Vol.2022-ARC-248 No.62
Vol.2022-SLDM-198 No.62
Vol.2022-EMB-59 No.62
2022/3/11

不揮発性メインメモリにおける効率的な整合性検証手法の検討

久保 龍哉[1,a)]　小池 亮[1,b)]　高前田 伸也[1,c)]

概要：DRAM の代替技術として急速に普及している不揮発性メモリ（NVM）技術は、盗聴、改ざん、リプレイ攻撃などのセキュリティ上の脅威にさらされている。データの暗号化と整合性検証がこれらの攻撃から NVM を保護するための標準的な技術であるが、いずれも性能上のオーバーヘッドが発生する。加えて、安全なシステムには、電源喪失時のデータ復元性も必要である。整合性のためのハッシュ木更新のオーバーヘッドが依然として重大な性能ボトルネックとなっていて、商用の NVM ではクラッシュが発生した場合、攻撃者にさらされる NVM にデータを排出することでデータを永続化させるため、不揮発性領域は依然として安全ではない。
本論文では、多重集合ハッシュ関数がメタキャッシュ上のハッシュノードを検証し、データの整合性更新とクラッシュ後の回復を効率的に行い、NVM を安全にする新しい整合性検証方式を提案する。メタデータキャッシュの状態を追跡するために、2 つの多重集合ハッシュを含むチップ上の 2 つの不揮発性レジスタを利用する。比較的単純な実装でありながら、整合性データの Write Back 更新を可能にし、メタデータのアクセス回数の削減に貢献する。本研究の性能を評価するため、Gem5 シミュレータで評価を行った。実験の結果、提案方式では、アプリケーションの実行時間を 18%削減し、NVM への書き込みを 6%削減することに成功した。

1. 序論

不揮発性メモリ（NVM）技術は、不揮発性とバイトアドレス性を備えつつ、低消費電力、高拡張性 [1], [2]、低遅延の特性を持つため、DRAM に代わる新たなストレージメディアとして近年注目されている。例えば、最大 3TB/ソケットの容量を持つ DIMM 互換の Intel Optane Persistent Memory は、クラウドサーバーなどに搭載された。

これらの特徴から、NVM は主記憶の一部としてメモリ階層を増強することが期待されているが、NVM は安全なメモリシステムにおける新たな課題をもたらす。まず、NVM は電源喪失時にもデータを保持するため、DRAM と比較して機密性や完全性などのセキュリティ攻撃に対して脆弱である [3]。データの機密性を保証し、バス・スヌーピングやメモリ・スキャン攻撃などの攻撃 [4], [5] を回避するためには、データの暗号化が必要で、データの改竄から NVM を保護するためには、整合性検証が必要である [6], [7]。第二に、いくつかの研究グループが DRAM システムのメモリ暗号化および整合性検証の設計に取り組んできたが、NVM においては、クラッシュ一貫性という新たな要件が発生する [8], [9], [10]。セキュアな NVM の設計では、システムクラッシュ時のデータとセキュリティメタデータのアトミックな永続性を保証する必要がある。

メモリコンテンツを悪意ある攻撃から守るために用いられるこれらのセキュリティ操作のうち、ストア操作はプログラム実行のクリティカルパスであり、最も大きな書き込みオーバーヘッドがかかる。なぜなら、クラッシュ一貫性のためには、ハッシュ木がメモリの現在の状態を反映し、分岐上のすべてのノードがアトミックに更新される必要があるからである。ハッシュ計算とそれに対応するメタデータの更新には、多数の CPU サイクルが必要であり [11], [12], [13]、ストア操作を完了するために相当数の追加書き込みが発生する。近年、セキュアな NVM システムにおいて、ストア操作のオーバーヘッドを削減する関心が高まっている。一つは、ハードウェアがサポートするロギングで、電源障害時にデータを復元するためにログを書き込みバッファに保持することで、メタデータキャッシュのライトバック・ポリシーを可能にするものである。しかし、ハードウェアロギングには余分な書き込みオーバーヘッドが含まれるため、安全なストア操作の書き込みオーバー

[1] 東京大学
The University of Tokyo, Bunkyo, Tokyo 113–8656, Japan
[a)] kubo-tatsuya535@g.ecc.u-tokyo.ac.jp
[b)] rkoike@g.ecc.u-tokyo.ac.jp
[c)] shinya@is.s.u-tokyo.ac.jp

© 2022 Information Processing Society of Japan

Vol.2022-ARC-248 No.62
Vol.2022-SLDM-198 No.62
Vol.2022-EMB-59 No.62
2022/3/11

ヘッドを削減するという課題が残される。書き込みオーバヘッドを削減するために、ハッシュ木の独特な構造を導入するアプローチがいくつかある。Log Hash [14] では、集合マルチセットハッシュ関数を利用することで、一連のストア操作の整合性検証が可能です。CacheTree [15] は、セキュリティメタデータキャッシュ上に追加のハッシュ木を構築することで、ライトバック・ポリシーを可能にし、NVM の書き込みオーバーヘッドを大幅に削減する。

上記のように、Log Hash は、安全かつ永続的な領域に集合ハッシュ値を保持し、安全でない一連のメモリ書き込みを後の時点で検証するための実行時オーバーヘッドを低く抑えている。しかし、整合性チェックの間隔が長いと、システムが攻撃に対して脆弱になることが示された [16]。つまり、Log Hash は、メタデータを更新するためのオーバーヘッドを大幅に削減しているが、整合性をチェックするためのオーバーヘッドは大きい。これに対し、CacheTree はメタキャッシュ上に小さなハッシュツリーを作成し、ライトバックポリシーを直接採用することで整合性更新の回数を減らしている。ハッシュツリーは、その整合性をチェックすると同時に効率的に更新するのに適したデータ構造であるのに対し、CacheTree の MACTree や HNodeTree の構造は、実行時にメタキャッシュ上の木の整合性をチェックする必要がないため、非効率である可能性がある。また、CacheTree の実装の複雑さは、HNodeCache の容量の制限を引き起こしている。

