Vol.2016-ARC-219 No.16
Vol.2016-SLDM-175 No.16
Vol.2016-EMB-40 No.16
2016/3/24

パス型ORAMのバンド幅オーバーヘッド削減手法の検討

藤枝 直輝[1,a)]　山内 涼[1]　市川 周一[1,b)]

概要：Oblivious RAM（ORAM）は，データそのものだけでなく，そのアクセスパターンが漏洩することも防止できるプロセッサ技術である．パス型 ORAM はその軽量なプロトコルの 1 つであるが，1 回の ORAM リクエストに対して多くのメモリアクセスを必要とし，バンド幅オーバーヘッドはなお大きい．本稿では，冗長なメモリアクセスの省略という観点から，パス型 ORAM に対する 2 種類のバンド幅オーバーヘッド削減手法を検討する．シミュレーションによる評価の結果，ブロックサイズが 4096 バイトのとき，平均のバンド幅オーバーヘッドはそれぞれ 5.6%，4.9%削減された．

Preliminary study on the reduction of bandwidth overhead for Path ORAM

Naoki Fujieda[1,a)]　Ryo Yamauchi[1]　Shuichi Ichikawa[1,b)]

1. はじめに

メモリの暗号化 [1]，すなわち，プロセッサから外部メモリへと送受信されるデータを暗号化することは，そのデータバスを観測されることによる情報流出を防ぐために重要である．XOM（eXecute Only Memory）[2] や AEGIS アーキテクチャ [3] などのセキュアプロセッサ・アーキテクチャでは，データはメモリからキャッシュされるときに復号され，キャッシュからメモリへと書き戻されるときに再び暗号化される．しかしながら，情報流出はデータそのものだけでなく，そのアクセスパターンを観測されることでも生じうる [4]．この脅威に対しては，メモリの暗号化に加え，アクセスパターンを隠蔽する手法が必要である．

Oblivious RAM（ORAM）[5] はダミーのアクセスなどを用いてアクセスパターンを隠蔽する手法である．近年その軽量なプロトコルとして提案されているのが，パス型 ORAM（Path ORAM）[6], [7] である．パス型 ORAM はそのシンプルさからハードウェアによる実装にも適しているが，ダミーのアクセスにより生じるバンド幅オーバーヘッドはなお大きく，その削減が望まれている．

本研究は，パス型 ORAM のバンド幅オーバーヘッドの削減を目的とする．パス型 ORAM においては，ORAM アクセスの際に参照される外部メモリ領域をパスとよび，1 回の ORAM アクセスは 1 つのパスの読み出しと書き込みを伴う．我々は，連続する ORAM アクセスにおいてパスの一部が重複することに着目した．もしこの重複部に関するデータがチップ内に残っていれば，その部分に対する外部メモリアクセスは，冗長なものとして省略できる．

そこで本稿では，それぞれ異なる戦略で冗長なメモリアクセスを省略する手法として，**Delay 方式**と **Reuse 方式**の 2 つを提案する．シミュレーションを用いて，各方式のオンチップ記憶容量とバンド幅に対するオーバーヘッドを評価する．

2. パス型 ORAM

2.1 パス型 ORAM の構成

パス型 ORAM（Path ORAM）[6] は軽量な ORAM プロトコルであり，その実装のひとつである PHANTOM [7] とともに提案されている．パス型 ORAM の主要なデータ構造は，図 1(a) に示す **ORAM Tree**，**Stash**，**Position Map** の 3 つからなる．

ORAM Tree は，暗号化されたデータを保持するための

[1] 豊橋技術科学大学
Toyohashi University of Technology
[a)] fujieda@ee.tut.ac.jp
[b)] ichikawa@ieee.org

Vol.2016-ARC-219 No.16
Vol.2016-SLDM-175 No.16
Vol.2016-EMB-40 No.16
2016/3/24

二分木データ構造であり，外部メモリにマッピングされる．ただし，根に近い部分は頻繁にアクセスされることから，チップ上にキャッシュされる場合もある．これを Treetop Caching [7] とよぶ．実データの格納されるブロック数を N とすると，木の高さ L は概ね $\log_2 N$ と定められ，根をレベル 0，葉をレベル L と番号付けされる．各頂点は Z 個のブロックを保持するので，ダミーを含めた ORAM Tree の総ブロック数は $(2^{L+1}-1)Z$ 個となる．葉には 0 から 2^L-1 までの ID がつけられ，根から葉 x までの経路上に含まれる頂点を $\mathcal{P}(x)$ と表記する．経路上の頂点の集合，ないしそれらが保持するブロックの集合をパスとよぶ．詳細は 2.2 節で述べるが，パス型 ORAM において 1 回の ORAM アクセスは，1 つのパスへの読み出しと書き戻しを伴う．パスに含まれるブロック数は $(L+1)Z$ である．

