AMD SEV/-ES/-SNPによるネストしたVMの保護方式

瀧口 和樹[1]　光来 健一[1]

概要：機密性の高い情報がクラウドでも扱われるようになり，クラウドの内部犯などから仮想マシン (VM) 内の機密情報を盗まれる危険性が増している．そのため，最近のクラウドはメモリを透過的に暗号化する AMD SEV と呼ばれる機能を用いた Confidential VM を提供している．SEV-ES と呼ばれる拡張ではレジスタの状態も暗号化され，SEV-SNP と呼ばれる拡張ではメモリの整合性なども保護される．しかし，VM 内で動作するネストした VM に対する SEV のサポートはまだ十分ではない．本稿では，ネストした VM を SEV，SEV-ES，SEV-SNP で保護することを可能にする Nested SEV を提案する．Nested SEV は SEV 仮想化と SEV パススルーの 2 つの保護方式を提供する．SEV 仮想化は仮想 SEV をネストした VM に適用し，外側の VM とは異なる鍵を用いてメモリとレジスタを暗号化する．一方，SEV パススルーは外側の VM に適用されている SEV をそのままネストした VM にも適用し，同じ鍵を用いてメモリとレジスタを暗号化する．これらの方式を KVM，BitVisor，Xen (準仮想化) に実装し，Nested SEV の性能を調べる実験を行った．

1. はじめに

ユーザに仮想マシン (VM) を提供するクラウドが様々な用途に活用されている．それに伴い，機密性の高い情報がクラウドでも扱われるようになり，クラウドの内部犯などから機密情報を盗まれる危険性が増している．そのため，Amazon Web Services，Google Cloud，Microsoft Azure などのクラウドは AMD EPYC プロセッサの Secure Encrypted Virtualization (SEV) [1] と呼ばれるセキュリティ機構を用いた Confidential VM を提供している [2–4]．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．第 2 世代以降では SEV-Encrypted State (SEV-ES) と呼ばれる拡張が提供されており，メモリに加えて CPU レジスタの状態も暗号化することができる．第 3 世代以降では SEV-Secure Nested Paging (SEV-SNP) と呼ばれる拡張が提供されており，メモリの整合性などを保護できる．SEV を用いることにより，クラウドの内部犯でさえ VM のメモリ内部の機密情報を盗聴したり，改ざんしたりすることはできなくなる．

一方，VM の中で VM を動作させるネストした仮想化 [5] と呼ばれる技術を用いて，仮想クラウド [6, 7] や監視機構 [8, 9] などの様々なシステムが提案されている．しかし，ネストした仮想化を用いるシステムにおいて SEV のサポートは十分ではない．Microsoft によるパッチ [10] ではネストした VM に SEV-SNP を適用することを可能にしているが，外側の VM には SEV を適用することができない．Hecate [11] や OpenHCL [12] は外側の VM にも SEV-SNP を適用することができるが，ネストした VM は 1 つのみに制限される．

本稿では，SEV で保護された VM の中で SEV を適用した VM の実行を可能にする Nested SEV を提案する．Nested SEV はシステム構成として，外側の VM と内側のネストした VM のメモリ暗号化に異なる鍵を用いる構成と同一の鍵を用いる構成をサポートしている．そのために，SEV 仮想化と SEV パススルーの 2 種類の保護方式を提供する．SEV 仮想化は SEV を仮想化した仮想 SEV をネストした VM に適用することにより，外側の VM とは異なる暗号鍵を用いてメモリとレジスタを暗号化する．一方，SEV パススルーは外側の VM に適用されている SEV をそのままネストした VM にも適用し，同じ鍵を用いてメモリとレジスタを暗号化する．

SEV 仮想化は外側の VM 内のハイパーバイザからネストした VM を保護することを可能にする．外側の VM に対して仮想 SEV を提供するために，SEV の暗号鍵などを管理する AMD Secure Processor (AMD-SP) や VM のメモリの破壊や割り当て変更を防ぐために用いられる Reverse Map Table (RMP) を仮想化する．一方，SEV パススルーはネストした VM をクラウドから保護しつつ，外側の VM 内のハイパーバイザからはアクセス可能にする．そのために，外側の VM 内のハイパーバイザはネストした VM に対して特殊な管理を行う．

[1] 九州工業大学

外側の VM を動作させるハイパーバイザとして KVM，外側の VM 内で動作するハイパーバイザとして KVM，BitVisor，Xen，ネストした VM 内で動作する OS として Linux を用いて Nested SEV を実装した．SEV の拡張である SEV-ES と SEV-SNP もサポートし，セキュリティと性能のトレードオフをとれるようにした．Nested SEV のオーバヘッドを調べるための実験を行い，Nested SEV の 2 種類の保護方式，3 種類の SEV，3 種類のハイパーバイザを用いた場合の性能を明らかにした．

以下，2 章では AMD SEV とネストした仮想化について述べる．3 章では提案する Nested SEV について述べる．4 章では SEV 仮想化，5 章では SEV パススルーの実装の詳細について述べる．6 章では Nested SEV を用いて行った性能測定について述べる．7 章で関連研究に触れ，8 章で本稿をまとめる．

2. AMD SEV とネストした仮想化

2.1 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．暗号化・復号化の処理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．SEV を有効にした VM では，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．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．VM が I/O に用いるメモリ領域などは，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．

2.2 SEV-ES

第 2 世代以降の AMD EPYC プロセッサは SEV-ES と呼ばれる SEV の拡張を提供しており，メモリに加えて CPU レジスタの状態も暗号化することができる．AMD プロセッサの仮想化支援機構である AMD-V を用いる場合，VM Exit が発生すると VM のレジスタの状態は VM Control Block (VMCB) と呼ばれるメモリ領域に格納されていた．SEV-ES ではレジスタの状態は VM Save Area (VMSA) と呼ばれるメモリ領域に暗号化されて格納される．ハイパーバイザは命令エミュレーションなどのために VM のレジスタを読み書きする必要があるが，VMSA に格納されたレジスタの状態にはアクセスすることができない．

そのため，SEV-ES では VM Exit の代わりに VM に対して VMM Communication (#VC) 例外が発生する．#VC 例外が発生するとゲスト OS の#VC 例外ハンドラが呼び出され，VM Exit の処理に必要となるレジスタの値を VM とハイパーバイザ間で共有されるメモリ領域である Guest Host Communication Block (GHCB) に格納する．そして，新たに導入された `VMGEXIT` 命令を実行して VM Exit を発生させ，ハイパーバイザが GHCB を参照して命令エミュレーションなどの処理を行う．ハイパーバイザが `VMRUN` 命令を実行して VM に復帰すると#VC 例外ハンドラから実行を再開し，#VC 例外ハンドラは `IRET` 命令により#VC 例外を発生させた処理に復帰する．

すべての VM Exit で#VC 例外ハンドラと GHCB を経由したゲスト OS による処理を行うと正常に VM の実行を行うことができなくなる．例えば，タイマ割り込みなどの外部割り込みによる VM Exit で#VC 例外を発生させると，VM がハイパーバイザに制御を戻さないことが可能になる．また，#VC 例外ハンドラで実行する `VMGEXIT` による VM Exit で#VC 例外が発生すると，無限ループに陥る．このような VM Exit は Automatic Exit (AE) と呼ばれ，#VC 例外は発生しない．それ以外の VM Exit は Non-Automatic Exit (NAE) と呼ばれる．

2.3 SEV-SNP

第 3 世代以降の AMD EPYC プロセッサは SEV-SNP と呼ばれる SEV の拡張を提供しており，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なども防ぐことができる．そのために，SEV-SNP は物理ページとその所有者の対応を管理する Reverse Map Table (RMP) を用いる．RMP はホスト物理ページごとにエントリを持ち，アドレス変換の際に得られたホスト物理ページに対応するエントリを用いて整合性チェックが行われる．各エントリは VM に割り当てられたアドレス空間識別子 (ASID)，ゲスト物理ページ番号，Assigned ビット，Validated ビット，VMSA ビットなどからなる．