本論文では、メタデータのライトバックポリシーを採用し、実行時に効率的にメタデータを更新するために、2 つの集合ハッシュ値がメタキャッシュの現在の状態を反映するという整合性検証方式を提案する。CacheTree のキャッシュ上の木構造を Log Hash スキームに置き換えることで、性能とアクセスのオーバーヘッドを大幅に削減し、HNodeCache などの整合性更新でライトバックを可能にする領域を、MAC と Merkle Tree（MT）キャッシュを含むキャッシュ全体に拡張することができる。本研究は集合ハッシュ関数 [17] を利用して、NVM 内の未更新な MAC と MT ノードを再計算し、Anubis 設計を用いてクラッシュ後にシステムを回復させる [18]。

提案手法の性能を評価するため、サイクルレベルシミュレータ Gem5 [19] を使用して、SPEC CPU2017 [20] のトレースを収集した。評価の結果、本手法はすべてのベンチマークにおいて、最先端の手法を上回る性能を発揮することが分かった。平均して、他の最先端手法と比較して、実行時間を 18%改善し、NVM 書き込み回数を 6%削減することに成功した。

2. 背景

2.1 脅威モデル

次に、本研究の脅威モデルについて紹介する。先行研究

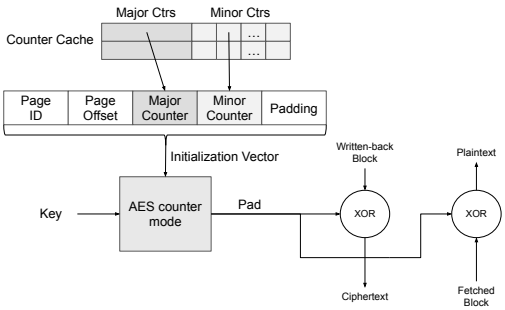


図 1: カウンターモード暗号化
Fig. 1 Counter mode encryption.

[9], [18], [21], [22] と同様に、プロセッサとキャッシュやレジスタなどの内部資源を利用して、攻撃者が観測・操作できない安全とされる TCB（Trusted Computing Base）を構築すると仮定し、データの機密性と整合性への攻撃を包含した脅威モデルを考える。さらに、メモリやメモリバスなど、コンピュータシステムの残りの部分はすべて信頼されないものであり、敵対者によって能動的に制御されると想定する。また、メモリの内容をスキャンしたり、メモリバスを盗み見ることができる受動的な攻撃だけでなく、チップ外データを改竄したり、メモリバス上のパケットを巻き戻すことができる能動的な攻撃者も想定する。さらに、我々の提案するソリューションは、信頼されていないオペレーティングシステム（OS）環境でも動作させることができる。最後に、差動電力攻撃と電磁波干渉攻撃は、本研究の範囲外とする。

2.2 暗号化

セキュアなプロセッサは、機密性を保護し、TCB 外のすべてのデータとコードを隠すために、メモリの暗号化を必要とする。最新設計では、図 1 に示すように、一般的にカウンターモードメモリ暗号化（CME） [23], [24], [25] が使用されている。CME は、まずカウンタ、秘密鍵、データラインのアドレスを入力として OTP（One Time Pad）を生成し、暗号化/復号化のレイテンシを、OTP と XOR しただけの暗号文/平文フェッチにオーバーラップさせる。CME のセキュリティは、秘密鍵で OTP を生成する初期化ベクトル（IV）を毎回変更し、二度と使用されないことで保証される。IV は図 1 に示すように、各ページのメジャーカウンタと各キャッシュラインの各マイナーカウンタで構成され、空間的・時間的な一意性を確保する。

2.3 メッセージ認証コード

能動的な攻撃は、値を直接置き換える（スプーフィング）、メモリ位置の値を入れ替える（スプライシング）、メモリの内容を古い値にロールバックする（リプレイング）などを

© 2022 Information Processing Society of Japan

行う可能性がある。このような能動的な攻撃から NVM を保護するために用いられる技術の 1 つが、メッセージ認証コード（MAC）である。安全なメモリの設計では、MAC と呼ばれる各データブロックの暗号署名を計算する。MAC が値とアドレスを入力とする場合、改ざんやスワップがあれば、保存されている MAC と再計算された MAC が不一致となる。そうすることで、スプーフィング攻撃やスプライシング攻撃を検知することができる。しかし、MAC は、値と対応する MAC の両方を古いバージョンに戻す、データリプレイ攻撃を検出することはできない。

2.4 整合性検証

ハッシュ木（MT）は、リプレイ攻撃から保護するために広く使われている方法である。MT のノードは、各子ノードの MAC 値から計算された MAC の集合で構成される。データの整合性をサポートするシステムは、データ上に MT を構築し、そのルートを攻撃者が改竄できないように、オンチップのセキュアレジスタに保存する。メモリの読み出しアクセスは、葉からルートまでのハッシュを再計算し、保存したものと比較することで検証できる。データを更新は、ルートを含む更新パス上のすべてのノードのハッシュ値を更新することで達成される。MT の考え方として、そのルートは全データのハッシュを効率的に表現する。つまり、MT は、各ノードがそのノード以下の全コンテンツの表現として動作するため、ルートの検証や更新を高速に行うことができる典型的なデータ構造である。しかし、主記憶装置内の全データの MT を維持するには、依然として膨大なオーバーヘッドが発生する。Bonsai Merkle Tree （BMT） [11] は、MT の更新のオーバーヘッドがその高さに対して線形であることと、MAC がそのカウンタを入力として取り、カウンタの整合性が保証される限り、データの完全性を確保できるという観測に基づいて導入された。図 2 に示すように、BMT はカウンタ上に MT を作成し、ナイーブ MT と比較してその高さを削減している。BMT は、SHA1 [26] などの安全なハッシュ関数を用いて計算した子の 64 ビットハッシュ値を 8 個連結して、64 バイトの各メルクル木ノードを算出する。

