一般社団法人　電子情報通信学会
THE INSTITUTE OF ELECTRONICS,
INFORMATION AND COMMUNICATION ENGINEERS

信学技報
IEICE Technical Report
CPSY2023-49,DC2023-115(2024-03)

Keystone Enclaveにおける高効率でセキュアなHost-Enclave間での大規模データ授受手法

齊木　昭大†　　木村　啓二†

† 早稲田大学
E-mail: †saiki@kasahara.cs.waseda.ac.jp, ††keiji@waseda.jp

あらまし　RISC-V における Trusted Execution Environment (TEE) 実装の 1 つである Keystone Enclave では，ホストと隔離環境間でのデータ授受が柔軟性を欠いており，高効率な大容量データの授受が困難という問題がある．本稿では，Keystone においてよりセキュアで高効率なデータ授受が可能な手法を提案する．提案手法を用いて大規模なデータ授受を行ったところ，既存の実装で可能な方法と比較して，約 2.3 倍の授受速度の向上が得られた．また本手法の活用例として，Keystone 上でのブートイメージに対する Secure Boot 署名計算を実装・評価した．その結果，データ授受のアプリケーション全体への影響が 3-5% と僅かであることが確認できた．

キーワード　TEE, RISC-V, Keystone Enclave, PMP, オーバーヘッド削減

An Efficient and Secure Data Transfer Method for Large Data between Host and Enclave on Keystone Enclave

Akihiro SAIKI† and Keiji KIMURA†

† Waseda University
E-mail: †saiki@kasahara.cs.waseda.ac.jp, ††keiji@waseda.jp

1　はじめに

近年の OS は，接続デバイスや機能の多様化のために大規模化・複雑化しており，内部に脆弱性を含む可能性が非常に高く，全面的な信頼を置くことが難しい．そのため，システムの信頼性保証や保護を行う技術が注目を集めている．

このような技術の 1 つとして，Trusted Execution Environment (TEE) がある．TEE は，ハードウェア的に隔離された環境でクリティカルな処理を安全に実行する技術である．その実装は，CPU ベンダーやアーキテクチャごとに様々なものが存在し，代表的なものとして Intel SGX [1], ARM TrustZone [2], RISC-V Keystone Enclave [3] が挙げられる．

しかし，TEE は高いセキュリティを実現する反面で，アプリケーションに対して様々な制約を課す．典型的なものとして，隔離環境で使用可能なメモリサイズの制限や，ハードウェアアクセスなど特権を要する操作の制限が挙げられる．本稿では，このような制約のうち，ホスト OS と TEE の間でのデータ授受における制約に注目する．

TEE は隔離環境であり，処理に利用するデータは隔離環境へ持ち込む必要がある．そのため，多くの TEE 実装では通常の実行環境である Rich Execution Environment (REE) と TEE 間でのデータの授受を行う機能を備える．しかし，授受機能は隔離に影響を及ぼさないように設計される必要があるため，柔軟性を欠き，様々な難点を抱えている．Intel SGX では，Intel の提供する SDK [4] の問題点として，ポインタを扱うデータ構造の安全な授受やデータ授受によるオーバーヘッド等についてが議論されており，その解決策が提案されている [5]〜[7].

Keystone では，Edge Call (Ecall) という，隔離環境である Enclave からホスト OS への一方向関数呼び出しによってデータ授受が可能である．Ecall は，データ授受にホスト OS と Enclave の双方からアクセス可能な共有バッファを使用するが，大規模な授受を行うことが想定されていない．そのため，大規模な領域を共有バッファとして使用することが出来ず，非効率な方法で授受を行わざるを得ない．また，Keystone は研究段階のフレームワークであり，実用例が少なく，大規模なデータ授受を伴うワークロードで利用されている事例も見受けられない．

本稿では，データ授受専用の保護された領域である Additional Data Memory (ADM) の導入を提案する．提案手法では，データ授受のシナリオを限定することで，可能な限りの保護をデータ授受領域に対しても適用する．これにより，大規模なバッファ

をデータ授受に使用可能となり，高効率かつ従来よりセキュアな大規模データ授受が可能となる．加えて，ADM の活用例としてブートイメージに対する Secure Boot 署名計算を Keystone 上で実装し，実際のワークロードにおけるデータ授受の影響を評価した．

