Vol.2021-ARC-244 No.2
Vol.2021-SLDM-194 No.2
Vol.2021-EMB-56 No.2
2021/3/25

整合性ツリーおよび暗号化機構を持つ不揮発性メインメモリエミュレータの実装

林 知輝[1,a)]　大森 侑[1]　木村 啓二[1]

概要：IoT デバイス等に近接して高速かつ低遅延な処理ができるエッジコンピューティングに注目が集まっている．エッジコンピューティングで機密性の高い計算を行う場合，盗聴・改竄から保護しなければならない．その方法として信頼実行環境（Trusted Execution Environment: TEE）が注目されている．既存の TEE では補助記憶の利用に制限があるが，TEE から不揮発性メインメモリ（Non-Volatile Main Memory: NVMM）を利用することで，複雑なデバイスドライバを経由することなく簡便にデータを保存可能となる．さらに，NVMM に暗号化や情報の整合性保証を行う機構を設けることにより NVMM 上のデータが保護できる．しかしながら，このようなシステムを評価できる環境が現在不足している．そこで本稿では，整合性ツリーと暗号で保護された NVMM のエミュレータを提案する．CPU と FPGA を搭載したボードである Zynq-7000 SoC ZC706 上に本エミュレータを実装し，FPGA 上に実装されたメモリ保護機構は 50[MHz] で動作することを確認した．

1. はじめに

クラウドコンピューティングでは通信速度や遅延の問題が生じる可能性があるため，高速かつ低遅延な処理が可能なエッジコンピューティングが注目を集めている．しかし，エッジコンピューティングではエッジデバイスで Deep Neural Network（DNN）[1] などの機密性の高い計算を行うことが求められる場合があり，データを盗聴や改竄から保護することで機密情報の流出や攻撃者による意図的な誤動作を防がなければならない．その方法として信頼実行環境（Trusted Execution environment: TEE）による実行コードの保護が注目されている．しかし，Intel SGX [2] や ARM TrustZone [3] など既存の TEE では補助記憶の利用に制限がある．そこで，補助記憶としての性質も併せ持つ不揮発性メインメモリ（Non-Volatile Main Memory: NVMM）を TEE から利用して簡便かつ安全にデータを保存することで，この課題を解決できると考えられる．また，NVMM は電源遮断後もデータが残留するので，TEE による実行コードの保護に加えて残留データの保護も必要となる．これは，暗号化やデータの整合性保証を行う機構を用いることで実現できる．しかし，現在メモリ保護と NVMM の利用を両立したシステムの評価環境が不足している．

本稿では，整合性ツリーと暗号で保護された NVMM のエミュレータを提案する．本エミュレータでは，整合性ツリーと暗号化の計算を行うメモリ保護機構を既存の NVMM エミュレータ [4], [5] に組み込むことで NVMM 保護機構を持つエミュレータを実現した．CPU と FPGA を搭載した評価ボードである Zynq-7000 SoC ZC706 [6] 上に本エミュレータを実装し，メモリ保護機構が 50[MHz] で動作することを確認した．本エミュレータ上でマイクロベンチマークを用いてメモリアクセスレイテンシを計測したところ，メモリ保護機構のオーバーヘッドが大きいことを確認できた．また，リード・ライトレイテンシの内訳を調査したところ，整合性ツリーの計算が最大のオーバーヘッドとなっていることが確認できた．

本稿は以下のように構成される．第 2 節ではメモリ保護や NVMM 評価環境の関連研究を紹介する．第 3 節では本エミュレータにおけるメモリ保護機構について説明する．第 4 節では本エミュレータの FPGA 評価ボードへの実装について説明する．第 5 節では本エミュレータ上での評価について述べる．最後に第 6 節でまとめを述べる．

2. 関連研究

信頼できない外部モジュールである DRAM メインメモリに対して，信頼できるチップ上のデータ・機構を用いた暗号化およびツリー構造によるメモリ保護を行う研究が行われてきた．

[1] 早稲田大学基幹理工学研究科
School of Fundamental Science and Engeneering, Waseda Univ
[a)] hayashi@kasahara.cs.waseda.ac.jp

AEGIS [7] はソフトウェア・ハードウェア双方の攻撃に対してセキュアなプロセッサアーキテクチャである．AEGIS は AES-128 [8] を用いた暗号化と Merkle Tree [9] による整合性保証の機構を有する．暗号化および復号はハードウェア実装されたモジュールによって行われ，Merkle Tree の計算はソフトウェアによって行われる．ただし，ソフトウェア実装ではハードウェアと比較して計算時間が大きくなるという課題がある．AEGIS には OS を信頼できる場合とそうでない場合の二つの実装が存在し，信頼できる場合は物理メモリ空間に対する整合性保証を，信頼できない場合は仮想メモリ空間に対する整合性保証を提供する．

Bonsai Merkle Tree（BMT）[10] は暗号化と Merkle Tree によるメモリ保護を提供するが，より低コストでメモリの整合性を保証できるものである．従来の Merkle Tree でリプレイ攻撃への耐性をもつ整合性保証を提供するには，DRAM 中のデータとそのページに対応したカウンタの双方を木で保護しなければならなかったため，サイズおよび計算時間の問題があった．しかし，BMT では木による保護をカウンタのみにした上で整合性保証の提供を実現した．また，ディスクにスワップされたページの保護にも対応している．BMT では暗号アルゴリズムは AES-128-CTR，MAC アルゴリズムは HMAC-SHA1 [11] を使用している．