メモリにアクセスする際に，RMP の対応するエントリに登録された ASID と実行中の ASID が異なる場合にはページフォールトが発生する．また，アクセスしようとした VM のゲスト物理アドレスと RMP に登録されたゲスト物理アドレスが一致しない場合にもページフォールトが発生する．Assigned ビットが 1 の時，そのページは VM に割り当てられていることを示しており，ハイパーバイザが書き込むことはできない．また，ハイパーバイザが RMP のエントリを変更した時に Validated ビットは 0 となり，そのメモリにアクセスすると#VC 例外が発生する．Validated ビットは VM が `PVALIDATE` 命令を実行することで 1 にすることができる．VMSA ビットが 1 の場合はそのページを VMSA に用いることができ，AMD-SP のコマンドか `RMPADJUST` 命令で 1 にすることができる．

SEV-SNP では VM Privilege Level (VMPL) と呼ばれる機構も導入され，VM のメモリアクセスを 4 つの特権レベ

ルに分割することができる．VMPL0 が最も権限が高く，PVALIDATE 命令や RMPADJUST 命令は VMPL0 のみが実行できる．RMP のエントリには VMPL ごとにページが読み書き実行できるかどうかを表す権限に関する情報も含まれ，AMD-SP のコマンドまたは RMPADJUST 命令によって変更できる．VMPL の情報は VMSA に含まれており，ハイパーバイザが VMSA を入れ替えることによって切り替えることができる．

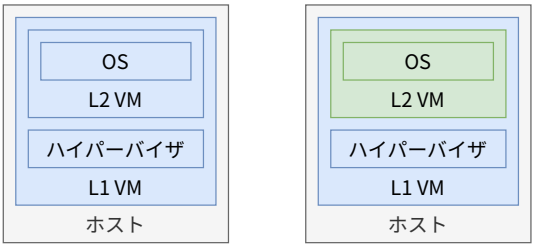
2.4 ネストした仮想化

ネストした仮想化 (Nested Virtualization) [5] 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 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 と呼ばれる．その VM の中で動作するハイパーバイザは L1 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 と呼ばれる．

これまで SEV とネストした仮想化の組み合わせは限定的であった．Microsoft によるパッチでは L0 で Hyper-V，L1 で KVM が動作し，L2 VM のみに SEV-SNP を適用することができる．L1 VM に SEV を適用することはできないため，L1 ハイパーバイザを保護することはできない．Hecate [11] や OpenHCL [12] では L0 で Hyper-V，L1 で KVM/QEMU や OpenVMM が動作し，L2 VM だけでなく，L1 VM にも SEV-SNP を適用することができる．しかし，VMPL を用いて実装されているため，SEV-SNP しかサポートすることができず，L2 VM は 1 つのみに制限される．また，VMPL0 で動作する L1 ハイパーバイザはより低い特権レベルで動作する L2 VM に自由にアクセスすることができるため，L2 VM を L1 ハイパーバイザから保護することはできない．

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．Nested SEV は SEV で保護された L1 VM の中でハイパーバイザを動作させることを可能にする．さらに，その上で動作する L2 VM にも SEV を適用可能にする．L1 ハイパーバイザ上では複数の L2 VM を実行することができる．Nested SEV は図 1 のような 2 種類のシステム構成をサポートしている．L1 ハイパーバイザが信頼できない場合は L2 VM のメモリ等の暗号化に L1 VM と異なる鍵を用いるが，信頼できる場合は同一の鍵を用いて L2 VM のメモリ等へのアクセスを許可することができる．



(a) 鍵が同一の構成　(b) 鍵が異なる構成

図 1 Nested SEV を用いたシステム構成

このような 2 種類のシステム構成を実現するために，Nested SEV は SEV 仮想化と SEV パススルーの 2 つの保護方式を提供する．SEV 仮想化は L1 ハイパーバイザから L2 VM を保護することを可能にする．L1 VM では信頼できない L1 ハイパーバイザを動作させることを想定しており，L2 VM は L1 VM とは異なる鍵で暗号化される．SEV 仮想化は L1 VM に対して仮想 SEV を提供し，L1 ハイパーバイザがそれを L2 VM に適用する．そのために，SEV の暗号鍵などを管理する AMD-SP も仮想化する．この仮想 AMD-SP は物理 AMD-SP が管理する暗号鍵を利用するため，L2 VM のメモリ等を安全に暗号化することできる．また，VM のメモリの破壊や割り当て変更を防ぐために用いられる RMP への操作も仮想化する．さらに，L1 VM と L2 VM それぞれで VMPL を制約なく用いることができる．MS パッチとは異なり，L1 VM にも SEV を適用することができ，L1 ハイパーバイザを L0 ハイパーバイザから保護することができる．

SEV 仮想化は保護された仮想パブリッククラウドの構築に用いることができる．パブリッククラウドのプロバイダが 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，SEV で保護する．L1 VM 内では L2 VM を一般のユーザに提供し，仮想 SEV で保護する．また，クラウドサービスを安全に監視するために利用することもできる．例えば，SEV-tracker [9] は SEV で保護された L1 VM 内でユーザの L1 ハイパーバイザを安全に実行し，仮想 SEV で保護された L2 VM 内でクラウドサービスを安全に実行する．これにより，クラウドとユーザの間で相互保護を実現しつつ，L1 ハイパーバイザで L2 VM の通信を安全に監視することができる．

一方，SEV パススルーは L1 VM を L0 ハイパーバイザから保護しつつ，L1 ハイパーバイザが L2 VM のメモリや CPU レジスタ等にアクセスすることを可能にする．L1 VM では信頼できるハイパーバイザを動作させることを想定しており，L1 VM と L2 VM のメモリ等は同一の鍵を用いて暗号化される．そのために，L1 VM に適用されている SEV を仮想化せずにそのまま L2 VM にも適用する．SEV パススルーは仮想 AMD-SP を提供しないため，ハイパーバイザが L2 VM に対して特殊な管理を行うことによ

り SEV を適用する．L2 VM に対して AMD-SP の操作を行わないため，準仮想化ゲストのように仮想化支援命令を用いずに動作する VM もサポートすることができる．SEV パススルーは L1 ハイパーバイザを信頼する点で Hecate や OpenHCL に似ているが，複数の L2 VM を作成することができる点などが異なる．また，SEV-SNP でしか提供されない VMPL に依存しないため，様々な種類の L2 VM をサポートすることができる．

SEV パススルーは保護された仮想プライベートクラウドの構築に用いることができる．L1 VM に仮想プライベートクラウドを構築し，SEV で保護するところまでは仮想パブリッククラウドと同様である．一方で，プライベートクラウドでは管理者とユーザが同一組織となるため，L2 VM は仮想 SEV で保護しない．また，L1 ハイパーバイザが L2 VM の内部を監視するために利用することもできる．例えば，L1 VM において VM イントロスペクションを用いて L2 VM を監視することができる．また，SEVmonitor [8] は L1 ハイパーバイザ内でエージェントを動作させて L2 VM のメモリデータを取得し，SEV で保護された別の VM で侵入検知を行うことを可能にしている．

Nested SEV は SEV，SEV-ES，SEV-SNP の 3 種類の SEV をサポートする．セキュリティは SEV-SNP が最も高く，SEV が最も低いが，性能は SEV が最も高く，SEV-SNP が最も低い．このように，この 3 種類の SEV の間にはセキュリティと性能のトレードオフがある．そのため，ユーザの要求に応じて使い分けることができる．例えば，非常に強いセキュリティが必要な場合には性能を犠牲にしてでも SEV-SNP を使うべきであるが，VM のメモリ暗号化が行われれば十分であれば SEV を使えばよい．