本稿の構成は以下のとおりである．2 節で Keystone のアーキテクチャに関連する RISC-V の仕様について，3 節で Keystone のアーキテクチャについて，それぞれ概要を述べる．4 節で提案手法である ADM について詳細を述べる．5 節で提案手法の性能及びセキュリティ評価を行い，提案手法の有効性を確認する．6 節で ADM を使用して Secure Boot 署名計算を実装し，実際のワークロードにおける有効性を確認する．

2 RISC-V の関連機能

本節では，Keystone Enclave の実装に関連する RISC-V ISA の仕様について概観する．本節の内容は，RISC-V ISA Privileged Architecture [8] の RV64 における仕様に準拠している．

2.1 特権レベル

RISC-V では, 特権レベルアーキテクチャとして Privilege Level が定義されている．基本的な構成のシステムでは，Machine, Supervisor, User の 3 種類の特権モードが主に使用される．最も高位の権限を持つモードは Machine Mode (M-Mode) で，ハードウェアへの低レベルなアクセスが可能である．Supervisor Mode (S-Mode) は M-Mode の次に高位であり，仮想メモリシステムなどが使用できる OS の動作を想定したモードである．最も低位なのは User Mode (U-Mode) で，OS 上のユーザーアプリケーションの動作を想定したモードである．

Keystone では，M-Mode において Security Monitor (SM) という制御ソフトウェアを動作させ，S,U-Mode の Monitor やメモリアクセス制御を行う．また Enclave では S, U-Mode の 2 モードが使用可能である．

2.2 Physical Memory Protection （PMP）

Physical Memory Protection (PMP) は RISC-V ISA で定義されたメモリ保護機能であり，コア単位で物理アドレス範囲へのアクセス権限を設定できる．PMP の設定は，M-Mode において PMP エントリを作成・有効化することにより行う．PMP エントリは，専用の Configuration and Status Register (CSR) の設定値から構成されるアドレス範囲とアクセス権限のペアである．また，指定可能な権限は Read, Write, Execute (RWX) の 3 種類となっている．

標準では，S-Mode, U-Mode でのページテーブルアクセスを含めた全てのメモリアクセスに対し，PMP によるアドレスマッチングが行われアクセス可否が決定される．PMP エントリを構成するレジスタは以下の 2 種類で，これらの設定値によってアドレスマッチング方法と対象範囲，権限が決定される．

- `pmpcfg`: Configuration Register (8-bit)
- `pmpaddr`: Address Register (54-bit)

Configuration Register は，PMP エントリの各種設定を保持する．このレジスタには静的に番号が振られており，若い順に優先度が高くなっている．異なるエントリで対象範囲の重複がある場合，優先度の高いエントリで指定された権限が優先される．

また，Configuration Register では Address Register の解釈方法を決定するアドレスマッチングモードを指定する．マッチングモードには，TOR と NAPOT の 2 種類が存在する．以下 i 番目の Configuration Register, Address Register をそれぞれ $\texttt{pmp}i\texttt{cfg}$, $\texttt{pmpaddr}_i$ と表記する．TOR は 2 本の Address Register を使い，直接対象のアドレス範囲を指定する方式である．$\texttt{pmp}i\texttt{cfg}$ に対応するのは $\texttt{pmpaddr}_{i-1}, \texttt{pmpaddr}_i$ であり，これらの間の領域がエントリの範囲となる．NAPOT は，アドレスマスクによって 1 本の Address Register で範囲の先頭アドレス及びサイズを指定する方式である．$\texttt{pmp}i\texttt{cfg}$ には $\texttt{pmpaddr}_i$ が対応する．このモードでは，`pmpaddr` の最下位から連続して立つビットがエントリの範囲を表し，途切れたビットより上位の部分がエントリの先頭アドレスを表す．そのため，2^nbytes のエントリには n-bit aligned の先頭アドレスが必要という制限が生じる．

