ	透過的 SEV	SEV パススルー	SEV 仮想化
Xen (PV)	不可	○	不可
Xen (HVM/PVH)	○	未実装	未実装
KVM	○	未実装	○
BitVisor	不可	○	未実装

Xen を動作させる場合，PVH は SEV の有無に関わらず動作しなかったので対象としていない．KVM と BitVisor は AMD-V を用いて VM を動作させており，BitVisor はデバイスアクセスに準パススルーを用いる．

4.1 L1 ハイパーバイザの SEV ゲスト対応

L1 ハイパーバイザに L1 OS と同等の SEV ゲスト対応を行い，SEV を有効にした L1 VM 内で実行できるようにした．KVM については，SEV ゲスト対応が含まれている Linux として動作するため，新たに SEV ゲスト対応を行う必要はなかった．SEV ゲスト対応として，PTE を作成・更新する際に適切に C ビットを設定する．逆に，PTE から物理アドレスを取得する際には物理アドレスの一部に含まれる C ビットを除去する．

L0 ハイパーバイザからアクセスが必要な VRAM などのメモリ領域が SEV によって暗号化されると，L1 ハイパーバイザを正常に動作させることができない．このようなページについては L1 ハイパーバイザの対応する PTE の C ビットに 0 を設定することにより，暗号化せずにアクセスする．L0 ハイパーバイザの仮想デバイスに対する DMA で使われるバッファについても同様であり，専用のバウンスバッファを用意して暗号化しないようにする．読み込みの際にはバウンスバッファから暗号化されている DMA バッファにデータをコピーし，書き込みの際には DMA バッファからバウンスバッファにデータをコピーする．64 ビットモードでなく PAE が無効の時はメモリが常に暗号化され，バウンスバッファを用いることができない．そのため，L1 ハイパーバイザはレガシーな BIOS ではなく UEFI を使って起動する．

L1 ハイパーバイザが仮想デバイスの MMIO 領域にアクセスすると VM Exit が発生する．その際に，アクセス先の L1 の物理アドレスが L1 VM を制御するために使われる Virtual Machine Control Block (VMCB) に格納される．L0 ハイパーバイザが MMIO 領域へのアクセスをエミュレートするにはこの情報に加えて実行された命令を知る必要がある．しかし，SEV ゲストのメモリは暗号化されているため，メモリ上にある命令をデコードすることはできない．そこで，AMD-V が提供している Decode Assists と呼ばれる機能を用いて，VM Exit の発生要因となった命令のバイト列を VMCB に格納させる．これにより，L0 ハイパーバイザは L1 VM のメモリを参照する必要がなくなるため，メモリが暗号化されていても命令をエミュレートすることができる．

しかし，L0 ハイパーバイザはこれらの機能を用いてもいくつかの命令をエミュレートすることができない．例えば，`MOVS` 命令のようにメモリからメモリへデータを転送する命令が挙げられる．これは，VMCB に格納できるアクセス先の物理アドレスは 1 つであり，オペランドにアドレスを 2 つ以上取る場合，すべての物理アドレスを格納することはできないためである．その上，MMIO 領域へのアクセスの転送元か転送先となる L1 VM のメモリは暗号化されているため，L0 ハイパーバイザからはアクセスすることができない．また，KVM は VM が SEV ゲストとして動作している場合に `REP` プレフィックスが付いたストリング操作命令のエミュレーションに対応していない．そこで，L1 ハイパーバイザが MMIO 領域にアクセスする際にはこれらの命令を使わないようにする．例えば，`memcpy` 関数ではストリング操作命令を使わずに 1 バイトずつデータをコピーする．

4.2 L1 ハイパーバイザ特有の SEV ゲスト対応

AMD-V を用いて L2 VM を作成する場合，L1 ハイパーバイザは `VMSAVE`，`VMLOAD`，`VMRUN` などの仮想化支援命令を実行する．これらの命令は L1 VM 内では実行することができないため VM Exit が発生し，L0 ハイパーバイザが代わりに命令を実行する．これらの命令は L2 VM を制御するために使われる VMCB を参照するが，VMCB が SEV によって暗号化されていると L0 ハイパーバイザからアクセスすることができない．そこで，L1 ハイパーバイザにおいて VMCB が格納されているページに対応する PTE の C ビットに 0 を設定し，VMCB を暗号化しないようにする．また，VMCB 経由で参照される I/O Permission Map (IOPM) と MSR Permission Map (MSRPM) も暗号化しないようにする．

