AMD SEV-ESによるネストしたVMの保護

瀧口 和樹[1]　光来 健一[1]

概要：機密性の高い情報をクラウドで扱うようになるにつれて，クラウドの内部犯から仮想マシン (VM) 内の機密情報を盗まれる危険性が増している．AMD のプロセッサは VM のメモリを透過的に暗号化する SEV と呼ばれる機能を提供しており，VM のメモリ上のデータの盗聴を防ぐことができる．我々は VM の中で VM を動作させるシステムにおいて，ネストした VM を SEV で保護することを可能にする Nested SEV を提案している．しかし，メモリに加えてレジスタも暗号化する SEV-ES には対応できていなかった．本稿では，ネストした VM を SEV-ES で保護することを可能にする Nested SEV-ES を提案する．Nested SEV-ES は SEV-ES 仮想化と SEV-ES パススルーの 2 つの方式を提供する．SEV-ES 仮想化は仮想 SEV-ES 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-ES パススルーは外側の VM に適用されている SEV-ES 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 に実装し，Nested SEV-ES 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，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，本稿ではクラウドが提供する外側の VM を L1 VM，その中で動作する VM を L2 VM と呼ぶ．例えば，クラウドの L1 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]．ネストした仮想化を用いるシステムにおいて L2 VM にも SEV を適用することを可能にするために，我々は Nested SEV を提案している [7]．第 2 世代以降の AMD EPYC プロセッサは SEV-Encrypted State (SEV-ES) と呼ばれる SEV の拡張を提供しており，メモリに加えてレジスタの状態も暗号化することができる．しかし，Nested SEV は SEV-ES には対応できていなかった．

本稿では，SEV-ES を適用した L1 VM の中で SEV-ES を適用した L2 VM を動作させることを可能にする Nested SEV-ES を提案する．Nested SEV-ES は用途に応じて，SEV-ES 仮想化と SEV-ES パススルーの 2 つの方式を提供する．SEV-ES 仮想化は SEV-ES を仮想化した仮想 SEV-ES を L2 VM に適用し，L1 VM と L2 VM で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-ES パススルーは L1 VM に適用されている SEV-ES をそのまま L2 VM にも適用し，L1 VM と L2 VM で同じ鍵を用いてのメモリとレジスタを暗号化する．これらの方式を KVM に実装し，Nested SEV-ES のオーバヘッドを測定した．

以下，2 章ではネストした仮想化に SEV を組み合わせる Nested SEV について述べる．3 章では Nested SEV を拡張した Nested SEV-ES を提案する．4 章では Nested SEV の実装について述べる．5 章では Nested SEV のオーバヘッドを調べ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. Nested SEV

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り，例えばクラウド上で仮想クラウドを実現するために用いられている [5]．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

[1] 九州工業大学

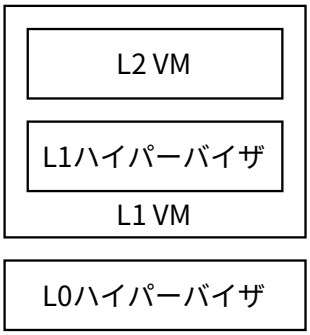
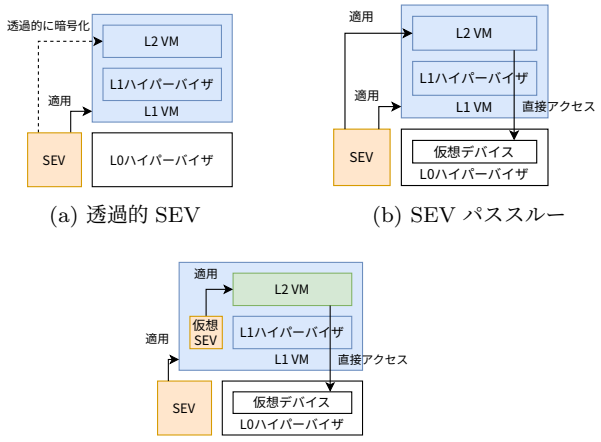


図 1 ネストした仮想化のシステム構成

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レベル 0 (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．

近年，クラウドにおいては内部犯から VM を保護するために AMD SEV が用いられるようになっている [2,3]．SEV は AMD のプロセッサに搭載されているセキュリティ機能であり，VM のメモリを透過的に暗号化することにより VM 外の内部犯によるメモリの盗聴を防ぐことができる．I/O に使う領域などが暗号化されていると VM の実行に支障が出るため，SEV では VM 内のページテーブルを用いてメモリ暗号化の制御を行う．ページテーブルエントリ (PTE) の C ビットを 1 にすると対応するページが暗号化され，0 にすると暗号化されない．一方，ページングが無効の場合のメモリアクセス，PTE へのアクセスや命令フェッチは常に暗号化される．