Keystone では，隔離される Enclave の領域を高優先度のエントリで，OS などの非信頼領域を最低優先度のエントリで扱うよう SM が制御している．

3 Keystone Enclave

本節では，Keystone Enclave [3] のアーキテクチャ及びデータ授受における問題点について述べる．

3.1 概　　観

Keystone は，2.1 節で述べた Privilege Level を用いて図 1 のような階層構造を構成する．図中における Keystone 固有の構成要素は以下の 4 つである．

- Security Monitor (SM)
- Runtime (RT)
- Enclave アプリケーション (Eapp)
- Host アプリケーション

Security Monitor (SM) は Keystone において中枢を担うソフトウェアである．SM は自身の動作する Platform のみを信頼し，また SM 自身は Root of Trust によって検証された上で Platform から信頼される．M-Mode 未満の権限レベルは，SM の下で Trusted Domain と Untrusted Domain に二分される．これらの領域は，SM が 2.2 節の PMP を用いて相互にアクセスができないように制御をする．Trusted に分類されるのは Enclave 内の RT, Eapp のみで，ホスト OS やその他アプリケーションはすべて Untrusted となる．Trusted Domain においては，RT が SM を信頼し，Eapp が RT 及び SM を信頼するという，信頼の階層構造をとる．

ホスト OS が Linux である場合の Enclave の作成・アプリケーションの実行は，次のように行われる．

（1） Host アプリケーションの起動
（2） カーネルモジュールを経由して Linux が Enclave 領域を確保
（3） Enclave 領域に Eapp, RT をコピー
（4） SM に領域情報を引き渡し Enclave 作成を指示
（5） SM が PMP を有効化し Enclave 内のデータを検証
（6） PMP を Trusted Domain 用の制御へ切り替えた上で RT

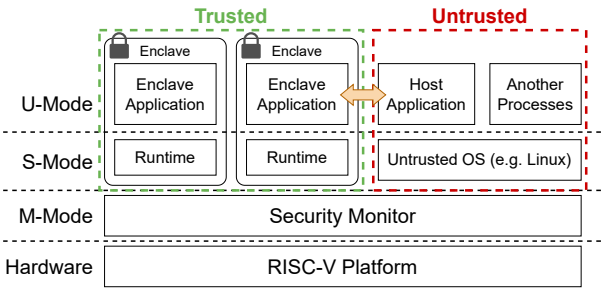

図 1　Keystone のアーキテクチャ

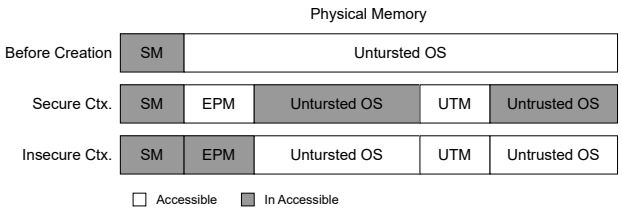

図 2　Keystone における PMP の制御

へ処理を移す

このように Enclave は Host アプリケーションが主導する形で作成・実行される．3.3 節で述べる Enclave-Host 間でのやりとりについても，この Host アプリケーションが受け口となる形で行われる．

3.2　メモリモデルと保護

Keystone では，以下の 2 種類の物理連続領域を使用して Enclave を構成する．

- Enclave Protected Memory (EPM)
- Untrusted Memory (UTM)

Enclave Protected Memory (EPM) は Enclave の実体であり，RT と Eapp のテキスト，及びこれらの実行時データが含まれる．この領域は PMP によって保護され，SM とその Enclave 自身からのみアクセスが可能となる．Untrusted Memory (UTM) は Trusted Domain と Untrusted Domain で唯一共有される領域であり，3.3 節の Edge Call においてデータ授受用領域として使用される．

これらの領域を，SM が PMP を用いて図 2 に示すように管理する．前提として，先述のとおり SM は Platform 以外を信頼しないため，SM の持つ領域は M-Mode 以外からのアクセスを常に許可しない．Enclave 作成前は SM 以外の領域をホスト OS が所有しているが，Enclave 作成時にここから EPM, UTM の領域を確保する．Enclave 作成後，Enclave へ処理が移る際はセキュアなコンテキストとなり，EPM と UTM に対して RWX の権限が与えられる．ホスト OS へ処理が移る際はセキュアでないコンテキストとなり，ホスト OS の領域と UTM にのみ RWX の権限が与えられる．