TEE からのメモリアクセスを保護する例として，Intel SGX における Memory Encryption Engine（MEE）[12] がある．ここでは整合性ツリーと暗号化によるメモリ保護がメモリコントローラのハードウェア拡張として実装されている．Merkle Tree ではなく，MAC 値の計算に親も用いるような木を使用することでセキュリティを強化している．暗号化を AES-128-CTR，MAC を Carter-Wegman MAC [13] で計算することで軽量かつ安全なメモリ保護を実現している．

同じく TEE からのメモリアクセス保護としては，RISC-V における Keystone の一部として実装されたメモリ保護 [14] が存在する．ここではソフトウェアによる Merkle Tree と暗号化計算が行われる．暗号化に AES-256-CTR，Merkle Tree のハッシュアルゴリズムに SHA-256 [15] を用いることでセキュリティを強化している．ただし AEGIS と同様に，ソフトウェア実装による大きな計算コストの課題が存在する．

また，BMT によるメモリ保護計算の高速化手法 [16] が研究されている．この研究では，ノード更新の計算順序に基づくセキュリティモデルに応じた高速化が提案されている．順序の変更が許されない場合にはパイプライン実行を，許される場合には OoO（Out of Order）実行や複数のノード更新の結合を行うことで高速化を図っている．

NVMM の評価を現実的な時間で行うための環境として，NVMM エミュレータを実装する研究が行われてきた．NVMM の評価環境として，Zynq-7000 SoC ZC706 [6] 上に実装された，組み込み向けの NVMM エミュレータ [4], [5] が存在する．このエミュレータは遅延挿入機構を持ち，DRAM メインメモリへのアクセス時に遅延を挿入することで NVMM へのアクセスをエミュレートしている．また，DRAM メインメモリと NVMM の両方を持つヘテロジニアスメモリシステムをエミュレートできる．

本稿では TEE からの NVMM 利用が想定されているので，TEE からのメモリ保護として実用化されている Intel SGX における MEE でのメモリ保護をベースとして使用する．このメモリ保護機構と NVMM エミュレータを統合することで現在不足している NVMM へのメモリ保護を行うことができる評価環境を実装する．

3. メモリ保護

不揮発性のオフチップモジュールである NVMM は外部からの攻撃から保護しなければ危険である．具体的には NVMM に対する改竄は攻撃者による意図的な誤動作を引き起こし，盗聴は機密情報の流出を引き起こしてしまう．そこで，上記の問題からシステムを保護するための機構として，

- 整合性ツリーを用いた整合性保証による改竄からの保護機構
- 暗号化による盗聴からの保護機構

を CPU キャッシュ-NVMM 間のトランザクションに対して提供する必要がある．本稿では整合性ツリーの構造とアルゴリズムおよび暗号アルゴリズムについて，Intel SGX におけるメモリ保護 [12] と同様のものを使用した．本機構は一つの整合性ツリーの要素計算を行うメモリ保護演算器と，この演算器を制御してツリー全体の処理を行い，かつ複数演算器を持つことで広いメモリ領域を保護するメモリ保護機構から構成されている．以下，本保護機構の詳細を説明する．

3.1 整合性ツリー

メモリ保護において信頼できるのは CPU チップのみである．そこでチップ上のデータと整合性検証機構を用いて，メモリへのリードおよびライトの整合性を保証しなければならない．しかし，チップ上の RAM は容量が小さい．そこで，ルートのみをチップ上に格納した整合性ツリーの利用が有効である [9].

3.1.1 整合性ツリーの構造

本稿における整合性ツリーを**図 1** に示す [12]．本ツリーでは各ノードが 8 個の子を持ち，レベルが 5 の構造を持つ．整合性ツリーでは Message Authentication Code（MAC）値を用いた整合性保証が提供される．この木が使用する領域は

- データを格納する D
- データの MAC タグを格納する T

Vol.2021-ARC-244 No.2
Vol.2021-SLDM-194 No.2
Vol.2021-EMB-56 No.2
2021/3/25

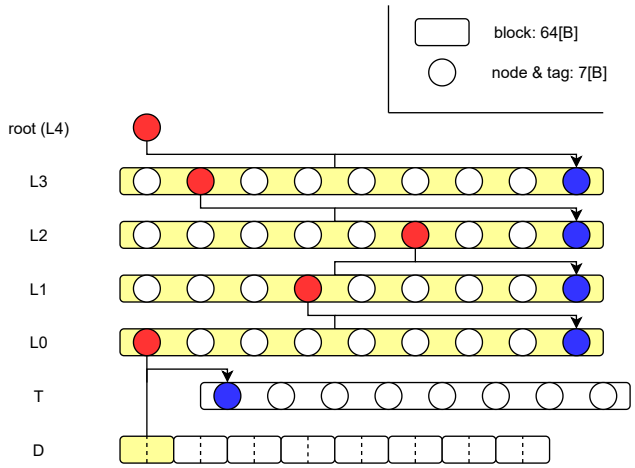

図 1 Intel SGX をベースとした整合性ツリーの構造 [12]
Fig. 1 Structure of integrity tree based on Intel SGX [12]