ネストした仮想化を用いるシステムにおいて L2 VM にも SEV を適用することを可能にするために，我々は Nested SEV を提案している [7]．Nested SEV は用途に応じて 3 つの方式を提供している．**透過的 SEV** は図 2(a) のように L1 VM に適用されている SEV の機能を用いて，L2 VM のメモリを透過的に暗号化する．L2 OS の対応は不要であるため，SEV 非対応の OS を用いることができる．**SEV パススルー**は図 2(b) のように 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 OS はページテーブルの C ビットを用いてメモリ暗号化を制御できるため，仮想 PCI デバイスへのパススルーアクセスが可能である．また，L2 VM がプロセッサの仮想化支援機構を用いて動作していない場合でも適用できる．**SEV 仮想化**は図 2(c) のように SEV を仮想化した仮想 SEV を L1 ハイパーバイザに提供し，L2 VM に適用する．AMD セキュアプロセッサ（AMD-SP）を仮想化した仮想 AMD-SP を L1 VM に提供することにより，L2 VM に対して独立した鍵管理などを行うことができる．



(a) 透過的 SEV　(b) SEV パススルー

(c) SEV 仮想化

図 2 Nested SEV の適用方式

透過的 SEV または SEV パススルーを用いると L1 VM と L2 VM の両方のメモリを同じ鍵で暗号化することができる．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，SEV 仮想化を用いると L1 VM と L2 VM の両方のメモリを異なる鍵で暗号化することができる．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．

Nested SEV を応用して安全な監視を行うシステムも提案されている．SEVmonitor [8] は SEV で保護された VM 内でエージェントを安全に動作させることにより，VM 外で侵入検知システム（IDS）を実行することを可能にしている．そのために，透過的 SEV または SEV パススルーを適用して L1 ハイパーバイザ内でエージェントを動作させる．そして，VM イントロスペクション（VMI） [9] を用いて監視対象システムが動作する L2 VM のメモリデータを取得し，IDS に転送する．また，SEV-tracker [10] はクラウド内にユーザの L1 ハイパーバイザを送り込み，クラウドサービスが動作する L2 VM の通信を追跡・制御する．クラウドとユーザ間の相互保護を実現するために，SEV 仮想化を適用することでクラウド内に送り込んだユーザ・ハイパーバイザをクラウドから保護しつつ，クラウドの L2 VM をユーザ・ハイパーバイザから保護する．

第 2 世代以降の AMD EPYC プロセッサは SEV の拡張である SEV-ES を提供している．SEV-ES は VM のメモリに加えて仮想 CPU レジスタの状態も暗号化する．そのため，ハイパーバイザは VM の仮想 CPU レジスタを直接参照して命令エミュレーションを行うことができず，SEV-ES 専用の対応が必要となる．Nested SEV は SEV-ES には対応しておらず，SEV-ES が有効になっている L1 VM の中で L2 VM を起動することができない．

3. Nested SEV の拡張

本稿では，Nested SEV を拡張して SEV-ES に対応させた Nested SEV-ES を提案する．

3.1 SEV-ES

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，ハイパーバイザ（ホスト）が `VMRUN` 命令を実行すると VM（ゲスト）に状態が遷移する．そして，外部割り込みや VM による I/O といった要因で VM Exit が発生するとハイパーバイザへ復帰する．例えば，I/O ポートへのアクセスをハイパーバイザがエミュレートする場合，VM がどのポートへどのレジスタを介して読み書きしようとしているかという情報が必要となる．VM Exit が発生した時に 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るため，ハイパーバイザが読み書きできる．一方，レジスタをどう使って I/O を行うかという情報を取得するためには VM Exit を発生させた命令をデコードする必要がある．SEV では VM のメモリが暗号化されているため命令のバイト列には直接アクセスできないものの，Decode Assists と呼ばれる機能によって命令のバイト列が VMCB に格納される．

しかし，SEV-ES では VM の仮想 CPU 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ため，ハイパーバイザはレジスタの状態にアクセスすることができない．また，SEV-ES では Decode Assists も利用できないため，VM Exit 発生時の命令のバイト列にもアクセスできない．そのため，SEV-ES では VM Exit の代わりに VM に VMM Communication (#VC) 例外が発生する．ゲスト OS 内のドライバが I/O を発生させると図 3 のような流れで処理が行われる．#VC 例外が発生するとゲスト OS の#VC 例外ハンドラが命令のデコードを行い，どのポートへどのレジスタを介して読み書きするかという情報を取得する．その情報を VM とハイパーバイザ間でデータを共有す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を行う．処理が完了すると `VMRUN` 命令を実行して VM に復帰し，#VC ハンドラから実行を再開する．#VC 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．

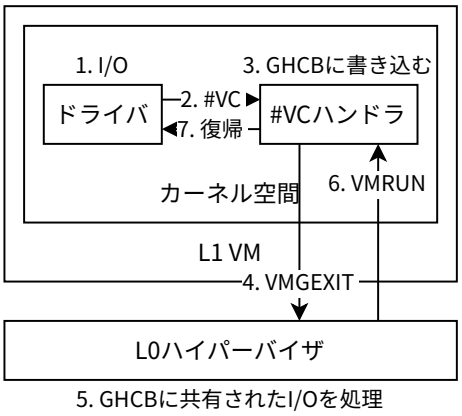


図 3 #VC 例外の処理の流れ

すべての VM Exit がこのような#VC ハンドラと GHCB を経由したゲスト OS による処理が必要というわけではない．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ってしまう．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，無限ループに陥ってしまう．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．