3.3　Edge Call

Keystone は，Enclave-Host 間でやりとりをする唯一の手段として Edge Call (Ecall) を実装している．Ecall は Enclave to Host の一方向関数呼び出しであり，通信用の共有バッファとして

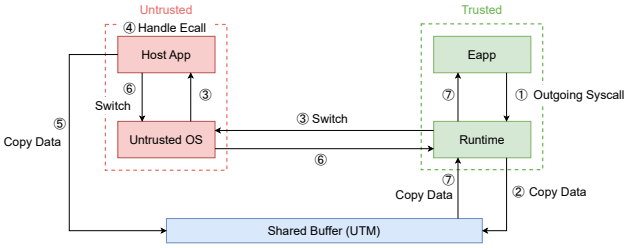

図 3　Edge Call の動作

UTM を使用する．Eapp がファイルシステムやデバイスへのアクセスを必要とする場合に，Host の機能を一部利用するといった形で使われる想定のものである．Ecall の動作する流れを図 3 に示す．

3.4　データ授受の問題

3.2 節で述べたように，Ecall の通信バッファとして用いる UTM は，常時読み書きが可能な非信頼領域である．柔軟な通信を可能とするためこのような設計となっているが，授受データの完全性や確実に授受されているかの保証ができない．

また，非信頼領域の権限が保護領域の権限を上書きすることを防ぐため，UTM を扱う PMP エントリはホスト OS と同様に最低優先度のエントリとなる．2.2 節で述べたアドレスマッチング方式について，TOR は 2 本の Address Register を用いるため，1 つ前のエントリに影響を与える可能性がある．そのため，最低優先度のエントリにおいて TOR を使用することは望ましくなく，Keystone では必ず NAPOT を使用して最低優先度のエントリを設定する．したがって，最低優先度のエントリには事実上 NAPOT における制約が適用されることとなり，2^n Bytes の領域を設定する場合先頭アドレスが n-bit aligned でなければならない．この制約は大規模な領域となるほど厳しくなり，確実に設定可能とするには，領域の確保時にアライメントを保証しなければならない．

Keystone におけるメモリ領域の確保は，ホスト OS である Linux に依存して行われており，Trusted なコンポーネントにおいてアライメントを保証することはできない．解決策として，SM にメモリをリザーブさせておく方法も考えられるが，これでは Keysotone の設計思想である Trusted Computing Base (TCB) 最小化に反してしまう．

大規模なデータ授受は，データを分割して Ecall での小規模な授受を繰り返すことでも可能である．しかし，図 3 で示すように，一回の Ecall でも多くのデータコピーやコンテキストスイッチを伴うため，分割しての授受は大きなオーバーヘッドをもたらす可能性がある．

以上より，大規模なデータをよりセキュアで高効率に授受する手法が必要である．そこで本稿では，4 節で述べる Additional Data Memory をデータ授受用のメモリとして導入する方法を提案する．

4　Additional Data Memory の導入

本節では，提案手法である Additional Data Memory (ADM) の

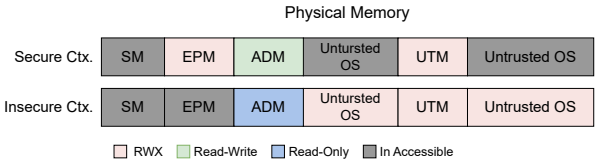

図 4　ADM を含めた PMP アクセス制御

導入について詳説する．ADM に対しては，特定のシナリオに従ったデータ授受に限定することで，授受に影響のない最大限の保護を適用する．これにより，3.4 節で述べた問題点を解消することができる．

4.1　想定するデータ授受のシナリオ

Enclave アプリケーションで大規模なデータに対する何らかの計算処理を行い，その結果を Host に返すというケースを想定する．また，処理対象のデータは Enclave アプリケーションの実行前に確定しているものとし，Enclave 内での処理に応じて要求するデータが動的に変化するケースは対象外とする．以上をふまえて，次のような条件を設定する．