4. SEV 仮想化

4.1 SEV 仮想化の実装

SEV を有効にした L1 VM 内で L1 ハイパーバイザを実行できるようにするために，L1 OS と同様の SEV 対応を行う．L1 ハイパーバイザは PTE を作成・更新する際に適切に C ビットを設定する．L0 ハイパーバイザからアクセスが必要な VRAM などのメモリ領域は PTE の C ビットに 0 を設定して暗号化しないようにする．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，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．

4.1.1 仮想化支援機構の仮想化

L1 ハイパーバイザは L2 VM の実行を制御するために用いられる $\mathrm{VMCB}_{1\to2}$ を暗号化しないようにする．L1 ハイパーバイザが L2 VM に対して `VMRUN` などの仮想化支援命令を実行すると L0 ハイパーバイザに対して VM Exit が発生する．その際に，L0 ハイパーバイザは L1 VM のメモリ上にある $\mathrm{VMCB}_{1\to2}$ を適切に変換して，L0 ハイパーバイザにおいて有効な $\mathrm{VMCB}_{0\to2}$ を生成する．この $\mathrm{VMCB}_{0\to2}$ を用いることにより，L0 ハイパーバイザが L2 VM に対して L1 ハイパーバイザの代わりに仮想化支援命令を実行することができる．

L1 ハイパーバイザは L2 VM 用の Nested Page Table (NPT) も暗号化しないようにする．ネストした仮想化を行う場合，L1 ハイパーバイザは L2 VM ごとに L2 物理アドレスから L1 物理アドレスに変換する $\mathrm{NPT}_{1\to2}$ を作成する．また，L0 ハイパーバイザは L1 VM ごとに L1 物理アドレスからホストの L0 物理アドレスに変換する $\mathrm{NPT}_{0\to1}$ を作成する．プロセッサはアドレス変換の際に 1 つの NPT しか参照できないため，L0 ハイパーバイザはこれら 2 つの NPT をマージして，L2 物理アドレスから L0 物理アドレスに変換する $\mathrm{NPT}_{0\to2}$（シャドウ NPT）を生成する．そのために，L0 ハイパーバイザが $\mathrm{NPT}_{1\to2}$ にアクセスできる必要がある．

VMCB や NPT はハイパーバイザによって管理されているため，暗号化しないようにすることによるセキュリティの低下はない．Nested SEV では L1 ハイパーバイザだけでなく L0 ハイパーバイザもアクセスできるようになるが，L0 ハイパーバイザが行える攻撃は L1 ハイパーバイザと同じである．

4.1.2 ASID 仮想化

L1 ハイパーバイザは L2 VM に仮想的な ASID を割り当てる．SEV において ASID は VM のメモリ暗号鍵に関連づけられる．L0 ハイパーバイザが L2 VM に実 ASID を割り当て，仮想 ASID と実 ASID の対応を管理する．L1 ハイパーバイザが L2 VM に対して `VMRUN` 命令を実行して VM Exit が発生した際に，L0 ハイパーバイザが仮想 ASID を実 ASID に変換する．L0 ハイパーバイザが L1 VM と L2 VM に別々の ASID を割り当てることにより，L1 VM と L2 VM の切り替え時に TLB フラッシュを行う必要がなくなる．KVM の SEV を用いないネストした仮想化の実装では，L1 VM と L2 VM に同一の ASID を割り当てるため ASID を節約することができるが，L1 VM と L2 VM の切り替え性能は低下する．

L0 ハイパーバイザは仮想 ASID と実 ASID の対応を任意に変更することができるが，ASID はメモリの暗号鍵に関連づけられているため，対応を変更すると L2 VM はメモリを正常に復号して実行することができなくなる．

4.1.3 仮想 AMD-SP

AMD-SP は PCI デバイスとして提供されているため，L0 ハイパーバイザは仮想 AMD-SP を仮想 PCI デバイスとして L1 VM に提供する．L1 ハイパーバイザは仮想 AMD-SP に対してコマンドを発行することによって L2 VM を動作させる．仮想 AMD-SP は L0 ハイパーバイザを呼び出してそのコマンドを物理 AMD-SP に転送して実行するため，暗号鍵の管理などは従来と同様に物理 AMD-SP の内部で

安全に行うことができる．

L1 ハイパーバイザは L2 VM を起動するために仮想 AMD-SP に対して一連のゲスト管理用コマンドを発行する．まず，LAUNCH_START コマンドを発行して L2 VM を識別するゲストハンドルを割り当て，メモリ暗号鍵を生成する．次に，ACTIVATE コマンドを発行し，ゲストハンドルに割り当てられているメモリ暗号鍵と L2 VM に割り当てられた仮想 ASID を関連づける．この時に，L0 ハイパーバイザが仮想 ASID を実 ASID に変換し，物理 AMD-SP のコマンドを呼び出して暗号鍵と実 ASID を関連づける．

そして，L1 ハイパーバイザは LAUNCH_UPDATE_DATA コマンドを発行することにより，UEFI イメージなどの L2 VM を動作させるために必要なメモリ領域をその L2 VM の鍵で暗号化する．その後，LAUNCH_MEASURE コマンドを発行して，暗号化したデータなどが L1 ハイパーバイザや L0 ハイパーバイザによって改ざんされていないことをゲスト所有者が検証するために使われる HMAC を取得する．最後に，LAUNCH_FINISH コマンドを発行すると VMRUN 命令を実行して L2 VM を動作させられる状態になる．

4.1.4 L2 VM とのメモリの共有

L1 ハイパーバイザと L2 VM の間でデータを共有する際には，そのメモリ領域を暗号化しないようにする．例えば，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である．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 を設定する．ただし，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 を設定する．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双方から暗号化せずにアクセスできるメモリ領域となる．このようにして共有するメモリ領域は L0 ハイパーバイザも盗聴できるため，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．

4.1.5 PCI パススルー

L2 VM 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．そのために必要なメモリマップト I/O (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する．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 物理アドレスが $\mathrm{VMCB}_{0\to2}$ に格納される．Nested SEV では L1 VM 内にある L2 VM 用の $\mathrm{NPT}_{1\to2}$ 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 物理アドレスを L1 物理アドレスに変換することができる．また，L0 ハイパーバイザは L2 VM の暗号化されたメモリ上にある命令をデコードできないため，AMD-V の Decode Assists 機能を用いて VM Exit の発生要因となった命令のバイト列も $\mathrm{VMCB}_{0\to2}$ に格納させる．これらの情報を用いて，L1 VM が仮想デバイスの MMIO 領域にアクセスした時と同様に，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．

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．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．

4.2 SEV-ES 仮想化の実装

SEV-ES を有効にした L1 VM 内で L1 ハイパーバイザを実行できるようにするために，L1 OS と同様の SEV-ES 対応を行う．主に#VC 例外ハンドラの実装が必要となる．

4.2.1 VMSA の管理

SEV-ES で導入された VMSA には VM の仮想 CPU レジスタの状態が暗号化されて格納されるが，ハイパーバイザは暗号化された状態であれば VMSA のメモリ領域を自由に読み書きすることができる．そのため，レジスタの状態を暗号化するだけでは状態のロールバック攻撃を行うことができてしまう．この攻撃を防ぐために，SEV-ES は VM Exit が発生した時に VMSA の CRC 値を計算し，Trusted Memory Region (TMR) と呼ばれる通常の方法では読み書きできない特殊なメモリ領域に格納する．ハイパーバイザが VMRUN 命令を実行した時，SEV-ES は VMSA の CRC 値を計算して TMR に格納されている値と比較を行い，一致しなければ実行を失敗させる．

L1 ハイパーバイザは L2 VM の起動時に VMSA を確保し，暗号化されていない仮想 CPU レジスタの初期値を格納する．そして，仮想 AMD-SP の LAUNCH_UPDATE_VMSA コマンドを実行すると，物理 AMD-SP にコマンドが転送される．物理 AMD-SP は L2 VM の鍵で VMSA を暗号化し，VMSA の CRC 値を計算して TMR に格納する．LAUNCH_UPDATE_VMSA コマンドを任意のタイミングで実行できてしまうと L1 または L0 ハイパーバイザによって L2 VM に任意の状態を設定できてしまうため，VM を実行で