2.5 クラッシュ一貫性

暗号化と整合性検証によりデータを安全に保護することができるが、NVM の場合には追加でこれらのセキュリティ機構にクラッシュ一貫性をサポートする必要性がある。クラッシュ一貫性のある整合性検証のためには、ストアのたびに対応するカウンター、MAC、MT ノードをアトミックに更新しなければならない。データを永続的に保持するための機能として、Intel は Asynchronous DRAM Refresh（ADR）とよばれる Write Pending Queue（WPQ）のデータを電源喪失時に NVM に流すことで永続性を提供する機能と、extended Asynchronous DRAM Refresh （eADR）とよばれるキャッシュを同様の方法で永続化させる機能を提供した。しかし、依然としてそれらのデータは安全ではなく、ハッシュ木のルートは最新の状態を反映している必要がある。

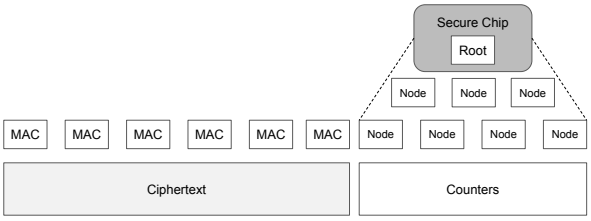


図 2: BMT による整合性検証
Fig. 2 Bonsai Merkle Tree.

2.6 メタキャッシュのライトバック更新

メタデータキャッシュは、暗号化と整合性検証を高速化する。しかし、メタキャッシュのライトバックポリシーを採用するだけでは、クラッシュ時に整合性の不一致が発生してしまう。Strict Persistency（SP）方式では、メタキャッシュがライトスルー・ポリシーを採用し、セキュア NVM のメタデータ・クラッシュ一貫性を実現するため、NVM の書き込みが増え、システム性能と NVM の寿命を大幅に低下させることになる。

Ye らは、カウンタのライトスルー更新のオーバーヘッドを減らすために、クラッシュ後に暗号化カウンタを回復する Osiris と呼ばれるスキームを提案した [9]。Osiris は、データとともに存在する ECC ビットを利用して正しいカウンタを見つけることで、カウンタのクラッシュ一貫性を保証し、カウンタキャッシュのライトバック・ポリシーを可能にして NVM の書き込みを削減する。本研究では、ADR をサポートするシステムが Osiris をセキュリティスキームと統合すると仮定する。

3. 関連研究

機密性、整合性、クラッシュ一貫性を保証するための操作のうち、整合性更新はプログラム実行のクリティカルパスであり、最大の書き込みオーバーヘッドが発生する。本節では、この課題に対処するための 2 つの研究を紹介する。まず、整合性検証のために多重集合ハッシュ関数を利用し、実行時に効率的にメタデータを更新する Log Hash を紹介する。次に、CacheTree を紹介する。これは、メタキャッシュ上に追加の MT を作成し、そのライトバック方針を採用し、システムクラッシュ後のリカバリーを可能にするも

© 2022 Information Processing Society of Japan

Vol.2022-ARC-248 No.62
Vol.2022-SLDM-198 No.62
Vol.2022-EMB-59 No.62
2022/3/11

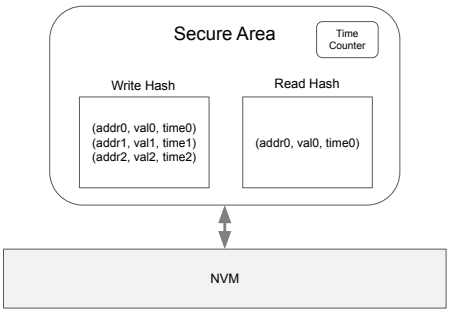


図 3: Log Hash の概要
Fig. 3 Overview of Log Hash.

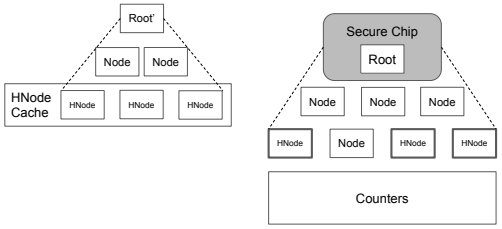


図 4: CacheTree の概要
Fig. 4 Overview of CacheTree.

のである。

3.1 Log Hash

図 3 に Log Hash の概要を示す。この方式では、多重集合ハッシュ関数を利用することで、ログを固定長で保持することができる。この関数は、集合に新しいメンバーを追加するという点でハッシュをインクリメンタルに更新することができ、追加されたメンバーの長さに対して線形な計算量でハッシュを更新することが可能である。Log Hash は、オンチップに格納されたハッシュに新しいログをインクリメントすることで、すべてのメモリ操作の読み取りログと書き込みログを維持し、実行時に最小限のオーバーヘッドで整合性の更新を可能にする。Log Hash が整合性をチェックするとき、ログハッシュはすべてのメモリコンテンツを書き込みログにインクリメントし、それを読み込みログと比較する必要がある。このように、ログハッシュは整合性をチェックするために大きなオーバーヘッドを必要とする。さらに、Log Hash では、セキュリティのために各データのタイムスタンプをメモリに保存する必要がある。

3.2 CacheTree

次に、メタキャッシュ上に追加の MT を構築し、MT ノードの更新回数を減らす CacheTree を紹介する。CacheTree は、HNode キャッシュと呼ばれるもう一つの上位 MT キャッシュを持ち、その上に小さな MT を構築する。図 4 に CacheTree の仕組みを示す。右のツリーがメインの MT、左が HNodeTree を表している。CacheTree では、あるノードが HNode キャッシュ上にキャッシュされると、メイン MT 上の葉からルートへのパスを更新する代わりに、HNodeTree 上のノードを更新する。この方式では、HNodeTree のルートを安全なオンチップ不揮発性レジスタに保存し、クラッシュ後に HNodeTree を再構築してその整合性を検証する。HNodeTree 上の経路がメイン MT よりも短いため、書き込みのオーバーヘッドを削減し、実行性能を向上させることができる。