（1） Enclave 作成前の 1 度のみ Host→Enclave 方向のデータ授受を行う
（2） Enclave 作成後は Host からの書き込みを禁止する
（3） Enclave→Host 方向の授受は任意に可能

4.2　アクセス制御

前節で設定した条件を基に，PMP によるアクセス制御を行う．このシナリオでは，Host からの書き込みを Enclave 作成前，すなわち PMP の有効化前に限定しているため，PMP 有効化後は Host に Read-Only 権限のみを与える．Enclave からは EPM と同様に自由にアクセス可能とするが，実行する命令列を格納することは想定しないため，実行権限 (X) は付与しないものとする．以上をふまえると，図 4 に示すようなアクセス制御となる．

本手法では，ADM を扱うための PMP エントリが 1 つ追加設定される．ADM のエントリは EPM よりも多くの権限をもつため，EPM よりも低い優先度で設定する．

4.3　データ構造

ADM で授受されるデータは，図 5 に示すデータ構造をとる．ADM の先頭には，領域の情報を持つ構造体 `adm_info` を配置する．これは領域サイズやデータ数，各データへのオフセットを持つ．また，各データについては，オフセットとデータサイズを情報としてもつ構造体 `adm_data` と共に配置する．この構造体は，データが Enclave から書き込まれたものかを識別する情報も保持している．

このような情報を付加した上で格納することで，複数データの取り扱いや書き込み時のデータ検証を可能とする．

4.4　実　　装

ADM を実装するため，Keystone の各コンポーネントに対して以下のような変更を加えた．Keystone リポジトリ [9] の Commit `a06b054` を変更のベースとして使用している．

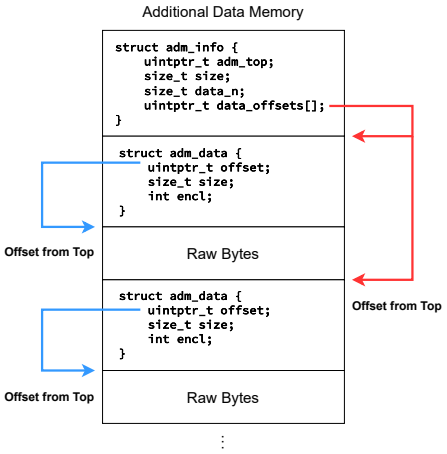

図 5　ADM 内のデータ構造

4.4.1　Security Monitor

Enclave の作成時とコンテキストスイッチ時に，4.2 節で示した PMP の制御を行うよう変更した．また，Enclave 起動前の validation プロセスに ADM に対する検証を追加した．検証内容は，データ構造のチェック，オフセットのオーバーフローチェック，及び Enclave からの書き込みを詐称していないかのチェックである．ADM 内のデータ数と各データの型についてもチェックし，これらは起動前に計算される Enclave Hash に含めた．これにより，ADM に対して想定されない入力があった場合，Attestation による検知，および秘匿データ漏洩の防止が可能である．

4.4.2　Linux カーネルモジュール

EPM, UTM と同様に，ADM 用の物理連続メモリを確保する処理を追加した．また，ホスト OS からのデータ読み書きのため，`mmap` で ADM の領域をマップ可能にした．

4.4.3　Eyrie Runtime

ブート時に ADM をカーネル空間へマップする処理を追加した．また，ユーザー空間から ADM を扱うためのインターフェースとしてシステムコールを追加した．

4.4.4　Keystone SDK

4.3 節のデータ構造に従ったデータ操作を行うライブラリを追加した．また，Host アプリケーションのライブラリへ，Enclave 作成・初期化時に ADM の確保・データ格納を行う処理，ADM から Enclave で書き込んだデータを読み取る処理を追加した．

4.5　動作フロー

ADM を用いたデータ授受の流れを図 6 に示す．まず，ADM 領域は Enclave 作成時に確保され，(1) Host アプリケーションが直接 ADM へデータを書き込む．その後，SM によって Enclave が初期化され，(2) PMP によるアクセス制御が有効となる．初期化が完了して Enclave へ処理が移ると，(3) RT のブート処理で ADM はカーネル空間へマップされる．そして RT から Eapp が実行され，(4) Eapp はシステムコールを発行して ADM をユーザー空間の仮想アドレスにマッピングし，ADM のデータを読み取る．Eapp からデータを書き込む際は，(5) システムコールを経由して RT が ADM へデータをコピーする．Eapp の書き

