日本総研
The Japan Research Institute, Limited

③実践・技術検証編

安全なデータ活用を可能にする「TEE」
― Confidential AIの基盤技術の動向と検証 ―

株式会社日本総合研究所　先端技術ラボ

2026年8月14日

［お問い合わせ］
執筆者：先端技術ラボ　藤後 英哲、森 毅
本レポートに関するお問い合わせにつきましては、当社ホームページの お問い合わせフォーム よりご連絡ください。

日本総研
The Japan Research Institute, Limited

目次

本シリーズの読み方

- 本シリーズは①概説・市場動向・戦略編、②詳細解説＜仕組み・理論＞編、③実践・技術検証編の3部構成から成る。
- 読者の役割や関心領域に応じ、推奨する参照ルートを以下に示す。

種別	主な内容	主な読者
①概説・市場動向・戦略編	TEEの概説・市場動向／ユースケース・戦略的示唆	経営層・企画部門・非技術者
②詳細解説＜仕組み・理論＞編	TEEの仕組み・信頼モデル・脅威モデル・性能理論の技術リファレンス	リードエンジニア・ITアーキテクト
③実践・技術検証編（本書）	Azure実機での構築手順・機能検証・性能実測と運用課題	開発エンジニア・検証担当者

想定読者別の読み方

- **経営層・企画：**
  ①概説・市場動向・戦略編を通読（必要に応じて②詳細解説編の第8章＜脅威モデルと攻撃手法＞、③実践・技術検証編の第7章＜まとめ＞を参照）
- **リードエンジニア・ITアーキテクト：**
  ①概説・市場動向・戦略編の第4章＜GPU TEE＞～第6章＜適用事例＞→ ②詳細解説編を通読 → ③実践・技術検証編から必要な結果を参照
- **開発エンジニア・検証担当者：**
  ③実践・技術検証編をメインに、①の第5章＜エコシステム＞、②詳細解説編の第5章＜アテステーション章＞等を参照
- **リスク管理・監査担当者：**
  ①概説・市場動向・戦略編の第2章＜背景＞・第7章＜技術展望と提言＞の残存リスク→ ②詳細解説編の第8章＜脅威モデルと攻撃手法＞

※ 役職によらず、TEEの基礎知識の習得には「①概説・市場動向・戦略編」の通読を推奨

日本総研
The Japan Research Institute, Limited

検証対象の全体像

- TEE(Trusted Execution Environment)のうち、SGX型TEE、CVM型TEE、Composite TEE構成を対象に、機能確認・アテステーション・性能 / ワークロードを整理する。
- SGX / CVM / GPUそれぞれで保護単位や信頼境界が異なるため、実行方式・証明方式・性能影響を同一観点で比較する。
- 性能値は単純な優劣比較ではなく、各TEEをどの用途に適用しやすいかを判断するための材料として扱う。

種別	対象	保護単位	機能確認	アテステーション	性能・ワークロード
SGX型TEE	Intel SGX	アプリの一部	Gramine実行	RA-TLS / DCAP、Azure MAA	CPU / Memory / File I/O、Nginx、Threadbench
CVM型TEE	SEV-SNP	VM全体	既存ワークロード実行	Azure MAA	CPU / Memory / File I/O、Nginx
CVM型TEE	TDX	VM全体	既存ワークロード実行	Azure MAA	CPU / Memory / File I/O、Nginx
Composite TEE	CPU側CVM + H100 CC	CPU側CVM + H100 GPU	Docker / PyTorch / vLLM	CPU側:Azure MAA GPU側:NVIDIA NRAS	GPU演算、転送、LLM推論

Intel SGX検証：検証目的と確認項目

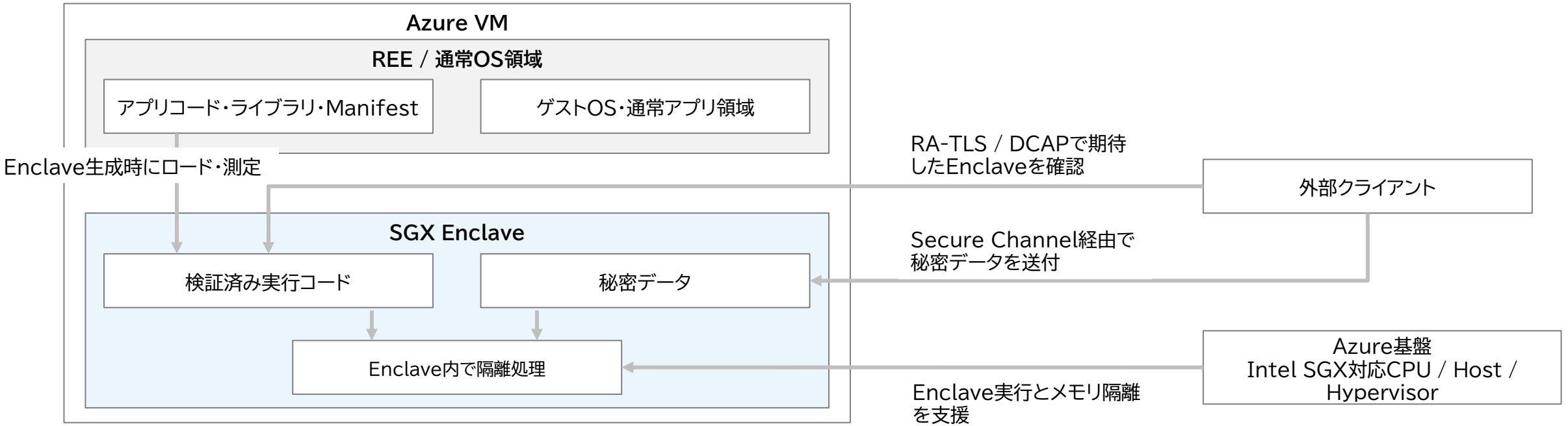
- Intel SGXでは、VM全体ではなくアプリケーションの一部をEnclave内に配置し、保護対象コード・データの隔離と外部からの真正性確認を行う。
- 本章では、SGXデバイス認識、GramineによるEnclave実行、RA-TLS / DCAP、Azure MAA連携、Gramine経由の既存アプリ実行性能、Intel SGX SDKによるSGX固有コストを確認した。

分類	確認項目	実施内容
環境確認	SGXデバイス認識	`/dev/sgx_enclave`、`/dev/sgx_provision` を確認
実行基盤	Gramine構築	SGX対応LibOSとしてGramine v1.7を構築
機能確認	Enclave実行	Hello Worldを gramine-sgx で実行
アテステーション	RA-TLS / DCAP正常系	クライアントからSGXサーバの真正性を確認
アテステーション	MRENCLAVE / MRSIGNER異常系	期待しないEnclave ID / Signer IDを拒否できるか確認
性能測定	Native / Gramine Direct / Gramine SGX比較	同一VM・同一バイナリ・同一条件で、起動時間、CPU、Memory、File I/Oを比較
性能測定	SGX SDK microbench	Gramineを使わないIntel SGX SDK Enclaveで、Enclave作成、ECALL、CPU、Memoryを測定
構成・性能確認	Enclaveサイズ設定・EPC利用条件	Gramine SGXの sgx.enclave_size 設定と、Enclave化したMemory処理の性能影響を確認

SGX型TEEの構成：アプリケーション単位の保護

- CVM型TEEがVM全体を保護単位とするのに対し、SGXはアプリケーション内の一部領域を保護単位とする。
- SGXでは、Enclave生成時に実行コードが測定され、投入された秘密データをEnclave内で隔離して処理する。
- 外部クライアントは、RA-TLS / DCAPで期待したEnclaveであることを確認した上で、Secure Channel経由で秘密データを送る。
- CVM型TEEと比べて保護単位を小さくできる一方、Enclave設計・実行設定・Memory性能が運用上の論点となる。

SGX型TEEの構成イメージ

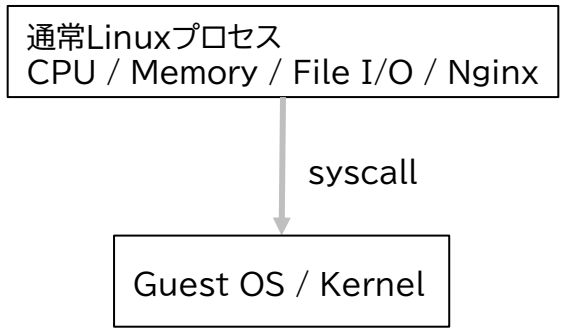


日本総研 The Japan Research Institute, Limited

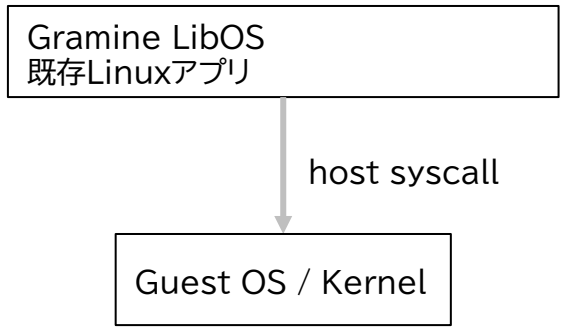
SGX実行方式の整理：Gramine SGXとIntel SGX SDK

- Gramine SGXは、既存LinuxアプリをGramine LibOS経由でSGX Enclave内に配置する方式であり、既存アプリ移行性を確認する。
- Intel SGX SDKは、アプリをEnclave内外に明示的に分割し、ECALL / OCALLで連携する方式であり、本検証ではECALLおよびEnclave内のCPU・Memory処理を確認する。
- 既存アプリ性能はNative / Gramine Direct / Gramine SGXで比較し、SGX固有コストはNative C / SGX SDK Enclaveで参考比較する。