きる状態に遷移すると LAUNCH_UPDATE_VMSA コマンドを実行することはできなくなる．

4.2.2 ハイパーバイザと VM 間の切り替え

SEV-ES を用いる場合，ハイパーバイザが VMRUN 命令を実行するとまず，命令ポインタなど最低限の状態がホスト保存領域に退避される．そして，VM のすべての状態が暗号化された VMSA から復元され，VM に状態が遷移する．一方，VM 内で VM Exit が発生した時にはまず，VM のすべての状態が暗号化されて VMSA に退避される．そして，ホスト保存領域から退避されていない状態も含めてすべての状態が復元され，ハイパーバイザに状態が遷移する．このように，VM の状態はハイパーバイザに対して秘匿される．

ネストした仮想化では L1 ハイパーバイザが VMRUN 命令を実行すると#VC 例外が発生するが，Nested SEV ではその代わりに直接，GHCB への設定を行って VMGEXIT 命令を実行して効率よく VM Exit を発生させる．L1 VM からの VM Exit 時に L0 ハイパーバイザは L1 VM の状態にアクセスできないため，L1 VM 用のホスト保存領域への状態の保存と復元の処理をエミュレートすることはできない．しかし，L1 VM のすべての状態は VMSA に保存されるため，L1 VM 用のホスト保存領域を使用せず，L1 VM の VMSA と VMCB を L2 VM のものに切り替える．L2 VM で VM Exit が発生した時には L2 VM の VMSA と VMCB を L1 VM のものに切り替える．ホスト保存領域を使用しなくても動作に支障はない．

4.2.3 NPT 仮想化

シャドウ NPT は NPT の書き換え時に常に同期を保つ方式，または書き換え時には同期を保たない方式で実装される．しかし，SEV-ES を適用した L1 VM では常に同期を保つ方式を用いることはできない．この方式では，L0 ハイパーバイザが $NPT_{1\to2}$ への書き込みを保護し，L1 ハイパーバイザが書き換えようとすると VM Exit が発生するように VMCB を設定する．VM Exit が発生すると，L0 ハイパーバイザは命令エミュレーションを行って書き換えようとした部分に対応する $NPT_{1\to2}$ と $NPT_{0\to2}$ の PTE を書き換える．そして，その次の命令から L1 ハイパーバイザの実行を再開させる．SEV-ES では L0 ハイパーバイザが直接，命令エミュレーションを行うことはできないため，#VC 例外を発生させる必要がある．しかし，NPT の読み込み時にも#VC 例外が発生するため，性能に大きな影響が出る．

そこで，Nested SEV では NPT の書き換え時に同期を保たない方式を用いる．この方式でも同様に L1 ハイパーバイザによる NPT の書き換え時に VM Exit を発生させるが，命令エミュレーションは行わない．VM Exit 発生時に L0 ハイパーバイザは書き込み保護を解除し，VM Exit を発生させた命令から L1 ハイパーバイザの実行を再開する．これにより，$NPT_{1\to2}$ のみが更新され，シャドウ NPT は最新の状態ではなくなる．そこで，L1 ハイパーバイザによる NPT の書き換え後に必ず行われる TLB フラッシュの際に同期を取る．L0 ハイパーバイザは TLB フラッシュの際に同期が取れていないシャドウ NPT の PTE を最新の状態に書き換え，再び書き込み保護を行う．L0 ハイパーバイザはこの処理を L1 ハイパーバイザが L2 VM に状態を遷移するために実行する VMRUN 命令によって VM Exit が発生した際に行う．

4.2.4 MMIO 仮想化

L2 VM 内での MMIO を仮想化するために，L1 ハイパーバイザは $NPT_{1\to2}$ の対応する PTE に無効な値を設定する．これにより，L2 VM がメモリ上の MMIO 領域にアクセスするとネストしたページフォールトによる VM Exit が発生する．SEV-ES ではネストしたページフォールトによって#VC 例外が発生し，L1 ハイパーバイザが#VC 例外ハンドラで命令のデコードを行う．そして，アクセス先のアドレスを GHCB に書き込み，VMGEXIT 命令を実行する．このように，MMIO のための VM Exit は#VC 例外が発生する NAE である必要がある．一方，デマンドページングを実現するには，ネストしたページフォールトによる VM Exit は#VC 例外が発生しない AE である必要がある．これは#VC 例外ハンドラの実行中に再びデマンドページングのための#VC 例外が発生すると正常に処理が行えなくなるためである．

そのため，L1 ハイパーバイザは L2 VM 内での MMIO 領域に対応する NPT の PTE の予約ビットを設定する．SEV-ES はネストしたページフォールトによる VM Exit で#VC 例外が発生するかどうかを NPT の PTE の予約ビットを用いて制御することができる．予約ビットが 1 の場合にのみ NAE が発生し，存在ビットが 0 のページへのアクセスや書き込み可能フラグが 0 のページへの書き込みの場合などには AE が発生する．さらに，Nested SEV は予約ビットの値もシャドウ NPT と同期をとる．

4.3 SEV-SNP 仮想化の実装

SEV-SNP を有効にした L1 VM 内で L1 ハイパーバイザを実行できるようにするために，L1 OS と同様の SEV-SNP 対応を行う．RMP が導入されたため，特定のメモリ領域を L0 ハイパーバイザと共有したい場合には C ビットを 0 にするだけでなく，VMGEXIT 命令を実行して特定の物理アドレス領域をハイパーバイザ所有に変更するための要求を L0 ハイパーバイザに対して行う．ハイパーバイザ所有の状態からゲスト所有に戻したい場合にも同様に VMGEXIT 命令を実行して要求を行い，さらに PVALIDATE 命令を実行して Validated ビットを 1 にする．

4.3.1 RMP 仮想化

RMP を書き換えるための RMPUPDATE 命令は L0 ハイ

パーバイザのみが実行することができる．L1 ハイパーバイザによる RMP の書き換えを仮想化するために，AMD64 Architecture Programmer's Manual [13] においてプロセッサ上に存在しない VIRT_RMPUPDATE MSR を使うことが推奨されている．L1 ハイパーバイザがこの MSR への書き込みを行うと，L0 ハイパーバイザに対して VM Exit が発生する．通常の MSR への書き込みとは異なり，RAX レジスタ，RDX レジスタ，R8 レジスタを入力に，RAX レジスタを出力に用いるため，L1 ハイパーバイザの#VC 例外ハンドラにおいて適切に GHCB の読み書きを行う必要がある．L0 ハイパーバイザは L1 物理アドレスから L0 物理アドレスへの変換と仮想 ASID から実 ASID への変換を行ってから RMPUPDATE 命令を実行する．2 MB ページ用の RMP エントリを 4 KB ページ用の RMP エントリに分割する PSMASH 命令も同様に VIRT_PSMASH MSR を用いて仮想化する．

RMP はハイパーバイザが直接メモリから読み込むことができるようになっているため，L1 ハイパーバイザは RMP を仮想化した仮想 RMP を作成する．仮想 RMP はハードウェアによって参照されることはない．ネストしていない場合と同様に，仮想 RMP はメモリの所有者を管理する．ゲスト所有メモリは VM の鍵で暗号化されたメモリ領域であり，ハイパーバイザ所有メモリは VM の鍵で暗号化されない共有メモリ領域である．ゲスト所有メモリに対応する仮想 RMP のエントリの Assigned ビットは 1 に設定し，L2 の仮想 ASID と L2 物理アドレスを格納する．ハイパーバイザ所有メモリに対応するエントリの Assigned ビットは 0 に設定する．仮想 RMP は L1 ハイパーバイザが VIRT_RMPUPDATE MSR への書き込みを行う際に更新し，実 RMP との同期を取る．