L1 ハイパーバイザは L2 の物理アドレスから L1 の物理アドレスへの変換を行うために NPT を用いる．NPT は L2 VM がメモリアクセスを行う時だけでなく，L0 ハイパーバイザが命令エミュレーションを行う時などにもアクセスされる．NPT が SEV によって暗号化されていると L0 ハイパーバイザからアクセスすることができないため，NPT を暗号化しないようにする．L1 ハイパーバイザが NPT のエントリを割り当てる際には，エントリが格納されている

ページに対応する PTE の C ビットに 0 を設定する．一方，NPT のエントリとして使用しなくなったページについては対応する PTE の C ビットに 1 を設定して，暗号化されるようにする．

4.2.1 Xen

L2 VM において Xen の DomU を HVM または PVH ゲストとして動作させる場合，ハイパーコールが実行された時などに L1 ハイパーバイザは L2 VM 用の NPT を参照する場合がある．NPT は暗号化しないようにしているため，PTE の C ビットを 0 に設定して L1 ハイパーバイザの仮想アドレス空間にマップしてアクセスする．その際に，L1 VM に複数の仮想 CPU が割り当てられている場合にはすべての仮想 CPU に割り込み（IPI）を送り，TLB をフラッシュする必要がある．そのため，L1 VM に割り当てる仮想 CPU が増えると性能が大幅に低下する．

そこで，Xen が PV ゲスト用に提供している map cache 機構を HVM ゲストと PVH ゲストでも用いる．map cache 機構は，物理メモリが 1 対 1 にマップされているダイレクトマップ領域にマップしきれない物理メモリを一時的に仮想アドレス空間に効率よくマップするための仕組みである．この機構は第 1 レベルのページテーブルエントリの不要になったマッピングを置き換える時にのみ，その CPU においてローカルに TLB フラッシュを行う．HVM ゲストと PVH ゲストでも動作するように map cache 機構を拡張し，暗号化を行わずにページをマップできるようにした．

4.2.2 KVM

KVM が動作する Linux にはページを一時的に仮想アドレス空間にマップする機能があるが，64 ビットアーキテクチャでは通常，Xen の map cache 機構に相当する機構は無効になっている．そのため，L1 ハイパーバイザが NPT を操作する際にはダイレクトマップ領域の PTE の C ビットを直接変更する．NPT のエントリを割り当てる際には，ダイレクトマップ領域の PTE の C ビットを 0 に設定して暗号化されないようにする．エントリの解放時には C ビットを 1 に設定して暗号化されるようにする．ダイレクトマップ領域には通常 1 GiB ページを使ってメモリがマップされているが，一部の PTE のみの C ビットを変更することにより，部分的に 2 MiB ページと 4 KiB ページに分割される．そのため，ページテーブルの段数がそれぞれ 1 段と 2 段増えるが，性能に与える影響はわずかである．

4.3 透過的 SEV

透過的 SEV を用いるために，L1 ハイパーバイザは L2 VM 用に使われる VMCB の VTE ビットを 1 に設定し，L2 VM に対して VTE を有効にする．KVM のネストした仮想化の実装は VTE に対応していないため，L1 ハイパーバイザが VTE ビットを設定しても L2 VM のメモリ暗号化には反映されない．そこで，L0 KVM を修正してネストした仮想化で用いられる VMCB の VTE ビットの設定が L2 VM に反映されるようにした．

透過的 SEV では，L2 VM に対しては SEV が有効になっていることを隠す．L2 OS に SEV が有効であるという情報を見せてしまうと，メモリ暗号化が機能しているかどうかのチェックに失敗したり，DMA の際にバウンスバッファへのデータコピーが発生して I/O 性能が低下したりするためである．L1 ハイパーバイザは L2 OS による `CPUID` 命令の実行時に VM Exit を発生させ，SEV 有効ビットを 0 に設定した値を返すことによって，SEV ゲストとして動作していないように見せかける．

L1 ハイパーバイザと L2 VM のメモリは同じ鍵で暗号化されるため，L1 ハイパーバイザは L2 OS の DMA バッファなどに暗号化されたままでアクセスすることができる．一方，L0 ハイパーバイザは L2 VM の暗号化されたメモリにアクセスすることはできないため，L2 VM のメモリ上のデータが L0 ハイパーバイザに漏洩する危険はない．その代わり，L2 OS は L0 ハイパーバイザの仮想デバイスに PCI パススルーを用いてアクセスすることはできない．そのため，準パススルーを用いる BitVisor では透過的 SEV を用いることはできない．

