Confidential Computing 解説 と そのユースケース紹介

Ring Minus Security LT #2

Acompany Inc.　平岡 拓海

Who are you?

平岡 拓海 (ヒラオカ タクミ) @takuuuuu_h__

- Acompany セキュアチャット Tech PdM
  - ソフトウェアエンジニアでもある
- Acompany新卒2年目
- 学生時代の研究
  - 学部: コンパイラ・プロセッサ
  - 修士: 差分プライバシー
- 趣味
  - 海外旅行 (最近はインドへ)
  - 霜降り明星

Agenda

1. Data in Use という空白
2. AMD SEV-SNP の仕組み
3. 検証する — Remote Attestation
4. プロダクトでの実装
5. まとめ

1. Data in Use という空白

データが守られていない瞬間がある

保護されてきたのは「保存中」と「転送中」だけ

場面	たとえ	守り方	状況
保存中 (at rest)	金庫の中の現金	ディスク暗号化	守れていた
転送中 (in transit)	現金輸送車	TLS	守れていた
使用中 (in use)	レジでお札を数えている瞬間	？	**空白**

計算するには、データを一度「裸」の状態でメモリに展開するしかない。
OS もハイパーバイザも、クラウドの管理者も、その瞬間のメモリを読める。

> **この空白を埋めるのが Confidential Computing**

Confidential Computing の定義

Confidential Computing Consortium (Linux Foundation)

**計算を hardware-based, attested な TEE の中で行うことによって
data in use を保護すること**

出典: Confidential Computing Consortium 『A Technical Analysis of Confidential Computing』 v1.3 — CC の定義 §2.1 / TEE の定義 §3.1

そして TEE とは、次の 3 つを保証する環境として定義される

Data Confidentiality

使用中のデータを
権限外の主体が
閲覧できない

Data Integrity

使用中のデータを
権限外の主体が追加・
削除・改変できない

Code Integrity

実行中のコードを
権限外の主体が追加・
削除・改変できない

TEE 単体では attestability は任意。attested を要求しているのは Confidential Computing の定義の側

2. AMD SEV-SNP の仕組み

なぜ「SNP」なのか — 3 段階の進化

暗号化だけでは足りなかった

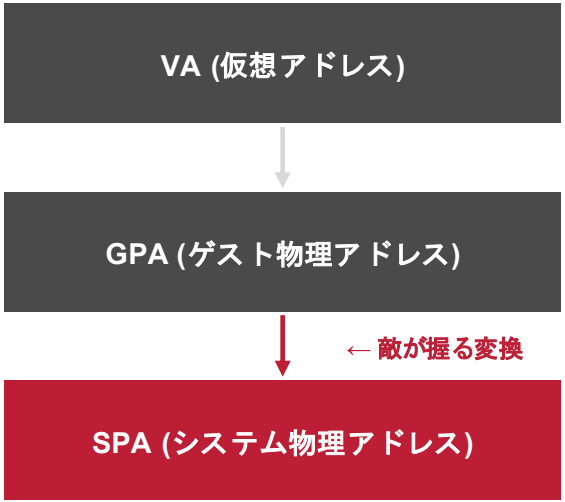
世代	追加された保護	残った穴
SEV	メインメモリを VM 固有 AES 鍵で自動暗号化	レジスタ状態が無防備
SEV-ES	レジスタ状態も暗号化 (Encrypted State)	完全性がない — 書き換え・リプレイ可
SEV-SNP	「常に最後に書き込まれた値を読む」を強制	物理攻撃・サイドチャネルは対象外 (Appendix)

暗号化は「読めなくする」だけ。読めないまま殴られるのは防げなかった。
SEV-SNP が足したのは 完全性 (Integrity)。

Secure Nested Paging = ネストページング (アドレス変換) を安全にする、という意味

なぜ完全性が難しいのか — アドレスの三層構造

GPA → SPA の変換表を握っているのはハイパーバイザ



アプリケーションが見る

OS が提供する抽象メモリ空間

ゲスト OS が見る

OS が「物理」と思い込んでいる領域 — 実際はハイパーバイザが見せている幻

CPU / ハイパーバイザが見る

DRAM 上の実際の物理位置。SEV-SNP はここで整合性をチェックする

暗号化だけでは「同じページを 2 箇所に見せる」「古いページに差し戻す」を止められない

RMP (Reverse Map Table)