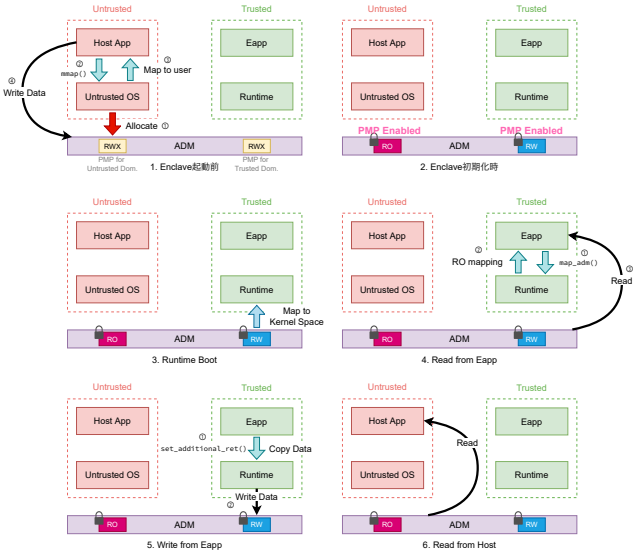

図 6　ADM を用いたデータ授受の流れ

表 1　評価環境 [11]

OS	Ubuntu Server 22.04 (Linux 5.15.0)
Board	SiFive HiFive Unamtched
CPU Core	4 x U74 Core
Privilege Level	Machine / Supervisor / User
Available PMP Entry	8
Frequency	1.2GHz
L1-I Cache	32KiB/core 4-way
L1-D Cache	32KiB/core 8-way
L2 Cache	2MiB 16-way
DRAM	DDR4 16GiB

込んだデータは，Host へ処理が戻った際，(6) Host アプリケーションが直接アクセスをして読み取ることができる．ただし，PMP によって書き込み権限は外されているため，データの改変はできない．

5　評　　価

5.1　データ授受性能の評価

オリジナルの Keystone における Ecall を用いたデータ授受と，提案手法によるデータ授受の性能を比較評価した．評価は，表 1 に示す HiFive Unmatched [10] 上で行った．

評価は，表 2 に示す 4 つのケースで行った．表中には，授受データのサイズと，その際に確保する Keystone の各種メモリ領域のサイズを示している．Ecall を用いた授受は，評価環境において確実に確保可能である 1MiB の UTM を確保し，1MiB 分割で評価を行った．評価アプリケーションは，Enclave の作成から Eapp にて授受データに自由にアクセス可能となるまでを実装している．それぞれの授受手法で，データ授受に関連する処理の合計時間，及び評価アプリケーション全体の実行時間を計測した．また，授受するデータは事前生成したランダムデータとした．

評価結果を図 7 に示す．アプリケーション全体の実行時間に

表 2　データ授受性能評価の条件

ケース	データサイズ	Ecall 使用		ADM 使用		
		EPM	UTM	EPM	UTM	ADM
small	64 B	1 MiB	1 MiB	1 MiB	1 MiB	4 KiB
large	32 MiB	36 MiB	1 MiB	1 MiB	1 MiB	36 MiB
mega	128 MiB	130 MiB	1 MiB	1 MiB	1 MiB	130 MiB
huge	512 MiB	515 MiB	1 MiB	4 MiB	1 MiB	512 MiB

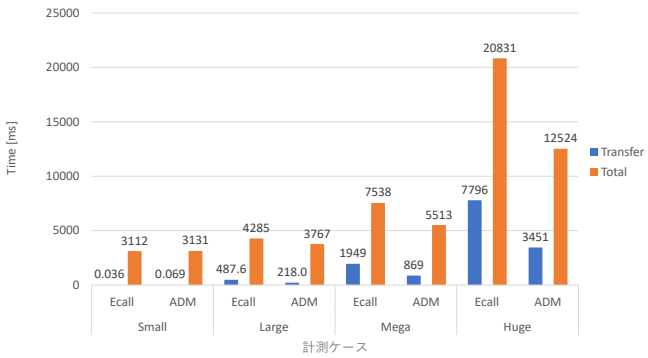

図 7　ADM によるデータ授受の性能評価

着目すると，小規模な small では Ecall を用いた場合でも，データ授受は全体の 10% にも満たない．しかし，large 以上のケースにおいてはデータ授受の割合が全体の約 10–37% と，無視できない程度まで増加している．したがって，授受サイズが大規模になるほど，授受オーバーヘッド削減の効果があることが分かる．