- 木の各ノードやその MAC タグを格納する L0 から L3
- 木のルートを格納する L4

に分けられる．L4 のデータのみチップ上の RAM に保存され，それ以外は NVMM 上に保存される．MAC タグとは，対象のデータやノードから算出される MAC 値である．この木では複数のノードやデータをまとめて 64[B] のブロックとして扱う．ブロックサイズは 64[B]，ノードサイズおよび MAC タグサイズは 7[B] となっている．ブロックは

- L0 から L3 に属する「ノード群ブロック」
- T に属する「タグブロック」
- D に属する「データブロック」

に分けられる．ノード群ブロックには 8 個のノードおよび，それらと親ノードから算出された MAC タグが格納される．タグブロックには対応するデータブロックと L0 のノードから算出された MAC タグが格納される．データブロックには暗号化された 32[B] のデータが 2 個格納される．これは本稿で対象とする CPU キャッシュラインサイズが 32[B] のためである．L1 から L4 の各ノードは子ノード群ブロックの 7[B] のカウンタを持っており，子の一つが更新されるごとにインクリメントされる．また，L0 のノードはそれぞれ対応するデータブロックの 7[B] のカウンタを持っている．

L4 のサイズは 7[B]，5 レベルのツリーで保護できる D の合計サイズは 256[KB] であるため，この木はオンチップデータ 7[B] で 256[KB] のデータに対して整合性を保証できる．

3.1.2 整合性ツリーの動作

リード要求が発行された際，要求されたアドレスのデータが前回のライト以後に改竄されていないか整合性検証が行われる．まず，データブロックと対応する L0 のノードから MAC 値を算出し，対応するタグブロック中の MAC タグとの一致比較を行う．続いて L0 から L3 のレベル順に，ノード群ブロック中の 8 個のノードと親ノードから MAC 値を算出し，ノード群ブロック中の MAC タグと一致比較を行う．全て一致した場合のみ検証成功となり，対象のデータが復号されリードされる．

ライト要求が発行されると，リード時と同様の事前検証の後に木が更新される．事前検証に成功した後に書き込みデータが暗号化される．その後，暗号化されたデータや対応のノードを用いてリード時と同様の順番で MAC 計算が行われ，MAC タグとして対応の領域に格納される．

3.1.3 MAC アルゴリズム

本稿では MAC アルゴリズムとして Carter-Wegman MAC [13] を使用した．アルゴリズムの概要を**図 2** に示す．Carter-Wegman MAC は以下の式で表される．

$$T = h_{k_1}(M) \oplus f_{k_2}(N) \tag{1}$$

T は MAC 値，M はメッセージ，N はノンスである．本稿では h_{k_1} は以下の式で表される関数を使用する．

$$h_{k_1}(x) = truncate(\sum_{i=0}^{7}(x[64*i+63:64*i] \otimes k_1[64*i+63:64*i])) \tag{2}$$

式 (2) における $\otimes$ は $\mathbb{F}_{2^{64}}$ 上の乗算であり，既約多項式は $x^{64}+x^4+x^3+x+1$ である．また，f_{k_2} は以下の式で表される関数を用いる．

$$f_{k_2}(x) = truncate(AES(k_2, x)) \tag{3}$$

式 (2) および式 (3) において，k_1 および k_2 はどちらも鍵でありチップ上に格納されている．また，$truncate$ は上位を切り捨てて下位 56[bit] のみを使用することを表す．

整合性ツリーにおける入力メッセージはデータブロックまたはノード群ブロックである．ただし，ノード群ブロックの MAC 値を求める際には 8 個のノードのみを用いるので，MAC タグが格納されているビットを 0 に置換することで MAC タグを無視して Carter-Wegman MAC への入力とする．また，ノンスは入力として使用するブロックのアドレスと対応するカウンタを合成した値を用いる．これによって空間および時間双方の面で一意なノンスとなる．

3.2 暗号化

NVMM 中のデータは，整合性保証と同様にチップ上の暗号化機構によって暗号化されている必要がある．リード時は検証後に読み出しデータが復号され，ライト時は事前検証後に書き込みデータが暗号化される．

本稿では暗号アルゴリズムとして AES-128-CTR [8] を使用した．アルゴリズムの概要を**図 3** に示す．AES-128-CTR ではノンスに対して AES 計算を行い，その結果と入力された平文の XOR をとることで暗号文を生成する．本稿における暗号化対象は 512[bit] (=64[B]) のデータブロックであるから，128[bit] ごとに本アルゴリズムを適用する