L0 ハイパーバイザが管理する実 RMP は従来のゲスト所有とハイパーバイザ所有の管理だけでは不十分であり，L2 所有，L1 所有，L0 所有の 3 種類を管理する必要がある．ネストした場合，ゲスト所有メモリは L1 所有と L2 所有に分かれるためである．L2 VM だけがアクセスできるゲスト所有メモリは L2 所有，L1 ハイパーバイザが L2 VM に割り当てていないハイパーバイザ所有メモリは L1 所有，L1 ハイパーバイザと L2 VM が共有するメモリは L0 所有になる．そのため，L2 VM が所有する L2 所有メモリに対応するエントリの Assigned ビットは 1 とし，L2 の実 ASID と L2 物理アドレスを格納する．L1 VM または L1 ハイパーバイザが所有する L1 所有メモリに対応するエントリは Assigned ビットを 1 とし，L1 VM の ASID と L1 物理アドレスを格納する．L0 ハイパーバイザが所有する L0 所有メモリに対応するエントリは Assigned ビットを 0 とする．

L1 ハイパーバイザは L1 所有メモリを L2 所有メモリに変換するために VIRT_RMPUPDATE MSR を用いる．L2 所有メモリから L1 所有メモリにするためには VIRT_RMPUPDATE MSR と PVALIDATE 命令を用いる．

4.3.2 仮想 AMD-SP の SEV-SNP 対応

SEV-ES までの AMD-SP はゲストハンドルと呼ばれる 32 ビット値によって VM を管理していたが，SEV-SNP ではゲストコンテキストと呼ばれるページを用いて管理する．L1 ハイパーバイザは仮想 AMD-SP に対してゲストコンテキストとして使われる L1 所有メモリの物理アドレスを指定して SNP_GCTX_CREATE コマンドを発行する．仮想 AMD-SP はそのアドレスをホスト物理アドレスに変換し，物理 AMD-SP に対して同じコマンドを発行する．このようにして作成されたゲストコンテキストは RMP によって保護されているため，L0 ハイパーバイザは改ざんすることができず，暗号化されているため盗聴することもできない．

次に，L1 ハイパーバイザは SNP_LAUNCH_START コマンドを発行して，ゲストコンテキストのアドレスやポリシーなどを指定してゲストコンテキストの設定を行う．また，ゲストコンテキストと仮想 ASID を指定して SNP_ACTIVATE コマンドを発行し，物理 AMD-SP 経由で暗号鍵をメモリコントローラにロードする．そして，SNP_LAUNCH_UPDATE コマンドを発行して，ファームウェアや VMSA などを L2 VM の鍵で暗号化する．このコマンドは渡された平文データを暗号化し，RMP を書き換えてゲスト所有に変更する．そのため，L1 所有メモリをハイパーバイザ所有に変更してから暗号化するデータをコピーする．最後に，SNP_LAUNCH_FINISH コマンドを発行すると L2 VM を実行することが可能になる．

L2 VM は VMGEXIT 命令を実行することで，L1 ハイパーバイザに仮想 AMD-SP の SNP_GUEST_REQUEST コマンドを発行させることができる．このコマンドはアテステーションなどのために，SNP_LAUNCH_UPDATE コマンドに渡されたデータのハッシュ値を取得するために使われる．返されるハッシュ値は物理 AMD-SP の内部に持っている鍵で署名されているため，L0 ハイパーバイザや L1 ハイパーバイザが改ざんすることはできない．

L1 ハイパーバイザは L2 VM が終了した際に SNP_DECOMMISSION コマンドを発行することによってゲストコンテキストを破棄する．L1 ハイパーバイザが正常終了しない場合に備えて，L0 ハイパーバイザは L2 VM のゲストコンテキスト一覧を保持しておく．そして，ゲストコンテキストとして使われている L1 物理メモリ領域が L1 VM から削除されようとした時に，SNP_DECOMMISSION コマンドを発行してゲストコンテキストを破棄する．

5. SEV パススルー

5.1 SEV パススルーの実装

仮想化支援機構の仮想化や PCI パススルーについては SEV 仮想化と同様であるが，SEV パススルーでは ASID を仮想化せず，仮想 AMD-SP も提供しない．

5.1.1 SEV の適用

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．L2 VM にも L1 VM と同じ ASID を割り当てることにより，同一の暗号鍵を使用する．一方，AMD-V を使わずに動作する L2 VM には ASID が割り当てられないため，L1 VM に適用されている SEV によって同一の鍵を用いて自動的にメモリが暗号化される．いずれの場合でも，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．L2 VM は仮想 AMD-SP を用いずに起動するため，L2 VM のアテステーションを行うことはできない．しかし，SEV パススルーでは L1 ハイパーバイザを信頼するため，アテステーションを行うことなく正しい起動が保証される．

5.1.2 メモリ共有

SEV パススルーでは，L1 ハイパーバイザと L2 VM の間で共有するメモリ領域は暗号化したままにすることができる．SEV 仮想化では L1 VM と L2 VM の暗号鍵が異なるため，暗号化しないようにする必要があった．SEV パススルーでは，L1 VM と L2 VM は同一の暗号鍵を用いるため，互いに暗号化されたメモリ領域にアクセスすることができる．例えば，暗号化されたバッファを用いて DMA を実行することができる．そのため，共有するメモリ領域を L0 ハイパーバイザから盗聴されることはない．また，DMA バッファと暗号化しないようにした DMA バウンスバッファとの間でコピーを行う必要がないため，DMA 性能を向上させることができる．SEV に対応した L2 OS は共有するメモリを暗号化しないようにするため，L2 OS を SEV パススルーに対応させる必要がある．

5.2 SEV-ES パススルーの実装

基本的に SEV 仮想化と同様であるが，VMSA に関連する処理が大きく異なる．

5.2.1 VMSA の管理

SEV パススルーを用いる場合，L1 VM に適用されている SEV-ES を L2 VM にも適用するため，L1 VM に仮想 AMD-SP は提供されない．L2 VM を起動する時点ではすでに L1 VM が実行中であるため，もはや AMD-SP の `LAUNCH_UPDATE_VMSA` コマンドを実行することはできない．そのため，L1 ハイパーバイザが L2 VM の起動時に VMSA を確保しても，それを L2 VM 用に用いることはできない．

そこで，L1 VM の起動時に L0 ハイパーバイザが L2 VM 用の VMSA も確保し，あらかじめ `LAUNCH_UPDATE_VMSA` コマンドを実行しておく．確保した VMSA は L1 VM のメモリ空間にマップし，L2 VM の起動時に利用する．L2 VM 用の VMSA は L1 VM に割り当てられた仮想 CPU 数だけ確保する．L1 VM の仮想 CPU 数より L2 VM の仮想 CPU の総数の方が多い場合もありえるため，L2 VM の仮想 CPU は L2 VM 用に確保した VMSA とは一対一に対応させず，その時に割り当てられている L1 VM の仮想 CPU に対応する L2 VM 用の VMSA を動的に利用する．

L1 VM の起動時に `LAUNCH_UPDATE_VMSA` コマンドを実行するため，L2 VM 用の VMSA の CRC 値も L1 VM の起動時に計算されて TMR に格納される．そのため，L2 VM の起動時に仮想 CPU レジスタの初期値を VMSA に格納すると，TMR に格納されている CRC 値と不一致を起こして L2 VM の起動に失敗してしまう．同様に，L2 VM の仮想 CPU に割り当てられる L1 VM の仮想 CPU が切り替わった時にも対応する VMSA を書き換える必要があるが，CRC 値が変化してしまうため `VMRUN` 命令の実行に失敗してしまう．

この問題を解決するために，VMSA を書き換える前に CRC 値を計算しておき，VMSA の一部の値を変更することで VMSA を書き換える前後で CRC 値が変化しないようする．SEV-ES パススルーでは L1 VM と L2 VM が同一の暗号鍵を用いるため，L1 ハイパーバイザが暗号化されている L2 VM 用の VMSA の値を読み書きすることができる．VMSA の途中に位置する VM Exit 時にのみ値が格納される領域を調整に用いる．