Native

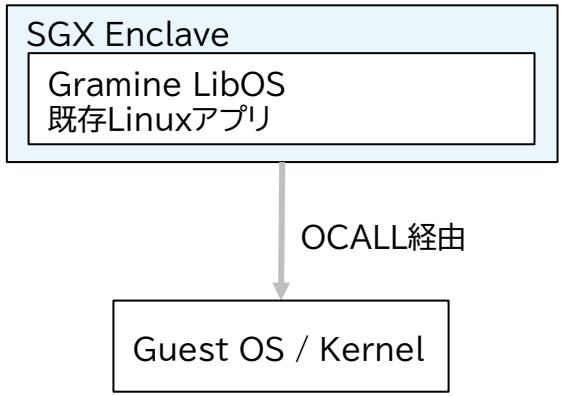


Gramine Direct



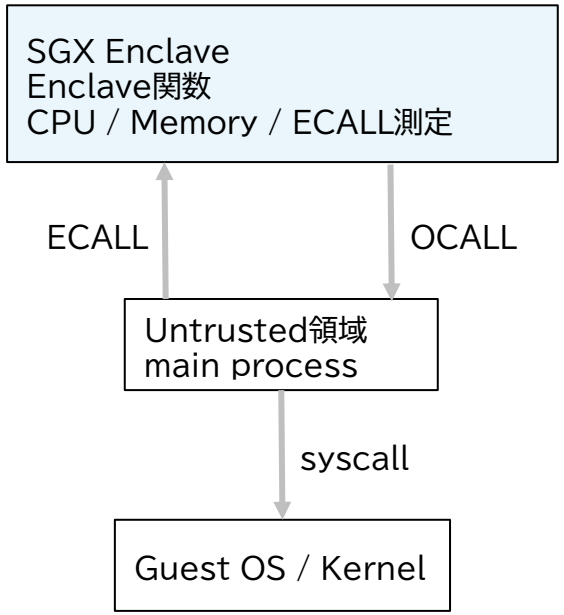
注：SGX Enclaveは使わない。
注：Gramine自体の影響を見る。

Gramine SGX



注：既存アプリをSGX化した場合の統合影響を見る。

Intel SGX SDK



注：Gramineを含まないSGX固有コストを見る。
注：NginxやFile I/Oの既存アプリ比較には向かない。

機能確認：GramineによるEnclave実行

- SGXで既存Linuxアプリを Enclave内で実行する方式として、Gramine manifestの作成・SGX用署名・gramine-sgxによる起動を行った。本検証では、Hello WorldアプリがGramine SGX上で起動し、SGX Enclave内で実行できることを確認した。
- Gramine SGXは既存アプリ移行に向く一方、CVM型TEEと比べるとmanifest・署名・利用ファイル設定などの事前設定が必要となる。

Gramine SGXでの実行フロー

1. アプリケーションを用意
   例：Hello World

2. Gramine manifestを作成
   Enclave内で利用するファイル・ライブラリ・実行設定を定義

3. SGX用に署名
   manifestをもとにEnclave構成を確定

4. gramine-sgxで起動
   アプリケーションをSGX Enclave内で実行

5. 実行結果を確認
   Enclave内でアプリが正常起動することを確認

```
   <user>@<server>     :~$ bash ~/sgx_screenshot_outputs/01_gramine_sgx_hello_short_view.sh \
  | tee ~/sgx_screenshot_outputs/01_gramine_sgx_hello_short_view.txt
=== Gramine SGX Hello World: summary ===

[0] Work directory
/home/ <user> /tee-hello

[1] Input files
-rwxrwxr-x 1  <user>   <user>  16K Apr 28 02:34 hello
-rw-rw-r-- 1  <user>   <user>  551 Apr 28 02:34 hello.manifest.template

[2] Manifest generation
gramine-manifest hello.manifest.template hello.manifest : PASS

[3] SGX signing
gramine-sgx-sign --manifest hello.manifest --output hello.manifest.sgx : PASS
Measurement:

[4] Run inside SGX Enclave
(host_framework.c:545:init_enclave) debug:     mr_enclave:   e7b1c4bbd54026b62ae75d4b599a9123ecd91b34807fc0ef3dba9fd882416b14
(host_framework.c:548:init_enclave) debug:     isv_prod_id:  0
(host_framework.c:549:init_enclave) debug:     isv_svn:      0
(pal_main.c:500:print_warnings_on_insecure_configs)   - sgx.debug = true                          (this is a debug enclave)
(libos_init.c:384:libos_init) debug: Host: Linux-SGX
(libos_fs_encrypted.c:410:get_or_create_encrypted_files_key) debug: Successfully retrieved special key "_sgx_mrenclave"
(attestation.c:364:init_sgx_attestation) debug: host is Linux-SGX and remote attestation type is 'none', skipping /dev/attestation/quote file
Hello, World from inside SGX enclave!

[5] Result
Application executed inside SGX Enclave : PASS
```

注：本検証では、debug enclaveで確認。運用時はrelease enclave設定での再確認が必要。

日本総研 The Japan Research Institute, Limited

参考:Gramine SGX Hello World確認スクリプト

```
     <user>@<server>     :~$ cat > ~/sgx_screenshot_outputs/01_gramine_sgx_hello_short_view.sh <<'SH'
#!/usr/bin/env bash
set -u

OUTDIR="$HOME/sgx_screenshot_outputs"
mkdir -p "$OUTDIR/logs"

CANDIDATE=$(find ~/gramine ~/sgx-unfinished ~ \
  -maxdepth 6 \
  -name "hello.manifest.template" \
  2>/dev/null | head -n 1)

if [ -z "$CANDIDATE" ]; then
  echo "ERROR: hello.manifest.template not found"
  exit 1
fi

APP_DIR=$(dirname "$CANDIDATE")
cd "$APP_DIR" || exit 1

echo "=== Gramine SGX Hello World: summary ==="
echo
echo "[0] Work directory"
pwd

echo
echo "[1] Input files"
ls -lh hello hello.manifest.template 2>/dev/null

echo
echo "[2] Manifest generation"
gramine-manifest hello.manifest.template hello.manifest \
  > "$OUTDIR/logs/gramine_manifest.log" 2>&1

if [ -f hello.manifest ]; then
  echo "gramine-manifest hello.manifest.template hello.manifest : PASS"
else
  echo "gramine-manifest hello.manifest.template hello.manifest : FAIL"
fi

echo
```

```
echo "[3] SGX signing"
gramine-sgx-sign --manifest hello.manifest --output hello.manifest.sgx \
  > "$OUTDIR/logs/gramine_sgx_sign.log" 2>&1

if [ -f hello.manifest.sgx ]; then
  echo "gramine-sgx-sign --manifest hello.manifest --output hello.manifest.sgx : PASS"
else
  echo "gramine-sgx-sign --manifest hello.manifest --output hello.manifest.sgx : FAIL"
fi

grep -E "Measurement|MRENCLAVE|MRSIGNER" \
  "$OUTDIR/logs/gramine_sgx_sign.log" | head -n 5 || true

echo
echo "[4] Run inside SGX Enclave"
gramine-sgx ./hello \
  > "$OUTDIR/logs/gramine_sgx_run.log" 2>&1

grep -E "Hello|SGX|enclave|process exited" \
  "$OUTDIR/logs/gramine_sgx_run.log" | tail -n 8 || true

echo
echo "[5] Result"
if grep -qi "hello" "$OUTDIR/logs/gramine_sgx_run.log"; then
  echo "Application executed inside SGX Enclave : PASS"
else
  echo "Application executed inside SGX Enclave : CHECK LOG"
fi
```

日本総研 The Japan Research Institute, Limited

アテステーション検証：RA-TLS / DCAP

- RA-TLS / DCAPにより、通常LinuxクライアントがSGX Quoteを含む証明書を検証し、TLS終端が期待するSGX Enclaveに結び付いていることを確認した。
- 正常系では、期待するMRENCLAVE / MRSIGNERと一致する場合に接続を許可し、5/5 PASSした。
- 異常系では、期待しないMRENCLAVE / MRSIGNERを指定した場合に接続を拒否できることを確認した。

RA-TLS / DCAPによる検証フロー

1. SGX Enclave内でサーバを起動
   ```
   $ gramine-sgx ./server
   ```

2. サーバ証明書にSGX Quoteを付与
   Enclaveの測定値・署名者情報と、TLS公開鍵との結び付きを含める

3. 通常LinuxクライアントがRA-TLS接続
   証明書とSGX Quoteを検証し、TLS接続先がQuoteで示されたEnclaveに結び付いていることを確認

4. DCAPでQuoteを検証
   Quote署名、PCK証明書チェーン、TCB status、Quote collateralを確認

5. MRENCLAVE / MRSIGNERを照合
   期待するEnclave / Signerの場合のみ信頼

検証項目	確認内容	結果
RA-TLS / DCAP正常系	Quote署名・証明書チェーン・TCB状態等を検証し、RA-TLS接続できるか確認	PASS
MRENCLAVE照合	期待するEnclave IDと一致するか確認	PASS
MRSIGNER照合	期待するSigner IDと一致するか確認	PASS
MRENCLAVE異常系	期待しないEnclave IDを拒否できるか確認	PASS（拒否確認成功）
MRSIGNER異常系	期待しないSigner IDを拒否できるか確認	PASS（拒否確認成功）

注：本検証はDebug Enclaveを許可した条件で実施。本番運用時はRelease Enclaveおよび本番用検証Policyでの再確認が必要。

アテステーション検証：MAA JWTのPolicy判定

- SGX DCAP QuoteをAzure MAAに送信し、検証結果をMAA署名付きJWTとして取得した。
- その後、Relying Party側Verifier相当のPythonスクリプトを作成し、JWT署名・jku / issuer・時刻・SGX claimsをPolicy照合した。
- release JWTはALLOW、debug enclave・改ざんJWT・issuer不一致・MRSIGNER不一致・MRENCLAVE不一致はDENYとなることを確認した。
- nonce / freshnessのchallenge-response、およびMAA custom policyによるtoken発行前拒否は今後課題。

SGX DCAP Quote → Azure MAA JWT → Verifier判定フロー

1. SGX Enclave（Azure SGX VM内）
   DCAP Quoteを生成

2. Azure MAA
   Quote / Evidenceを検証｜MAA署名付きJWTを発行

3. Verifier（Relying Party側）
   JWT署名検証、jku / issuer、exp / nbf / iat確認
   debug、MRENCLAVE / MRSIGNER / ProductID / SVNを
   Policy照合
   Policy一致時のみ ALLOW

検証項目	結果
MAA JWT取得	成功
JWT署名 / jku / issuer / 時刻検証	PASS
release JWT + strict policy	ALLOW
debug enclave	DENY
改ざんJWT	DENY
issuer不一致	DENY
MRSIGNER / MRENCLAVE不一致	DENY

注：今回のVerifierは同一SGX VM上のPython実装。実運用ではAPI / KBS等の利用側サービスに配置する想定。

参考:SGX MAA JWT Verifier確認ログ

```
      <user>@<server>      :~/sgx-unfinished/01_maa/microsoft-azure-attestation/sgx.attest.sample.intel.sdk/validatequotes.core$ echo "=== p10 SGX MAA JWT External Verifier: summary ==="
[column -t -s $'\t' "$ART/04_results/summary.tsv"
=== p10 SGX MAA JWT External Verifier: summary ===
test                          decision  failed_checks         error
T01_release_strict_allow.json ALLOW
T02_debug_deny.json           DENY      debug_policy_ok
T03_tampered_jwt_deny.json    DENY      signature_ok,time_ok  JWT verification failed: InvalidSignatureError: Signature verification failed
T04_wrong_issuer_deny.json    DENY      issuer_ok
T05_wrong_mrsigner_deny.json  DENY      mrsigner_ok
T06_wrong_mrenclave_deny.json DENY      mrenclave_ok
```

T01 ： release JWTを正しいPolicyで検証し、すべての条件が一致するためALLOWとなることを確認
T02 ： debug enclaveのJWTを検証し、allow_debug=falseのPolicyに反するためDENYとなることを確認
T03 ： payloadを改ざんしたJWTを検証し、署名検証エラーによりDENYとなることを確認
T04 ： 期待するissuerを意図的に不一致にし、issuer_ok=falseでDENYとなることを確認
T05 ： 期待するMRSIGNERを意図的に不一致にし、mrsigner_ok=falseでDENYとなることを確認
T06 ： 期待するMRENCLAVEを意図的に不一致にし、mrenclave_ok=falseでDENYとなることを確認