4. 提案手法

4.1 モチベーション

ここでは、提案手法の設計の動機付けとして、NVM におけるメモリ認証のオーバーヘッドを考察する。NVM の書き込みエネルギーとレイテンシは、読み出しエネルギーとレイテンシよりも高く、特にマルチレベル／トリプルレベルセル（MLC/TLC）NVM では、従来の DRAM よりも高い値である [27]。さらに、ほとんどの NVM は寿命が限られている。しかし、メモリ認証には余分なメモリアクセスがあり、データの読み出しや NVM への追加書き込みが滞り、システムの IPC を劣化させる。したがって、セキュリティメタデータの更新と暗号化処理のオーバーヘッドにより、NVM のエネルギー／レイテンシーのオーバーヘッドが大きくなり、寿命を劣化させる。

セクション 3 で述べた研究は、主に MT 更新のオーバーヘッドを削減し、NVM の実行性能、エネルギー効率、寿命を向上させる必要性によって動機づけられている。私たちは、以下の見解に基づいて、提案手法の設計をした。

- Log Hash では、インクリメント可能な多重集合ハッシュ関数を用いて、実行時に高速な整合性メタデータの更新を可能にしている。Log Hash が必要とするメタデータ更新のオーバーヘッドは、キャッシュブロックの多重ハッシュ計算一回である。
- Log Hash は、アプリケーションの実行中に定期的に NVM から読み込まれる一連の値の整合性を検証し、整合性チェックのために NVM の内容全体を読み込む必要がある。
- CacheTree は、メタデータキャッシュのライトバックポリシーを採用し、NVM ライトオーバーヘッドを削減するために、メタデータキャッシュ上に余分な MT を構築するが、CacheTree の MACTree および HNodeTree の構造は、実行時にメタキャッシュ上の余分のツリーの整合性をチェックする必要がないため非効率である可能性がある。また、CacheTree の実装が

© 2022 Information Processing Society of Japan

Vol.2022-ARC-248 No.62
Vol.2022-SLDM-198 No.62
Vol.2022-EMB-59 No.62
2022/3/11

複雑なため、HNodeCache の容量に制限が発生する。

つまり、Log Hash は最小限のオーバーヘッドで整合性のためのメタデータを更新できるが、その整合性を検証するためにすべてのメモリ内容を読み込む必要がある。しかし、CacheTree 方式では、実行時に余分な MT の整合性をチェックする必要はない。以上の分析から、提案手法は CacheTree の余分な MT を Log Hash で使用する機構に置き換え、メタデータキャッシュ全体へのライトバック・ポリシーを採用して領域を拡張する。言い換えれば、提案手法はメタデータの内容を追跡するために 2 つの多重集合ハッシュ値をオンチップの不揮発性レジスタに保持し、低コストのインクリメント型多重集合ハッシュ値を活用することで MT 更新のオーバーヘッドを大幅に削減することを可能にする。

4.2 Read Hash と Write Hash の更新

提案手法は、メタキャッシュの現在の状態を反映する 2 つの不揮発性多重集合ハッシュ値を用いることで、メタデータキャッシュのライトバックポリシーと電源喪失後のメタデータの回復を可能にする。2 つのハッシュ値を Log Hash と同様に Read Hash と Write Hash と名付け、多重集合ハッシュ関数として MSet-Add-Hash [17] を選択し、ハッシュの計算と増分に MD5 演算が 1 回必要であることを示す [28]。キャッシュ内容の更新には、キャッシュされたデータを直接更新する場合と、データを退避させて新しいものに置き換える場合とがある。メタキャッシュの状態から見ると、この 2 つのケースは古いデータを新しいデータに置き換えるという点では同じ操作である。図 5 に、提案手法で MT を更新する際に行われる動作の概要を示す。データとアドレスのペアを表すノードを N_m とする。メタキャッシュには N_0、N_1、N_2、N_3 があり、図 5- (a) のように Read Hash の値は (N_0, N_1, N_2, N_3) の集合、Write Hash の値は空集合を表すと仮定する。メモリ書き込みの場合、メモリコントローラはまずデータが LLC にあるかどうかを確認し、ない場合は対応するセキュリティメタデータを更新する。ライトバックポリシーで N_1 を N_1' として更新する場合、セキュリティコントローラは N_1' を Read Hash、N_1 を Write Hash にインクリメントする（図 5- (b)）。メタキャッシュからデータを退避させる場合、Read Hash と Write Hash はライトバックポリシーでのデータ更新と同じ動作をする（図 5- (c)）。直感的には、提案手法は Read Hash でキャッシュへの書き込みログを取り、Write Hash で書き込まれる値のログを取る。メタキャッシュの整合性をチェックする場合、我々の設計では、Write Hash にキャッシュ上のすべてのデータをインクリメントし、Read Hash と比較する。キャッシュ内のすべてのデータが有効であれば、2 つのマルチセットハッシュ値は同じになるはずで、電源喪失後に MT 内のダーティノードを再計算してその整合性を確認することでリカバリーが可能になる。我々の設計の多重集合要素には、タイムスタンプが含まれていない。しかし、キャッシュされる時と退避される時の間には間隔があり、キャッシュされる古いデータはと同じアドレスの更新されるデータの両方を同時に決定することはできないため、このことは安全性を損なわない。