Stash は，ORAM Tree から読み出され，復号された実データを保持するキャッシュであり，チップ上に搭載される．ORAM アクセスに先立って，Stash にはパス 1 つ分，すなわち $(L+1)Z$ 個の空きブロックが必要である．また，ORAM アクセスの際，書き戻せなかったブロックは Stash に残される．もし Stash に残されたブロックが多く，空きブロックが不足してしまうと，読み出しの際に実データを Stash に格納しきれなくなるおそれが生じる．この状況を破綻（Failure）とよぶ．Stash のブロック数（以下，Stash サイズとよぶ）は破綻の可能性を考慮して定める．具体的には，パラメータ λ を定め，破綻をきたす確率が $2^{-\lambda}$ を下回るように Stash サイズを定める．

Position Map は，実データの格納されるブロックそれぞれに対応する ORAM Tree の葉の ID を保持するテーブルである．その容量は NL ビットである．Position Map はチップ上に搭載されることを想定している．もしチップ上に搭載するには大きすぎる場合，Position Map 自体をもう 1 つのパス型 ORAM により保持する，再帰的なアプローチをとる．

2.2 パス型 ORAM の動作

パス型 ORAM における ORAM アクセスの手順は，以下の 5 つの手順による．ただし，外部メモリへのアクセス時はデータの暗号化・復号を行う．

(1) **オンチップの探索**．対象ブロックが既にチップ上の Stash または Treetop キャッシュに格納されているかをチェックする．もしそうならばチップ上のブロックにアクセスして終了する．この場合，外部メモリへのアクセスは生じない．

(2) **Position Map の読み出しと更新**．対象ブロックが属する葉の ID を取得し，0 から 2^L-1 までのランダムな番号で置き換える．取得した葉の番号を x とおく．

(3) **パスの読み出し**．$\mathcal{P}(x)$ に属する頂点が保持する全てのブロックを読み出し，ダミーでないブロックを Stash に移動する．

(4) **対象ブロックへのアクセス**．Stash に移動された対象ブロックにアクセスする．

(5) **パスの書き戻し**．$\mathcal{P}(x)$ に属する頂点に対し，葉から順に以下の処理を繰り返す．まず，Stash 内のブロックのうち，そのブロックの属するパスが対象の頂点を含むものを抽出する．もし Z 個以上のブロックが抽出されたら，そのうち Z 個を選び，対象の頂点に書き戻す．そうでなければ，抽出されたブロックを対象の頂点に書き戻し，不足分にはダミーブロックを書き込む．

パスの書き戻しの処理は，直感的には，読み出したブロックをなるべく葉に近い場所に書き戻す，とも表現できる．以上の手順の繰り返しを外部から観察すると，もはやランダムなパスへの読み書きを繰り返しているようにしか見えず，これによりアクセスパターンの隠蔽を達成する．