日本総研 The Japan Research Institute, Limited

性能特性：Native / Gramine Direct / Gramine SGX比較

- Azure上のIntel SGX対応VMにおいて、Native / Gramine Direct / Gramine SGXを同一VM・同一バイナリ・同一条件で測定した。
- CPU loopは3方式でほぼ同等となり、アプリ内部の単純CPU処理では大きな低下は確認されなかった。
- 一方、Gramine SGXではEnclave起動・初期化込みのwall timeが大きく、File I/O readやMemory seq系でNative比の低下が見られた。

主要結果

- 起動時間：Native 0.57ms → Gramine SGX 約105秒
- CPU処理性能：Gramine SGXはNative比 約100.2%
- Memory 512MB seq_read：Gramine SGXはNative比 約85.4%
- Memory 512MB seq_rw：Gramine SGXはNative比 約86.8%
- File I/O 書込：Native比 約94.8～98.1%
- File I/O 読込：Native比 約68.8～73.3%

SGXではCPU演算はほぼ同等。一方、Enclave起動・初期化、I/O read、Memory seq系が主要な性能論点となる。

実行方式の違い

Native：通常のLinuxプロセスとして直接実行

Gramine Direct：Gramine経由で実行するが、SGX Enclaveは使わない

Gramine SGX：Gramine経由でSGX Enclave内にアプリを起動

観点	指標	Native	Gramine Direct	Gramine SGX	SGX / Native	解釈
起動時間	wall_ms_avg	0.569 ms	5.014 ms	約105秒	約18万倍	Enclave cold start・初期化の影響大
CPU	ops/sec	215.1M ops/s	214.9M ops/s	215.5M ops/s	100.2%	ほぼ同等
Memory	512MB seq_read	10,690.3 MB/s	10,648.3 MB/s	9,131.0 MB/s	85.4%	低下
Memory	512MB seq_rw	8,846.2 MB/s	8,816.2 MB/s	7,677.7 MB/s	86.8%	低下
File I/O	256MB 書込 MB/s	139.2 MB/s	139.1 MB/s	131.9 MB/s	94.8%	概ね近い
File I/O	256MB 読込 MB/s	4,194 MB/s	4,163 MB/s	3,073 MB/s	73.3%	低下
File I/O	1024MB 書込 MB/s	135.3 MB/s	135.7 MB/s	132.7 MB/s	98.1%	概ね近い
File I/O	1024MB 読込 MB/s	4,400 MB/s	4,386 MB/s	3,028 MB/s	68.8%	低下

注：Gramine SGXは sgx.enclave_size=2048M、sgx.debug=true。Memoryは同一バイナリ・CPU固定・順序ランダム化・cache flush条件で10回再測定した平均値。

日本総研 The Japan Research Institute, Limited

Intel SGX SDKによるEnclave実行

- Gramine SGXはGramine LibOSの影響も含むため、SGX固有コストを切り分ける目的でIntel SGX SDK Enclaveを測定した。
- Enclave作成は平均約1.57秒、Empty ECALLは平均約5.0μs/callだった。
- 単純CPUループはNative C比約99.4%で、Gramineを含まないSGX Enclave内CPU処理そのものの性能低下は限定的だった。
- Memory seq_rwは、malloc・pre-touchを測定前に分離してseq_rw本体のみを測定した結果、Native C比約88～89%となった。

指標	Native C	SGX SDK Enclave	SGX / Native	解釈
Enclave作成	-	1,567.6 ms	-	Enclave作成コスト
Empty ECALL	-	4.955 μs/call	-	Enclave境界呼び出しコスト
CPU loop	386.2 M ops/s	383.9 M ops/s	99.4%	ほぼ同等
Memory 64MB seq_rw	12,559.0 MB/s	11,204.0 MB/s	89.2%	低下
Memory 128MB seq_rw	11,606.1 MB/s	10,220.3 MB/s	88.1%	低下
Memory 256MB seq_rw	11,487.6 MB/s	10,105.7 MB/s	88.0%	低下

注：Memoryは、Native C / SGX SDK Enclaveで共通処理を使用し、malloc・pre-touchを測定前に分離したうえで、seq_rw本体のみを10回測定した平均値。CPU固定・順序ランダム化・cache flush条件を適用。

性能特性：Nginx Webワークロード / マルチスレッド

- Nginx Webワークロードでは、Gramine SGXで10KB / 1MB配信を実行し、HTTPレスポンス性能を確認した。
- Gramine SGXは、Gramine Direct比で10KBは約17.5%、1MBは約9.2%となり、Web / I/O系ではオーバーヘッドが大きく出た。
- 一方、pthreadベースのマルチスレッドCPU処理では、SGXでも1→4 threadsまではスケールし、4→8 threadsでは伸びが鈍化した。

Nginx結果表

対象	Native	Gramine Direct	Gramine SGX	SGX / Direct	解釈
10KB req/sec	75,493	121,636	21,310	17.5%	低下大
10KB Latency	0.85ms	0.66ms	3.00ms	4.5倍	遅延増
1MB req/sec	9,404	9,189	844	9.2%	低下大
1MB Latency	6.94ms	6.73ms	76.6ms	11.4倍	遅延増

Threadbench結果表

threads	Direct iters/sec	SGX iters/sec	SGX / Direct	SGX 1thread比	解釈
1	555.2M	535.0M	96.4%	1.00倍	ほぼ同等
2	1,111.1M	1,067.9M	96.1%	2.00倍	良好にスケール
4	2,213.0M	1,957.2M	88.4%	3.66倍	伸びはある
8	2,134.7M	1,913.9M	89.7%	3.58倍	頭打ち傾向

SGXではCPU並列処理は一定程度スケールする一方、Web / I/O系ワークロードではGramine Direct比で性能低下が大きく出た。既存アプリ移行時の性能影響はGramine SGXで評価し、SGX固有のECALL / CPU / MemoryコストはSGX SDK microbenchで補足的に確認した。

注：Native NginxとGramineサンプルではNginxバージョン・設定が異なるため、厳密な性能比較ではなく傾向評価として扱う。
注：Nginxなど既存LinuxアプリのSGX化は、SGX SDK単体ではなくGramine SGXのようなLibOS経由で評価する。

日本総研
The Japan Research Institute, Limited

検証目的と確認項目の整理

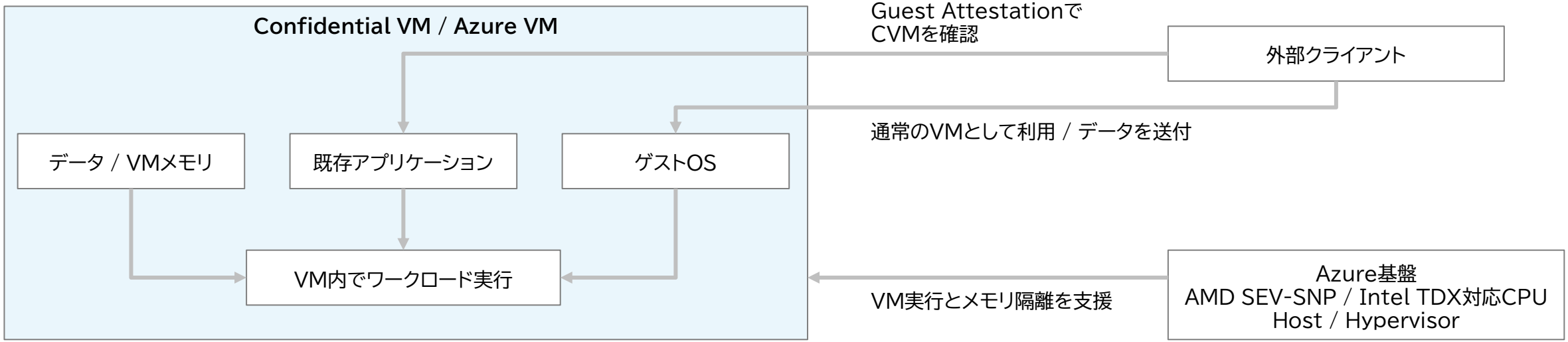
- CVM型TEEでは、アプリケーション単位ではなくVM全体を保護対象とし、既存アプリを大きく変更せずに実行できるかを検証した。
- 本章では、AMD SEV-SNPとIntel TDXについて、CVM作成、Secure Boot / vTPM、TEE関連情報、既存ワークロード実行、Azure MAA Guest Attestation、性能影響を確認した。
- SGXと比べて導入容易性が高い一方、信頼境界がゲストOS全体に広がるため、保護単位・TCB・運用上の前提が異なる。

分類	確認項目	実施内容
CVM作成	SEV-SNP / TDX CVMの作成	Azure上でConfidential VMを作成
起動設定	Secure Boot / vTPM	VM設定およびゲストOS内から確認
TEE確認	SEV-SNP / TDX関連情報	CPU情報、dmesg、TEE関連ログを確認
vTPM確認	vTPMデバイス・Properties	`tpm2_*`系コマンドでvTPMを確認
機能確認	既存ワークロードの実行	既存スクリプトを大きく変更せず実行
性能測定	Native VMとの比較	プロセス起動 / 初期化、CPU、Memory、File I/Oを比較
アテステーション	Azure MAA Guest Attestation	MAA JWT取得・decode・主要claim確認

CVM型TEEの構成

- CVM型TEEでは、VM全体を保護対象とし、ゲストOS・アプリケーション・VMメモリをまとめて保護する。
- SEV-SNP / TDX対応CPUにより、Hypervisor / Host OSからVM内部のメモリ内容を原則として平文で直接参照できない構造を実現する。
- SGXのようにアプリケーションをEnclave単位に分割する必要がなく、既存ワークロードを大きく変更せずに実行しやすい。
- 一方、ゲストOSも信頼境界に含まれるため、SGXとは保護単位・TCB・運用上の前提が異なる。

CVM型TEEの構成イメージ
SEV-SNP / TDX 共通



日本総研
The Japan Research Institute, Limited

機能確認：既存アプリの透過性

- CVM型TEEでは、SGXのようにアプリケーションをEnclave向けに分割・修正せず、通常VMに近い形で実行できるかを確認した。
- 本検証では、Python / NumPy / ベンチマークスクリプトをSEV-SNP / TDX上で実行し、既存ワークロードが大きな変更なしで動作することを確認した。

既存ワークロードの実行イメージ

SEV-SNP / TDX 共通

- 通常VM向けに作成したPythonスクリプトを、SEV-SNP / TDX上でそのまま実行
- SGXのようなEnclave分割、manifest作成、署名処理は不要
- CPU演算、Memory処理、File I/Oなどの基本ワークロードを通常Linuxプロセスとして実行
- 既存アプリを大きく変更せず、CVM環境へ移行しやすいことを確認
- ただし、ゲストOSも信頼境界に含まれるため、OS設定・パッチ管理は重要