3.2 Nested SEV-ES

Nested SEV-ES は SEV-ES を適用した L1 VM の中で SEV-ES を適用した L2 VM を動作させることを可能にする．Nested SEV-ES では，L1 ハイパーバイザを L0 ハイパーバイザに対する SEV-ES ゲストとして動作させられるようにする．そのために，L0 ハイパーバイザがアクセスする必要のある VMCB や GHCB などのメモリ領域や DMA バウンスバッファを暗号化しないようにする．それに加えて，#VC 例外が発生した時に#VC 例外ハンドラを用いて処理を行うようにする．また，L1 ハイパーバイザを L2 VM に対する SEV-ES ホストとして動作させられるようにする．そのために，L2 VM 用の VMSA の管理を行えるようにしたり，L2 VM の#VC 例外ハンドラで実行される `VMGEXIT` 命令の処理を行えるようにしたりする．

Nested SEV-ES は用途に応じて，SEV-ES 仮想化と SEV-ES パススルーの 2 つの方式を提供する．

- **SEV-ES 仮想化**
  SEV-ES を仮想化した仮想 SEV-ES を L1 VM に提供し，L1 ハイパーバイザが仮想 SEV-ES を L2 VM に適用する．L1 VM と L2 VM で異なる鍵を用いてメモリとレジスタを暗号化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ができる．そ

のために，暗号鍵を管理する AMD-SP を仮想化した仮想 AMD-SP を L1 VM に提供する．仮想 AMD-SP は SEV が必要とするゲスト管理用コマンドに加えて，VMSA を暗号化するために用いられる SEV-ES 用のコマンドを提供する．

- **SEV-ES パススルー**
  L1 VM に適用されている SEV-ES を L2 VM にもそのまま適用する．L1 VM と L2 VM は同一の鍵を用いてメモリとレジスタを暗号化するため，L1 ハイパーバイザが L2 VM のメモリやレジスタにアクセスすることができる．そのため，GHCB などを L1 VM と L2 VM の共通の鍵で暗号化することができ，L0 ハイパーバイザからはアクセスできないようにすることができる．また，L1 ハイパーバイザは L2 VM に対して命令エミュレーションを行うことができるため，#VC 例外を用いずに効率のよいメモリマップド I/O (MMIO) を実現することができる．

Nested SEV では同様の方式である SEV 仮想化と SEV パススルーに加えて，L2 OS の SEV 対応が不要な透過的 SEV と呼ばれる方式も提供していた．しかし，SEV-ES では透過的 SEV で用いていた SEV の機能である Virtual Transparent Encryption (VTE) を利用して L2 VM のメモリを透過的に暗号化することができない．また，L2 OS に#VC ハンドラを用意する必要があるため，L2 OS の SEV-ES 対応が必須である．

4. 実装

L0 ハイパーバイザおよび L1 ハイパーバイザとして KVM と QEMU を用いて Nested SEV-ES を実装した．

4.1 L2 VM 用の 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 のチェックサムを計算し，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 のチェックサムを計算して 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．

ハイパーバイザは VM の起動時に VMSA に暗号化されていない仮想 CPU レジスタの初期値を格納し，AMD-SP の `LAUNCH_UPDATE_VMSA` コマンドを用いて VM の鍵で VMSA を暗号化する．このコマンドではさらに TMR にチェックサムを格納する領域を確保して VMSA にその参照を書き込み，VMSA のチェックサムを計算して 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ハイパーバイザによって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．

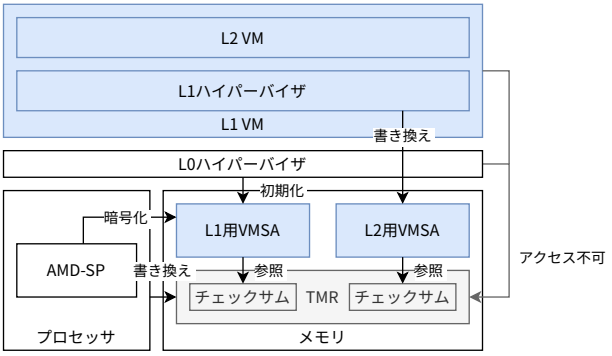


図 4 SEV-ES パススルーにおける VMSA の管理

4.1.1 SEV-ES 仮想化の場合

SEV-ES 仮想化では L1 VM に仮想 AMD-SP を提供するため，L0 ハイパーバイザと同様に L1 ハイパーバイザでも L2 VM 用の VMSA の管理を行うことができる．L1 ハイパーバイザは仮想 SEV-ES を L2 VM に適用し，L2 VM の起動時に VMSA を確保する．VMSA に暗号化されていない仮想 CPU レジスタの初期値を格納し，仮想 AMD-SP に対して `LAUNCH_UPDATE_VMSA` コマンドを実行する．仮想 AMD-SP は L0 ハイパーバイザを呼び出し，物理 AMD-SP にコマンドを転送する．物理 AMD-SP は L2 VM 用の VMSA のチェックサムを計算して TMR に格納する．