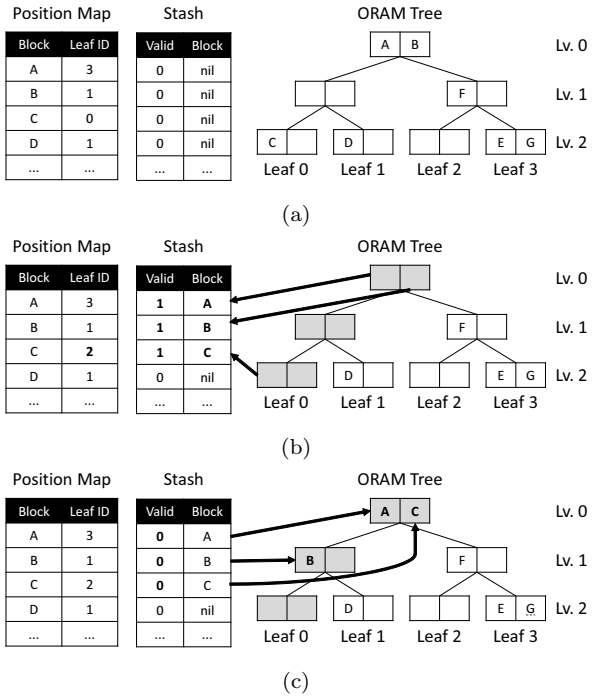


図 1 パス型 ORAM の構造と動作．

パス型 ORAM の動作例について，図 1 を用いて説明する．二分木の高さ L を 2，各頂点が持つブロック数 Z を 2 とする．実データのブロックを A，B，...，G とし，他はダミーブロックとする．実データのうち，A と B は根，C は葉（Leaf）0，D は葉 1 に置かれており，葉の ID はそれぞれ $3, 1, 0, 1$ が割当てられている．

ブロック C に対する ORAM アクセスが実行されたとし，このうち当該ブロックへのアクセスまでを完了したとする．このときの状態を図 1(b) に示す．ブロック C は Stash には存在しないので Position Map を読み出し，葉の ID として 0 を得る．Position Map の該当エントリはランダムな値（ここでは 2）で更新される．ブロック C はグレーで示す $\mathcal{P}(0)$ 上のどこかにあるので，このパスに含ま

Vol.2016-ARC-219 No.16
Vol.2016-SLDM-175 No.16
Vol.2016-EMB-40 No.16
2016/3/24

れるブロックを読み出す．その結果，ブロック A，B，C が Stash に移動され，ブロック C へのアクセスが可能になる．

つづいて，パスの書き戻しを完了したときの状態を図 1(c) に示す．まず，葉 0 を含むパスをもつ，すなわち対応する葉の ID が 0 であるブロックを Stash から探索する．そのようなブロックは存在しないので，葉 0 には 2 つのダミーブロックが書き込まれる．次に，レベル 1 の左の頂点は $\mathcal{P}(0)$ と $\mathcal{P}(1)$ に含まれるので，対応する葉の ID が 0 または 1 であるブロックを探索する．これには B が該当するので，この頂点には B とダミーブロックが書き込まれる．最後に，根は全てのパスに含まれるので，根には任意のブロックを書き込める．したがって，残る A と C は根に書き戻される．今回は全てのブロックが書き戻されたが，最終的に書き戻せないブロックがあれば，それらは Stash に残される．

パス型 ORAM のバンド幅オーバーヘッド，すなわち ORAM アクセス回数と外部メモリへアクセスするブロック数との比は，Treetop Caching を考慮せず，オンチップの探索に常に失敗するとすれば，$2(L+1)Z$ である．これは，1 回の ORAM アクセスにつき $(L+1)Z$ ブロックの読み出しと書き込みが発生するためである．

2.3 関連研究

パス型 ORAM の改善に関わる関連研究として，Ring ORAM [8] が挙げられる．通常のパス型 ORAM [6] では，ORAM Tree のある頂点が保持する Z 個のブロックが読み出し対象となった場合，それらは全て読み出される．Ring ORAM ではアルゴリズムの改良により，1 回の ORAM アクセスである頂点から読み出されるブロックを 1 つだけに制限する．これにより，バンド幅オーバーヘッドの削減を達成する．しかしながら，Ring ORAM は通常のパス型 ORAM と比べてダミーブロックをより多く必要する可能性があり，容量効率が低下しうる．この点について，実用上は影響が少ないと報告されているものの，その詳細については未報告である [8].

3. バンド幅オーバーヘッド削減手法

本稿では，パス型 ORAM のバンド幅オーバーヘッド削減手法として，Delay 方式と Reuse 方式の 2 つを提案する．いずれの方式もパス型 ORAM における冗長なメモリアクセスに注目するが，その削減のための戦略は両者で異なる．本節では，まず冗長なメモリアクセスについて実例を述べたあと，各手法の詳細について述べる．