CVM型TEEでは、Enclave向けの個別修正なしに、通常Linuxプロセスとして既存ワークロードを実行できることを確認した。

```
    <user>@<server>     :~/tdx_screenshot_outputs$ bash 01_tdx_existi
ng_workload_view.sh | tee 01_tdx_existing_workload_view.txt
=== Existing workload execution on TDX CVM ===

[0] VM
 Static hostname:      <server>
Operating System: Ubuntu 22.04.5 LTS
          Kernel: Linux 6.8.0-1053-azure-fde
    Architecture: x86-64

[1] Run workload without SGX-specific changes
$ python3 cvm_existing_workload.py
=== CVM existing workload demo: TDX ===
python: 3.10.12
numpy : 1.21.5

[1] CPU / Memory workload
NumPy matrix multiply 1024x1024 : PASS (573.75 ms)

[2] File I/O workload
write 64 MiB to /tmp         : PASS (31.94 ms)
read  64 MiB from /tmp       : PASS (19.82 ms)

[3] Result
Existing Python / NumPy workload on TDX CVM : PASS

[2] Point
No enclave split / no Gramine manifest / no SGX signing required
```

参考：TDX CVMにおける既存ワークロード実行スクリプト

```
      <user>@<server>       :~/tdx_screenshot_outputs$ cat > 01_tdx_existing_workload_view.sh <<'SH'
#!/usr/bin/env bash
set -euo pipefail

OUTDIR="$HOME/tdx_screenshot_outputs"
PY="$OUTDIR/cvm_existing_workload.py"

cat > "$PY" <<'PY'
import os
import sys
import time
from pathlib import Path

import numpy as np

def elapsed_ms(start):
    return (time.perf_counter() - start) * 1000

print("=== CVM existing workload demo: TDX ===")
print(f"python: {sys.version.split()[0]}")
print(f"numpy : {np.__version__}")

# CPU / Memory
x = np.random.rand(1024, 1024)
start = time.perf_counter()
y = x @ x
cpu_ms = elapsed_ms(start)

# File I/O
size_mib = 64
data = np.random.bytes(size_mib * 1024 * 1024)
path = Path("/tmp/cvm_io_test.bin")
```

```
start = time.perf_counter()
_ = path.read_bytes()
read_ms = elapsed_ms(start)

path.unlink(missing_ok=True)

print()
print("[1] CPU / Memory workload")
print(f"NumPy matrix multiply 1024x1024 : PASS ({cpu_ms:.2f} ms)")

print()
print("[2] File I/O workload")
print(f"write {size_mib} MiB to /tmp         : PASS ({write_ms:.2f} ms)")
print(f"read  {size_mib} MiB from /tmp       : PASS ({read_ms:.2f} ms)")

print()
print("[3] Result")
print("Existing Python / NumPy workload on TDX CVM : PASS")
PY

echo "=== Existing workload execution on TDX CVM ==="
echo
echo "[0] VM"
hostnamectl | grep -E "Static hostname|Operating System|Kernel|Architecture"

echo
echo "[1] Run workload without SGX-specific changes"
SHho "No enclave split / no Gramine manifest / no SGX signing required"
```

アテステーション検証：Azure MAA Guest Attestation

- SEV-SNP / TDX CVMについて、Azure MAA Guest Attestationを実行し、CVMの起動状態・TEE情報を検証した。
- Attestation Providerへゲストレポートを送信し、Azure MAA JWTとして検証結果を取得できることを確認。
- SEV-SNP / TDXともにJWT取得・decode・主要claim確認まで完了し、CVM型TEEとして外部検証可能であることを確認した。

Azure MAA Guest Attestationの検証フロー

1. CVMとしてVMを起動
   SEV-SNP / TDX Confidential VMを作成

2. ゲストOS内でTEE状態を確認
   Secure Boot / vTPM / TEE関連情報を確認

3. Guest Attestationを実行
   ゲストレポートをAzure MAAへ送信

4. Azure MAA JWTを取得
   Attestation Providerが検証結果をJWTで返却

5. JWT claimを確認
   issuer、TEE種別、Secure Boot状態、compliance状態などを確認

```
     <user>@<server>      :~/cvm-unfinished/cvm-attestation/azure-guest-attestation-sdk$
bash ~/tdx_screenshot_outputs/02_tdx_guest_attestation_short_view.sh \
  | tee ~/tdx_screenshot_outputs/02_tdx_guest_attestation_short_view.txt
=== TDX Azure MAA Guest Attestation: summary ===

[0] VM / TEE state
 Static hostname:        <server>
Operating System: Ubuntu 22.04.5 LTS
          Kernel: Linux 6.8.0-1053-azure-fde
    Architecture: x86-64

SecureBoot enabled
crw-rw---- 1 tss root  10,   224 Jun  9 12:52 /dev/tpm0
crw-rw---- 1 tss tss  253, 65536 Jun  9 12:52 /dev/tpmrm0

[1] Azure MAA Guest Attestation claims
"secureboot": true
"x-ms-attestation-type": "azurevm"
"x-ms-attestation-type": "tdxvm"
"x-ms-compliance-status": "azure-compliant-cvm"

[2] Result
JWT acquired      : PASS
decode            : PASS
TEE type claim    : PASS
compliance claim  : PASS
secure boot claim : PASS

Azure MAA Guest Attestation on TDX CVM : PASS
```

日本総研
The Japan Research Institute, Limited

参考：TDX CVMにおけるAzure MAA Guest Attestation確認ログ

```
      <user>@<server>       :~/cvm-unfinished/cvm-attestation/azure-guest-attestation-sdk$ cat > ~/tdx_screenshot_outputs/02_tdx_guest_attestation_short_view.sh <<'SH'
#!/usr/bin/env bash
set -u

LOG="$HOME/tdx_screenshot_outputs/tdx_maa_guest_attest_full.log"

echo "=== TDX Azure MAA Guest Attestation: summary ==="
echo

echo "[0] VM / TEE state"
hostnamectl | grep -E "Static hostname|Operating System|Kernel|Architecture"
echo
mokutil --sb-state 2>/dev/null || true
ls -l /dev/tpm* 2>/dev/null || true

echo
echo "[1] Azure MAA Guest Attestation claims"
grep -oE \
'"secureboot"[ ]*:[ ]*(true|false)|"x-ms-attestation-type"[ ]*:[ ]*"[^"]+"|"x-ms-compliance-status"[ ]*:[ ]*"[^"]+"|"x-ms-runtime"[ ]*:[ ]*"[^"]+"|"tdxvm"|"azure-compliant-cvm"' \
"$LOG" | sort -u

echo
echo "[2] Result"
if grep -Eqi "tdxvm|tdx" "$LOG"; then
  echo "JWT acquired      : PASS"
  echo "decode            : PASS"
  echo "TEE type claim    : PASS"
else
  echo "TEE type claim    : CHECK LOG"
fi

if grep -qi "azure-compliant-cvm" "$LOG"; then
  echo "compliance claim  : PASS"
else
  echo "compliance claim  : CHECK LOG"
fi

if grep -Eqi '"secureboot"[ ]*:[ ]*true|secureboot.*true' "$LOG"; then
  echo "secure boot claim : PASS"
else
  echo "secure boot claim : CHECK LOG"
fi

echo
echo "Azure MAA Guest Attestation on TDX CVM : PASS"
SH
```

日本総研 The Japan Research Institute, Limited

性能比較の前提条件：CVM / Native baseline

- SEV-SNP / TDXとも、対応するNative VMを用意し、vCPU・OS・Disk・測定ツールを可能な範囲で揃えた。
- SEV Native / TDX NativeともにSecure Boot / vTPMを有効化し、CVM側の起動設定に近いNative baselineとして測定した。
- ただし、CVM側はFDE※ kernel、Native側はAzure standard kernelであり、ホスト配置差も残るため、厳密な同一ハードウェア比較ではなく傾向評価として扱う。

観点	SEV-SNP CVM	SEV Native	TDX CVM	TDX Native
Region / Zone	East US / なし	East US / なし	West US / なし	West US / なし
VM size	Standard_EC4as_v5	Standard_E4as_v5	Standard_EC4es_v6	Standard_E4s_v6
vCPU / Memory	4 / 30GiB	4 / 31GiB	4 / 30GiB	4 / 31GiB
OS	Ubuntu 22.04.5 LTS	Ubuntu 22.04.5 LTS	Ubuntu 22.04.5 LTS	Ubuntu 22.04.5 LTS
Kernel	6.8.0-1046-azure-fde	6.8.0-1052-azure	6.8.0-1053-azure-fde	6.8.0-1052-azure
Secure Boot / vTPM	有効 / 有効	有効 / 有効	有効 / 有効	有効 / 有効
OS Disk	StandardSSD_LRS / 30GB / ReadWrite	StandardSSD_LRS / 30GB / ReadWrite	StandardSSD_LRS / 30GB / ReadWrite	StandardSSD_LRS / 30GB / ReadWrite
Resource Disk	なし	なし	なし	なし
Tools	nginx / wrk / OpenSSL / sysbench同一	nginx / wrk / OpenSSL / sysbench同一	nginx / wrk / OpenSSL / sysbench同一	nginx / wrk / OpenSSL / sysbench同一
残る主な差分	CVM / FDE kernel	Native / Azure kernel	CVM / FDE kernel	Native / Azure kernel

注：vCPU、OS、Disk、Secure Boot、vTPM、測定ツールは可能な範囲で統一した。一方、Azure上では物理ホスト、BIOS/UEFI、ファームウェア、ストレージ配置等を同一条件にできないため、Native比はTEE単体の純粋なオーバーヘッドではなく、本構成における傾向評価として扱う。

※FDE（Full Disk Encryption）：ディスク全体を暗号化する方式。本資料では、CVMで使用したazure-fdeを指す。

日本総研 The Japan Research Institute, Limited

性能特性：SEV-SNP単体

- SEV-SNP CVMとAMD Native VMに対し、プロセス起動 / 初期化、CPU、Memory、File I/Oを比較した。
- CPUはNative比98.5%、Memoryは88.7～98.1%となり、CPUは概ね同等、Memoryは一部サイズで低下が見られた。
- File I/Oは読込98.7～100.7%、書込97.8～104.8%となり、概ねNative同等水準だった。File I/OはOSディスク、キャッシュ、ホスト配置差の影響を含むため、傾向評価として扱う。

主要結果

- CPU処理性能：Native比 約98.5%
- Memory性能：Native比 約88.7～98.1%
- File I/O読込 ：Native比 約98.7～100.7%
- File I/O書込 ：Native比 約97.8～104.8%
- プロセス起動 / 初期化：Native比 約1.12倍、絶対差 約0.12ms

SEV-SNP CVMではCPU・File I/Oは概ねNative同等水準だった。一方、Memoryでは一部サイズで低下が見られた。

注：SEV-SNP CVMとNative baselineは同一リージョン・同一vCPU数で比較しているが、VMサイズ、OSイメージ、カーネルが完全同一ではないため、厳密なハードウェア性能比較ではなく傾向評価として扱う。