CRC を逆算するために，調整に使う領域より前方のバイト列については通常の生成多項式で計算する．これは `CRC32` 命令を用いて高速に計算することができる．一方，調整に使う領域より後方のバイト列は係数を反転させた生成多項式で CRC を計算するため，この命令を利用することができない．そこで，2 つの 128 ビットの SSE レジスタと第 3 世代の EPYC プロセッサで導入された `VPCLMULQDQ` 命令を用いて GF(2) 有限体上の乗算を行う．同一の CRC 値であるがより短いバイト列を乗算によって求めていき，最終的に乗算によって剰余を求める Barret Reduction を用いて CRC 値を求める．端数バイト列には Slicing-by-N [14] を用いることにより，$8 \times n$ ビットを命令レベルの並列性を活かして計算する．

5.2.2 SIPI 仮想化

複数のプロセッサがある場合，最初に起動するプロセッサである Bootstrap Processor (BSP) 以外のプロセッサを Application Processor (AP) と呼ぶ．AP を立ち上げるためにプロセッサ間割り込みである Startup Inter Processor Interrupt (SIPI) が用いられており，KVM と QEMU はプロセッサ間割り込みを仮想化した機構を提供している．SIPI は AP で最初に実行されるアドレスを指定する．しかし，SEV-ES を有効にした VM では実行中にハイパーバイザが VMSA の中の命令ポインタレジスタの値を変更することはできないため，SIPI をエミュレートすることはできない．

そのため，従来は UEFI のイメージに AP で最初に実行されるアドレスを含めておき，ハイパーバイザが VM の起動時に AMD-SP の LAUNCH_UPDATE_VMSA コマンドを実行して AP の初期状態に設定する．VM 内で実行される UEFI は SIPI を用いてあらかじめ AP を立ち上げておき，UEFI の後で起動するゲスト OS が SIPI を発行するまで待機する．ゲスト OS は AP Jump Table と呼ばれるメモリ領域に AP で実行するアドレスを書き込み，SIPI を発行する．UEFI は SIPI を検知すると，SIPI で指定されたアドレスは無視し，AP Jump Table で指定されたアドレスにジャンプする．これにより，SEV-ES でも SIPI と同等の処理を行えるようになっており，SEV 仮想化でも同様の手法を用いている．

SEV-ES パススルーでは L1 ハイパーバイザが L2 VM 用の VMSA を変更することができるため，L2 VM 内で SIPI が発行されると，L1 ハイパーバイザが SIPI で指定されたアドレスを L2 VM 用の VMSA に書き込む．その際に，CRC が変わらないように VMSA の調整を行う．その後，VMRUN 命令を実行すると AP は指定したアドレスにジャンプする．

5.2.3 MMIO 仮想化

SEV-ES パススルーの場合には，MMIO を SEV 仮想化の場合と同様に仮想化することができる．L1 ハイパーバイザが L2 VM 用の VMSA を参照することができ，命令エミュレーションを行うことができるためである．MMIO のメモリ領域に対応する PTE の予約ビットを 0 に設定することで AE とし，#VC 例外を発生させずに直接，VM Exit を発生させる．L1 VM と L2 VM は同じ暗号鍵を用いるため，L1 ハイパーバイザは L2 VM 用の VMSA にアクセスして MMIO アクセスに用いられた CPU レジスタの値を取得することができる．

5.3 SEV-SNP パススルーの実装

SEV-SNP では，RMPADJUST 命令で RMP のエントリの VMSA ビットを 1 にすることにより VMSA を動的に確保できるため，SEV-ES パススルーとは異なり，L2 VM の起動時に VMSA を初期化する．VMSA の CRC 値も TMR に格納されないため，VMSA の書き換える時に CRC の逆算は行わない．

5.3.1 排他的な物理アドレス割り当て

SEV パススルーの大きな特徴は，L1 ハイパーバイザが L2 VM のメモリにアクセスできることである．同じ鍵を用いて暗号化されるようにするために，L2 VM にには L1 VM と同じ ASID が割り当てられる．SEV-SNP パススルーにおいては，L1 ハイパーバイザが L2 VM のメモリにアクセスする際に RMP によるチェックが行われるが，ASID の同一性判定には成功する．しかし，実 RMP のエントリには L2 物理アドレスが登録されており，対応する物理ページの L1 物理アドレスは一般的には異なる．そのため，ゲスト物理アドレスの同一性判定には失敗し，L1 ハイパーバイザは L2 VM のメモリにアクセスすることができない．一方，同一の L1 物理アドレスや L2 物理アドレスを持つ異なる物理ページが存在する場合には，L1 VM と L2 VM 間や L2 VM 間で物理ページを入れ替えることが可能になり，整合性の保護が不完全なものとなる．

そのため，SEV-SNP パススルーでは L1 VM とすべての L2 VM の間で利用する物理アドレス空間の範囲が重ならないように割り当てる．しかし，x86 アーキテクチャは物理アドレス 0xFFFFFFF0 から動作を開始する仕様になっており，QEMU で使われる UEFI の実装である OVMF もそれを前提に実装されている．一方で，Linux カーネルは DYNAMIC_MEMORY_LAYOUT と KASLR が有効であれば，ダイレクトマップも含めて物理アドレスの配置に縛られずに動作することができる．

そのため，Linux カーネルをブートさせることに特化した独自のファームウェアを qboot [15] をベースに実装した．このファームウェアは特定のアドレスを開始アドレスとして動作を開始し，#VC 例外ハンドラの設定，チップセットの初期化，ACPI テーブルのロードと配置，カーネルなどのロード，メモリマップの取得などを行う．これらの情報は QEMU Firmware Configuration (fw_cfg) Device とやり取りすることで取得する．最後に，Linux カーネルに渡すパラメータを設定し，Linux カーネルのエントリポイントにジャンプして初期化を行う．

L2 VM に割り当てる物理アドレスとして，L1 VM で使われていない連続したメモリ領域が必要となる．この領域を確保するために，あらかじめ L0 QEMU が物理アドレス 0x300000000000 以降の特定のメモリ領域を L1 VM に割り当てておく．そのメモリ領域をマップすることにより L1 QEMU に割り当てる．そのために，L1 Linux に専用のデバイス/dev/snp-mem を実装した．L1 QEMU については 2^{32} より高いアドレスだけでも動作できるようにした．QEMU は 32 ビットの ACPI テーブルを生成するため，ACPI テーブルの RSDT の 64 ビット版である XSDT を用い，DSDT，FACS への参照を 64 ビットアドレス対応にした．また，ファームウェアを 0x300000000000 から割り当てるようにした．hugepage を用いるとこのような特殊なメモリ管理を行わずとも確保できると考えられるが，実装は行っていない．

5.3.2 AP の初期化

INIT-SIPI による AP の初期化は指定されたコードセグメントを用いてリアルモードで動作する．そのためには下位 1 MiB の物理アドレス空間が必要であるが，排他的に物理アドレスを割り当てた L2 VM では利用できないため，代わりに ACPI v6.4 で追加された Wakeup Mailbox を用いる．この領域に命令ポインタ，CPU 番号，コマンドを書き

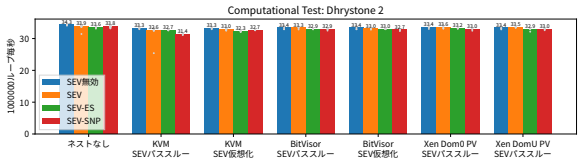


図 2 CPU 性能

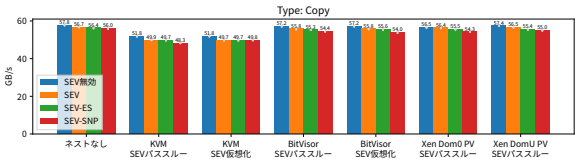


図 3 メモリコピー性能

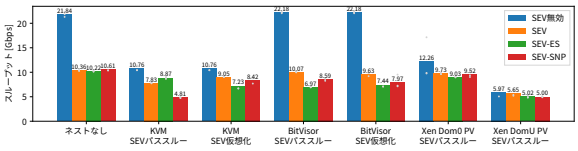


図 4 TCP スループット (1 並列)