3.1 パス型 ORAM における冗長なメモリアクセス

図 1 で示したブロック C へのアクセスに続いて，ブロック D にアクセスすることを考える．パスの読み出しにおいて，ブロック D の存在する $\mathcal{P}(1)$ 上の全てのブロックが読み出される．しかし，レベル 0，1 の頂点は先にアクセスされた $\mathcal{P}(0)$ にも含まれており，これらが保持するブロックは前回 Stash から書き戻されたものである．もしもこうしたデータを Stash に残しておくことができれば，パスへのアクセスの一部が省略できる可能性がある．また，同じ条件下で今度はブロック B にアクセスすることを考える．ブロック B は先のアクセスで Stash に読み出されている．この場合，Stash からデータを供給することができれば，そもそも外部メモリへのアクセス自体を省略できる．

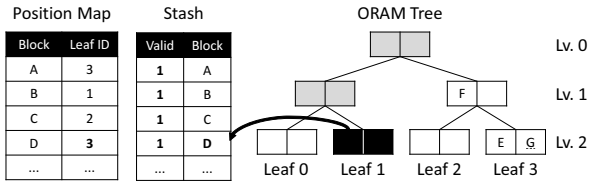


図 2 Delay 方式におけるパス読み出しの動作．

ここで，本稿で用いるいくつかの用語を定義する．2 つの連続する ORAM Tree へのアクセスに注目して，前回アクセスされたパスを先行パス，現にアクセスしようとしているパスを後続パスと定義する．また，先行パスと後続パスが共有する頂点の集合をパスの共通部，共通部を除いた頂点の集合をパスの独立部と定義する．

本研究で省略の対象としようとしているのは，パスの共通部へのアクセスである．パスの共通部を求めるためには，対応する葉の ID を 2 進法で最上位ビットから比較する．最上位ビットから連続して一致したビット数を k とおけば，レベル 0 から k までが共通部となり，レベル $k+1$ から L は独立部となる．

なお，パスの共通部へのアクセスを省略することで，パス型 ORAM によるアクセスパターンの秘匿性が崩れることはないことに注意されたい．これは，省略が ORAM Tree 上でアクセスされるパスの系列に対してのみ依存しており，ORAM アクセスの系列がパスの系列に変換された時点でアクセスパターンの隠蔽は完了しているためである．

3.2 Delay 方式

Delay 方式は，パスの書き戻しを ORAM Tree へのアクセス 1 回分だけ遅延させることで，最後にアクセスされたパスを常に Stash に残しておく方法である．このとき，パスの共通部に対してはそのアクセスを省略する．すなわち，1 回目の ORAM Tree へのアクセス時にはパスの書き戻しは行われず，2 回目以降のアクセス時には，後続パスの独立部のみを Stash に読み出し，先行パスの独立部のみを ORAM Tree に書き戻す．アクセス対象のブロック自体は書き戻しの対象外とする．

Delay 方式におけるパスへのアクセスの例を図 2 に示す．3.1 節と同様に，図 1(c) の状態からブロック D にアクセス

Vol.2016-ARC-219 No.16
Vol.2016-SLDM-175 No.16
Vol.2016-EMB-40 No.16
2016/3/24

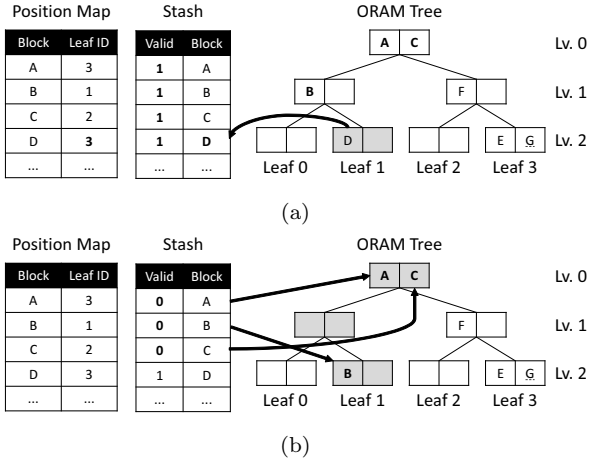


図 3 Reuse 方式におけるパスアクセスの動作.