観点	指標	Native	SEV-SNP CVM	Native比	解釈
プロセス起動 / 初期化	wall_ms_avg	0.982 ms	1.097 ms	1.12倍	相対差はあるが絶対差は約0.12ms
CPU	ops/sec	535.4M	527.4M	98.5%	概ね同等
Memory	64MB MB/s	19,781.1	17,537.1	88.7%	低下
Memory	512MB MB/s	17,288.9	16,027.5	92.7%	やや低下
Memory	1024MB MB/s	14,049.4	13,776.2	98.1%	概ね同等
File I/O	256MB 書込 MB/s	110.3	115.6	104.8%	概ね同等
File I/O	256MB 読込 MB/s	1,704.4	1,715.5	100.7%	同等
File I/O	1024MB 書込 MB/s	132.1	129.2	97.8%	概ね同等
File I/O	1024MB 読込 MB/s	1,760.2	1,736.9	98.7%	概ね同等

日本総研 The Japan Research Institute, Limited

性能特性：TDX単体

- TDX CVMとIntel Native VMに対し、プロセス起動 / 初期化、CPU、Memory、File I/Oを比較した。
- CPUはNative比100.2%、Memoryは97.3～97.6%となり、基本的な演算・メモリ処理は概ねNative同等水準だった。
- File I/Oは読込99.2～100.5%で概ね同等だった一方、書込は69.0～86.3%となり低下が見られた。File I/OはOSディスク、キャッシュ、ホスト配置差の影響を含むため、傾向評価として扱う。

主要結果

- CPU処理性能：Native比 約100.2%
- Memory性能：Native比 約97.3～97.6%
- File I/O読込 ：Native比 約99.2～100.5%
- File I/O書込 ：Native比 約69.0～86.3%
- プロセス起動 / 初期化：Native比 約1.61倍、絶対差 約0.69ms

TDX CVMでは、CPU・Memory・File I/O読込は概ねNative同等水準だった。一方、File I/O書込では低下が見られた。

注：プロセス起動・CPU・Memoryは同一コード・同一条件で測定した。File I/OはOSディスク上の既存測定結果であり、ページキャッシュや仮想ストレージ経路、ホスト配置差の影響を含むため、傾向評価として扱う。

観点	指標	Native	TDX CVM	Native比	解釈
プロセス起動 / 初期化	wall_ms_avg	1.15 ms	1.84 ms	1.61倍	相対差は大きいが、絶対差は約0.69ms
CPU	ops/sec	585.0 M ops/s	586.2 M ops/s	100.2%	同等
Memory	64MB MB/s	11,891 MB/s	11,569 MB/s	97.3%	概ね同等
Memory	512MB MB/s	11,774 MB/s	11,494 MB/s	97.6%	概ね同等
Memory	1024MB MB/s	12,216 MB/s	11,922 MB/s	97.6%	概ね同等
File I/O	256MB 書込 MB/s	129.7	111.9	86.3%	低下
File I/O	256MB 読込 MB/s	4,513.3	4,534.6	100.5%	同等
File I/O	1024MB 書込 MB/s	138.9	95.8	69.0%	低下
File I/O	1024MB 読込 MB/s	4,629.0	4,592.2	99.2%	同等

日本総研 The Japan Research Institute, Limited

性能特性：CVM上のNginx Webワークロード

- SEV-SNP / TDX CVMと各Native VM上でNginxを起動し、10KB / 1MBファイル配信のHTTPレスポンス性能を比較した。
- 10KB配信ではNative比でSEV-SNP約73.0%、TDX約61.8%となり、小容量ファイルを高頻度に配信する条件で性能低下を確認した。
- 1MB配信ではSEV-SNP約92.7%、TDX約104.1%となり、10KB配信と比べて性能差が小さくなった。TDXはNativeと概ね同等水準だった。

測定条件

対象

- SEV-SNP CVM / TDX CVM
- 各CVMに対応するNative VM
- 各方式でCVM / Nativeを比較

ワークロード

- Nginxによる静的ファイル配信
- 10KB / 1MB
- Nginx 1.18.0（各VM共通）
- worker_processes 1 / sendfile on

測定

- `$ wrk -t4 -c64 -d30s`
- 同一VM内のループバック通信
- 事前に10秒間のウォームアップ
- 各条件10回平均

評価

- Requests/sec
- Latency
- Transfer/sec
- Native比（Requests/sec）

系統	target	Native req/sec	CVM req/sec	Native比	Latency Native → CVM	Transfer/sec Native → CVM	解釈
SEV-SNP	10KB	26,840	19,607	73.0%	2.41 → 3.31 ms	268.5 → 196.2 MB/s	低下
SEV-SNP	1MB	4,606	4,268	92.7%	14.43 → 15.49 ms	4,607.0 → 4,270.1 MB/s	やや低下
TDX	10KB	65,707	40,624	61.8%	0.98 → 1.58 ms	657.4 → 406.4 MB/s	低下
TDX	1MB	8,160	8,496	104.1%	7.96 → 7.56 ms	8,162.3 → 8,498.2 MB/s	概ね同等

本検証構成では、小容量ファイルを高頻度に配信する条件で性能低下が見られた一方、大容量配信では影響が相対的に小さい傾向が見られた。

注：Native比はRequests/sec平均から算出。VMサイズ、OSイメージ、カーネル、物理ホストが完全には一致しないため、厳密なハードウェア性能比較ではなく傾向評価として扱う。

CVM型TEE：SEV-SNP / TDX比較まとめ

- SEV-SNP / TDXはいずれもVM全体を保護単位とするCVM型TEEであり、既存ワークロードを大きく変更せずに実行できることを確認した。
- 両方式ともAzure MAA Guest Attestationにより、CVMの起動状態・TEE情報をJWT claimとして取得できることを確認した。
- 性能面ではCPU / File I/Oは概ね同等水準だった一方、Memoryの一部条件とNginx小レスポンス大量処理で性能低下が確認された。

CVM型TEEとして確認できたこと

- VM全体を保護対象とする構成を確認
- Secure Boot / vTPM / TEE関連情報をゲストOS内から確認
- Python / NumPy / ベンチマークスクリプトを大きな変更なしで実行
- Azure MAA Guest AttestationによりJWT取得・decode・主要claim確認まで完了
- CPU / Memory / File I/O / Nginxで性能傾向を確認

CVM型TEEは、既存ワークロードを移行しやすく、VM単位で保護とアテステーション情報を取得できる一方、Memory / Web / I/O系ではワークロード特性に応じた性能確認が必要となる。

観点	SEV-SNP	TDX	まとめ
保護単位	VM全体	VM全体	どちらもCVM型TEE
既存ワークロード	実行可能	実行可能	大きなコード変更なしで実行
Secure Boot / vTPM	確認済み	確認済み	CVMの起動設定を確認
Azure MAA	Guest Attestation成功	Guest Attestation成功	MAA JWTとしてTEE種別・Secure Boot状態等を取得
CPU / Memory	CPUは概ね同等、Memoryは一部低下	Native比で概ね同等	基本CPU / Memory処理では大きな低下は限定的
File I/O	概ね同等	書込で低下、読込は同等	ストレージ・キャッシュ条件差に注意
Nginx 10KB	Native比 約73.0%	Native比 約61.8%	小レスポンス大量処理で低下
Nginx 1MB	Native比 約92.7%	Native比 約104.1%（参考値）	大容量では低下は限定的

Composite TEE：検証目的と確認項目

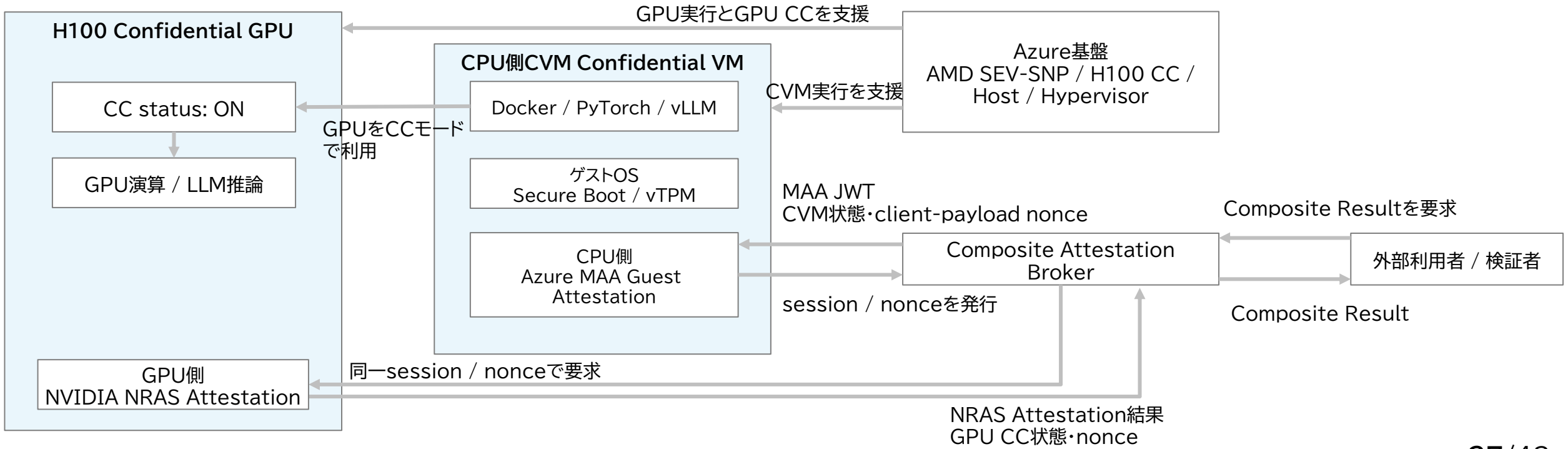
- Composite TEE構成では、CPU側CVMでVM内処理を保護し、H100 GPU CCでGPU上の計算処理を保護する構成を確認した。
- 本検証では、Secure Boot有効のCVM上でCPU側のAzure MAA Guest Attestationと、H100 GPU側のGPU Attestationを取得・確認した。
- 同一CVM上でDocker / PyTorch / vLLMを実行し、CPU側CVM + H100 CC構成でGPU / LLMワークロードが動作し、vLLM推論性能を条件別に測定できることを確認した。

分類	確認項目	実施内容
構成確認	CPU側CVM + H100 GPU CC	Confidential VM上でH100 Confidential GPUを利用する構成を確認
起動設定	Secure Boot / vTPM / TEE関連情報	Secure Boot有効状態、vTPM、TEE関連情報を確認
CPU側証明	Azure MAA Guest Attestation	CVMの起動状態・TEE情報をAzure MAA JWTとして確認
GPU側証明	GPU Attestation	H100のCC状態、Ready State、Attestation成功を確認
コンテナ実行	Docker / H100利用	コンテナ内からH100 GPUを認識し、GPUワークロードを実行
MLワークロード	PyTorch / CUDA	GPU上でCUDA / PyTorch演算が動作することを確認
LLM推論	vLLM	Qwen2.5 1.5B / 7Bで、prompt長・出力token数・batch size別の推論性能を確認
性能測定	GPU演算・転送・LLM	GEMM、Host-GPU転送、vLLM条件別推論性能を測定