4.4 SEV パススルー

SEV パススルーは L2 VM が AMD-V を使っているかどうかに関わらず適用することができる．AMD-V を使って動作する L2 VM に対しては，L1 ハイパーバイザが L2 VM 用に使われる VMCB の SEV 有効ビットを 1 に設定する．KVM のネストした仮想化の実装は SEV 有効ビットの設定を L2 VM に反映させているため，透過的 SEV の場合と異なり，L0 KVM の修正は不要であった．SEV パススルーでは，L2 OS はページテーブルの C ビットを設定することでメモリ暗号化を制御できる．ただし，64 ビットでなく PAE が無効の OS の場合はすべてのメモリが暗号化されるため，透過的 SEV と同等になる．一方，AMD-V を使わずに動作する L2 VM は常に SEV パススルーの状態になり，L1 VM に適用されている SEV によって自動的にメモリが暗号化される．この場合でも，L2 OS はページテーブルの C ビットを設定することでメモリ暗号化を制御できる．

SEV パススルーを適用した L2 VM では，L0 ハイパーバイザが提供する仮想デバイスへの PCI パススルーを行うことができる．L2 VM が AMD-V を用いている場合，MMIO のパススルーは L0 の仮想デバイスの MMIO 領域を L2 VM の物理アドレス空間にマップすることで実現される．L2 VM 内でこの MMIO 領域にアクセスすると L0 ハイパーバイザに対して VM Exit が発生し，アクセス先の L2 の物理アドレスが L2 VM 用の VMCB に格納される．L1 VM 内にある L2 VM 用の NPT は暗号化しないようにしているため，L0 ハイパーバイザは L2 の物理アドレス

を L1 の物理アドレスに変換することができる．このため，L1 VM が仮想デバイスの MMIO 領域にアクセスした場合と同様に，L0 ハイパーバイザが L2 VM による MMIO 領域へのアクセスをエミュレートすることができる．L2 VM が AMD-V を用いていない場合は，L1 VM 内で MMIO 領域にアクセスした場合と同様に命令エミュレーションを行うことができる．

L2 VM が PCI パススルーを用いて L0 ハイパーバイザの仮想デバイスに DMA でアクセスする場合，L0 ハイパーバイザによって提供される仮想 IOMMU（AMD-Vi や VT-d）が L2 の物理アドレスを L1 の物理アドレスに変換する．L1 ハイパーバイザが IOMMU 用のページテーブルなどを暗号化しないようにし，L2 VM が DMA バッファを暗号化しないようにすることにより，L0 ハイパーバイザからアクセスできるようにする．Xen の PV ゲストのように L1 と L2 の物理アドレスの変換を L2 OS が行う場合や，BitVisor のように L1 と L2 の物理アドレスが一致している場合には IOMMU は必須ではないが，L2 VM に割り当てられているメモリに対してのみ DMA を許可するためには IOMMU が必要となる．

SEV パススルーは SEV ゲストに対応した OS であれば L2 VM で動作させることができる．しかし，従来の SEV ゲストは仮想デバイスに DMA でアクセスする際にはバッファを暗号化しないようにしている．SEV パススルーでは L1 ハイパーバイザと L2 VM のメモリは共通の鍵で暗号化されるため，バッファを暗号化しないようにする必要はない．DMA バッファを暗号化しないようにすると，例えば，BitVisor の透過的にディスクを暗号化する機能を利用しても，L0 ハイパーバイザから DMA バッファを盗聴される．この問題を解決するには，SEV パススルー用の SEV ゲストを開発し，DMA バッファ等を暗号化したままにする必要がある．PCI パススルーが必要ない場合には透過的 SEV を用いればこのような問題は発生しない．

4.4.1 Xen

Linux にはすでに SEV ゲスト対応が含まれているが，Xen の PV ゲストとして起動する場合には SEV ゲストのための処理は実行されない．これはネストした仮想化を使って SEV パススルーを行わない限り，AMD-V を使わずに動作する VM は SEV を利用することができないためである．そこで，Xen が L1 ハイパーバイザとして動作している場合には，Linux を Xen の PV ゲストとして起動する際のカーネル初期化処理で SEV ゲストのために必要となる処理を行う．Dom0 を起動する際にはハイパーバイザが初期ページテーブルを作成するため，ハイパーバイザにおいて C ビットを設定する．PV ゲストとして動作する DomU を起動する際には，初期ページテーブルは Dom0 のユーザ空間のライブラリ `libxenguest` で作成されるため，このライブラリにおいて C ビットを設定する．

PV ゲストではページテーブルの書き換えはハイパーコールを介してハイパーバイザが行うため，L1 ハイパーバイザが L2 OS のページテーブルに C ビットを自動的に付与することも可能である．しかし，ページテーブルの参照に関しては PV ゲストがハイパーコールを介さずに行うことができる．そのため，L2 OS がページテーブルの物理アドレスを参照する際に C ビットを除去する処理が必要となる．このことから，PV ゲストであっても C ビットの操作を行う SEV ゲスト対応が必要である．