Vol.2021-ARC-244 No.2
Vol.2021-SLDM-194 No.2
Vol.2021-EMB-56 No.2
2021/3/25

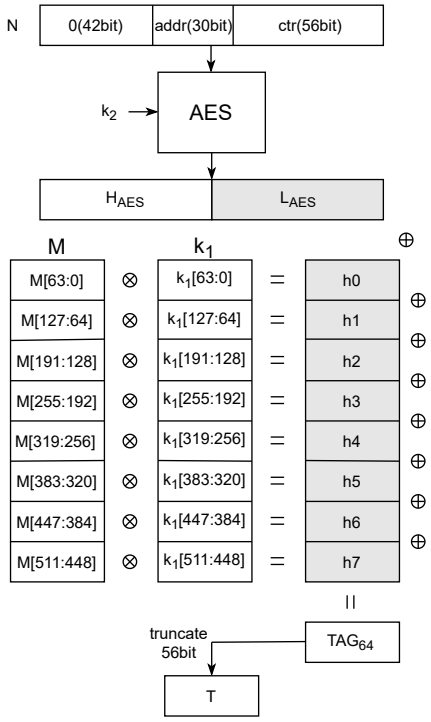

図 2 Carter-Wegman MAC のアルゴリズム [13]
Fig. 2 Algorithm of Carter-Wegman Mac [13]

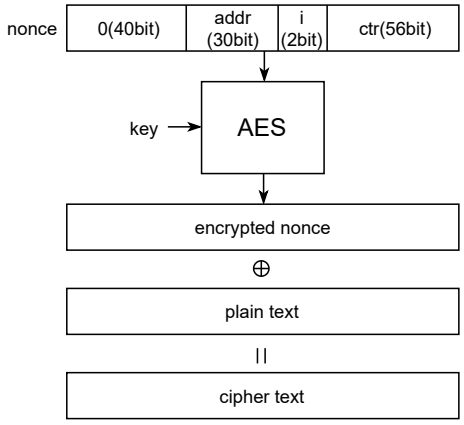

図 3 AES-128-CTR のアルゴリズム [8]
Fig. 3 Algorithm of AES-128-CTR [8]

ことで暗号化を行う．また，Carter-Wegman MAC と同様にアドレスとカウンタを合成した値をノンスとして用いる．ノンスに $i = 2'b00, 01, 10, 11$ を持たせることで，128[bit] ごとの暗号化に対してノンスの重複を防いでいる．CTR モードの暗号では暗号化と復号が同じ動作をするので，一つの回路で実現できる．また，Carter-Wegman MAC における式 (3) の計算と回路を共有することができる．

3.3 メモリ保護演算器

整合性ツリーの処理と暗号化の処理を行うメモリ保護演算器の概要を**図 4** に示す．この演算器は

- 要求から対応するブロックのアドレスを算出するアドレス演算器

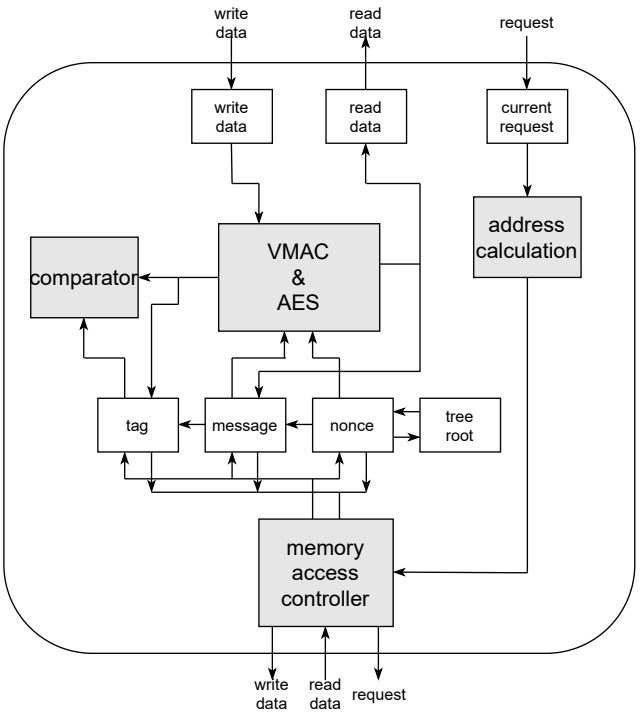

図 4 メモリ保護演算器
Fig. 4 Memory protection calculator

- NVMM に対するブロックのリードおよびライトを制御するコントローラ
- Carter-Wegman MAC および AES-128-CTR の演算器
- 算出された MAC 値と格納された MAC タグを比較する比較器

で構成される．リードあるいはライト要求が入力されると，整合性ツリーおよび暗号化計算で必要となるブロックのアドレスがアドレス演算器によって算出される．また，整合性ツリーの構造に従って NVMM へのブロックのリードおよびライト要求が発行され，各レベル分の Carter-Wegman MAC および AES-128-CTR の計算，MAC 値と MAC タグの比較が順に行われる．

3.4 メモリ保護機構

メモリ保護演算器を複数用いることで広範なメモリ領域に対して暗号化および整合性保証が提供できる．本稿ではメモリ保護計算器を 8 個用いて 2[MB] の NVMM 領域を保護する．

3.4.1 メモリ保護機構の構造

本稿におけるメモリ保護機構を**図 5** に示す．本機構は

- CPU キャッシュからのリクエストおよび書き込みデータ，CPU キャッシュへの書き込み応答および読み出しデータを格納するキュー
- 8 個のメモリ保護演算器
- メモリ保護回路から NVMM へのリクエストおよび書き込みデータを格納するキュー
- メモリアクセスレイテンシを抑えるためのブロック