する．先行パスは $\mathcal{P}(0)$ で，後続パスは $\mathcal{P}(1)$ であり，先行パスの独立部は葉 0 のみ，後続パスの独立部は葉 1 のみである．パスの読み出しでは，葉 1 に含まれるブロック D が読み出される．パスの書き戻しでは，葉 0 に書き戻せるブロックがないので，葉 0 に 2 つのダミーブロックを書き戻す．また，ブロック B へアクセスする場合は Stash からデータを供給できる．

Delay 方式において検討すべき事項として，Stash にパス 1 個分のブロックを残すことによる Stash のサイズ増加がある．悲観的な予測をすれば，後続パスの読み出しのために予約されている $(L+1)Z$ 個に加えて，先行パスを保持するための LZ 個（必ず共通部となる根を除く）のブロックの追加が必要である．しかし，より楽観的に，先行パスに含まれる実ブロック数の分布を考慮すれば，増加するブロック数は $L+1$ の定数倍にとどまるとも考えられる．本稿の 5 章では，シミュレーションを用いて Stash サイズの増加量を評価する．

3.3 Reuse 方式

Reuse 方式は，パスの書き戻しにより Stash 上で無効化されたブロックを，必要に応じて再度有効化する方法である．ブロックが書き戻された時点では Stash 上の有効ビットをネゲートされるだけで，データ自体は上書きされるまで保持されている．Reuse 方式ではブロックが ORAM Tree のどのレベルに書き戻されたかを記憶しておくことで，パスの共通部に含まれるブロックを有効化する．

Reuse 方式におけるオンチップの探索においては，共通部に書き戻されたブロックも探索する．ブロックが共通部で見つかれば一時的にそれを有効化する．もし当該の ORAM アクセスが読み出しであれば，データを供給してそのまま改めて無効化してもよい．書き込みであれば，データの一貫性を保つため有効化したままとする．これにより，アクセスパターンによっては必要な Stash サイズの増加の可能性がある．増加量は Delay 方式と比べれば軽微と考えられるが，その検討は今後の課題である．

パスの読み出しにおいては，独立部のブロックは通常通り ORAM Tree から読み出す一方，共通部のブロックは，ORAM Tree から読み出すかわりに Stash の該当エントリを再度有効化する．パスの書き戻しは，どこにどのブロックを書き戻したかを，何らかの追加ハードウェアで記憶するほかは，通常と同様である．

Reuse 方式におけるパスへのアクセスの例を図 3 に示す．想定する状況は図 2 と同じである．まず，ブロック D へアクセスする場合を考える．図 3(a) に示すパスの読み出しにおいては，共通部に含まれるブロック A, B, C に対する Stash エントリを再度有効化する．また，葉 1 を読み出し，ブロック D を Stash へ移動する．パスの書き戻しは通常のパス型 ORAM と同様であるので，図 3(b) のように葉 1 にブロック B，根にブロック A，C が書き戻される．また，ブロック B へアクセスする場合は，Stash 上のブロック B を再度有効化すればそこからデータを供給できる．

4. 評価

4.1 Delay 方式の容量オーバーヘッド

Delay 方式における Stash サイズの増加について評価を行う．パスの書き戻しを終えた後に Stash に残ったブロック数の分布を測定する．2.1 節から，ある Stash サイズ S に対する破綻確率は，Stash に残ったブロック数が $S' = S - (L+1)Z$ 個を超える確率と定義されるので，この分布から S' に対する破綻確率が得られる．それをもとに，破綻確率を $2^{-\lambda}$ とした時の λ の近似式を求める．いくつかの λ について近似式から必要な S' を求め，比較する．

評価には C++で記述したシミュレータを用いる．木の高さ L は 13，15，17，19 の 4 つを用いる．各頂点の保持するブロック数 Z を 4 とし，実ブロック数を 2^{L+1} とする．最も Stash に対する負担が大きい [6] アクセスパターンとして，全ブロックに対して順番にアクセスを繰り返すパターンを用いる．1 億アクセスをスキップした後の 25 億アクセスについて測定を行い，これを 8 回繰り返すことで，計 200 億アクセスを対象とする．評価対象は，通常のパス型 ORAM（Original）と，Delay 方式を追加したパス型 ORAM（Delay）である．

図 4，5 に，Stash に残されたブロック数の分布から求めた，Stash サイズと破綻確率との関係のプロットを示す．縦軸は破綻確率の対数をとり符号反転したもので，λ に対応する．横軸はパスの保持に必要な $(L+1)Z$ 個を除く Stash サイズで，S' に対応する．また，図 4 では $5 \leq S' \leq 25$，図 5 では $30 \leq S' \leq 60$ での結果から近似式を求めた．