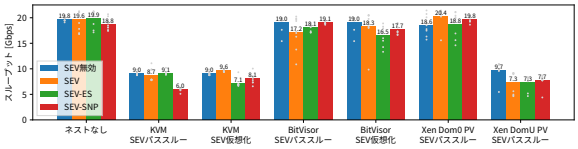


図 5 TCP スループット (2 並列)

込むと，物理アドレスと仮想アドレスが一致する Identity Paging で AP がロングモードで起動する．この機能への対応は Intel TDX のために Linux カーネルにすでに導入されているが，SEV-SNP パススルーの実装で動作するように修正した．また，L1 QEMU が生成する ACPI テーブルの MADT に Wakeup Mailbox の情報を追加し，Wakeup Mailbox のための MMIO 領域の実装を行った．

6. 実験

Nested SEV を用いて L2 VM の性能を調べるための実験を行った．この実験では，Nested SEV の保護方式として SEV 仮想化と SEV パススルーの 2 種類，SEV の種類として SEV，SEV-ES，SEV-SNP の 3 種類，L1 ハイパーバイザとして KVM，BitVisor，Xen の 3 種類，L0 ハイパーバイザとして KVM を用いた．比較のために，ネストした仮想化を用いない場合と SEV を適用しない場合についても調べた．ネストしない場合には L1 VM の性能を測定した．

この実験には，AMD EPYC 9334 の CPU，DDR5-4800 RDIMM 64 GiB のメモリ 2 枚を搭載したマシンを用いた．L0 ハイパーバイザとして Linux/KVM 6.11.0 と QEMU 9.1.0 を，L1 ハイパーバイザとして Linux/KVM 6.11.0 と QEMU 9.1.0 を用いた．L2 VM の OS には Linux 6.11.0 を，UEFI ファームウェアには OVMF を用いた．L1 VM には 12 個の仮想 CPU と 16 GiB のメモリを割り当て，L2 VM には 6 個の仮想 CPU と 8 GiB のメモリを割り当てた．ネストした仮想化を用いない場合は，L1 VM に 6 個の仮想 CPU と 8 GiB のメモリを割り当てた．

6.1 CPU 性能

Dhrystone ベンチマークを用いて CPU 性能の測定を行った．ベンチマーク結果を図 2 に示す．ネストした場合は 3〜4%程度のオーバヘッドがあった．SEV の種類と適用方式で大きな差は見られなかったが，SEV-SNP パススルーのみ 6%程度のオーバヘッドがあった．VM で動作させる場合の最大のオーバヘッドである VM Exit は割り込みなどを除けばほとんど発生しなかったため，性能差は大きくなかったと考えられる．

6.2 メモリ性能

STREAM ベンチマークを用いてメモリ性能の測定を行った．L3 キャッシュサイズが 128MiB であることから，それより十分大きな値として 2GiB の領域同士でのメモリコピーのテストを行った．ベンチマーク結果を図 3 に示す．ネストしない場合と BitVisor，ネストしない場合と Xen での性能差はあまりなく，SEV 無効，SEV，SEV-ES，SEV-SNP の順で性能が 3%程度低下した．ネストしない場合と KVM を比較すると，SEV 無効で 10%と性能の低下幅が大きかった．SEV-SNP を有効にすると SEV 無効からさらに 7%性能が低下した．

6.3 ネットワーク性能

iperf3 クラアイントを実行し，L0 Linux 上で動作する iperf3 サーバに接続して TCP のスループットの測定を行った．測定結果を図 4 に示す．ネストしない場合に SEV を有効にするとスループットが半減した．バウンスバッファを介する必要があるためではないかと考えられる．ネストした場合においても，BitVisor では同様の傾向が見られた．KVM と Xen の場合は SEV が無効の場合でもネストするとスループットが半減した．

次に，iperf3 クライアントと iperf3 サーバを 2 組同時に動作させた．2 並列でのスループットを図 5 に示す．ネストしない場合と BitVisor の場合は 2 並列となったことでオーバヘッドが吸収されたと考えられる．また，Xen の Dom0 でも性能が向上した．一方，KVM の場合は大きな性能の向上はみられなかった．

6.4 HTTP サーバ性能

HTTP サーバに 2 種類の GET リクエストを送信し，それぞれのリクエスト処理性能を測定した．リクエストの対象として，小さな HTML ファイル (45 バイト) と比較的大

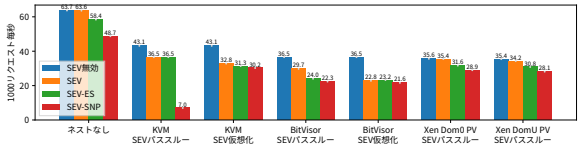


図 6 HTTP サーバのリクエスト処理性能 (小さいファイル)

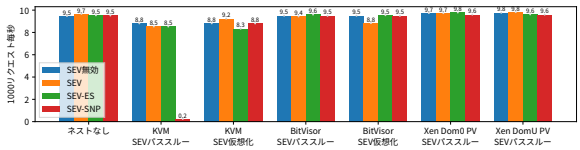


図 7 HTTP サーバのリクエスト処理性能 (大きいファイル)

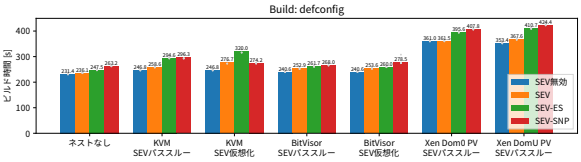


図 8 カーネルビルド時間

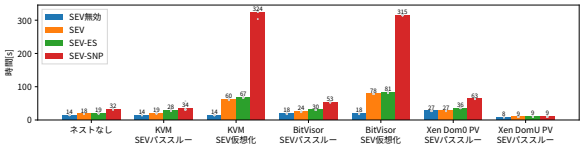


図 9 VM 起動から ssh 疎通までの時間

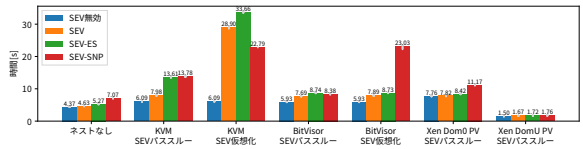


図 10 カーネル起動時間

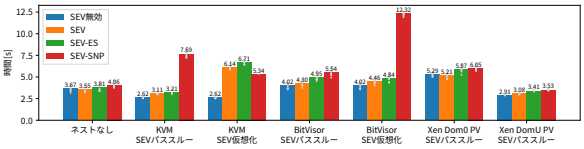


図 11 サービス起動時間

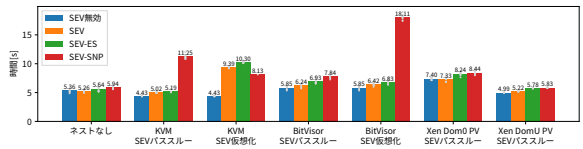


図 12 graphical.target 起動時間

きな HTML ファイル (115KB) を用いた．HTTP サーバとして Apache HTTP Server 2.4.62 を動作させ，HTTP リクエストを送るためのベンチマークとして bombardier v1.2.6 を用いた．ベンチマークは 10GbE のネットワークで接続された Intel Xeon Silver 4110 を 2 ソケット，DDR4-2666 RDIMM 32 GiB のメモリを 8 枚搭載したマシンで実行した．

200 並列でリクエストを送信した際のリクエスト処理性能を図 6 と図 7 に示す．SEV-SNP パススルーの性能が極端に低くなっていることが分かる．SEV パススルーの場合に L2 VM に重複しないように物理アドレスを割り当て，独自ファームウェアで動かした場合も同様に性能が低かったため，この実装に原因があると考えられる．概ねネストすると性能が低下し，SEV，SEV-ES，SEV-SNP の順に性能が低下した．SEV パススルーより SEV 仮想化の方が性能が低い傾向にあった．

