Hiding Decryption Latency in Intel SGX using Metadata Prediction

Daulet Talapkaliyev

Thesis submitted to the Faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of

Master of Science
in
Computer Engineering

Xun Jian, Co-chair
Cameron D. Patterson, Co-chair
Paul E. Plassmann

December 12, 2019
Blacksburg, Virginia

Keywords: High-performance Architecture, Intel SGX, Encryption, Metadata Prediction

Hiding Decryption Latency in Intel SGX using Metadata Prediction

Daulet Talapkaliyev

(ABSTRACT)

Hardware-Assisted Trusted Execution Environment technologies have become a crucial component in providing security for cloud-based computing. One of such hardware-assisted countermeasures is Intel Software Guard Extension (SGX). Using additional dedicated hardware and a new set of CPU instructions, SGX is able to provide isolated execution of code within trusted hardware containers called enclaves. By utilizing private encrypted memory and various integrity authentication mechanisms, it can provide confidentiality and integrity guarantees to protected data. In spite of dedicated hardware, these extra layers of security add a significant performance overhead. Decryption of data using secret OTPs, which are generated by modified Counter Mode Encryption AES blocks, results in a significant latency overhead that contributes to the overall SGX performance loss. This thesis introduces a metadata prediction extension to SGX based on local metadata releveling and prediction mechanisms. Correct prediction of metadata allows to speculatively precompute OTPs, which can be immediately used in decryption of incoming ciphertext data. This hides a significant part of decryption latency and results in faster SGX performance without any changes to the original SGX security guarantees.

Hiding Decryption Latency in Intel SGX using Metadata Prediction

Daulet Talapkaliyev

(GENERAL AUDIENCE ABSTRACT)

With the exponential growth of cloud computing, where critical data processing is happening on third-party computer systems, it is important to ensure data confidentiality and integrity against third-party access. Sometimes that may include not only external attackers, but also insiders, like cloud computing providers themselves. While software isolation using Virtual Machines is the most common method of achieving runtime security in cloud computing, numerous shortcomings of software-only countermeasures force companies to demand extra layers of security. Recently adopted general purpose hardware-assisted technology like Intel Software Guard Extension (SGX) add that extra layer of security at the significant performance overhead. One of the major contributors to the SGX performance overhead is data decryption latency. This work proposes a novel algorithm to speculatively predict metadata that is used during decryption. This allows the processor to hide a significant portion of decryption latency, thus improving the overall performance of Intel SGX without compromising security.

Dedication

To my family

Acknowledgments

I am extremely grateful to my advisor, Dr. Xun Jian, for his mentorship and support. I would also like to thank Dr. Cameron D. Patterson and Dr. Paul E. Plassmann for being a part of my advisory committee. Thank you to all members of HEAP Lab for working with me. Lastly, thank you to all my friends and family for providing me with all the support I need.

Contents

List of Figures

List of Tables

List of Abbreviations

CL Cache Line or Cache Block

EPC Enclave Page Cache

JIT Just-in-Time

LLC Last Level Cache

MC Memory Controller

MEE Memory Encryption Engine

OTP One Time Pad

PCT Program Counter Table

PQ Prediction Queue

PRM Processor Reserved Memory

PRMRR PRM Range Register

RQ Relevel Queue

SGX Intel Software eXtension

TEE Trusted Execution Environment

Chapter 1

Introduction

Today, cloud computing is the predominant way to store and process enormous amounts of data. The ubiquitous nature of using third-party computer resources without direct active management makes cloud computing extremely appealing for diverse applications and end users [22, 25]. With exponential growth of outsourced cloud computing, even processing of sensitive data becomes cheaper and more convenient on externally available cloud services. However, this raises the concerns of security and privacy. With data on machines that are out of users control, protecting data confidentiality and integrity against third-party access becomes even more critical. This includes both outsiders, like external malicious attackers, and insiders in the form of cloud computing providers. Numerous countermeasures have been proposed to ensure the security of computing systems. Establishing trusted execution environment through Virtual Machine isolation is the most common method for achieving a runtime security. However, with the complexity and severity of modern attacks, pure software-based countermeasures are simply not enough, particularly against attacks where the adversary has a physical access to the device. Thus, to provide such a closed computing system that ensures data processing in a trustworthy manner, hardware support is fundamental. Hardware-Assisted TEE, which reduces the size of the trusted computing base and minimizes attack surfaces, allows us to mitigate many limitations of software-only solutions [11, 19, 23]. One of the examples of general-purpose hardware-assisted security technology is Intel's hardware-based isolation and memory encryption – Software Guard Extensions.