通常のパス型 ORAM においては，先行研究 [6] と異なり，わずかに L が λ に影響を与えるという結果を得た．λ は S' についての 1 次式 $\lambda = (-0.00815L + 0.9317)S' + 7.203$ で近似できた．この式から $\lambda = 80, 128, 256$ における S' を

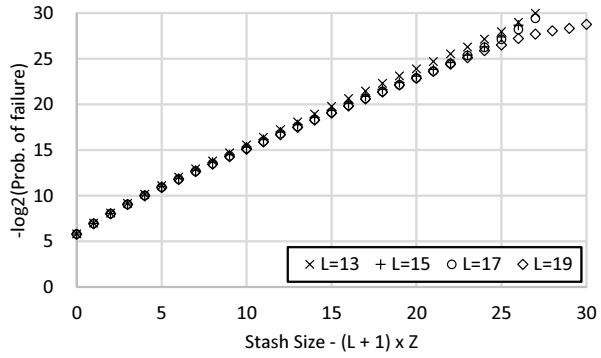


図 4　既存パス型 ORAM における Stash サイズと破綻確率.

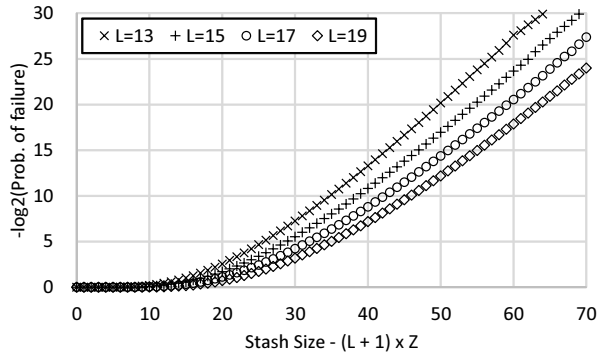


図 5　Delay 方式における Stash サイズと破綻確率.

表 1　Delay 方式におけるパラメータ λ の近似結果.

L	Approx. Formula
13	$\lambda = 0.0032S'^2 + 0.3935S' - 7.4829$
15	$\lambda = 0.0035S'^2 + 0.2907S' - 6.4346$
17	$\lambda = 0.0040S'^2 + 0.1931S' - 5.2035$
19	$\lambda = 0.0042S'^2 + 0.1125S' - 4.0737$

求めると，$L = 13$ としたときに先行研究 [6] とほぼ一致する結果が得られた．

Delay 方式においては，λ は表 1 に示す S' についての 2 次式により近似ができた．これは，ORAM Tree に書き戻せなかったブロック数の多寡が，Delay 方式により保持されるブロック数，すなわち最後にアクセスされたパスに含まれる実ブロック数の分布に異なる影響を与えていることを示唆している．

これらの近似式から，$\lambda = 32, 64, 96, 128$ における S' を求めた結果を表 2 に示す．小数部は切り上げている．必要な Stash サイズの増加量は，Original と Delay との差によって求められ，$\lambda = 32$ 前後で最大（35 ～ 48 個）をとる．Delay 方式での 2 次式での近似が正しければ，ここから徐々に差は減少していき，$\lambda = 150$ 前後で追い抜くことになる．しかしながら，実際にそのようになるとは考えにくいため，どこかで差の減少が頭打ちになることが考えられる．いずれにしても，Stash サイズの増加について考えるときは最大値についてのみ考えればよいので，Stash サイズの増加は高々 $2.5(L + 1)$ 個程度と結論づけられる．

表 2　いくつかの λ における，必要 Stash サイズの比較.

λ	Method	$L = 13$	$L = 15$	$L = 17$	$L = 19$
32	Original	31	31	32	32
	Delay	66	72	76	80
64	Original	69	71	72	74
	Delay	100	107	110	114
96	Original	108	110	112	115
	Delay	128	136	137	141
128	Original	147	150	153	156
	Delay	152	160	160	163

4.2　バンド幅オーバーヘッド

続いて，提案手法によるバンド幅オーバーヘッドの削減率を評価する．評価には，Memory Scheduling Championship [9] のワークロードとして公開されている，16 個のシングルスレッドプログラムに対するメモリアクセスのトレースを利用する．プログラムには PARSEC，SPEC CINT2006，BioBench の各ベンチマーク，および商用のプログラムを含む．メモリアクセスは，容量 512 KiB，ブロックあたり 64 バイトのキャッシュによりフィルタされている [10].