前ページの脅威を、RMP と Page Validation がどう止めるのか

RMP エントリ — 4KB ページごとに 1 件。専用 CPU 命令でしか更新できない

ページテーブルは敵が持ち、RMP は CPU が持つ。この二重帳簿の照合が SNP の完全性の正体

VMPL — VM の「中」をさらに 4 層に割る

ゲスト OS が乗っ取られても、鍵とアテステーションは守られる

VMPL0	SVSM / パラバイザ — 鍵管理・vTPM・アテステーション
VMPL1	中間層 (用途別)
VMPL2	中間層 (用途別)
VMPL3	通常 OS — Linux 等の Rich OS・アプリケーション

VMPL3 は VMPL0 のメモリに
絶対アクセスできない
(ハードウェア保護)

Azure Confidential VM は
VMPL0 にパラバイザと vTPM を置く設計。
だからゲスト OS を無改造のまま
TEE 内で動かせる。

出典: Microsoft Learn — About Azure confidential VMs / CVM guest attestation design

TCB Versioning と VCEK — 「建物のセキュリティ等級」

Measured Boot だけでは実行基盤の正当性を検証できない

Measured Boot — 入室者の指紋認証

- 検証対象: ゲスト VM 内の初期イメージ
- ファームウェア / カーネル / initrd
- 課題1: 許可リストが爆発する
- 課題2: ロールバック攻撃を許しうる

TCB Versioning — 建物の等級

- 検証対象: 実行基盤 (AMD-SP FW・マイクロコード)
- 64 ビット構造体に各 SVN を統合
- 「バージョン 2.0 以上なら OK」と書ける
- 古い FW では新しい VCEK を生成できない

VCEK = チップ製造時の秘密 + 現在の TCB Version
「そのチップ」で「そのファームウェアバージョン」下でのみ生成できる鍵

だからロールバック攻撃は、運用ミスではなく暗号学的に不可能になる

信頼境界

SEV-SNP が引き直した TCB の線

信頼する (TCB)

Platform TCB

AMD Hardware (SoC)
AMD-SP
AMD-SP Firmware API
CPU Microcode Patch
Migration Agent

Guest TCB

Guest Firmware (OVMF)
Guest Kernel
Kernel Command Line
initrd

敵とみなす (Untrusted)

Hypervisor (KVM / QEMU)
Host BIOS
Device Drivers
External PCI Devices
Cloud Management SW
Other VMs

従来は「信頼するしかなかった」ハイパーバイザとクラウド管理ソフトを、敵の側に置ける

3. 検証する — Remote Attestation

Remote Attestation — 「信じる」のではなく「検証する」

SEV-SNP のアテステーションフロー

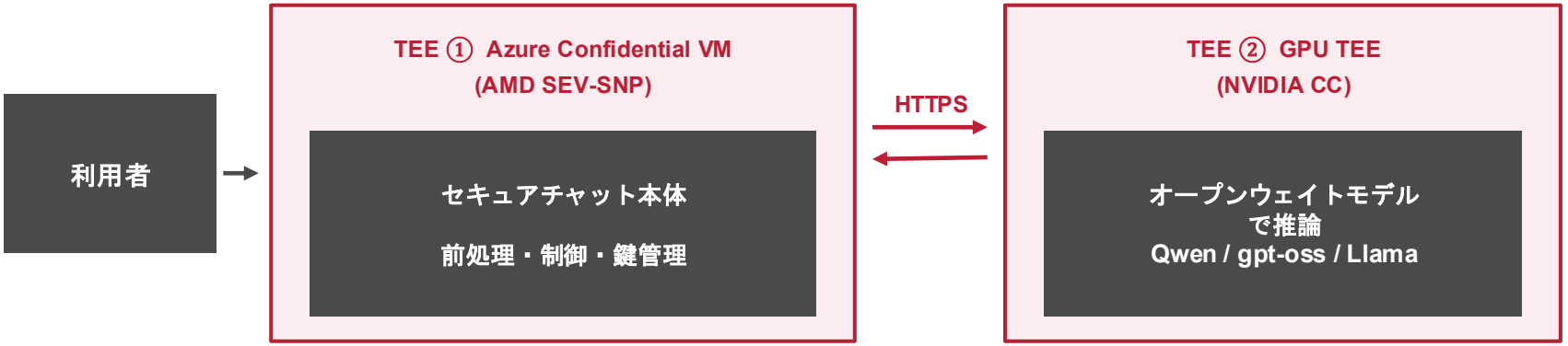
1	**計測値の確定**	AMD-SP が Launch Digest (SHA-384) を確定させる
2	**nonce を投げる**	検証者が challenge を送る — リプレイ攻撃対策。省略した解説が多い
3	**VCEK 生成 → 署名**	CPU が現在の TCB Version から VCEK を生成し、ATTESTATION_REPORT (計測値 + TCB + ポリシー + nonce) に署名
4	**チェーン検証**	検証者が AMD KDS から証明書を取得し ARK → ASK → VCEK を検証。TCB Version も照合
5	**鍵を渡す**	一致して初めて、鍵を解放する / データを送る