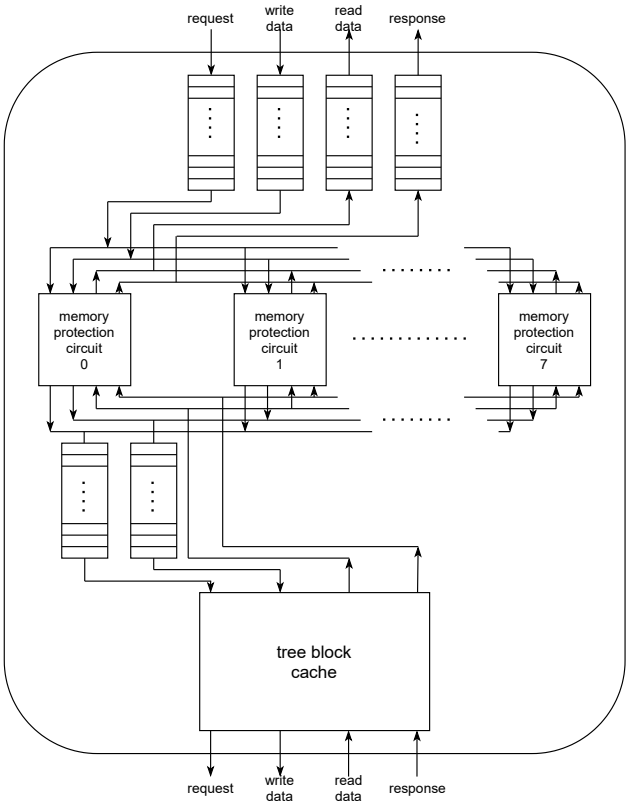

図 5 メモリ保護機構
Fig. 5 Memory protection system

キャッシュ
から構成される．

CPU キャッシュから渡されたリクエストおよび書き込みデータはキューに格納される．各リクエストおよび書き込みデータは，対応するアドレスのデータを処理するメモリ保護演算器に渡される．メモリ保護演算器によって NVMM に対してブロックのリードおよびライトリクエストが発行される．このリクエストおよび書き込みデータはキューに格納される．

また，本機構ではメモリアクセスレイテンシを抑えるためのブロックキャッシュが存在する．ブロックキャッシュは整合性ツリーの各レベルの処理に必要な NVMM 中のブロックをキャッシュするもので，8-way セットアソシアティブのライトスルーキャッシュでありサイズは 32[KB] である．

複数のメモリ保護演算器を同時に使用することにより，メモリアクセス並列性を利用することもできるが，本稿の実装では簡単のため，一度に一つのメモリ保護計算器のみを動作させるシーケンシャルな実装とした．

3.4.2 メモリ保護性能とトレードオフ

メモリ保護機構における様々なパラメータはトレードオフの関係にあり，実装するためには最適なパラメータを選択しなければならない．保護するデータ領域サイズを大きくする方法として，整合性ツリーのレベルを増やすことでメモリ保護演算器一つあたりの保護できるデータ領域サイズを大きくすることが考えられる．これによりルートの数を減らすことができ，保護するデータ領域サイズに対するチップ上の RAM 消費量を抑えることができる．しかし，レベルが増えると検証・更新コストが増加し，メモリアクセスレイテンシが大きくなる．レベルを増やさずに保護するデータ領域サイズを大きくするには木の本数を増やす必要がある．しかしメモリ保護演算器の個数が増えて回路規模が大きくなる，ルートの合計サイズが増加してチップ上の RAM をより多く消費する，といった問題が生じる．例として，2[MB] のデータ領域を保護するために 5 レベルと 6 レベルの整合性ツリーを利用することを考える．5 レベルの場合 8 個，6 レベルの場合 1 個のメモリ保護演算器がそれぞれ必要となる．そのためルートの合計サイズはそれぞれ 56[B]，7[B] となり，6 レベルの場合のチップ上の RAM 消費量は 5 レベルの場合の 0.125 倍となる．整合性ツリーの検証および更新に要する時間は 6 レベルの場合，5 レベルの場合の約 1.2 倍となる．

4. FPGA 評価ボードへの実装

第 3 節で示したメモリ保護機構を NVMM エミュレータと統合し，CPU と FPGA を搭載したボードである Zynq-7000 SoC ZC706 上に実装した．メモリ保護機構を持った NVMM エミュレータの概要を**図 6** に示す．また，本エミュレータの詳細について**表 1** に示す．ZC706 は Processing System（PS）と Programmable Logic（PL）から構成されており，PS は CPU と周辺機構，PL は FPGA を持つ．PS と PL それぞれに DIMM スロットがあり DRAM メモリが搭載されている．本稿では PL 側の DRAM の一部を NVMM としてエミュレートし，メモリ保護機構で保護する．本エミュレータにおけるアドレス割り当てを**表 2** に示す．本エミュレータでは 2[MB] の NVMM を保護し，メモリ保護機構のメタデータ領域を含めて 4[MB] の PL 側の DRAM を確保している．また本エミュレータの実装に伴い，AES コアとして東北大学の青木研究室が公開している IP [17] を利用した．本実装で使用したボードリソースを**表 3** に示す．ソフトウェアスタックとしては，Xilinx が提供する Linux カーネル [18] と NVMM エミュレータ向けに提供されるソフトウェア群 [19] を利用した．