L2 OS は仮想 CPU のモデル固有レジスタ（MSR）の一つである `SEV_STATUS` の値を取得して SEV が有効かどうかを判定する．Xen はこの MSR の値の取得を許可していなかったため，この MSR へのアクセスを許可し，`RDMSR` 命令の結果をそのまま返すようにした．

また，L2 OS は `CPUID` 命令を用いて C ビットの位置など SEV ゲスト対応に必要な情報を取得する．この命令は特権命令ではないため，PV ゲストの場合にはハイパーバイザは基本的には介入することができない．そこで，Linux では `CPUID` 命令の前に `UD2` 命令を配置して強制的にハイパーバイザに `CPUID` 命令をエミュレートさせている．

しかし，この手法では PV ゲスト内で `CPUID` 命令を実行するプロセスには適切な値を返すことができない．IvyBridge 以降の Intel プロセッサでは MSR を操作することによって `CPUID` 命令でフォールトを発生させることができ，一部の Intel プロセッサや AMD プロセッサでは MSR の操作で `CPUID` 命令の一部の返り値だけを変更することができる．Xen はこの機能をサポートしているが，KVM 上で L1 ハイパーバイザとして動作させる場合には一部の返り値を変更する機能を利用することができない．

そこで，L1 ハイパーバイザとして Xen を動かす場合にも CPUID フォールトを利用できるようにした．Xen は AMD プロセッサ上で動作する場合には CPUID フォールトを使わないが，KVM の CPUID フォールトのエミュレーションを利用して実装した．SEV 情報を取得するための機能番号を指定して `CPUID` 命令を実行した場合に Xen は 0 を返していたため，PV ゲストに正しい SEV 情報を返すようにする．

DomU が準仮想化デバイスを用いる場合には，図 4 に示す DomU のフロントエンドドライバと Dom0 のバックエンドドライバの通信は共有メモリ経由で行われる．Dom0 と DomU が動作する 2 つの L2 VM のメモリは SEV によって同じ鍵で暗号化されているため，L0 ハイパーバイザに盗聴されることなく通信を行うことができる．Dom0 においてはデバイスが仮想化されないため，PCI パススルーの場合と同様に L0 ハイパーバイザの仮想デバイスに直接アクセスする．

Linux の PV ゲスト実装では，PCI パススルーの際に IOMMU を用いない場合には SWIOTLB が用いられる．

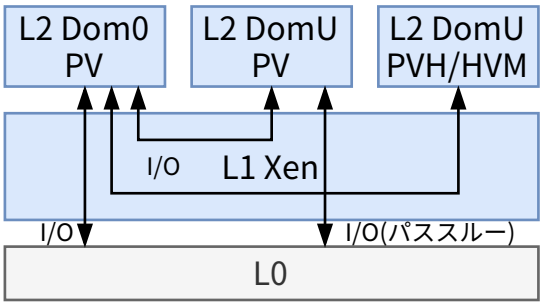


図 4 Xen の I/O

SWIOTLB は DMA にバウンスバッファを用いるが，L2 VM で確保したメモリ領域は L2 の物理アドレスとしては連続していても L1 の物理アドレスとして連続しているとは限らない．そこで，ハイパーコールを呼び出すことにより，L1 の物理アドレスとして連続したメモリ領域を確保する．そして，そのメモリ領域を暗号化しないようにすることで，L0 ハイパーバイザからアクセスできるようにする．

4.4.2 BitVisor

L2 VM が PCI パススルーのための MMIO 領域にアクセスした時，L0 ハイパーバイザは L2 VM 用の NPT を参照することにより，アクセス先の L2 の物理アドレスを L1 の物理アドレスに変換して命令エミュレーションを行うことができる．しかし，KVM はアクセス先の L2 の仮想アドレスから L2 の物理アドレスに変換し，それを L1 の物理アドレスに変換して命令エミュレーションを行う．SEV ゲストのページテーブルは常に暗号化されるため，L0 ハイパーバイザは L2 の仮想アドレスを L2 の物理アドレスに変換することはできない．そこで，L0 KVM を修正して L2 の物理アドレスを用いて L1 の物理アドレスに変換するようにした．

L0 KVM を修正しない方法として，UEFI から取得したメモリマップを用いて MMIO 領域を推定し，BitVisor がアクセスを代行する方法が考えられる．L2 VM において空き領域と予約領域には物理メモリを割り当てないようにし，L2 VM で MMIO 領域にアクセスした時に VM Exit を発生させる．BitVisor が同じメモリ領域にアクセスすると，L0 KVM が MMIO の処理を行う．しかし，この実装ではフレームバッファ領域などのアクセスの代行が不要な領域も BitVisor がアクセスを代行してしまい性能が大幅に低下した．フレームバッファ領域へのアクセスを代行しないように修正すると実装が複雑になる．