> **RA が証明するのは「計測値どおりのコードが改ざんされず動いている」こと。それが良いコードかは証明しない**

4. プロダクトでの実装

セキュアチャット — プライベートモード

オープンウェイトモデルを CC 環境でホストし、推論を TEE 内で完結させる



外部 LLM ベンダーへの送信が発生しない — データもモデルも、2 つの TEE の外には出ない

https://service.acompany.tech/cas/secure-chat/

外部フロンティアモデルを使いたい場合は、PII をマスキングして送るモードで補完する

SKR (Secure Key Release)

Remote Attestation が実務でどう効いているか

① VM 起動時 — シークレットの取り出し

1	**事前: 暗号化**	運用者がシークレットを AES-256-GCM で暗号化し、Key Vault に置く
2	**VM 起動**	VMSS が起動し cloud-init が動き出す
3	**SKR で鍵を要求**	Secure Key Release — ここでアテステーションが走る。「正しい計測値の TEE である」ことを AMD のチップが署名した証明を提示
4	**TEE 内で復号**	Key Vault から ciphertext を取得し、TEE の中で復号してアプリへ渡す

② 暗号化 DB からデータを抜き出すとき

Always Encrypted の列を復号する鍵 (CMK) も、同じアテステーション検証を通って初めて解放される

Terraform state にも cloud-init にも平文の秘密は出てこない。契約書ではなく暗号学的な証拠でゲートしている

ただし — 「TEE を使えば安全」ではない

ハードウェアが守るのは境界であって、アプリケーションの設計ではない

起こりうること

- TEE 上でも「秘密を裸で出力する」プログラムは書けてしまう
- 運用者が SSH で入れれば、メモリ上の復号済み PII に触れられる
- 保護処理が失敗したとき、そのまま続行すれば守られないまま外へ出る

→

セキュアチャットの実運用

- TEE 境界を越えるデータフローを規約で明文化
- 本番では SSH を完全に閉じている (enable_ssh_access = false)
- アテステーション検証などの安全側パスは fail-closed (エラー時は停止)

TEE の「中」に自分で穴を開けたら、そこで終わり

5. まとめ

Confidential Computing とは、結局なにか

「誰も信じなくていい技術」ではなく、
「信じなければいけない相手を、できるだけ少なくする技術」

1	**信頼を消さず、集約する**	OS・ハイパーバイザ・クラウド事業者を信頼する代わりに、対象をハードウェアと TCB へ最小化する
2	**何を前提に置いているかを明示できる**	だから「何を守らないか」も同時に言える。物理アクセスを持つ攻撃者やサイドチャネルは守備範囲の外
3	**秘匿 (TEE) と検証 (RA) は両輪**	秘匿だけならブラックボックス化しただけ。検証できて初めて「信じる」が「確かめる」に変わる

物理攻撃・性能オーバーヘッド・機構とプロダクトの対応は Appendix に収録

まとめ

in use の空白を埋める技術

RMP が「誰のページか」を守る

VCEK が「何を起動したか」を署名する

秘匿と検証が揃って成立する

Appendix

限界 — 何を守ると言っていないのか

誠実に語るために

守備範囲の外にあるもの

- サイドチャネル攻撃と高度な物理攻撃
- 可用性 — 見られないだけで、止められるのは防げない
- サプライチェーン攻撃
- ハードウェアベンダーへの信頼は前提として残る

2025-2026 の物理攻撃

- BadRAM: SPD 改変で偽のメモリサイズを報告
- Battering RAM: $50 のインターポーザで RA を破る
- TEE.Fail: DDR5 バスの物理プロービング
- いずれも「物理アクセスを持つ攻撃者」= 想定外の相手