4.1.2 SEV-ES パススルーの場合

SEV-ES 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，図 4 に示すように，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．

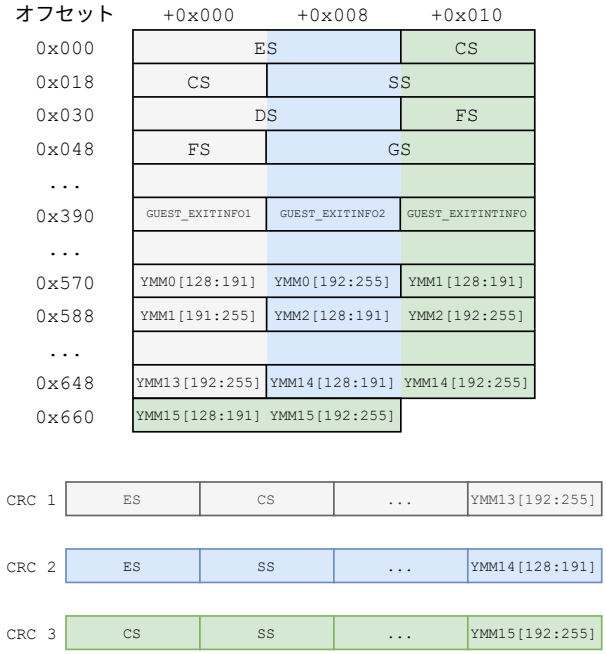


図 5 第 3 世代 EPYC における VMSA のチェックサム

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 のチェックサムも 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 に格納されているチェックサム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，チェックサム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 を書き換える前にチェックサムを計算しておき，VMSA の一部の値を変更することで VMSA を書き換える前後でチェックサム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．この処理を実装するために，AMD-SP の内部で用いられているチェックサムの計算方法を調べた．調査の結果，第 3 世代の EPYC では図 5 のように 3 つの独立した CRC-32C を計算していることが分かった．

CRC は文献 [11] の手法で通常の CRC の計算と同程度の計算量で逆算することが可能である．ただし, 3 つの 32 ビットの CRC の調整を行うためには，VMSA の中の 3 つの 32 ビットの領域が必要となる．予約済み領域やレジスタの値が格納されている領域は調整に利用できないため，VMSA の途中に位置する GUEST_EXITINFO1，GUEST_EXITINFO2，GUEST_EXITINTINFO の領域を調整に用いる．これらの領域は VM Exit 時にのみ値が格納されるため，VMRUN 命令の実行前に書き換えても動作に影響はない．

CRC の逆算は，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．SSE4.2 には CRC-32C の計算を行う CRC32 命令が含まれており，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，CRC32 命令を利用することができない．CRC を 1 ビットずつ計算していく手法が最も単純であるが，テーブルを用いて複数ビットを一度に処理することで高速化することができる [12]．ただし，多くのビットを処理できるようにすると指数的にテーブルの要素数が増加してしまうため，一般的には 8 ビットずつ計算する．その場合には 2^8 要素のテーブルが用いられる．テーブルの要素数を大幅に増やすことなく高速化する手法として Slicing-by-N [13] がある．$2^8 \times n$ 要素のテーブルを用意すると $8 \times n$ ビットを命令レベルの並列性を活かして計算できる．

x86 プロセッサの PCLMULQDQ 命令を使うとオーバヘッドを償却できるような長さのバイト列であればさらに高速に計算できる [14]．この命令は 2 つの 128 ビットの SSE レジスタを用いて GF(2) 有限体上の乗算を行う．同一の CRC 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 Barrett Reduction を用いて CRC を求める．第 3 世代の EPYC プロセッサからは新たに VPCLMULQDQ 命令が導入され，256 ビットの AVX のレジスタを用いた乗算が行えるため，さらに 2 倍の高速化を行える．そこで，VPCLMULQDQ 命令と端数バイト列のためのテーブル方式を併用した CRC の逆算の実装を行った．

4.2 仮想化支援命令の SEV-ES 対応

SEV-ES が有効でない場合，ハイパーバイザが仮想化支援命令の VMRUN 命令を実行するとまず，命令ポインタなど必要最低限の状態がホスト保存領域に退避される．そして，VMCB に保存された VM の状態が復元され，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 の最低限の状態が VMCB に退避される．そして，ホスト保存領域の状態が復元され，ハイパーバイザに状態が遷移する．VMRUN 命令で復元されない FPU レジスタなどの状態は通常のコンテキストスイッチのようにストア命令やロード命令を用いて退避と復元を行う．特殊な状態は仮想化支援命令である VMSAVE 命令と VMLOAD 命令を用いて退避と復元を行う．

SEV-ES が有効になっている場合は，ハイパーバイザが VMRUN 命令を実行すると SEV-ES が無効の場合と同様に，まず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 に保存される．そして，すべての状態がホスト保存領域から復元されるか初期化されるため，VM の状態は秘匿される．