4.5 SEV 仮想化

SEV 仮想化では，L2 VM において L1 VM とは異なる暗号鍵を使えるようにするために AMD-SP を仮想化する．

4.5.1 AMD-SP のコマンド

AMD-SP はゲスト管理用コマンド，プラットフォーム管理用コマンド，デバッグ用コマンドを提供している．ゲスト管理用コマンドは VM のメモリを暗号化するための鍵の生成などを行う．プラットフォーム管理用コマンドは初期化，ファームウェア更新，証明書関連などに用いられる．デバッグ用コマンドはポリシーで許容されている場合に SEV ゲストのメモリ領域の暗号化・復号を行う．ポリシーは LAUNCH_MEASURE コマンドでゲスト所有者が検証可能である．

ハイパーバイザは SEV ゲストを動作させるために AMD-SP に対して一連のゲスト管理用コマンドを呼び出す．まず，LAUNCH_START コマンドを呼び出して SEV ゲストを識別するハンドルを割り当て，メモリ暗号鍵を生成する．次に，ACTIVATE コマンドを呼び出し，ハンドルに割り当てられているメモリ暗号鍵と VM に割り当てられた ASID を関連付ける．そして，LAUNCH_UPDATE_DATA コマンドを呼び出すことにより，UEFI イメージなどの SEV ゲストを動作させるために必要なメモリ領域をそのゲストの鍵で暗号化する．

その後，LAUNCH_MEASURE コマンドを呼び出して，SEV ゲストの鍵で暗号化したデータなどがハイパーバイザによって改ざんされていないかをゲスト所有者が検証するために使われる HMAC とノンスを取得する．HMAC の計算に使われる鍵はメモリの暗号鍵とは別であり，LAUNCH_START コマンドの実行後に AMD-SP とゲスト所有者の間で共有される．LAUNCH_MEASURE コマンドを実行した後は LAUNCH_UPDATE_DATA コマンドは実行できなくなる．

LAUNCH_SECRET コマンドはディスクの暗号鍵などハイパーバイザに秘密データを注入するために使われ，LAUNCH_MEASURE コマンドを呼び出した後でのみ使うことができる．もし HMAC が一致しなければゲスト所有者が LAUNCH_SECRET に必要なデータを送信しないことによって起動を拒否することができる．LAUNCH_FINISH コマンドの実行後に VMRUN 命令を実行できる状態になる．この状態から LAUNCH_SECRET コマンドを実行できる状態に戻ることはできない．

4.5.2 仮想 AMD-SP

AMD-SP は PCI デバイスとして提供されているため，仮想 AMD-SP も L0 ハイパーバイザが仮想 PCI デバイスとして提供する．L1 ハイパーバイザは仮想 AMD-SP を使って L2 の SEV ゲストを動作させる．PCI デバイス ID，何番の割り込みが使われるか，SEV 以外のレジスタの機能，SEV レジスタの MMIO のベースアドレスなどについては仕様書が見つからなかったため，Linux に実装されている AMD-SP ドライバを基にメールボックスモードの必要最小限の部分のみを実装した．

仮想 AMD-SP は L0 ハイパーバイザを呼び出して実際の AMD-SP に対してコマンドを呼び出すことにより，L2 VM 用の暗号鍵の生成やハンドルと ASID との関連付けな

どを行う．L1 ハイパーバイザは L2 VM に仮想的な ASID を割り当て，L0 ハイパーバイザが仮想 ASID と実 ASID の対応を管理する．そして，L1 ハイパーバイザが実行した VMRUN 命令を L0 ハイパーバイザが処理する際や，ASID をパラメータに含む AMD-SP コマンドを実行する際に，仮想 ASID から実 ASID に変換する．KVM のネストした仮想化の実装では，L1 VM と L2 VM の ASID は同一であるため L1 と L2 の切り替え時に TLB フラッシュを行う必要がある．ASID を L1 VM と L2 VM に別々に割り当てることにより，TLB フラッシュを行う必要がなくなり，VM Exit での切り替えを高速化することができる．