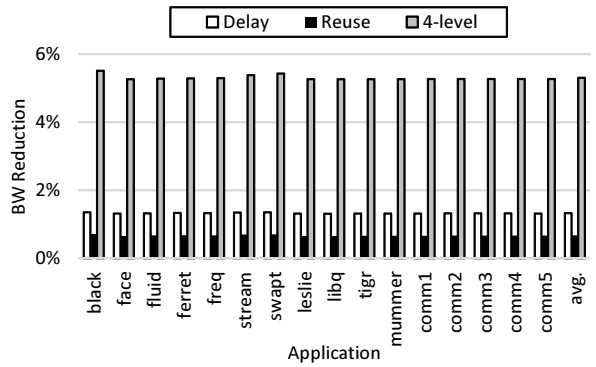
評価方法を以下に示す．パス型 ORAM はブロックサイズを 64 または 4096 バイトとし，実データが合計 256 MiB となるよう，64 バイトブロックでは $N = 2^{22}$，$L = 21$，$Z = 4$，4096 バイトブロックでは $N = 2^{17}$，$L = 16$，$Z = 4$ と定める．また，レベル 0 から 2 までの 3 段の Treetop Caching [7] を施す．各トレースの全体をシミュレートし，外部メモリのブロックがアクセスされた回数をトレースごとに評価する．評価対象は以下の 4 つである．

- Original: 通常のパス型 ORAM．評価結果はこれを基準とした相対値で示す．
- Delay: Delay 方式を追加したもの．
- Reuse: Reuse 方式を追加したもの．
- 4-level: Treetop Caching [7] をレベル 3 までの 4 段に変更したもの．オンチップに 32 ブロックの記憶容量が追加で必要であり，Delay 方式と同等の容量オーバーヘッドで比較するために用いる．

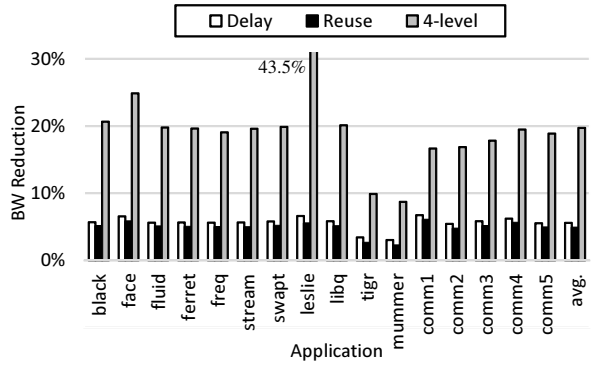
バンド幅オーバーヘッドの評価結果を図 6 に示す．(a), (b) はそれぞれブロックサイズを 64 バイト，4096 バイトとした場合の結果である．縦軸は Original と比較したバンド幅オーバーヘッドの削減率であり，横軸はトレース名である．ただし左端の avg. は算術平均である．

64 バイトブロックでは，キャッシュラインとブロックが 1 対 1 対応している．そのため，参照の局所性が活かせず，ほぼ全ての ORAM アクセスが外部メモリへのアクセスを必要とする．つまり，図 6(a) で得られたバンド幅オーバーヘッドの削減率は，アプリケーションにほとんど依存せず，1 回の ORAM アクセスでアクセスされるブロック数の期待値の減少とほぼ一致している．通常はレベル 3 か

Vol.2016-ARC-219 No.16
Vol.2016-SLDM-175 No.16
Vol.2016-EMB-40 No.16
2016/3/24



(a) 64 Bytes / Block



(b) 4096 Bytes / Block

図 6 Delay 方式および Reuse 方式のバンド幅オーバーヘッド削減率.