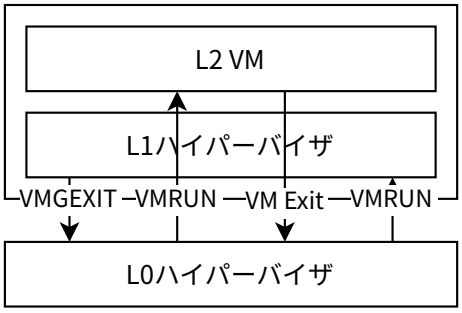


図 6 VMRUN 命令の処理の流れ

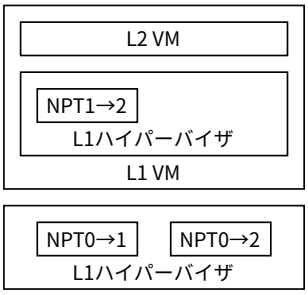


図 7 NPT の仮想化

Nested SEV-ES では，L1 ハイパーバイザがこれらの仮想化支援命令を実行した時に VM Exit を発生させるようにし，L0 ハイパーバイザが代わりに処理する．これらの VM Exit はいずれも NAE であるため，L1 ハイパーバイザに#VC 例外ハンドラを用意して処理する必要がある．ただし，L1 ハイパーバイザは SEV-ES ゲストとして L1 VM 内で動作しているため，VMRUN 命令で#VC 例外が発生することはあらかじめ分かっている．そのため，VMRUN 命令を実行して#VC ハンドラで VM Exit を処理する代わりに，直接，GHCB への設定を行って VMGEXIT 命令を実行することにより効率化を図る．この場合，図 6 のような流れで L2 VM が実行される．

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，Nested SEV-ES では L1 VM 用のホスト保存領域を使用しない．L1 ハイパーバイザが VMRUN 命令を実行した時には 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く，L1 ハイパーバイザで特殊な状態の退避するための VMSAVE 命令と復元するための VMLOAD 命令を実行する必要もない．

4.3 マルチプロセッサ初期化の SEV-ES 対応

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 (IPI) が用いられており，ハイパーバイザは IPI を仮想化した機構を提供している．INIT IPI を送ると AP がリセットされ，Startup IPI (SIPI) を送信すると指定したアドレスから実行を開始する．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仮想 CPU のレジスタを更新することはできないため，これら IPI をエミュレートすることはできない．

QEMU ではあらかじめ UEFI のイメージに AP で最初に実行するアドレスを含めておき，VM の起動前に LAUNCH_UPDATE_VMSA コマンドで AP の初期状態に設定する．そして，SIPI に指定されたアドレスは無視する．UEFI は SIPI を使いあらかじめ AP を立ち上げておき，UEFI が起動した後に実行されるゲスト 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 AP Jump Table に指定されたアドレスにジャンプする．

SEV-ES 仮想化の場合，L2 VM の場合もこの仕組みで AP を立ち上げる．一方，SEV-ES パススルーの場合は 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，より単純な SIPI 仮想化を行うことができる．そこで，最初の SIPI に関しては，SIPI で指定されたアドレスを L1 ハイパーバイザが L2 VM 用の VMSA に書き込み，チェックサム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．2 回目以降の SIPI は AP Jump Table を利用する．

4.4 NPT 仮想化の SEV-ES 対応

仮想アドレスから物理アドレスに変換するためにページテーブルが用いられるが，仮想化を行う場合はさらに VM の物理アドレスからホストの物理アドレスに変換するネストしたページテーブル (NPT) が用いられる．ネストした仮想化を行う場合，図 7 のように NPT を仮想化する．L1 ハイパーバイザは L2 VM の物理アドレスから L1 VM の物理アドレスに変換する $NPT_{1\to2}$ を用意し，L0 ハイパーバイザは L1 VM の物理アドレスからホスト (L0) の物理アドレスに変換する $NPT_{0\to1}$ を用意する．プロセッサはこの 2 つの NPT を使って L2 VM の物理アドレスからホストの物理アドレスへの変換を行うことはできない．そのため，L0 ハイパーバイザは $NPT_{1\to2}$ を基にして，L2 VM の物理アドレスからホストの物理アドレスに変換する $NPT_{0\to2}$ をシャドウページテーブルの手法を用いて生成する．

シャドウページテーブルの実装には，書き換え時に常に同期を保つ方式と書き換え時には同期を保たない方式がある．同期を保つ方式では PTE を書き込み保護し，L1 ハイ

パーバイザが書き換えようとすると VM Exit が発生するようにしておく．VM Exit が発生すると，L0 ハイパーバイザが命令エミュレーションを行って書き換えようとした PTE に対応するシャドウ PTE を書き換え，次の命令から実行を再開する．

同期を保たない方式でも同様に書き換え時に 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 発生時に書き込み保護を解除し，VM Exit を発生させた命令から実行を再開する．これにより，$NPT_{0\to2}$ が最新の状態ではなくなるが，書き換え後に行われる TLB フラッシュの際に同期を取ることでつじつまを合わせる．TLB フラッシュの際には同期が取れていないシャドウ PTE を最新の状態に書き換え，再び書き込み保護を行う．この処理は L1 ハイパーバイザが `VMRUN` 命令を実行して L2 VM に状態を遷移する際に行う．