L1 ハイパーバイザが L0 で動作する仮想 AMD-SP のコマンドを呼び出す際に，L1 VM の鍵で暗号化されたコマンドバッファを渡しても仮想 AMD-SP では復号することができない．そこで，L1 Linux の AMD-SP ドライバと L1 KVM を改変して，L1 VM が SEV ゲストとして動作している場合にはコマンドバッファを暗号化しないようにした．この際に，DMA で用いられているような暗号化されない固定のバウンスバッファを用意する方式では LAUNCH_UPDATE_DATA コマンドが動作しない．このコマンドは L2 VM の鍵でデータを暗号化して返すが，バウンスバッファからコマンドバッファにデータをコピーして物理アドレスが変わると正しく復号できなくなるためである．

L1 ハイパーバイザとして KVM を用いる場合に，仮想 AMD-SP を L0 QEMU の PCI デバイスとして実装した．仮想 AMD-SP が L0 KVM を呼び出すために，仮想 SEV 用の ioctl を追加した．

4.5.3 L1-L2 間でのデータ共有

L1 ハイパーバイザが提供する仮想デバイスに L2 VM が DMA でアクセスする場合などに，L1 VM と L2 VM の間でデータを共有する必要がある．SEV 仮想化では L1 VM と L2 VM の暗号鍵は異なるため，透過的 SEV や SEV パススルーのように共通の L1 VM の鍵で暗号化することはできない．そこで，L1 VM と L2 VM のそれぞれのページテーブルにおいて対応する PTE の C ビットに 0 を設定し，データを共有するためのメモリ領域を暗号化しないようにする．L1 ハイパーバイザがこのようなメモリ領域を特定するのは容易ではないため，L2 VM に割り当てるすべてのページに対応する PTE の C ビットに 0 を設定しておく．L2 VM においても PTE の C ビットに 0 を設定したページだけが暗号化されないメモリ領域となる．ただし，L0 ハイパーバイザもこのメモリ領域のデータを盗聴できるため，必要に応じて L2 OS がデータを暗号化する必要がある．

L1 ハイパーバイザとして KVM を用いる場合に，L1 VM 内に PTE の C ビットに 0 が設定されたメモリ領域を確保するためのデバイス /dev/sev-shared を L1 Linux に実装した．このデバイスを mmap して L2 VM のメモリとして使うように L1 QEMU に指定することにより，L1 QEMU から暗号化されていないメモリとしてアクセスすることができる．

表 3 実験に用いたハイパーバイザと OS

レイヤ	システムソフトウェア
L0	Linux/KVM 6.0.0, QEMU v7.1.0-748-gf1d33f55c4
L1	Xen 4.16-6d6aee437e Linux/KVM 6.0.0, QEMU v6.2.0（無改変） BitVisor a7de71da1f1b
L2	Linux 6.0.0

5. 実験

L2 VM に SEV を適用した場合としなかった場合について，ブロックデバイスに対する読み出し性能を測定する実験を行った．比較として，ネストした仮想化を用いず，L1 VM に SEV を適用した場合としなかった場合についても測定した．この実験には，AMD EPYC 7443P（24 コア）の CPU を 1 基，DDR4-3200 RDIMM 128 GiB のメモリを搭載したマシンを用いた．ハイパーバイザと OS に関しては表 3 のものを利用した．L2 VM に SEV を適用する際には，表 2 の実装済みの方式を用いた．

L0 においては SME によるメモリ暗号化は用いなかった．CPU の周波数スケーリング設定には performance を指定し，周波数のブースト機能は無効にした．L1 VM には 6 個の仮想 CPU と 8 GiB のメモリを割り当て，L0 で使われていない物理 CPU を割り当てた．L2 VM には 2 個の仮想 CPU と 2 GiB のメモリを割り当て，L1 VM で使われていない仮想 CPU を割り当てた．Xen の DomU を動作させる場合は，Dom0 と DomU の 2 つの L2 VM を作成した．L1 VM と L2 VM で用いる SWIOTLB のバッファサイズは 32 MiB とした．SEV 仮想化では TLB フラッシュの最適化は用いなかった．

この実験では SEV を適用することに伴う最大のオーバヘッドを調べるために，L0 に 10 GiB の RAM ディスクを作成し，この RAM ディスクにアクセスする virtio-blk-pci デバイスを QEMU に新たに追加した．このブロックデバイスの処理を行う QEMU のスレッドには専用の物理 CPU を 1 コア割り当て，POSIX AIO を使うように指定した．L1 VM でも，L0 によって提供されるブロックデバイスにアクセスする virtio-blk-pci デバイスを同様の設定で QEMU に新たに追加した．BitVisor を動作させる場合は，L2 VM は L0 のブロックデバイスにパススルーで直接アクセスした．L2 VM で Xen の Dom0 を動作させる場合にも L0 のブロックデバイスに直接アクセスした．L2 VM で Xen の DomU を動作させる場合には準仮想化デバイスを用い，Dom0 経由で L0 のブロックデバイスにアクセスした．