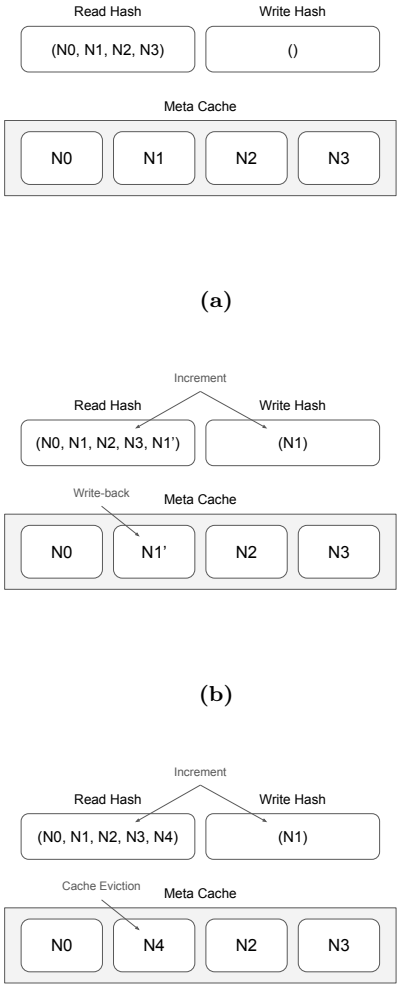


図 5: (a) Read Hash と Write Hash の初期状態
(b) N_1 の N_1' への更新
(c) N_1 の N_4 への退避

Fig. 5 (a) A state of Read Hash, Write Hash, and the meta cache.
(b) Operation of direct updating N_1 as N_1'.
(c) Operation of evicting N_1, replaced with N_4.

4.3 動作フロー

提案手法では、MT のパス上のノードがチップ上にキャッシュされているのを発見次第、データの整合性検証を完了するという従来の最適化を Read 操作で行っている。この

© 2022 Information Processing Society of Japan

Vol.2022-ARC-248 No.62
Vol.2022-SLDM-198 No.62
Vol.2022-EMB-59 No.62
2022/3/11

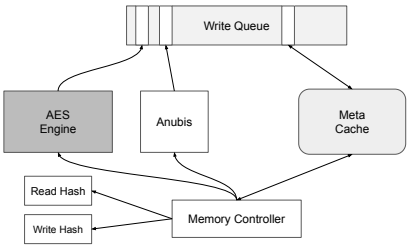


図 6: 操作フロー
Fig. 6 Operation Flow

最適化により、キャッシュされたノードを木のルートであるかのように扱うことができるため、安全性を損なうことはない。ここでは、提案手法の詳細な Write 処理を説明する。本方式では、Anubis 機構を利用してメタキャッシュのコンテンツのアドレスを保存し、クラッシュリカバリを可能にしており、リカバリのプロセスについては後述する。図 6 に示すように、メモリコントローラにおける書き込み要求の処理は、以下の手順で構成される。1. LLC から退避したデータブロックがメモリコントローラに到着すると、NVM またはメタデータキャッシュからそれぞれのカウンタと MT ノードをフェッチして暗号化と整合性検証を行う。2. 対応するカウンタからルートまでの経路上の MT ノードを更新し、メタキャッシュに格納されているノードに到達するまで繰り返す。3. キャッシュされたノードを発見した場合、一連の更新は停止し、Read Hash および Write Hash は、セクション 4.2 に示される方法でメタキャッシュの変更を反映させる。4. 最後に、暗号化されたデータ、カウンター、Anubis データ、キャッシュされていない MT ノードを含む、すべての永続的な書き込みが WPQ に配置される。eADR をサポートするシステムは、eADR の機能ですべてのキャッシュされたコンテンツが永続的であるため、WPQ を使用してステップ 4 で永続性をサポートする必要がないことに注意する。メタキャッシュが NVM にデータを退避させる場合、メモリコントローラは Read Hash と Write Hash だけでなく、退避させたノードから次のキャッシュノードまでの NVM 内の MT の経路もアトミックに更新しなければならない。

4.4 リカバリー

次に、Read Hash と Write Hash を用いて、システムクラッシュや電源障害からメタデータを復旧させる方法について、以下の 2 つのケースで説明する。

4.5 ADR 対応システムの場合

ADR をサポートするシステムでは、WPQ は永続領域であり、クラッシュが発生した場合、その内容は NVM にフラッシュされる。メタキャッシュのライトバックポリシーを採用しているため、システムがクラッシュすると、MAC、カウンタ、MT ノードが失われる。これらの内容を回復する手順は以下のとおり。1 提案手法は、Osiris を通じて NVM 内のカウンタを回復する。2 Anubis にアドレスが含まれるすべての MAC と MT ノードを再計算し、Write Hash にインクリメントする。3 提案手法は Read Hash と Write Hash が等しいかどうかをチェックする。不一致の場合、攻撃を警告する。

4.6 eADR 対応システムの場合

eADR をサポートするシステムでは、クラッシュが発生した場合、NVM に内容をフラッシュすることで、キャッシュ全体を永続的なドメインに追加する。この場合、システムは Osiris を利用しないことに注意する。また、システムクラッシュ時には、キャッシュされていない各 MT ノードを計算し、フラッシュされたカウンタに従って NVM に排出すると仮定する。しかし、eADR をサポートするシステムは、キャッシュ全体をフラッシュするための大規模なバッテリーバックアップを持っており、残りの MT ノードを計算するために必要な余分なバッテリー量は比較的小さいため、この仮定は妥当である。メタデータを復元するためには、ADR 対応の場合と同じ 2 ステップと 3 ステップを行う。

4.7 セキュリティ分析

悪意のある攻撃者がカウンターを変更した場合、BMT はルートの不一致を検出し、または Osiris はそれを回復することができる。暗号文や MAC が変更されれば、MAC の不一致が発生する。MT ノードに関しては、それぞれのカウンタから再計算できるため、MT ノードへの攻撃は無意味となる。したがって、クラッシュ後に変更されたり未更新の MAC とカウンターを再構築すれば十分である。提案手法は、対応する暗号文とカウンターを使用してダーティな MAC と MT ノードを再計算し、2 つの多重集合ハッシュ値を再構築して保存されているものと比較することにより、暗号文とカウンターが妥当であることを確認することができる。最後に、理論上、攻撃者は同じハッシュを生成する値を計算することで、MAC の不一致なしにデータを変更することができるが、同じハッシュを生成する 2 つの値を見つけることは困難であることは示されている [11] 。

5. 評価

5.1 評価方法

SPEC CPU2017 のベンチマークをトレース収集に使用し、サイクルレベルシミュレータ Gem5 のシステムコールエミュレーション（SE）モード上で提案手法を評価した。すべての実験において、アプリケーションを一定サイクル

© 2022 Information Processing Society of Japan

表 1: シミュレーション設定
Table 1 Configuration of the Simulated System.