ら 21 の 19 レベル分にアクセスするが，Delay 方式ではこれが読み書きともに 18.75 レベル分に削減され，削減率は 1.3%であった．Reuse 方式では読み出しだけが 18.75 レベル分に削減され，削減率は 0.7%であった．4-level では読み書きともに 18 レベルに削減され，削減率は 5.3%であった．この条件下では，読み書きともに省略を行う Delay 方式が優勢である．しかしながら，オンチップ記憶容量の増加を考慮すると Delay 方式は 4-level に及ばない．また，読み出しのみしか削減できない Reuse 方式の効果は小さい．

一方，4096 バイトブロックでは，キャッシュラインとブロックが多対 1 対応している．そのため，参照の局所性があり，オンチップの探索に成功するケースが多い．バンド幅削減効果はアプリケーションに依存し，平均では Delay 方式で 5.6%，Reuse 方式で 4.9%となった．2 方式の差は 64 バイトブロックの場合とほぼ変わらないことから，双方に上乗せされた 4.2 ∼ 4.3%はオンチップの探索に成功する割合が増加したことに由来すると考えられる．この条件下では，Reuse 方式の書き込みを削減できないデメリットは相対的に小さくなり，オンチップ記憶容量の追加が少ないメリットが活かせると考えられる．

5. おわりに

本稿では，アクセスパターンの漏洩を防ぐ ORAM の軽量なプロトコルであるパス型 ORAM に対して，冗長なメモリアクセスの省略という観点から，Delay 方式と Reuse 方式の 2 つの手法を提案した．トレースによるシミュレーションを用いた評価では，ブロックサイズ 4096 バイトの場合，Delay 方式で平均 5.6%，Reuse 方式で 4.9%のバンド幅オーバーヘッド削減効果が確認された．

今後は，局所性の活用という観点から新たな手法を検討していく予定である．パス型 ORAM には，最近アクセスされたブロックほど根の近くに保持されやすい性質がある．図 6(b) では，既存手法でオンチップ記憶容量を増やした場合である 4-level において，19.7%のバンド幅オーバーヘッド削減効果が確認されており，この結果は，局所性を活かすことで，更なるオーバーヘッド削減が可能であることを示唆している．また，手法の更なる解析と，プログラムの実行性能の面での評価も今後の課題である．

謝辞

本研究の一部は JSPS 科研費 26870278 の支援による．

参考文献

[1] Henson, M. and Taylor, S.: Memory Encryption: A Survey of Existing Techniques, *ACM Comput. Surv.*, Vol. 46, No. 4, pp. 53:1–53:26 (2014).

[2] Thekkath, D. L. C., Mitchell, M., Lincoln, P., Boneh, D., Mitchell, J. and Horowitz, M.: Architectural support for copy and tamper resistant software, *Proc. of ASPLOS-IX*, pp. 168–177 (2000).

[3] Suh, G. E., Clarke, D., Gassend, B., van Dijk, M. and Devadas, S.: AEGIS: architecture for tamper-evident and tamper-resistant processing, *Proc. of ICS '03*, pp. 160–171 (2003).

[4] Islam, M. S., Kuzu, M. and Kantacioglu, M.: Access Pattern disclosure on Searchable Encryption: Ramification, Attack and Mitigation, *Proc. of NDSS '12* (2012).

[5] Goldreich, O.: Towards a Theory of Software Protection and Simulation by Oblivious RAMs, *Proc. of STOC '87*, pp. 182–194 (1987).

[6] Stefanov, E., van Dijk, M., Shi, E., Fletcher, C., Ren, L., Yu, X. and Devadas, S.: Path ORAM: An Extremely Simple Oblivious RAM Protocol, *Proc. of CCS '13*, pp. 299–310 (2013).

[7] Maas, M., Love, E., Stefanov, E., Tiwari, M., Shi, E., Asanovic, K., Kubiatowicz, J. and Song, D.: PHANTOM: Practical Oblivious Computation in a Secure Processor, *Proc. of CCS '13*, pp. 311–324 (2013).

[8] Ren, L., Fletcher, C., Kwon, A., Stefanov, E., Shi, E., van Dijk, M. and Devadas, S.: Constants Count: Practical Improvements to Oblivious RAM, *Proc. of USENIX Security '15*, pp. 415–430 (2015).

[9] The Journal of Instruction Level Parallerism: 3rd JILP Workshop on Computer Architecture Competitions (JWAC-3): Memory Scheduling Championship (MSC), (online), available from ⟨http://www.cs.utah.edu/ rajeev/jwac12/⟩ (accessed 2016-01-12).

[10] Chatterjee, N., Balasubramonian, R., Shevgoor, M., Pugsley, S. H., Udipi, A. N., Shafiee, A., Sudan, K., Awasthi, M. and Chishti, Z.: USIMM: the Utah SImulated Memory Module, Technical Report UUCS-12-002, University of Utah (2012).