Composite TEE構成：CPU側CVM + H100 CC

- Composite TEE構成では、CPU側はCVMとしてVM全体を保護し、GPU側はH100 Confidential GPUのCC機能でGPU上の計算処理を保護する。
- 外部利用者 / 検証者は、Composite Attestation Brokerが束ねたCPU側のAzure MAA Guest Attestation結果とGPU側のGPU Attestation結果により、CVM状態とGPU CC状態をまとめて確認する。
- アプリケーションはCVM内のDocker / PyTorch / vLLMからH100をCCモードで利用し、CPU側CVMとGPU側H100 CCを組み合わせた実行構成となる。

Composite TEEの構成イメージ



アテステーション検証：CPU側MAA + GPU Attestation

- CPU側ではAzure MAA JWT、GPU側ではNVIDIA NRAS EAT/JWTを取得・検証した。
- CPU/GPU tokenが同一session / nonceに束ねられ、両側のPolicyがPASSした場合のみComposite PASSとなることを確認した。
- 異常系として、別session由来tokenへの差し替え、片側Policy FAIL、CPU / GPU token改ざん、issuer不一致、nonce / session_id不一致、Composite Result JWT改ざんでFAILとなることを確認した。

CPU MAA + GPU Attestationの判定フロー

① CPU側 Azure MAA JWT

署名検証 client-payload nonce確認

issuer確認 Policy確認

② GPU側 NVIDIA NRAS EAT/JWT

署名検証 eat_nonce確認

issuer確認 Policy確認

③ Composite Broker

CPU / GPU nonce一致確認 | CPU client-payloadのsession_id整合性確認 | CPU / GPU両方のPolicy PASS確認

全ての条件一致時のみ Composite PASS

検証項目	結果
正常CPU + 正常GPU + 同一session / nonce	PASS
別session token差し替え	FAIL
CPU / GPUの片側がPolicy FAIL	FAIL
CPU / GPU token改ざん	FAIL
nonce / session_id不一致	FAIL
CPU / GPU issuer不一致	FAIL
Composite Result JWT改ざん	FAIL

注：本検証は、CPU側MAA JWTのclient-payloadとGPU側NRAS EATのeat_nonceを同一session / nonceで照合する実用版である。SEV-SNP REPORT_DATAへのnonce直接binding、およびCVMとGPUの割当関係を証跡レベルで確認することは未実施。

日本総研 The Japan Research Institute, Limited

参考：Composite Attestation 判定ログ

```
      <user>@<server>      :~/dual-tee/cpu-attestation/azure-guest-attestation-sdk$ cat "$ART/summary/p29_composite_log_view.log"
=== Composite Attestation: PASS / FAIL summary ===

Test  Result OK?   Meaning
----  ------ ---   -------
C01   PASS   true  正常CPU + 正常GPU + 同一session/nonce
C02   FAIL   true  別session GPU token差し替え
C03   FAIL   true  別session CPU token差し替え
C04   FAIL   true  CPUのみPASS / GPU FAIL
C05   FAIL   true  GPUのみPASS / CPU FAIL
C06   FAIL   true  CPU MAA JWT改ざん
C07   FAIL   true  GPU NRAS EAT/JWT改ざん
C08   FAIL   true  GPU Policy条件不一致
C09   PASS   true  CPU/GPU nonce一致
C10   FAIL   true  expected nonce不一致
C11   FAIL   true  session_id不一致
C12   FAIL   true  CPU issuer不一致
C13   FAIL   true  GPU issuer不一致
C14   FAIL   true  Composite Result JWT改ざん

Key points:
  - C01/C09: 正常なCPU/GPUが同一session/nonceで束ねられる場合はPASS
  - C02/C03/C10/C11: session/nonceのbindingが崩れるとFAIL
  - C04/C05: 片側だけPASSではComposite FAIL
  - C06/C07/C14: token/JWT改ざんはFAIL
```

C01/C09　：　正常なCPU / GPUが同一session / nonceで束ねられる場合はPASS
C02/C03　：　別session由来tokenの差し替えはFAIL
C04/C05　：　片側だけPASSではComposite FAIL
C06/C07　：　CPU / GPU token改ざんはFAIL
C10/C11　：　nonce / session_id不一致はFAIL
C14　：　Composite Result JWT改ざんはFAIL

日本総研 The Japan Research Institute, Limited

機能確認：Docker / PyTorch / vLLM実行

- CPU側CVM + H100 GPU CC構成上で、Dockerコンテナ内からH100 GPUを認識し、GPUワークロードを実行できることを確認した。
- PyTorchではCUDA利用可能であることを確認し、H100上でテンソル演算が実行できることを確認した。
- vLLMではQwen2.5 1.5B / 7Bの推論を実行し、Composite TEE構成上でもLLM推論ワークロードが動作し、後続の条件別性能測定に進めることを確認した。

Docker / PyTorch / vLLMの実行確認フロー

1. Docker環境を確認
   Docker / nvidia-container-toolkitを利用

2. コンテナ内GPU認識を確認
   nvidia-smiでH100 GPUを確認

3. PyTorchワークロードを実行
   CUDA利用、GPU上のテンソル演算を確認

4. vLLM推論を実行
   Qwen2.5 1.5B / 7Bで生成・条件別測定を確認

5. 実行結果を確認
   GPUが利用され、ワークロードが完走することを確認

```
    <user>@<server>     :~/dual-tee$ bash screenshot_outputs/03_function_docker_pytorch_vllm_view.sh | tee screenshot_outputs/03_funct
ion_docker_pytorch_vllm_view.txt
=== Function check: Docker / PyTorch / vLLM on Dual TEE ===

[0] VM / GPU
hostname:     <server>
name, uuid, driver_version
NVIDIA H100 NVL, GPU-  <UUID>                          , 580.126.09

[1] Docker GPU recognition
| NVIDIA-SMI 580.126.09             Driver Version: 580.126.09     CUDA Version: 13.0     |
|   0  NVIDIA H100 NVL                On  |   00000001:00:00.0 Off |                    0 |

[2] PyTorch CUDA smoke
torch: 2.9.0a0+50eac811a6.nv25.09
cuda available: True
gpu: NVIDIA H100 NVL
matmul shape: (4096, 4096)
elapsed_sec: 0.487154
RESULT: OK

[3] vLLM inference summary
--- Qwen2.5-1.5B-Instruct ---
model,tag,runs,load_time_sec,generation_time_sec_mean,generation_time_sec_std,output_tokens_per_sec_mean,output_tokens_per_sec_std
Qwen/Qwen2.5-1.5B-Instruct,qwen15b,5,16.898338794708252,0.4655512809753418,0.0010846387505921322,1099.7762080933742,2.561298956171046

--- Qwen2.5-7B-Instruct ---
model,tag,runs,load_time_sec,generation_time_sec_mean,generation_time_sec_std,output_tokens_per_sec_mean,output_tokens_per_sec_std
Qwen/Qwen2.5-7B-Instruct,qwen7b,5,153.2982714176178,0.9415751457214355,0.002318452155992923,530.1938320470873,10.98772184023422

[4] Result
Docker GPU recognition : PASS
PyTorch CUDA workload  : PASS
vLLM 1.5B inference    : PASS
vLLM 7B inference      : PASS
Dual TEE workload      : Docker / PyTorch / vLLM confirmed
```

日本総研
The Japan Research Institute, Limited

参考:Docker / PyTorch / vLLM実行確認ログ

```
      <user>@<server>       :~/dual-tee$ cat > screenshot_outputs/03_function_docker_pytorch_vllm_view.sh <<'SH'
#!/usr/bin/env bash
cd ~/dual-tee

echo "=== Function check: Docker / PyTorch / vLLM on Dual TEE ==="
echo
echo "[0] VM / GPU"
echo "hostname: $(hostname)"
nvidia-smi --query-gpu=name,uuid,driver_version --format=csv

echo
echo "[1] Docker GPU recognition"
grep -E "NVIDIA-SMI|Driver Version|CUDA Version|NVIDIA H100 NVL" \
  secureboot_true/logs/docker_nvidia_smi_true.log

echo
echo "[2] PyTorch CUDA smoke"
grep -E "torch:|cuda available:|gpu:|matmul shape:|elapsed_sec:|RESULT: OK" \
  secureboot_true/logs/pytorch_smoke_true.log

echo
echo "[3] vLLM inference summary"
echo "--- Qwen2.5-1.5B-Instruct ---"
cat secureboot_true/rich/results/vllm_qwen15b_summary_true.csv

echo
echo "--- Qwen2.5-7B-Instruct ---"
cat secureboot_true/rich/results/vllm_qwen7b_summary_true.csv

echo
echo "[4] Result"
echo "Docker GPU recognition : PASS"
echo "PyTorch CUDA workload  : PASS"
echo "vLLM 1.5B inference    : PASS"
echo "vLLM 7B inference      : PASS"
echo "Dual TEE workload      : Docker / PyTorch / vLLM confirmed"
SH
```

日本総研 The Japan Research Institute, Limited

性能特性：GPU内演算・Host-GPU転送

- CPU側CVM + H100 GPU CC構成上で、GEMMによるGPU内部演算と、Host-GPU間のH2D / D2H転送性能を測定した。
- GEMM 8192×8192 FP16では平均約440.5 TFLOPSとなり、GPU内部演算ワークロードが安定して実行できることを確認した。
- Host-GPU転送では、1024MiB pageableでH2D約8.54GB/s、D2H約9.65GB/sとなり、転送性能の基礎値を取得した。

主要結果

- GEMM 8192×8192 FP16：
  平均 約440.5 TFLOPS
- GEMM 4096×4096 FP16：
  平均 約400.7 TFLOPS
  ※1回目が遅く、参考値扱い
- H2D 1024MiB pageable：
  平均 約8.54 GB/s
- D2H 1024MiB pageable：
  平均 約9.65 GB/s
- H2D / D2Hともに10回測定し、ばらつきは小さい

分類	条件	runs	平均	標準偏差	解釈
GEMM	8192×8192 FP16	10	440.5 TFLOPS	13.0	GPU内部演算は安定
H2D	1024MiB pageable※1	10	8.54 GB/s	0.01	Host→GPU転送
D2H	1024MiB pageable	10	9.65 GB/s	0.01	GPU→Host転送
H2D	1024MiB pinned※2	10	8.22 GB/s	0.01	参考値
D2H	1024MiB pinned	10	9.61 GB/s	0.01	参考値