Processor	
CPU	1 out-of-order core, RISCV, 2 GHz
L1 i-cache	16KB, 2-way, 64B block, 2 cycles
L1 d-cache	64KB, 2-way, 64B block, 2 cycles
Main Memory	
NVM	8GB, Single Channel, 1 rank/channel, Read = 60ns, Write = 150ns
Memory Controller	
Meta Cache	128KB, 4-way, 64B block, 2 cycles
BMT	8 levels, 8-ary, 64B each node
Security Parameter	
Encryption Latency	24 cycles
MAC Latency	80 cycles
Multiset Hash Latency	160 cycles

表 2: 各ワークロードにおける MemWrite の割合
Table 2 Ratio of MemWrite Instruction.

Benchmark	Proportion of MemWrite Instruction
deepsjeng	46.65%
gcc	13.49%
leela	13.74%
mcf	15.72%
omnetpp	7.05%
perlbench	15.33%
x264	2.35%
xalancbmk	11.12%
xz	14.64%

早送りし、その後 10M 命令をシミュレートしている。表 1 に示すように、8GB NVM メモリシステムを搭載したシングルコアのアウトオブオーダープロセッサをモデル化し、暗号化および整合性検証スキームを含む、提案手法の設計をエミュレートするセキュリティユニットを追加するようにシミュレータを構成した。NVM の Read 遅延は 60ns、Write 遅延は 150ns に設定している [29]。提案手法の設計の性能を評価するために、以下の 4 つの方式を比較した。

- Baseline: 暗号化と整合性検証のための BMT を実装した、セキュアベースライン NVM の設計。
- ASSURE: ASSURE [7] はマルチルート MT を搭載することで、NVM への書き込み回数を減らし、整合性検証の性能を高めている。本研究でも 2 つの不揮発性レジスタをオンチップで必要とするため、1 つのホット MT を持つ分散型ダイナミックマルチルート Merkle Trees を採用した。
- CacheTree: CacheTree は、メタキャッシュ上に追加の MT を維持し、MAC と MT キャッシュのライトバック・ポリシーを採用する。
- 提案手法: 提案手法とメタキャッシュのライトバック・ポリシーを採用。

先行研究 [9], [30] と同様に、AES 暗号化待ち時間を 24 サイクル、MAC 計算を 80 サイクル [12], [31] と仮定している。また、本研究では、MSet-Add-Hash 関数が多重集合ハッシュの計算またはインクリメントに 160 サイクルを要すると仮定する [31]。シミュレーションでは、カウンタキャッシュ、MAC キャッシュ、MT キャッシュを組み合わせたメタキャッシュを設計する。各データのキャッシュの方針は、パラメータと同位置に配置することで制御した。5.3 項では、メタキャッシュのサイズについてセンシティブな分析を行った。

5.2 実行時間

図 1 は、提案手法がシステム性能に与える影響を、異なる方式と比較したものである。正規化実行時間は、1 命令あたりのサイクル数（CPI）をベースライン CPI で正規化したものである。提案手法はベースライン方式と比較して、平均で 53%実行時間が向上している。ASSURE と CacheTree は、ベースラインと比較して、平均してそれぞれ 25%、34%実行時間が改善された。すべてのベンチマークにおいて、提案手法は CacheTree を上回り、他の 3 つの方式の中で最も優れている。特に、書き込みの多い Deepsjeng のワークロードでは、CacheTree の HNodeCache よりも MT キャッシュのライトバック・ポリシーを採用する領域が大きいため、提案手法がさらに優れた性能を発揮している。さらに、多重集合のハッシュ値を維持するシンプルさが、CacheTree に対する性能上の優位性につながっていると考えられる。ASSURE は、ホットルートがルートに近いため、利点が限定的である。ベースラインと ASSURE を CacheTree や提案手法と比較した場合、メタキャッシュのライトバック・ポリシーは、メタデータ書き込みの回数を減らすことで、大幅な性能向上につながる。この結果から、提案手法はセキュア NVM の整合性検証のオーバーヘッドを大幅に軽減することがわかった。

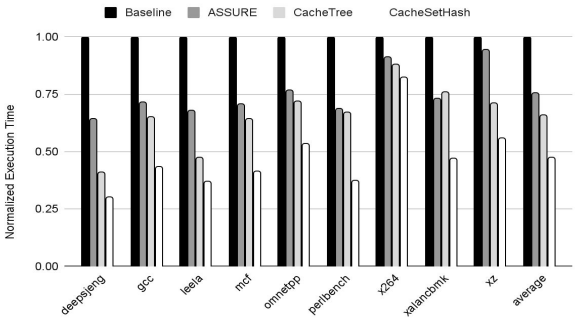


図 7: 実行時間
Fig. 7 Execution Performance.