L2 VM において，このブロックデバイスから 10 GiB のデータを読み出すスループットを測定するベンチマークを

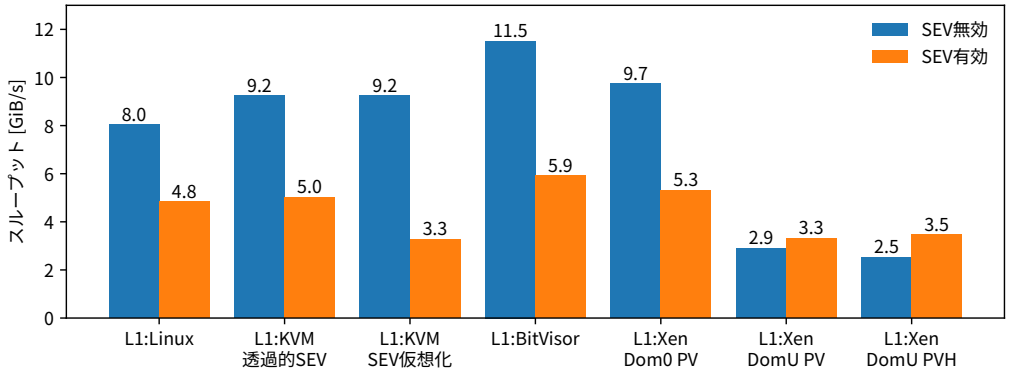


図 5 実験結果

実行した．read システムコールで一度に読み出すブロックサイズは 20 MiB とした．ネストした仮想化を用いない場合には，このベンチマークを L1 VM で実行した．このベンチマークを 500 回実行した時の平均スループットを図 5 に示す．

ネストした仮想化を用いない場合でも，L1 VM に SEV を適用するとスループットが 40%低下した．メモリ暗号化のオーバヘッドと DMA にバウンスバッファを使うオーバヘッドのためであると考えられる．一方，L1 ハイパーバイザとして Xen を動作させて Dom0 に SEV パススルーを適用した場合や，BitVisor を動作させて SEV パススルーを適用した場合，KVM を動作させて透過的 SEV を適用した場合にはスループットが 45〜49%低下した．また，KVM を動作させて SEV 仮想化を適用した場合にはスループットが 74%低下した．これは L2 VM と L1 VM で二重にバウンスバッファを使うためであると考えられる．

一方，L1 ハイパーバイザとして Xen を動作させて DomU に SEV パススルー（PV ゲストの場合）や透過的 SEV（PVH ゲストの場合）を適用した場合，SEV を適用しない場合よりも性能がよくなった．SEV を適用しない場合の性能が異常に低いように思われるが，この原因は不明である．また，L1 ハイパーバイザで KVM を動作させた方がネストした仮想化を用いない場合よりも性能が 15%よくなっているが，この原因も現在調査中である．

6. 関連研究

AMD SEV は Trusted Execution Environment (TEE) の一実装であるが，別の実装である Intel SGX については VM 内で利用するための SGX 仮想化 [11] が提案されている．SGX はエンクレイヴと呼ばれる保護領域でプログラムを安全に実行することができるプロセッサのセキュリティ機構である．SGX を仮想化するために，エンクレイヴ・ページキャッシュ（EPC）の一部だけを VM に提供し，CPUID 命令と MSR のエミュレーションを行う．Xen 用のパッチは現在のところ利用できなくなっているが，KVM と QEMU については標準でサポートされている．

ネストしたエンクレイヴ [12] は SGX ハードウェアを拡張することにより，エンクレイヴの中でエンクレイヴを動作させることができる．外側のエンクレイヴからは内側のエンクレイヴにアクセスすることはできないが，内側のエンクレイヴは外側のエンクレイヴに自由にアクセスすることができる．また，内側のエンクレイヴ同士は隔離される．これにより，エンクレイヴ内で動作するアプリケーションを信頼できないサードパーティのライブラリから保護したりすることができる．

Ryoan [13] はクラウド内に SGX のエンクレイヴを作成し，Google NaCl [14] を用いてその中にサンドボックスを構築する．NaCl がサンドボックス内で実行されるコードを検査したりランタイムチェックを行ったりすることにより，サンドボックス内でクラウドのサービスを安全に実行することができる．これにより，サンドボックスの外部で安全にサービスの通信履歴を取得したり，サービスのアクセス制限を行ったりすることができる．