※1 pageable：OSが必要に応じて移動・退避できる通常のCPU側メモリ。

※2 pinned：移動・退避されないよう固定されたCPU側メモリで、GPUとのデータ転送に適する。

日本総研 The Japan Research Institute, Limited

性能特性：LLM推論

- CPU側CVM + H100 GPU CC構成上で、vLLMによりQwen2.5のLLM推論性能を条件別に測定した。
- Qwen2.5 1.5Bはbatch size 16で最大約4.3K output tok/s、Qwen2.5 7Bはbatch size 8で最大約1.1K output tok/sとなった。
- prompt長、出力token数、batch sizeを変えても推論は完了し、H100 CC上でLLM推論ワークロードが安定して動作することを確認した。

測定条件

構成

- CPU側CVM + H100 GPU CC

実行基盤

- vLLM Python API / bfloat16

モデル

- Qwen2.5-1.5B-Instruct
- Qwen2.5-7B-Instruct

測定条件

- **prompt**：short / mid / long
- **max output**：64 / 128 / 256 tokens
- **batch size**：1 / 4 / 8 / 16（一部7Bは8まで）

評価指標

- **output tok/s**：1秒間に生成した出力token数
- **total tok/s**：1秒間に処理した入力＋出力token数
- **req/s**：1秒間に完了したリクエスト数

※いずれもbatch全体の集約値。単一リクエストの応答時間ではない。

モデル	prompt	output	batch	output tok/s	total tok/s	req/s	解釈
Qwen2.5 1.5B	short	128	1	299	337	2.34	単発基準
Qwen2.5 1.5B	short	128	16	4,292	5,004	33.5	batchで大きく向上
Qwen2.5 1.5B	long	128	16	3,914	36,123	30.6	長文promptでも安定
Qwen2.5 7B	short	128	1	143	161	1.12	単発基準
Qwen2.5 7B	short	128	8	1,088	1,270	8.50	7Bもbatchで向上
Qwen2.5 7B	long	128	4	547	5,052	4.27	長文promptでも完了

H100 CC上でも、vLLM推論はモデルサイズ・prompt長・batch sizeを変えて安定動作し、batch化により出力token/sが大きく向上した。

日本総研
The Japan Research Institute, Limited

性能特性の横断比較

- TEEの性能影響は一律ではなく、ワークロード特性によって見え方が異なる。
- CPU演算はSGX / CVMとも概ね同等だった一方、I/O・Web小レスポンス・メモリ条件では低下が見られた。
- Composite TEE構成では、CPU側CVM + H100 CC上でCUDA / PyTorch / vLLMが動作することを確認した。

性能影響の見え方

- CPU演算
  SGX / CVMとも概ね同等
- 起動 / 初期化
  SGXはEnclave起動で増加
  CVMは差が小さい
- Memory
  SGXはEnclave設定の影響あり
  CVMは一部条件で低下
- Web / I/O
  SGX / CVMとも小レスポンス・I/Oで低下傾向
- GPU / LLM
  Composite TEE上でGPU演算・LLM推論が動作

TEE種別	CPU / Memory	I/O / Web	GPU / LLM	主な性能論点
SGX	CPUは概ね同等	I/O read・Webで低下	対象外	起動・I/O・メモリ制約
SEV-SNP	CPUは概ね同等、Memoryは一部低下	Nginx 10KBで低下	対象外	Memory条件・小レスポンス
TDX	CPU / Memoryは概ね同等	File書込、Nginx 10KBで低下	対象外	Memory条件・小レスポンス
Composite TEE	CPU側CVM上で動作	Host-GPU転送を確認	GEMM / PyTorch / vLLM動作	CPU/GPU双方の確認

各TEEの性能影響は、CPU演算よりも起動・I/O・Web小レスポンス・メモリ条件で差が出やすい。

注：Native比を含む性能比較は、Azure上で条件を可能な範囲で揃えた傾向評価である。比較条件と制約はp.21を参照。

日本総研 The Japan Research Institute, Limited

性能特性の横断比較：SGX / SEV-SNP / TDXの相対性能値

- CPU性能への影響は3方式とも小さく、各基準環境比で概ね同等であった。
- MemoryではSGXでやや低下が見られる一方、SEV-SNP / TDXの影響は比較的小さい。
- File I/OではSGXの読込・TDXの書込で低下が見られ、Nginxでも各基準環境に対する影響に差が確認された。

マイクロベンチマーク：各基準環境に対する相対性能

観点	SGX	SEV-SNP	TDX	傾向
CPU	100.2%	98.5%	100.2%	いずれも概ね同等
Memory 512MB	85.4～86.8%	92.7%	97.6%	SGXでやや低下
File I/O 1024MB 書込	98.1%	97.8%	69.0%	TDXで低下
File I/O 1024MB 読込	68.8%	98.7%	99.2%	SGXで低下

注：SGXはNative実行、CVMは各対応Native VMを基準として100%に正規化。実行環境・CPU・OS等が異なるため、絶対性能ではなく各TEE導入時の性能影響を比較する参考値として扱う。

Nginx：各基準環境に対する相対性能（SGXは参考値）

レスポンスサイズ	SGX※（参考）	SEV-SNP	TDX	傾向
10KB	17.5%	73.0%	61.8%	CVMではTDXの低下が大きい
1MB	9.2%	92.7%	104.1%	CVMでは影響は比較的小さい

注：SGXはGramine Directを100%としたGramine SGXの相対値、SEV-SNP / TDXは各Native VMを100%としたCVMの相対値。SGXはNginxのバージョン・設定も異なるため、3方式の直接比較ではなく参考値として掲載する。

日本総研
The Japan Research Institute, Limited

アテステーション方式の比較

- TEEごとにAttestationで確認する対象は異なり、SGXはEnclave、CVMはVM全体、H100 CCはGPUのデバイス・ソフトウェア状態を確認する。
- 本検証では、SGXはRA-TLS / DCAPとAzure MAA、CVMはAzure MAA Guest Attestation、H100はNVIDIA NRASによるRemote GPU Attestationを確認した。
- Composite TEE構成では、CPU側MAA JWTとGPU側NRAS EAT/JWTを同一session / nonceの証跡として対応付け、両方のPolicyがPASSした場合のみComposite PASSとした。

アテステーションで確認する対象

- SGX
  Enclave内で期待するコードが動作し、接続先がそのEnclaveに結び付いているか
- CVM
  VMが期待するTEE・起動構成で動作しているか
- Composite TEE
  CPU側CVMとGPU側H100がそれぞれ期待状態であり、両証跡が同一session / nonceに属しているか

TEE種別	証明対象	検証方式	確認できたこと
SGX	Enclave	RA-TLS / DCAP	Enclave真正性、MRENCLAVE / MRSIGNER照合
SGX	Enclave	Azure MAA（SGX / DCAP）	DCAP QuoteをMAAで検証しJWT取得
CVM	VM全体	Azure MAA Guest Attestation	SEV-SNP / TDXのVM状態をJWTで確認
Composite TEE	CPU側CVM＋H100 GPU	Azure MAA（CPU） / NVIDIA NRAS（GPU）/ Composite Broker	CPU / GPU証跡を同一session / nonceで対応付け、両方のPolicyがPASSした場合のみComposite PASS

日本総研 The Japan Research Institute, Limited

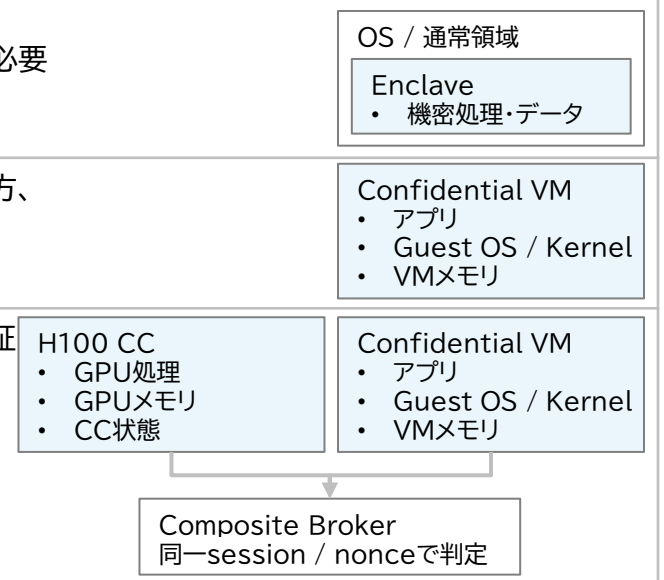
TCB・メモリ制約・信頼境界の比較

- TEEごとに保護単位と信頼境界が異なり、TCBに含まれる範囲も変わる。
- SGXは保護範囲をEnclaveに絞れる一方、EPC設計・アプリ分割・ Gramine設定が必要となる。
- CVMは既存ワークロードを載せやすい一方、ゲストOSを含むVM全体を信頼境界として扱う。
- Composite TEEでは、CPU側CVMをMAAで、GPU側H100 CCをNRASでそれぞれ検証し、両証跡を同一session / nonceで束ねて判定する。

注：CPU側Linux上で動作するGPU Attestation ClientやComposite Brokerについて、Linux IMAの測定ログとvTPMのPCRを用いた、実行ファイルの検証は未実施である。

TEE方式ごとの保護単位・信頼境界

TEE種別	保護単位	TCB / 信頼境界	メモリ制約・運用論点	保護・検証範囲
SGX	アプリの一部 / Enclave	ゲストOSやHypervisorは基本的に信頼外。Enclave内のアプリと実行基盤を信頼対象とする（Gramine利用時はLibOS等も含む）	Enclaveメモリ / EPCの設計、Manifest、署名、アプリ分割が必要	細粒度に保護できる一方、アプリ分割・Enclave設計が必要
CVM	VM全体	ゲストOS・アプリはCVM内の信頼境界に含まれる。Hypervisor / Host OSは信頼外	既存アプリを動かしやすいが、OS管理・パッチ管理が重要	既存アプリを移行しやすい一方、VM全体が信頼境界となる
Composite TEE	CPU側CVM + H100 CC	CPU側CVMとGPU側CCに加え、GPU Attestation Client、Composite Broker / Verifier、関連ライブラリ・設定、Policyも信頼・管理対象となる	CPU / GPU個別のAttestation、session / nonce、Policy、鍵、ログを一体的に管理する必要がある	CPU / GPU双方を保護・検証できる一方、Policy・鍵・証跡・ログ管理が複雑化



Composite TEEの位置付けと構成候補

- Composite TEEは、CPU側をCVMで保護し、GPU側をGPU CCで保護する構成である。
- 本検証では、Azure SEV-SNP＋H100 CC上で、CPU側MAA JWTとGPU側NRAS EAT/JWTを同一session / nonceで対応付けるBroker型Composite判定を実装した。
- 今後は、Linux IMAの測定ログとvTPMのPCRを用いたGPU Attestation Client / Composite Brokerの実行ファイル検証、CVM–GPU間のより強い証跡結合、Intel TDX＋H100の公式統合方式との比較、BlackwellおよびAMD Instinct MI300X系への拡張を検討する。

Composite TEEの位置付け

- **CVM型TEE**
  CPU側のアプリ・データ・Guest OSをVM全体で保護する。既存アプリを移行しやすい一方、GPU上の処理は別途保護が必要となる。