5.3 メタデータ書き込み量

図 2 では、各方式で正規化した余分な NVM 書き込みを比較している。この図では、暗号化および整合性チェックのためのセキュリティ操作による書き込みのみを考慮している。提案方式では、ベースライン方式に比べて NVM への書き込み回数が平均 32%減少。また、ASSURE と CacheTree

© 2022 Information Processing Society of Japan

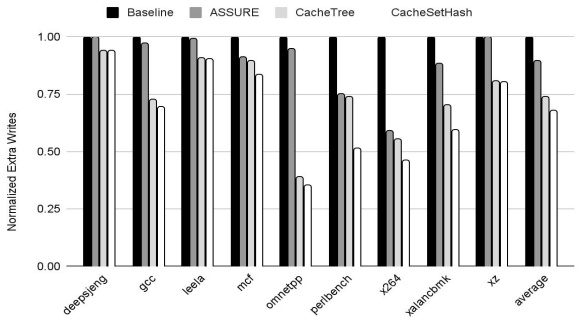
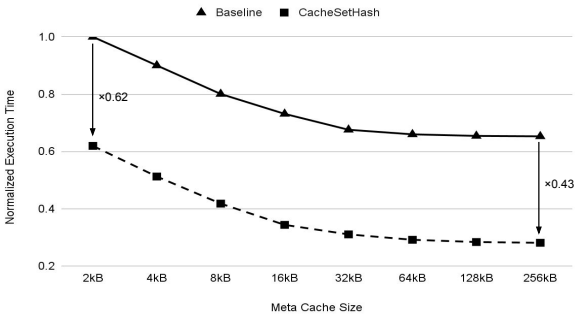


図 8: メモリ書き込み量
Fig. 8 Metadata Writes.

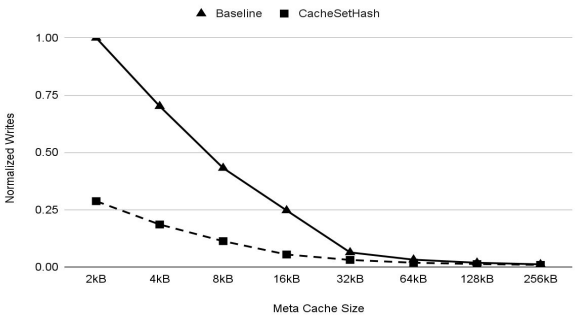
は、MT のショートパスへのアクセスにより、平均してそれぞれ 11%、26%の書き込みオーバヘッドを改善した。この図 2 は、CacheTree と提案手法のライトバックポリシーが、メモリ書き込みの数を減らすことで、書き込みオーバヘッドを改善することも示している。CacheTree が ASSURE より優れているのは、ルートに近いホットルートを持ち、CacheTree は MT の最終レベルに複数の HNode を持ち、NVM ライトの削減を可能にしているからと考えられる。本研究で提案した方式は、ライトバック・ポリシーを採用したキャッシュのサイズにより、すべてのベンチマークで CacheTree を上回っていることがわかる。MT で使用されるパスが短いほど、セキュリティ操作に必要な書き込み回数が少なくなる。以上のことから、提案手法は NVM チップの寿命延長に有効であることがわかる。

5.4 メタキャッシュ量の影響

また、メタキャッシュのサイズを変化させることによる分析も行った。図 9 では、メタキャッシュのサイズが最小のベースライン（2kB）の設定に対して正規化した結果である。すべてのプロセスで GCC ワークロードが実行されている。図 9- (a) は、実行性能に関する結果を示しており、提案手法はメタキャッシュのサイズが大きいほど、ベースラインより性能が優れていることがわかる。この図では、キャッシュヒット数が多いほど MT 更新のオーバーヘッドを削減する機会が増えるため、メタキャッシュのライトバック・ポリシーの貢献によってこの傾向を説明することができる。一方、図 9- (b) では、NVM 書き込みに関する結果を示している。この図からわかるように、提案手法はベースラインよりもメタキャッシュの容量が少ない方が性能が良いという傾向が見て取れる。キャッシュサイズを大きくすることで、ベースラインの比較的高い書き込みオーバーヘッドを隠していることが推察される。メタキャッシュのサイズに関しては、実行時性能と NVM の寿命の間にトレードオフがあることがわかる。



(a)



(b)

図 9: (a) メタキャッシュのサイズと実行時間の関係
(b) メタキャッシュのサイズとメモリ書き込み量の関係
Fig. 9 (a) Sensitivity of the meta cache size on the execution performance.
(b) Sensitivity of the meta cache size on the write overhead.

© 2022 Information Processing Society of Japan

6. 結論

メモリ暗号化、整合性検証、クラッシュ一貫性は、安全な計算機システムを実現するために不可欠なプリミティブである。しかし、性能上のオーバーヘッドの大部分は、ハッシュ木の根をアトミックに更新することにある。本稿では、この課題を解決し、NVM を用いたセキュリティプリミティブを効果的に実装するスキームを提案した。提案手法は、メタキャッシュの状態を追跡するために 2 つの多重集合ハッシュ値を保持し、メタキャッシュのライトバックポリシーの採用を可能にすることで、NVM 書き込みを大幅に削減することができる。提案手法は、他の最先端ソリューションと比較して、最大 18%の性能向上と、6%の NVM 書き込み量の削減を達成した。

謝辞

本研究の一部は，JSPS 科研費 19H04075，18H05288，および キオクシア奨励研究の支援により行われたものである．

参考文献

[1] Lee, B. C., Zhou, P., Yang, J., Zhang, Y., Zhao, B., Ipek, E., Mutlu, O. and Burger, D.: Phase-change technology and the future of main memory, *IEEE micro*, Vol. 30, No. 1, pp. 143–143 (2010).

[2] Li, Z., Zhou, R. and Li, T.: Exploring high-performance and energy proportional interface for phase change memory systems, *2013 IEEE 19th International Symposium on High Performance Computer Architecture (HPCA)*, IEEE, pp. 210–221 (2013).

[3] Swami, S., Rakshit, J. and Mohanram, K.: STASH: Security architecture for smart hybrid memories, *2018 55th ACM/ESDA/IEEE Design Automation Conference (DAC)*, IEEE, pp. 1–6 (2018).

[4] Awad, A., Wang, Y., Shands, D. and Solihin, Y.: Obfusmem: A low-overhead access obfuscation for trusted memories, *Proceedings of the 44th Annual International Symposium on Computer Architecture*, pp. 107–119 (2017).

[5] Chhabra, S. and Solihin, Y.: i-NVMM: A secure non-volatile main memory system with incremental encryption, *2011 38th Annual international symposium on computer architecture (ISCA)*, IEEE, pp. 177–188 (2011).