6.5 カーネルビルド時間

Linux カーネル 6.1.0 をデフォルトの構成でビルドするのにかかる時間を測定した．ビルド時間を図 8 に示す．ネストしない場合と比べると，Xen の PV ゲストの性能が特に悪かった．Linux カーネルは非常に規模の大きなソフトウェアであり，ビルドには大量のプロセス生成をともなうため，ページテーブルへの書き込みにハイパーコールを用いる Xen の PV ゲストの性能が悪くなる傾向にあると考えられる．

6.6 起動時間

L2 VM の起動から ssh サーバへの接続が可能になるまでの時間を 10 回測定した．この時間は VM の初期化からファームウェアの起動，Linux カーネルのロードなどの時間も含まれる．測定結果を図 9 に示す．また，systemd を採用している Ubuntu Server 22.04 を用いていることから systemd-analyze コマンドによりユーザ空間に到達するまでにかかった時間と ssh.service が起動するまでにかかった時間，systemd によりユーザ空間の初期化にかかった時間 (graphical.target) も計測した．これらの測定結果を図 10，図 11，図 12 に示す．これらの時間はカーネルの初期化が終わってからの経過時間を示している．

SEV-SNP で起動が遅い原因として，利用するメモリに対してあらかじめ PVALIDATE 命令を実行する必要があるためと考えられる．事前に UEFI が 2 GB 程度の領域を PVALIDATE し，それ以外の領域は `accept_memory=eager` が指定されていなければ実行時に PVALIDATE される．BitVisor の SEV-SNP 仮想化については Linux カーネル起動前に全領域が PVALIDATE される．一方で，カーネル起動時間とサービス起動時間は SEV-ES 仮想化より高速であった．これは，事前に PVALIDATE 命令が実行された結果，NPT のエントリがあらかじめ作成され，ネストしたページフォールトによる VM Exit が減ったためである可能性がある．BitVisor の SEV 仮想化が遅くなった原因として，100MB 程度ある initramfs を AMD-SP 経由で暗号

化していることが挙げられる．SEV-SNP パススルーは現在の実装では最初に L2 VM を起動する時に L1 ハイパーバイザが PVALIDATE 命令を実行しているため，初回起動ではもう 20 秒ほど遅くなっている．

7. 関連研究

Microsoft のパッチ [10] は Hyper-V 上で L1 ハイパーバイザとして KVM を動作させ，L2 VM に SEV-SNP を適用可能にしている．そのために，L0 ハイパーバイザの Hyper-V は仮想化した AMD-SP を L1 VM に提供する．ただし，仮想 AMD-SP のコードは公開されていない．Nested SEV とは異なり，L1 VM に対して SEV を有効にすることはできず，L1 VM 内の KVM は L0 ハイパーバイザから保護されない．L0 ハイパーバイザが L1 VM のメモリに自由にアクセスできるため，Nested SEV よりも容易に実装することができる．L1 VM に SEV を適用できないため，L2 VM と同一の暗号鍵を用いる SEV パススルーは実現できない．

Hecate [11] は Hyper-V 上で L1 ハイパーバイザとして KVM を動作させ，L1 VM と L2 VM の両方に SEV-SNP を適用可能にしている．Hecate は SEV-SNP に対応していない OS を L2 VM で動作させることを目的としている．Hecate は VMPL を用いて実装されているため，SEV-SNP しかサポートできず，複数の L2 VM を動作させるのは難しい．L1 ハイパーバイザは最も高い特権レベルの VMPL0 で動作するため，SEV 仮想化のように L2 VM を保護することはできない．OpenHCL [12] は Hecate に似たシステムであり，paravisor と呼ばれる機能を実現している．SEV-SNP 以外に，Intel CPU における SEV に対応する機能である Intel TDX にも対応している．

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 [16] 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．KVM と QEMU について標準でサポートされているが，Intel TDX によって保護された VM 内では SGX を利用することはできない．

ネストしたエンクレイヴ [17] 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 [18] はクラウド内に SGX のエンクレイヴを作成し，Google NaCl [19] 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．同様に，AccTEE [20] は SGX のエンクレイヴ内で WebAssembly [21] 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を可能にする．

8. まとめ

本稿では，SEV を適用した L1 VM の中で SEV を適用した L2 VM を動作させることを可能にする Nested SEV を提案した．Nested SEV はシステム構成として，L1 VM と L2 VM の暗号化に異なる鍵を用いる構成と同一の鍵を用いる構成をサポートする．そのために，Nested SEV は SEV 仮想化と SEV パススルーの 2 種類の保護方式を提供する．SEV の拡張である SEV-ES と SEV-SNP もサポートし，セキュリティと性能のトレードオフをとることができる．L1 VM を動作させるハイパーバイザとして KVM，L1 VM 内で動作するハイパーバイザとして KVM，BitVisor，Xen，L2 VM 内で動作する OS として Linux を用いて Nested SEV を実装した．Nested SEV のオーバヘッドを調べるための実験を行い，Nested SEV の 2 種類の保護方式，3 種類の SEV，3 種類の L1 ハイパーバイザを用いた場合の性能を明らかにした．

今後の課題は，Nested SEV の性能を向上させることや Xen の完全仮想化に対応することなどが挙げられる．また，Intel TDX をネストした仮想化に対応させ，Nested SEV と同様の手法を用いることができるかを確認することも必要である．

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] Amazon Web Services, Inc: AMD SEV-SNP for Amazon EC2 instances, `https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/sev-snp.html` (2024).
[3] Google Cloud: Confidential VM documentation, `https://cloud.google.com/confidential-computing/confidential-vm/docs` (2025).
[4] Microsoft Corporation: About Azure con-

fidential VMs, https://learn.microsoft.com/en-us/azure/confidential-computing/confidential-vm-overview (2024).
[5] 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).
[6] Williams, D., Jamjoom, H. and Weatherspoon, H.: The Xen-Blanket: Virtualize Once, Run Everywhere, *Proc. European Conf. Computer Systems*, pp. 113–126 (2012).
[7] Liu, C. and Mao, Y.: Inception: Towards a Nested Cloud Architecture, *Proc. Workshop on Hot Topics in Cloud Computing* (2013).
[8] 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).
[9] 安東尚哉，光来健一：AMD SEV を用いてネストした VM を保護することによる安全な通信の追跡・制御，第 104 回 CSEC 研究会 (2024).
[10] Piotrowski, J.: [RFC PATCH v2 0/7] Support nested SNP KVM guests on Hyper-V, https://lore.kernel.org/lkml/20230213103402.1189285-1-jpiotrowski@linux.microsoft.com/ (2023).
[11] Ge, X., Kuo, H.-C. and Cui, W.: Hecate: Lifting and Shifting On-Premises Workloads to an Untrusted Cloud, *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, CCS '22, p. 1231–1242 (2022).
[12] Microsoft Corporation: microsoft/openvmm: Home of OpenVMM and OpenHCL., https://github.com/microsoft/openvmm (2024).
[13] Advanced Micro Devices, Inc.: AMD64 Architecture Programmer's Manual Revision 4.08 (2024).
[14] Kounavis, M. and Berry, F.: A systematic approach to building high performance software-based CRC generators, *10th IEEE Symposium on Computers and Communications (ISCC'05)*, pp. 855–862 (online), DOI: 10.1109/ISCC.2005.18 (2005).
[15] Paolo Bonzini: bonzini/qboot: Minimal x86 firmware for booting Linux kernels, https://github.com/bonzini/qboot (2022).
[16] Huang, K.: Introduction to Intel SGX and SGX Virtualization, Xen Project Developer and Design Summit (2017).
[17] 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).
[18] 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).
[19] Google, Inc.: Native Client, https://developer.chrome.com/docs/native-client/ (2016).
[20] 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).
[21] 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).