SEV-SNP が破れたのではなく、SEV-SNP がもともと守ると言っていない領域を突かれた。
信頼境界の図に「DIMM に物理的に触れる人」は最初から入っていない。

出典: BadRAM / Battering RAM / TEE.Fail

だから CC は万能薬ではなく、多層防御の一部として設計する

性能オーバーヘッド

「遅くて使えない」は、もう過去の話

計測対象	オーバーヘッド	備考
LLM 推論 (NVIDIA Hopper GPU)	**7% 未満**	典型的なクエリの大半。モデルが大きいほどゼロに漸近
LLM 推論 (モデル横断平均)	**9% 未満**	主因は GPU 内の計算ではなく PCIe 転送の暗号化
国内モデルでの検証 (TTFT)	**最大 約10%**	SEV-SNP + NVIDIA CC 環境での自社 PoC

オーバーヘッドの主因はメモリ暗号化ではなく CPU-GPU 間のデータ転送。
バッチが大きいほど、相対的なコストは下がる。

出典: 上 2 行 — Zhu et al. 『Confidential Computing on NVIDIA Hopper GPUs』 arXiv:2409.03992 / 最下行は自社 PoC 実測

ただしライブマイグレーションは SEV-SNP / TDX とも非対応。「止められる」前提の運用設計が要る

Measured Boot — 起動時に「指紋」を採る

ゲスト CPU が最初の命令を実行する前に、測定は完了している

1	**初期イメージロード**	ハイパーバイザがブートコード等をメモリに配置 (暗号化なし・機密データを含まない前提)
2	**測定 + 暗号化**	SNP_LAUNCH_UPDATE — AMD-SP がページを計測し Launch Digest を更新。ページ内容だけでなく GPA (配置場所) とページ種別もハッシュに含める → Re-mapping 攻撃を防ぐ
3	**Launch Digest 確定**	SNP_LAUNCH_FINISH — 全初期ページのロード完了時点で計測が確定。これが VM の「指紋」になる
4	**ID ブロック検証**	所有者が事前署名した Identity Block の期待値と実 Launch Digest を AMD-SP が照合。不一致なら VM は正当に起動できない

計測後、ページは VEK で現地暗号化され所有権が VM へ移る (Guest-Valid 状態)

SEV-SNP の機構と保護対象

技術パートのまとめ

技術要素	保護対象
メモリ暗号化 (VEK)	データ機密性 — メモリコントローラ層で透過的に暗号化
RMP	メモリ完全性、所有者検証
Page Validation	ゲストによる明示的な使用許可
ASID	ゲスト間分離 — HW 回路で所有者 ASID を照合
VMPL	VM 内の特権分離
TCB Versioning / VCEK	ファームウェアの信頼性検証・ロールバック防止

プライベートモードで実際に何を守っているかは次ページ

SEV-SNP の機構は、実際に何を守っているのか

解説してきた機構と、プライベートモードの対応

SEV-SNP の機構	プライベートモードで守っているもの
メモリ暗号化 (VEK)	プロンプト・KV キャッシュ・応答、そしてモデルの重みそのもの
RMP / Page Validation	ハイパーバイザによるページ差し替え・リプレイ・エイリアシング
ASID	同一物理ホスト上の他テナント VM からの覗き見
VMPL0 (パラバイザ + vTPM)	ゲスト Linux が乗っ取られても鍵とアテステーション基盤は分離
Measured Boot + VCEK	Golden Image・モデル配布物の改ざん、古い FW へのロールバック
GPU TEE (NVIDIA CC)	GPU の HBM 上に展開された重みと中間活性 (CVM とは別 TEE として分離)

モデルの重みを守れることが、オープンウェイトを「自社の統制下でホストする」前提になる

参考 — Intel TDX との対比

同じ CVM でも、信頼の設計思想が違う

項目	AMD SEV-SNP	Intel TDX
仲介者	**AMD-SP (ハードウェア)**	TDX Module (Intel 署名済み SW)
完全性機構	**RMP**	Secure EPT + TME-MK
Launch measurement	**MEASUREMENT**	MRTD
Runtime measurement	**vTPM の PCR (ソフトウェア)**	RTMR[0-3] (ハードウェア)
署名チェーン	**AMD CA: ARK → ASK → VCEK**	Intel CA: Root → Interm. → PCK
プライベート判定	**C-bit = 1**	Shared ビット = 0

SEV-SNP は仲介者をハードウェアに、TDX はソフトウェアに置いた