KVM は性能上の理由により，レベル 1 ページテーブルは書き換え時に同期を取らない方式を，それより上のレベルのページテーブルは同期を取る方式を採用している [15, 16]．同期を取る方式では命令エミュレーションを行う必要があるため，SEV-ES を適用した L1 VM では利用できない．そこで，Nested SEV-ES では NPT 全体で書き換え時に同期を取らないようにすることで，NPT の仮想化を SEV-ES に対応させる．

4.5 L2 VM における MMIO の SEV-ES 対応

MMIO 領域は VM 内でのアクセス時に VM Exit を発生させることで仮想化される．対応する NPT の PTE に無効な値を設定することにより，ネストしたページフォールトによる VM Exit を発生させることができる．SEV-ES ではネストしたページフォールトで#VC 例外が発生し，#VC 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` 命令を実行することでハイパーバイザによって MMIO がエミュレートされる．しかし，ネストしたページフォールトのすべてが NAE であるとデマンドページングを行うことができない．そのため，NPT の PTE の予約ビットが 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 が発生するようになっている．

KVM のシャドウページテーブル実装では，NPT の PTE に予約ビットを設定してもシャドウ PTE には反映されず，L2 VM 内での MMIO 領域へのアクセスで NAE を発生させることができなかった．そのため，PTE の予約ビットの値もシャドウ PTE と同期を取るようにした．SEV-ES パススルーの場合には，L1 ハイパーバイザが命令エミュレーションを行うことができるため，PTE の予約ビットを 0 に設定することで AE を発生させ，効率よく MMIO を仮想化することが可能である．

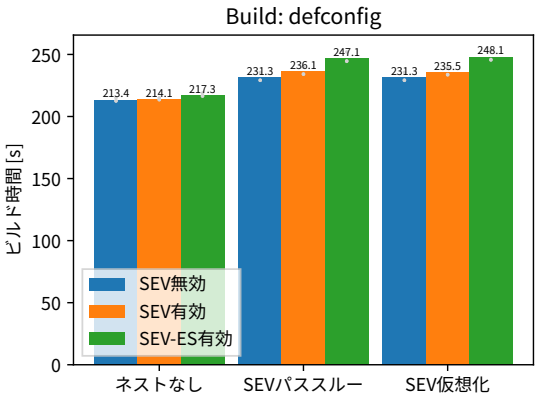


図 8 カーネルビルド時間

5. 実験

Nested SEV-ES のオーバヘッドを調べる実験を行った．この実験には，AMD EPYC 7443P（24 コア）の CPU を 1 基，DDR4-3200 RDIMM 128GiB のメモリ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 6.0.0 と QEMU v7.1.0-748-gf1d33f55c4 を，L1 ハイパーバイザとして Linux/KVM 6.0.0 と QEMU v6.2.0 を用いた．L2 の OS には Linux 6.0.0 を，UEFI として OVMF edk2-stable201903-5039-g0b633b1494 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，L0 で使われていない物理 CPU を割り当てた．L2 VM には 6 個の仮想 CPU と 8 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．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．MMIO 仮想化については，SEV-ES 仮想化では#VC 例外による方法を，SEV-ES パススルーでは命令エミュレーションによる方法を用いた．

5.1 カーネルビルド時間

Linux カーネル 6.1.0 を `defconfig` でビルドしたときにかかる時間を測定した．カーネルビルド時間を図 8 に示す．L1 VM と L2 VM のどちらにも SEV を適用しない場合と比べて，SEV 仮想化ではビルド時間が 1.8%増加し，SEV-ES 仮想化では 7.3%増加した．一方，SEV パススルーではビルド時間が 2.1%増加し，SEV-ES パススルーでは 6.8%増加した．このことから Nested SEV と比べて，Nested SEV-ES の方がいずれの方式でも少しオーバヘッドが大きいことが分かる．ネストした仮想化を用いない場合には，SEV のオーバヘッドは 0.3%，SEV-ES のオーバヘッドは 1.8%であった．