同様に，AccTEE [15] は SGX のエンクレイヴ内で WebAssembly [16] を用いて双方向サンドボックスを構築する．WebAssembly も NaCl と同様にサンドボックス内でプログラムを安全に実行することができる．これにより，サンドボックス外部でプログラムによるリソース利用情報を安全に記録することができる．記録された利用情報はエンクレイヴ外部からもサンドボックス内のプログラムからも改ざんされることはない．

7. まとめ

本稿では，ネストした仮想化に SEV を組み合わせることを可能にする Nested SEV を提案した．L1 VM と L2 VM への SEV の適用方法によって考えられる 4 種類のシステム構成を示した．L2 VM に SEV を適用するために，Nested SEV は透過的 SEV，SEV パススルー，SEV 仮想化の 3 つの方式を提供する．これらを Xen，KVM，BitVisor に実装し，I/O 性能を調べる実験を行った．

今後の課題は，それぞれのハイパーバイザに未実装のSEV 適用方式を実装することである．L2 VM にのみ SEV を適用するシステム構成についても実装を行う．その上で，Nested SEV の性能を詳細に調べ，性能を改善する．さらに, セキュリティを向上させるために SEV-ES や SEV-SNP にも対応する必要がある.

謝辞

本研究の一部は，JST，CREST，JPMJCR21M4 の支援を受けたものである．また，本研究の一部は，国立研究開発法人情報通信研究機構の委託研究（05501）による成果を含む.

参考文献

[1] Advanced Micro Devices, Inc.: Secure Encrypted Virtualization API Version 0.24 (2020).
[2] Google LLC: Confidential VM documentation, `https://cloud.google.com/compute/confidential-vm/docs` (2020).
[3] Microsoft Corporation: Azure Confidential VM options on AMD, `https://learn.microsoft.com/en-us/azure/confidential-computing/virtual-machine-solutions-amd` (2021).
[4] Ben-Yehuda, M., Day, M. D., Dubitzky, Z., Factor, M., Har'El, N., Gordon, A., Liguori, A., Wasserman, O. and Yassour, B.-A.: The Turtles Project: Design and Implementation of Nested Virtualization, *Proc. Symp. Operating Systems Design and Implementation*, pp. 423–436 (2010).
[5] Williams, D., Jamjoom, H. and Weatherspoon, H.: The Xen-Blanket: Virtualize Once, Run Everywhere, *Proc. European Conf. Computer Systems*, pp. 113–126 (2012).
[6] Liu, C. and Mao, Y.: Inception: Towards a Nested Cloud Architecture, *Proc. Workshop on Hot Topics in Cloud Computing* (2013).
[7] 能野智玄，光来健一：AMD SEV で保護された VM の隔離エージェントを用いた安全な監視，コンピュータセキュリティシンポジウム 2022 (2022).
[8] 安東尚哉，光来健一：AMD SEV とネストした仮想化を用いた安全な通信履歴の取得，第 155 回 OS 研究会 (2022).
[9] Garfinkel, T. and Rosenblum, M.: A Virtual Machine Introspection Based Architecture for Intrusion Detection, *Proc. Network and Distributed Systems Security Symp.*, pp. 191–206 (2003).
[10] Madhavapeddy, A., Mortier, R., Rotsos, C., Scott, D., Singh, B., Gazagnaire, T., Smith, S., Hand, S. and Crowcroft, J.: Unikernels: Library Operating Systems for the Cloud, *Proc. Int. Conf. Architectural Support for Programming Languages and Operating Systems*, pp. 461–472 (2013).
[11] Huang, K.: Introduction to Intel SGX and SGX Virtualization, Xen Project Developer and Design Summit (2017).
[12] Park, J., Kang, N., Kim, T., Kwon, Y. and Huh, J.: Nested Enclave: Supporting Fine-grained Hierarchical Isolation with SGX, *Proc. Annual Int. Symp. Computer Architecture* (2020).
[13] Hunt, T., Zhu, Z., Xu, Y., Peter, S. and Witchel, E.: A Distributed Sandbox for Untrusted Computation on Secret Data, *Proc. Symp. Operating Systems Design and Implementation* (2016).
[14] Google, Inc.: Native Client, `https://developer.chrome.com/docs/native-client/` (2016).
[15] Goltzsche, D., Nieke, M., Knauth, T. and Kapitza, R.: AccTEE: A WebAssembly-based Two-way Sandbox for Trusted Resource Accounting, *Proc. Int. Middleware Conf.* (2019).
[16] Haas, A., Rossberg, A., Schuff, D., Titzer, B., Holman, M., Gohman, D., Wagner, L., Zakai, A. and Bastien, J.: Bringing the Web Up to Speed with WebAssembly, *Proc. Conf. Programming Language Design and Implementation* (2017).