[6] Aga, S. and Narayanasamy, S.: Invisimem: Smart memory defenses for memory bus side channel, *ACM SIGARCH Computer Architecture News*, Vol. 45, No. 2, pp. 94–106 (2017).

[7] Rakshit, J. and Mohanram, K.: Assure: Authentication scheme for secure energy efficient non-volatile memories, *Proceedings of the 54th Annual Design Automation Conference 2017*, pp. 1–6 (2017).

[8] Yan, C., Englender, D., Prvulovic, M., Rogers, B. and Solihin, Y.: Improving cost, performance, and security of memory encryption and authentication, *ACM SIGARCH Computer Architecture News*, Vol. 34, No. 2, pp. 179–190 (2006).

[9] Ye, M., Hughes, C. and Awad, A.: Osiris: A Low-Cost Mechanism to Enable Restoration of Secure Non-Volatile Memories., Technical report, Sandia National Lab.(SNL-NM), Albuquerque, NM (United States) (2018).

[10] Liu, S., Kolli, A., Ren, J. and Khan, S.: Crash consistency in encrypted non-volatile main memory systems, *2018 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, IEEE, pp. 310–323 (2018).

[11] 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)*, IEEE, pp. 183–196 (2007).

[12] Lehman, T. S., Hilton, A. D. and Lee, B. C.: PoisonIvy: Safe speculation for secure memory, *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, IEEE, pp. 1–13 (2016).

[13] Yang, F., Chen, Y., Mao, H., Lu, Y. and Shu, J.: ShieldNVM: An Efficient and Fast Recoverable System for Secure Non-Volatile Memory, *ACM Transactions on Storage (TOS)*, Vol. 16, No. 2, pp. 1–31 (2020).

[14] Suh, G. E., Clarke, D., Gasend, B., Van Dijk, M. and Devadas, S.: Efficient memory integrity verification and encryption for secure processors, *Proceedings. 36th Annual IEEE/ACM International Symposium on Microarchitecture, 2003. MICRO-36.*, IEEE, pp. 339–350 (2003).

[15] Chen, Z., Zhang, Y. and Xiao, N.: CacheTree: Reducing Integrity Verification Overhead of Secure Non-Volatile Memories, *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* (2020).

[16] Shi, W., Lee, H.-H. S., Lu, C. and Ghosh, M.: Towards the issues in architectural support for protection of software execution, *ACM SIGARCH Computer Architecture News*, Vol. 33, No. 1, pp. 6–15 (2005).

[17] Clarke, D., Devadas, S., Van Dijk, M., Gassend, B. and Suh, G. E.: Incremental multiset hash functions and their application to memory integrity checking, *International conference on the theory and application of cryptology and information security*, Springer, pp. 188–207 (2003).

[18] Zubair, K. A. and Awad, A.: Anubis: ultra-low overhead and recovery time for secure non-volatile memories, *Proceedings of the 46th International Symposium on Computer Architecture*, pp. 157–168 (2019).

[19] Binkert, N., Beckmann, B., Black, G., Reinhardt, S. K., Saidi, A., Basu, A., Hestness, J., Hower, D. R., Krishna, T., Sardashti, S. et al.: The gem5 simulator, *ACM SIGARCH computer architecture news*, Vol. 39, No. 2, pp. 1–7 (2011).

[20] Bucek, J., Lange, K.-D. and v. Kistowski, J.: SPEC CPU2017: Next-generation compute benchmark, *Companion of the 2018 ACM/SPEC International Conference on Performance Engineering*, pp. 41–42 (2018).

[21] Awad, A., Ye, M., Solihin, Y., Njilla, L. and Zubair, K. A.: Triad-nvm: Persistency for integrity-protected and encrypted non-volatile memories, *Proceedings of the 46th International Symposium on Computer Architecture*, pp. 104–115 (2019).

[22] Awad, A., Suboh, S., Ye, M., Zubair, K. A. and Al-Wadi, M.: Persistently-secure processors: Challenges and opportunities for securing non-volatile memories, *2019 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, IEEE, pp. 610–614 (2019).

© 2022 Information Processing Society of Japan

[23] Awad, A., Manadhata, P., Haber, S., Solihin, Y. and Horne, W.: Silent shredder: Zero-cost shredding for secure non-volatile main memory controllers, *ACM SIGPLAN Notices*, Vol. 51, No. 4, pp. 263–276 (2016).
[24] Swami, S., Rakshit, J. and Mohanram, K.: SECRET: Smartly encrypted energy efficient non-volatile memories, *2016 53nd ACM/EDAC/IEEE Design Automation Conference (DAC)*, IEEE, pp. 1–6 (2016).
[25] Young, V., Nair, P. J. and Qureshi, M. K.: DEUCE: Write-efficient encryption for non-volatile memories, *ACM SIGARCH Computer Architecture News*, Vol. 43, No. 1, pp. 33–44 (2015).
[26] Eastlake, D. and Jones, P.: US secure hash algorithm 1 (SHA1) (2001).
[27] Arjomand, M., Kandemir, M. T., Sivasubramaniam, A. and Das, C. R.: Boosting access parallelism to PCM-based main memory, *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*, IEEE, pp. 695–706 (2016).
[28] Rivest, R. and Dusse, S.: The MD5 message-digest algorithm (1992).
[29] Lee, B. C., Ipek, E., Mutlu, O. and Burger, D.: Architecting phase change memory as a scalable dram alternative, *Proceedings of the 36th annual international symposium on Computer architecture*, pp. 2–13 (2009).
[30] Zuo, P., Hua, Y. and Xie, Y.: Supermem: Enabling application-transparent secure persistent memory with low overheads, *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*, pp. 479–492 (2019).
[31] Jeong, J., Park, C. H., Huh, J. and Maeng, S.: Efficient hardware-assisted logging with asynchronous and direct-update for persistent memory, *2018 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, IEEE, pp. 520–532 (2018).

© 2022 Information Processing Society of Japan