データ授受時間に着目すると，最小の授受サイズである small 以外のケースでは，提案手法により速度向上が得られた．small では，Ecall による授受が 0.036ms，ADM による授受が 0.069ms と，提案手法により速度が低下した．しかしその差はごく僅かで，アプリケーション全体の 3000ms 強と比較すれば無視できる程度のオーバーヘッドである．したがって，より確実なデータ授受を求める場合には，小規模なデータ授受においても提案手法が有効といえる．一方 large 以上のケースでは，いずれのケースでも提案手法により約 2.3 倍の速度向上が得られた．これは，提案手法によって，Ecall を用いた手法に存在する多くのコンテキストスイッチやデータコピーが削減されているためである．

5.2　セキュリティ評価

本研究の脅威モデルは，オリジナルの Keystone と同様である [3]．本節では，ADM と関連する実装において新たに考えられる脅威について考察する．

5.2.1　メモリ保護

ADM の追加により，Untrusted Domain から読み取り可能な領域が増加する．ADM の PMP エントリに対し，本来 Untrusted Domain からアクセス不可能な EPM をオーバーラップするよう設定することで，機密へのアクセスを試みる攻撃が考えられる．しかし，ADM の PMP エントリについては，SM において指定範囲がチェックされる．また，必ず EPM のエントリより優先度が低くなるよう設定されるため，EPM に対する保護をオー

表 3　署名計算アプリケーションの評価条件

オリジナルのハッシュ	SHA-256
ターゲットのハッシュ・署名	SHA3-384 / ED25519
暗号ライブラリ	wolfSSL v5.5.6-stable
計算対象	Linux initramfs / kernel
ブロックサイズ	80KiB

表 4　署名計算アプリケーションの評価結果

対象	初期化 [s]	検証・署名 [s]	データ授受 [s]	全体 [s]
initramfs	4.956	8.133	0.669	15.555
kernel	4.179	2.365	0.197	7.472

バーライドすることはできない.

5.2.2　不正なオフセットの指定

4.3 節で述べたように，ADM 内のデータは構造体によりラップされ，先頭からのオフセットで管理される．Enclave 起動前に Untrusted Domain からデータを書き込む際，不正なオフセットや他パラメータを設定することが可能である．不正なオフセットの指定により，意図しない動作を引き起こすことを目論む攻撃が考えられるが，全てのオフセットは Enclave 起動前に検証され，不正なものが存在すれば実行がキャンセルされる．

6　Secure Boot への活用

ADM の活用例として，並列化 Secure Boot [12] のブートイメージに対する署名計算を Keystone を用いて実装し，データ授受が及ぼす影響を評価した．

6.1　実　　装

署名計算アプリケーションは，通常の手法で計算されたハッシュ値とともにイメージが配布され，更新を行うために Secure Boot 署名を付与するシナリオを想定し，実装する．Enclave 内でのプログラムの進行は次のようになる．

（1） ADM よりターゲットイメージとオリジナルのハッシュ値を読み込み，検証
（2） ブートイメージのハッシュ値を算出
（3） ハッシュ値から署名を導出し，値を ADM へ書き込み

6.2　評　　価

前節のシナリオにしたがったプログラムを，ADM を使用した Encalve アプリケーションとして実装し，Enclave の初期化やデータ授受，検証・署名にかかる時間を計測した．評価環境は表 1 と同様である．検証・署名に関連する評価条件を表 3 に示す．計算対象のイメージには，Linux のブートに必要な initramfs, kernel のイメージを選択した．それぞれ 97MiB, 29MiB のファイルサイズを持つイメージであり，ADM による大規模データ授受の対象に適したものである．