5.2 HTTP サーバの性能

VM 内の Apache HTTP Server 2.4.48 に bombardier [17] を用いて VM 内から並列にリクエストを送信した．Web

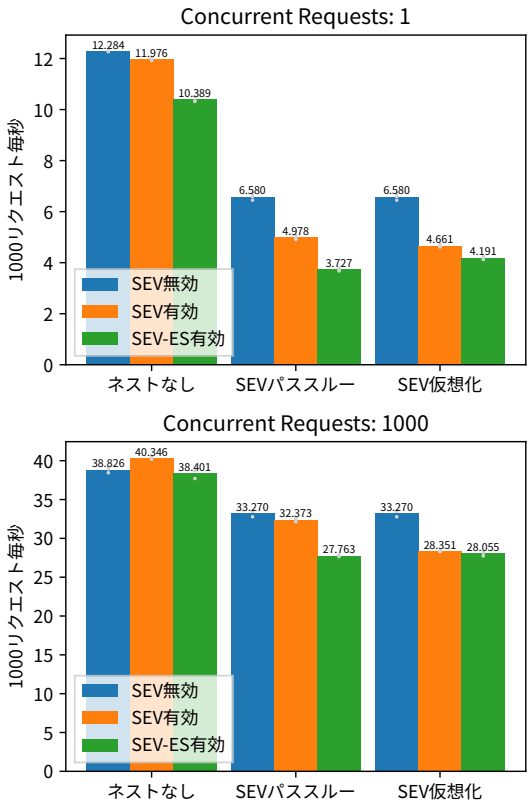


図 9 HTTP サーバのリクエスト処理性能

サーバのリクエスト処理性能を図 9 に示す．リクエストを 1 並列で送信した場合，SEV を適用しない場合と比べて，SEV 仮想化では性能が 29%低下し，SEV-ES 仮想化では 36%低下した．一方，SEV パススルーでは性能が 24%低下し，SEV-ES パススルーでは 43%低下した．このことから，Nested SEV の場合とは逆に，SEV-ES 仮想化の方が性能がよいことが分かった．ネストした仮想化を用いない場合には，SEV のオーバヘッドは 2.5%，SEV-ES のオーバヘッドは 15%であった．ネストした仮想化により性能が大幅に低下していることが分かる．

リクエストを 1000 並列で送信した場合，SEV を適用しない場合と比べて，SEV 仮想化では性能が 15%低下し，SEV-ES 仮想化では 16%低下した．この場合には SEV と SEV-ES のオーバヘッドはほぼ同程度であった．一方, SEV パススルーでは性能が 2.7%低下し，SEV-ES パススルーでは 17%低下した．SEV パススルーではオーバヘッドが非常に小さかったが，SEV-ES パススルーでは SEV-ES 仮想化と同程度のオーバヘッドとなった．ネストした仮想化を用いない場合には，SEV を適用するとむしろ高速化し，SEV-ES を適用してもオーバヘッドは 1.1%であった．ネストした仮想化によるオーバヘッドは 1 並列の場合よりも小さくなっている．

この実験は VM 外部への I/O を行わず VM 内で完結させたが，割り込み要因による VM Exit が多く発生した．SEV を無効にした場合，`VMSAVE` 命令と `VMLOAD` 命令の実行に仮想化支援機構が使えるため，VM Exit のオーバーヘッドは小さくなった．一方，SEV を有効にした場合はプロセッサによる仮想化支援を受けられないため，性能の低下が大きくなった．SEV-ES の場合はレジスタの状態の VMSA への退避によるオーバーヘッドがあるため，さらなる性能低下がみられた．

5.3 I/O 性能

暗号化されたメモリに対して DMA を行うことはできないため，SEV では暗号化しないようにしたバウンスバッファに対して DMA を行う．その際に，バッファとバウンスバッファとの間でコピーが行われるため，I/O 性能が低下する．SEV-ES パススルーでは L1 VM と L2 VM が同じ暗号鍵を用いるため，L1 ハイパーバイザと L2 VM の間では暗号化されたメモリに対する DMA を行うことができる．一方，SEV-ES 仮想化では L1 ハイパーバイザと L2 VM の間と L0 ハイパーバイザと L1 ハイパーバイザの間の両方でバウンスバッファが使われるため，性能がさらに低下する．

I/O スループットの最大の性能低下を調べるために，ブロックデバイスに対する読み出しスループットを測定した．ホストに 10 GiB の RAM ディスクを作成し，このブロックデバイスにアクセスする仮想デバイスを `virtio-blk-pci` を使って L0 QEMU に追加した．L1 VM でも同様に，L0 QEMU が提供するブロックデバイスにアクセスする仮想デバイスを `virtio-blk-pci` を使って L1 QEMU に追加した．

`readv` システムコールを用い，物理アドレスが連続していない 4 KiB の 1024 ページをバッファとした際の読み出しスループットを図 10 に示す．SEV を適用しない場合と比べて，SEV 仮想化では性能が 46%低下し，SEV-ES 仮想化では 48%低下した．一方，SEV パススルーでは性能が 28%低下し，SEV-ES パススルーでは 32%低下した．これらのことから，SEV-ES を適用することによる I/O 性能への影響は SEV を適用する場合とそれほど変わらないことが分かった．ネストした仮想化を用いない場合，SEV を適用すると性能が 16% 低下し，SEV-ES を適用すると 13%低下した．

2 MiB の 2 ページをバッファとした際の結果を図 10 に示す．現在の実装では，仮想 SEV と仮想 SEV-ES ではゲストに 2 MiB ページを割り当てることができていないため，この測定では除外している．SEV を適用しない場合と比べて，SEV パススルーでは性能が 36%低下し，SEV-ES パススルーでは 40%低下した．4 KiB の 1024 ページをバッファとして用いた場合と比べて性能は改善したが，SEV による性能低下は少し大きくなった．