Vol.2021-ARC-244 No.2
Vol.2021-SLDM-194 No.2
Vol.2021-EMB-56 No.2
2021/3/25

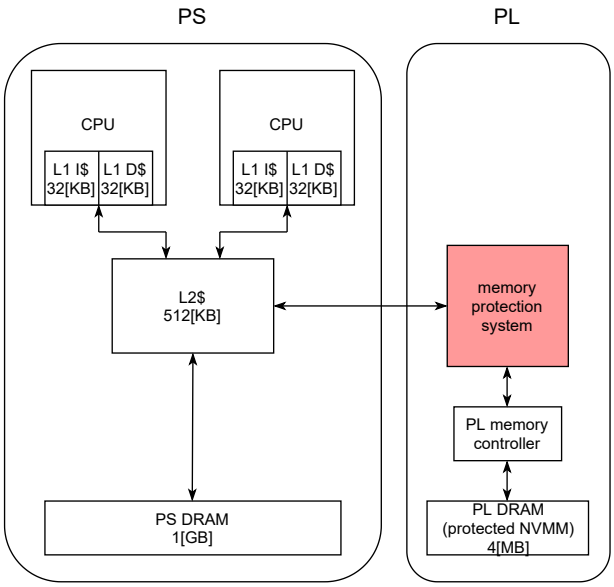

図 6 メモリ保護機構を持つ NVMM エミュレータ
Fig. 6 NVMM emulator with memory protection system

表 1 エミュレータ詳細
Table 1 Details of the emulator

ボード	Zynq-7000 SoC ZC706
CPU	ARM Cortex-A9 Dual Core 667 [MHz]
L1 キャッシュ	Instruction: 32[KB] Data: 32[KB]
L2 キャッシュ	512[KB]
PS DRAM	DDR3-1066, 1[GB]
PL DRAM (NVMM)	DDR3-1600, 4[MB]
メモリ保護機構 動作周波数	50[MHz]
PL メモリコントローラ 動作周波数	200[MHz]
OS	GNU/Linux 4.14.0-xilinx-00081-g88cc987 Ubuntu 16.04 LTS

表 2 エミュレータのアドレス割り当て
Table 2 Address assignment of the emulator

PS DRAM		0x00000000 - 0x3FFFFFFF
PL DRAM (NVMM)	D	0x80000000 - 0x801FFFFF
	L0	0x80200000 - 0x8023FFFF
	T	0x80240000 - 0x8027FFFF
	L1	0x80280000 - 0x80287FFF
	L2	0x80288000 - 0x80288FFF
	L3	0x80289000 - 0x802891FF
	unused	0x80289200 - 0x803FFFFF

5. 評価

本エミュレータの NVMM 領域に対するリードおよびライトのレイテンシを，マイクロベンチマークを用いて計測したのでその結果を報告する．

表 3 ボードリソース使用率
Table 3 Utilization of board resources

リソース	使用量	使用可能量	使用率 %
LUT	176,139	218,600	80.58
LUTRAM	3,523	70,400	5.00
FF	139,991	437,200	32.02
BRAM	132.50	545	24.31
DSP	32	900	3.56
IO	127	362	35.08
BUFG	9	32	28.13
MMCM	2	8	25.00
PLL	1	8	12.50

5.1 マイクロベンチマーク

メモリ保護の目的はオフチップモジュールである NVMM を盗聴および改竄から保護することであり，メモリ保護機構は CPU キャッシュ-NVMM 間のトランザクションに対してメモリ保護計算をするものである．そこでキャッシュミスした際に行われるメモリ保護計算のオーバーヘッドを調査するために，マイクロベンチマークを用いてメモリアクセスレイテンシを計測した．使用したマイクロベンチマークを**図 7** に示す．アクセスサイズを 8[KB] から 2[MB] で 2 倍ずつ変化させ，ストライド 1[KB] でのストライドアクセスを行った．また，各メモリアクセスは_DMB() 関数によって順番が保証される．また，レイテンシは 1 回のメモリアクセスに対する値を計測する．

5.2 レイテンシ計測結果

第 5.1 節に示したマイクロベンチマークを用いて計測したメモリアクセスレイテンシを，メモリ保護機構を持つ本エミュレータとメモリ保護機構を持たないエミュレータ間で比較した．本稿では純粋なメモリ保護機構のオーバーヘッドのみを調査するために遅延を挿入せずに計測を行った．リードレイテンシ計測結果を**図 8** に，ライトレイテンシを**図 9** に示す．

512[KB] 以上のメモリアクセスでリードレイテンシおよびライトレイテンシともに大幅に増加している．メモリ保護ありの場合のリードレイテンシはメモリ保護なしの場合（456.74[ns]）と比較して最大で約 22.3 倍（10,188.04[ns]），メモリ保護ありの場合のライトレイテンシはメモリ保護なしの場合（475.39[ns]）と比較して最大で約 44.1 倍（20,964.28[ns]）となっている．これは L2 キャッシュサイズが 512[KB] であり，これを超えると NVMM へのアクセスが頻繁に発生し，それに伴いメモリ保護計算が行われるためである．