評価結果を表 4 に示す．いずれのターゲットにおいても，データ授受で消費した時間は全体の 3–5% 程度に留まっている．したがって，ADM によるデータ授受が全体に与える影響は非常に小さいことが分かる．しかし，メモリ確保などの Enclave 初期化処理は依然としてアプリケーションの大部分を占める．

7　ま　と　め

本稿では，Keystone Enclave におけるセキュアで高効率な大規模データ授受手法を提案した．新たにデータ授受専用領域として Additional Data Memory を導入し，オリジナルの Keystone における，大規模データ授受に要する十分な領域を安定して確保できない問題を解決した．データ授受のシナリオを限定することで，大規模データ授受で問題となる制約の回避を，また授受するデータについても従来より高い信頼度を保つことを実現している．実際に大規模なデータ授受を提案手法で行ったところ，既存の実装を用いる手法より約 2.3 倍の速度向上を得た．

また，提案手法の活用例として，並列化 Seucre Boot のための署名計算アプリケーションを Encalve アプリケーションとして実装・評価した．ADM を用いることで，アプリケーション全体の 3–5% 程度という非常に少ないオーバーヘッドでのデータ授受が実現されたことを確認した．

謝辞　本研究の成果の一部は JSPS 科研費 JP23K11040 の助成を受けたものです．

文　　献

[1] V. Costan and S. Devadas, “Intel sgx explained,” Cryptology ePrint Archive, Paper 2016/086, 2016. https://eprint.iacr.org/2016/086
[2] B. Ngabonziza, D. Martin, A. Bailey, H. Cho, and S. Martin, “Trustzone explained: Architectural features and use cases,” 2016 IEEE 2nd International Conference on Collaboration and Internet Computing (CIC), pp.445–451, 2016.
[3] D. Lee, D. Kohlbrenner, S. Shinde, K. Asanović, and D. Song, “Keystone: an open framework for architecting trusted execution environments,” Proceedings of the Fifteenth European Conference on Computer Systems, pp.1–16, EuroSys ’20, Association for Computing Machinery, New York, NY, USA, 2020. https://doi.org/10.1145/3342195.3387532
[4] Intel, “Intel(r) software guard extensions for linux* os,” `https://github.com/intel/linux-sgx`.
[5] O. Weisse, V. Bertacco, and T. Austin, “Regaining lost cycles with hotcalls: A fast interface for sgx secure enclaves,” ACM SIGARCH Computer Architecture News, vol.45, p.10, 2017. https://doi.org/10.1145/3079856.3080208
[6] F. Dreissig, J. Röckl, and T. Müller, “Compiler-aided development of trusted enclaves with rust,” ACM International Conference Proceeding Series, p.10, Aug. 2022. https://dl.acm.org/doi/10.1145/3538969.3538972
[7] A. Ghosn, J.R. Larus, E. Bugnion, and A.R.G.J.L. EPFL, “Secured routines: Language-based construction of trusted execution environments,” 2019 USENIX Annual Technical Conference (USENIX ATC 19), pp.571–586, 2019. https://www.usenix.org/conference/atc19/presentation/dice
[8] A. Waterman, K. Asanović, and J. Hauser, “The risc-v instruction set manual volume ii: Privileged architecture,” 2021. https://github.com/riscv/riscv-isa-manual/releases/download/Priv-v1.12/riscv-privileged-20211203.pdf
[9] “Keystone: An Open-Source Secure Enclave Framework for RISC-V Processors”. https://github.com/keystone-enclave/keystone
[10] SiFive inc., “SiFive HiFive Unmatched,” 2023. https://www.sifive.com/boards/hifive-unmatched
[11] SiFive inc., “SiFive FU740-C000 Manual v1p6,” 2022. https://www.sifive.com/document-file/freedom-u740-c000-manual
[12] A. Saiki, Y. Omori, and K. Kimura, “Parallel verification in risc-v secure boot,” 2023 IEEE 16th International Symposium on Embedded Multicore/Many-core Systems-on-Chip (MCSoC), pp.568–575, IEEE Computer Society, Los Alamitos, CA, USA, dec 2023. https://doi.ieeecomputersociety.org/10.1109/MCSoC60832.2023.00089