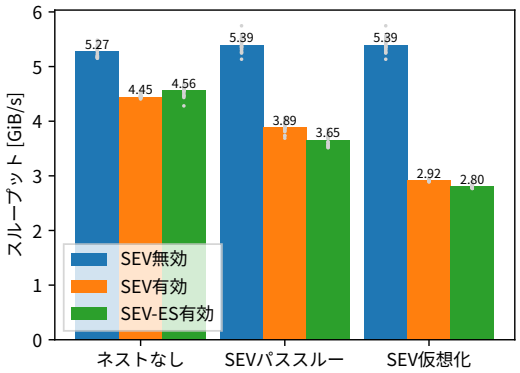


図 10 ブロックデバイスの読み出しスループット（バッファ: 4KiB × 1024)

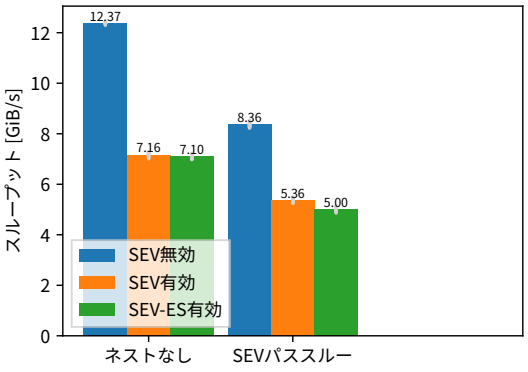


図 11 ブロックデバイスの読み出しスループット（バッファ: 2MiB × 2)

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 [18] 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 [19] 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 [20] はクラウド内に SGX のエンクレイヴを作成し，Google NaCl [21] 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 [22] は SGX のエンクレイヴ内で WebAssembly [23] 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．

Hyper-V は AMD-SP を仮想化することにより，L1 ハイパーバイザとして動作する KVM 上の L2 VM に SEV-Secure Nested Paging (SEV-SNP) を適用可能にしている [24]．SEV-SNP は SEV-ES の拡張であり，リプレイ攻撃やメモリデータの破壊，VM の物理ページの入れ替えによる攻撃なども防ぐことができる．Nested SEV-ES とは違い，L1 VM 内の KVM は SEV ホストとしてのみ動作し，SEV ゲストとしては動作しない．そのため，L1 VM 内で動作する Linux カーネルや QEMU は SEV で保護されていない．また，SEV-ES パススルーのような，L1 VM と L2 VM が同一の暗号鍵を用いる構成はサポートしていない．

7. まとめ

本稿では，SEV-ES を適用した L1 VM の中で SEV-ES を適用した L2 VM を動作させることを可能にする Nested SEV-ES を提案した．Nested SEV-ES は SEV-ES 仮想化と SEV-ES パススルーの 2 つの方式を提供する．SEV-ES 仮想化は SEV-ES を仮想化した仮想 SEV-ES を L2 VM に適用し，L1 VM と L2 VM で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-ES パススルーは L1 VM に適用されている SEV-ES をそのまま L2 VM にも適用し，L1 VM と L2 VM で同じ鍵を用いてのメモリとレジスタを暗号化する．これらの方式を KVM に実装し，Nested SEV-ES のオーバヘッドを測定した．

今後の課題は，ネストした VM に SEV-SNP を適用できるようにし，リプレイ攻撃やメモリデータの破壊，VM の物理ページの入れ替えによる攻撃などを防げるようにすることである．また，Nested SEV では対応していた Xen や BitVisor を L1 ハイパーバイザとして用いることができるようにすることも予定している．

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] 瀧口和樹，光来健一：Nested SEV：ネストした仮想化への AMD SEV の適用，第 34 回コンピュータシステム・シンポジウム (2022).
[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] 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] 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).
[11] Stigge, M., Plötz, H., Müller, W. and Redlich, J.-P.: Reversing CRC – Theory and Practice, HU Berlin Public Report SAR-PR-2006-05, Humboldt University Berlin (2006).
[12] Sarwate, D. V.: Computation of Cyclic Redundancy Checks via Table Look-Up, *Commun. ACM*, Vol. 31, No. 8, p. 1008–1013 (online), DOI: 10.1145/63030.63037 (1988).
[13] 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).
[14] Gopal, V., Ozturk, E., Guilford, J., Wolrich, G., Feghali, W., Dixon, M. and Karakoyunlu, D.: Fast CRC Computation for Generic Polynomials Using PCLMULQDQ Instruction.
[15] Guida, G.: [Xen-devel] [PATCH 0/4] Out-of-sync L1 shadows., https://lists.xenproject.org/archives/html/xen-devel/2008-06/msg00659.html (2008).
[16] Tosatti, M.: RFC: out of sync shadow, https://lore.kernel.org/all/20080906184822.560099087@localhost.localdomain/ (2008).
[17] bombardier: Fast cross-platform HTTP benchmarking tool written in Go, https://github.com/codesenberg/bombardier.
[18] Huang, K.: Introduction to Intel SGX and SGX Virtualization, Xen Project Developer and Design Summit (2017).
[19] 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).
[20] 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).
[21] Google, Inc.: Native Client, https://developer.chrome.com/docs/native-client/ (2016).
[22] 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).
[23] 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).
[24] 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).