Vol.2021-ARC-244 No.2
Vol.2021-SLDM-194 No.2
Vol.2021-EMB-56 No.2
2021/3/25

```
#define min_size (8*1024)
#define max_size (2*1024*1024)
#stride 1024

base=(return of mmap);
for(size=min_size;size<=max_size;size=size*2){
    time0=get_seconds();
    for(addr=base;addr<base+size;addr=addr+stride){
        _DMB();
        #if defined(read) a=*((volatile int *)addr);
        #elif defined(write) *((volatile int *)addr)=1;
        #endif
    }
    time1=get_seconds();
    avg_ns=(time1-time0)/(csize/stride);
}
```

図 7 レイテンシ計測用マイクロベンチマーク
Fig. 7 Micro benchmark for latency measurement

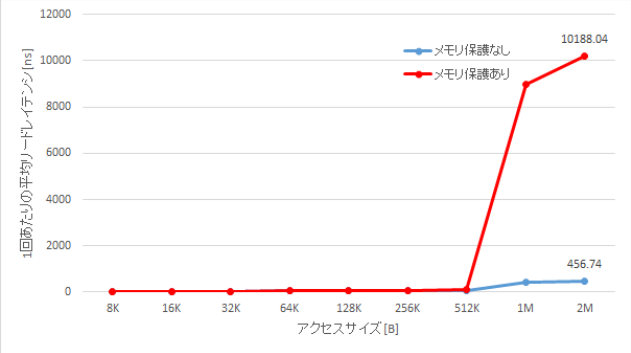

図 8 リードレイテンシ
Fig. 8 read latency

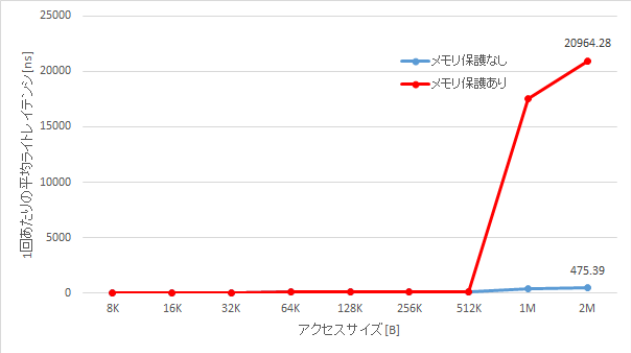

図 9 ライトレイテンシ
Fig. 9 write latency

5.3 レイテンシの内訳

マイクロベンチマークを動作させた際に CPU キャッシュ-NVMM 間に発行されるリードおよびライトリクエストの回数を計測した．アクセスサイズが 2[MB] の場合に発行されたリクエスト回数を**表 4** に示す．キャッシュミスが発生した際，リード時には対象のキャッシュラインを NVMM からフェッチするだけなのでリードリクエストのみが発行されるが，ライト時にはキャッシュラインのフェッチと NVMM への書き戻しが生じるためリードおよびライトリクエストが両方発行される．

表 4 リード・ライトリクエスト回数
Table 4 number of read and write request

	リードリクエスト回数	ライトリクエスト回数
リード	387,656	0
ライト	403,151	201,744

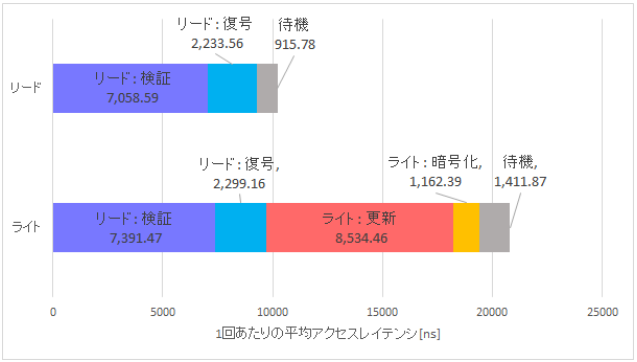

図 10 メモリアクセスレイテンシの内訳
Fig. 10 Memory access latency breakdown

続いて，メモリ保護機構の各内部状態のサイクル数をカウントすることで 1 回のメモリアクセスにおける各処理のレイテンシを調査した．アクセスサイズが 2[MB] の場合のメモリアクセスレイテンシの内訳を**図 10** に示す．ただし，待機とはメモリ保護機構が動作していない時間を意味する．

リード時にはリードレイテンシ合計（11,207.94[ns]）のうち整合性ツリーでの整合性検証が約 69%（7,058.59[ns]）を占めている．ライト時にはライトレイテンシ合計（20,799.36[ns]）のうち整合性検証が約 35%（7,391.47[ns]），ツリーの更新が約 40%（8,534.46[ns]）を占めている．このことから，メモリ保護機構において整合性ツリーの計算時間が最も大きいオーバーヘッドとなっていることがわかる．

6. まとめ

本稿では整合性ツリーと暗号化によるメモリ保護機構を持った NVMM エミュレータを実装した．Intel SGX でのメモリ保護に基づいたメモリ保護機構を作成し，既存の NVMM エミュレータに統合することでこれを実現した．本エミュレータ上ではメモリ保護機構が 50[MHz] で動作することが確認できた．