- **本検証のComposite TEE**
  Azure SEV-SNP CVMとH100 CCを組み合わせ、CPU側MAA JWTとGPU側NRAS EAT/JWTを同一session / nonceで対応付けて判定した。

- **今後の検討課題**
  IMA＋vTPMによるCPU側実行アプリの検証、CPU / GPU証跡のハードウェアレベルでの結合、Intel TDX＋H100の公式方式との比較、BlackwellおよびAMD Instinct MI300X系GPUへの展開を検討する。

構成候補	内容	本検証との関係
CVM単体	CPU側VM全体を保護	SEV-SNP / TDXで機能・性能・Attestationを検証
本検証のComposite TEE	Azure SEV-SNP CVM＋NVIDIA H100 CC	MAA JWTとNRAS EAT / JWTを同一session / nonceで対応付け、両Policy PASS時のみComposite PASS
公式統合方式の参考例	Intel TDX CVM＋NVIDIA H100＋Intel Trust Authority	CPU / GPUを一つのComposite Attestationフローで検証
将来候補：Blackwell	CVM＋Blackwell世代のCC対応GPU	H100で確認したGPU Attestation・性能・運用を次世代GPUへ拡張
将来候補：AMD Instinct MI300X系	CVM＋AMD Instinct GPU	AMD GPU環境でのAI処理・GPU保護構成を検討

【参考】公式統合方式と本検証環境の違い
Intel Trust Authorityでは、Intel TDX＋NVIDIA H100のComposite Attestationが提供されている。一方、今回の構成はSEV-SNP＋H100であるため直接適用できず、同等の統合方式や証跡結合の実現方法は今後の検討課題となる。

日本総研 The Japan Research Institute, Limited

各TEEの現実的な適用領域

- TEE選定では、保護対象がアプリの一部か、VM全体か、GPU / LLM処理まで含むかを最初に整理する。
- アプリの一部を小さく保護したい場合はSGX、既存ワークロードを大きく変更せず移行したい場合はCVM、AI / LLM処理まで含めて保護・確認したい場合はComposite TEEが候補となる。
- 性能値だけで優劣を決めるのではなく、改修負荷、TCB、アテステーション方式、運用・鍵管理の複雑さを合わせて判断する。

TEE選定の観点

1. 保護したい対象はどこか？
   - アプリ内の一部処理 → SGX
   - VM上の既存アプリ全体 → CVM
   - GPU / LLM処理まで含む → Composite TEE
2. 外部に実行環境の証跡を示す必要があるか？
   - あり → 対象TEEに応じたRemote Attestationと証跡の検証を運用設計に組み込む
   - なし → PoC・テスト段階として、機能・性能確認を中心に評価
3. 運用上の制約はどこにあるか？
   - 改修負荷を下げたい → CVM
   - TCBを小さくしたい → SGX
   - AI / LLMを扱いたい → Composite TEE
   - コスト・共有性が重要 → GPU利用形態を追加検討

判断軸	SGX	CVM	Composite TEE
保護単位	アプリの一部 / Enclave	VM全体	CPU側CVM + H100 Confidential GPU
既存アプリ移行	修正・manifest対応が必要	移行しやすい	コンテナ / GPU基盤整備が必要
TCB	小さくしやすい	ゲストOSを含む	CPU側VM + GPU側まで拡大
アテステーション	Enclave単位のRemote Attestation（DCAP / RA-TLS等）	VM単位のAttestation（SEV-SNP / TDXの証跡検証、Attestationサービス等）	CPU / GPUのAttestation結果を同一session / nonceで束ね、Policyを評価
性能論点	起動時間、I/O、メモリ制約	Webレスポンス、環境差	GPU転送、LLM条件、運用複雑性
運用負荷	高め	中程度	高い

注：本検証では、Azure上の実装例としてRA-TLS / DCAP、Azure MAA、NVIDIA NRASを検証した。

日本総研
The Japan Research Institute, Limited

TEE選定の判断軸

- SGXは、秘密鍵操作や署名処理など、保護対象を小さく限定できる処理に向きやすい。
- CVMは、既存のWeb / API / バッチ処理など、VM単位で既存ワークロードを移行したい用途に向きやすい。
- Composite TEEは、機密データを用いたAI分析やLLM推論など、CPU側処理とGPU側処理の双方を保護・確認したい用途に向きやすい。

用途	推奨TEE	理由	注意点
秘密鍵操作 / 署名処理	SGX	保護対象をEnclave内に限定しやすい	Enclave設計、署名、メモリ制約がある
小さな機密ロジックの隔離	SGX	TCBを小さくしやすい	アプリ分割・Gramine設定が必要
既存Web / API	CVM	既存アプリを大きく変更せず移行しやすい	ゲストOS管理、パッチ管理が重要
バッチ処理 / データ加工	CVM	VM単位で処理環境を保護できる	I/O性能とストレージ条件を要確認
機密データを用いたAI分析	Composite TEE	CPU側処理とGPU側処理の双方を確認できる	証跡・ログ・鍵管理が複雑
LLM推論 / 生成AI基盤	Composite TEE	vLLM等のGPU推論を保護構成で動かせる	batch、prompt長、同時実行数などの追加評価が必要

規制対応との接点

- EU金融分野のICTレジリエンスを求めるDORAでは、保存中・転送中に加えて、処理中のデータについても機密性・完全性等を維持することが求められる。
- 保存中・転送中は暗号化技術が広く普及している一方、処理中は通常、CPU / GPU上で平文に戻す必要があり、保護が難しい領域となる。
- 処理中データを保護する技術には**TEE**、**準同型暗号**、**MPC**等があるが、それぞれ性能、汎用性、導入方法が異なる。
- 既存アプリやAIワークロードへの適用性を踏まえると、**現時点ではTEEが実運用に適用しやすい選択肢**となる。
- ただし、TEEはCPU / GPUのハードウェアやファームウェアを信頼の起点とする技術であり、ハードウェアベンダーやAttestation基盤を含むTCBへの信頼が前提となる。

データ保護の対象

保存中	転送中	処理中
想定する脅威 • 媒体の持ち出し • ディスク漏えい	**想定する脅威** • 通信の盗聴 • 通信の改ざん	**想定する脅威** • OS / Hypervisor • クラウド管理
主な保護技術 • ディスク暗号化 • DB暗号化 • KMS / HSM • アクセス制御	**主な保護技術** • TLS / HTTPS • VPN / IPsec • 証明書管理	**主な保護技術** • TEE • 準同型暗号 • MPC

処理中データの保護技術	仕組み	強み	実運用上の主な制約
準同型暗号（HE / FHE）	データを暗号化したまま計算	実行環境から平文を隠せる	計算負荷が大きく、対応する演算やモデルに制約
MPC	データを複数者に分割し、共同で計算	特定の一組織へデータを集約しない	複数者間の通信、専用プロトコル、構成変更が必要
TEE	CPU / GPUの保護領域内で平文で処理	既存アプリやAI処理を比較的適用しやすい	ハードウェアやTCBへの信頼、Attestation・運用設計が必要

検証結果のまとめ

- Azure上でSGX、SEV-SNP、TDX、H100 Confidential GPUを構築し、動作、Attestation、性能特性を確認した。
- 保護単位、アプリ改修、性能、TCB、運用負荷が異なるため、用途に応じた選定が必要となる。

方式別の検証結果

【Intel SGX】

- Gramineによる既存LinuxアプリのEnclave実行と、RA-TLS / DCAP / MAAによる外部検証を確認した。
- 保護単位を小さくできる一方、Enclave設計、manifest設定、起動・I/O・Memory性能が論点となる。

【SEV-SNP / TDX CVM】

- 既存Linuxアプリを大きく変更せず移行でき、MAAによりCVM種別、Secure Boot、vTPM等を確認できた。
- CPU・Memory・大容量処理はNativeに近い一方、小容量レスポンスの大量処理では性能低下を確認した。

【H100 Confidential GPU / Composite TEE】

- CVM内のDocker / PyTorch / vLLMから、H100をCCモードで利用できることを確認した。
- CPU MAAとGPU NRASを同一session / nonceで束ね、Composite PASS / FAILを判定できた。

横断的に得られた示唆

【保護範囲】

- SGXはアプリの一部、CVMはVM全体、Composite TEEはCPU / GPU処理を保護する。
- 保護範囲が広がるほど、TCBと構成・運用の複雑性も増加する。

【性能】

- TEEの性能影響は一律ではなく、起動、I/O、小容量レスポンス、GPU転送などで現れ方が異なる。
- 実際のデータサイズ、同時実行数、ワークロードを用いた評価が必要となる。

【Attestation・運用】

- Attestationは取得だけでなく、署名、issuer、nonce、Policyの外部検証が必要となる。
- ログ保全、鍵払い出し、変更管理まで含めて運用設計へ組み込む必要がある。

次フェーズの検証課題

- 次フェーズでは、単体TEEの機能・性能確認から、AI基盤としてのスケール、運用統合、鍵管理まで検証範囲を拡張する。
- 特に、Blackwell世代、マルチGPU、厳密なComposite Attestation、K8s CoCo、ミドルウェア運用が未評価項目となる。

AIワークロード・スケール検証

【Blackwell世代のCC対応GPU】

- H100 CCで確認したGPU Attestation、性能、運用条件について、Blackwell世代のCC対応GPUで再評価する。
- GPU CCの世代差、Attestationフロー・claim、性能影響、対応するクラウドVMの種類を比較する。

【マルチGPU / 分散推論】

- 単一H100検証から複数GPU構成へ拡張し、CCモード、GPU間通信、Attestation、配置制約、転送性能を検証する。
- vLLM等で、tensor parallel / pipeline parallel利用時の性能と保護境界を確認する。

【厳密なComposite Attestation】

- 現行PoCでは、CPU側MAAとGPU側NRASの証跡を同一session / nonceで相関し、両方のPolicyがPASSした場合にComposite Resultを発行するところまで実施した。
- 次フェーズでは、nonceや公開鍵をCPU / GPU双方のハードウェア証跡へ直接結び付け、CVMとGPUの割当関係やCPU–GPU間の通信路まで確認する。

運用・プラットフォーム統合検証

【Kubernetes / Confidential Containers(CoCo)】

- Kubernetes上で、Pod単位のワークロードをConfidential VM内で実行する構成を検証する。
- Kata Containers、GPU Operator、Attestation、Secret Deliveryの連携を確認する。

【ミドルウェア評価】

- Key Vault / HSM、外部Verifier、Policy Engine、ログ基盤とAttestationを接続する。
- Composite PASS時のみ、データ暗号鍵・モデル復号鍵・アクセストークンを払い出す運用を検証する。

【本番運用設計】

- Attestation結果、nonce、Policy version、署名検証結果を監査ログとして保存する。
- 異常系、ローテーション、更新、障害時の再Attestation、鍵失効を含めて確認する。

日本総研

The Japan Research Institute, Limited