また，本エミュレータ上でマイクロベンチマークを用いて NVMM へのメモリアクセスレイテンシを計測したところ，メモリ保護なしの場合と比較してリード時に最大で約 22.3 倍，ライト時に最大で約 44.1 倍となり，メモリ保護機構のオーバーヘッドが大きいことが確認できた．またレイテンシの内訳を調査したところ，リードレイテンシのうち整合性ツリーでの整合性検証が約 69%，ライトレイテンシ

のうち整合性検証が約 35%，整合性ツリーの更新が約 40%を占めており，整合性ツリーの計算が最大のオーバーヘッドとなっていることが確認できた．

謝辞 本研究の一部はキオクシア株式会社と早稲田大学との組織連携活動の一貫として実施した．

参考文献

[1] Isakov, M., Gadepally, V., Gettings, K. M. and Kinsy, M. A.: Survey of Attacks and Defenses on Edge-Deployed Neural Networks, *2019 IEEE High Performance Extreme Computing Conference (HPEC)*, IEEE, pp. 1–8 (2019).

[2] Intel Corporation: Intel® Software Guard Extensions, Intel Corporation (online), available from ⟨https://software.intel.com/content/www/us/en/develop/topics/software-guard-extensions.html⟩ (accessed 2019-12-20).

[3] Arm Limited: TrustZone, Arm Limited (online), available from ⟨https://developer.arm.com/ip-products/security-ip/trustzone⟩ (accessed 2019-12-20).

[4] 大森侑，木村啓二：不揮発性メインメモリエミュレータの評価，研究報告システムと LSI の設計技術 (SLDM)，Vol. 2019, No. 19, pp. 1–8 (2019).

[5] Omori, Y. and Kimura, K.: Performance Evaluation on NVMM Emulator Employing Fine-Grain Delay Injection, *2019 IEEE Non-Volatile Memory Systems and Applications Symposium (NVMSA)*, IEEE, pp. 1–6 (2019).

[6] Xilinx: Xilinx Zynq-7000 SoC ZC706 Evaluation Kit, Xilinx (online), available from ⟨https://www.xilinx.com/products/boards-and-kits/ek-z7-zc706-g.html⟩ (accessed 2020-12-02).

[7] Suh, G. E., Clarke, D., Gassend, B., van Dijk, M. and Devadas, S.: AEGIS: Architecture for Tamper-Evident and Tamper-Resistant Processing, *ACM International Conference on Supercomputing 25th Anniversary Volume*, New York, NY, USA, Association for Computing Machinery, p. 357–368 (online), DOI: 10.1145/2591635.2667184 (2003).

[8] Nechvatal, J., Barker, E., Bassham, L., Burr, W., Dworkin, M., Foti, J. and Roback, E.: Report on the development of the Advanced Encryption Standard (AES), *Journal of Research of the National Institute of Standards and Technology*, Vol. 106, No. 3, p. 511 (2001).

[9] Gassend, B., Suh, G. E., Clarke, D., Van Dijk, M. and Devadas, S.: Caches and hash trees for efficient memory integrity verification, *The Ninth International Symposium on High-Performance Computer Architecture, 2003. HPCA-9 2003. Proceedings.*, IEEE, pp. 295–306 (2003).

[10] Rogers, B., Chhabra, S., Prvulovic, M. and Solihin, Y.: Using Address Independent Seed Encryption and Bonsai Merkle Trees to Make Secure Processors OS- and Performance-Friendly, *40th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO 2007)*, pp. 183–196 (online), DOI: 10.1109/MICRO.2007.16 (2007).

[11] Krawczyk, H., Bellare, M. and Canetti, R.: HMAC: Keyed-hashing for message authentication (1997).

[12] Gueron, S.: A Memory Encryption Engine Suitable for General Purpose Processors. (2016).

[13] Wegman, M. N. and Carter, J. L.: New hash functions and their use in authentication and set equality, *Journal of computer and system sciences*, Vol. 22, No. 3, pp. 265–279 (1981).

[14] Andrade, G., Lee, D., Kohlbrenner, D., Asanović, K. and Song, D.: Software-Based Off-Chip Memory Protection for RISC-V Trusted Execution Environments.

[15] Rachmawati, D., Tarigan, J. and Ginting, A.: A comparative study of Message Digest 5 (MD5) and SHA256 algorithm, *Journal of Physics: Conference Series*, Vol. 978, No. 1, IOP Publishing, p. 012116 (2018).

[16] Freij, A., Yuan, S., Zhou, H. and Solihin, Y.: Persist Level Parallelism: Streamlining Integrity Tree Updates for Secure Persistent Memory, *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, IEEE, pp. 14–27 (2020).

[17] Aoki Laboratory, GSIS, Tohoku University: Cryptographic Hardware Project in Aoki Lab., Tohoku Univ., Tohoku University (online), available from ⟨http://www.aoki.ecei.tohoku.ac.jp/crypto/index.html⟩ (accessed 2018-11-09).

[18] Xilinx: Xilinx/linux-xlnx: The official Linux kernel from Xilinx, Xilinx (online), available from ⟨https://github.com/Xilinx/linux-xlnx⟩ (accessed 2020-12-02).

[19] uyiromo: uyiromo/nvmtest: NVMM emulator on ZC706 board, uyiromo (online), available from ⟨https://github.com/uyiromo/nvmtest⟩ (accessed 2020-12-02).