1.1 Motivation

Security guarantees, like data confidentiality, integrity, tamper resistance and replay prevention come with a significant performance overhead. Even with dedicated hardware and tailored cryptographic primitives, Intel SGX can suffer a significant slowdown, especially for memory bandwidth heavy applications [21, 41]. Encryption of commodity memory relies on security metadata, specifically on version counters in AES Counter Mode of SGX. For every fetched data cache line (ciphertext) its counter is also fetched. This means that the secret One Time Pad, which is required for decryption, cannot be generated before the counter arrives from the memory. With a relatively high latency of 100 cycles [30], pad generation is the major part of decryption latency. To improve the performance, SGX uses a small on-chip counter cache, which helps to cache versions. Caching the versions allows encryption engine to precompute the pad before the data arrives, thus hide a significant portion of the decryption latency. However, given a small size of counter cache and the fact that it is shared between encryption versions and integrity tree counters, it can be completely ineffective for cache-sensitive applications [40]. Frequent autonomous integrity tree traversals required in integrity checking alone can significantly increase miss-rates for counter cache, which leads to cache pollution and therefore performance degradation. With the above concerns in mind, the goals of this thesis are as follows:

- Hide decryption latency by predicting the versions
- Minimize any overhead caused by new design (e.g. memory bandwidth, storage)
- No modifications to any existing Memory Encryption Engine components
- Maintain the original SGX security guarantees - confidentiality, integrity, replay protection

1.2 Contribution

To hide decryption latency of cache-sensitive applications and improve the overall performance of SGX, metadata prediction extension to existing Memory Encryption Engine is proposed. Metadata prediction extension relies on version releveling and prediction to speculatively precompute correct decryption OTP and thus, reduce performance degradation due to considerable decryption latency. Chapter 2 provides background information on hardware side of Intel SGX. Metadata prediction extension is described in Chapter 3, followed by prediction accuracy evaluation using Intel Pin tool functional simulation in Chapter 4. Chapter 5 discusses related work. Lastly, conclusions and future work are presented in Chapters 6.

Chapter 2

Background

2.1 Intel SGX

Intel Software Guard Extensions introduce a set of CPU instructions that enable hardware-assisted isolation through enclaves. Enclaves are execution environments that guarantee integrity, and confidentiality to trusted programs running inside them [17, 18]. It minimizes the TCB by only trusting the CPU internals and isolating enclaves from all other processes on the system using hardware-based access control mechanism. This ensures that all data running inside a certain enclave is completely out of reach to any potentially malicious processes, like compromised drivers, operating system, hypervisor, and even other enclaves running on the same system [10, 26, 32]. This work examines the low-level hardware architecture of Intel SGX; thus, background information focuses solely on hardware primitives and associated hardware-based control mechanisms.

2.1.1 Processor Reserved Memory

SGX reserves an isolated trusted memory region in DRAM of limited size called Processor Reserved Memory (PRM). PRM is a union of three primary regions: Enclave Page Cache (EPC), MEE integrity tree and metadata. EPC stores encrypted protected data and code at 4KB enclave page granularity, which are assigned by CPU at enclave initialization. Each page is assigned to only one enclave, meaning that only the owner enclave can operate on that page while being completely protected from the rest of the enclaves and software, no matter what privilege level they have. The original SGX design offers 128MB PRM region where 96MB is protected user-available data, 24MB is metadata, while the rest is split between external levels of the integrity tree (excluding a root) and the reserved gaps. Reserved gaps are used to create natural alignments of all PRM sub-regions making it easier for hardware implementation of sub address and field derivation [23, 24]. All data and counter blocks are at the granularity of 512-bit, called cache lines (CL hereafter).

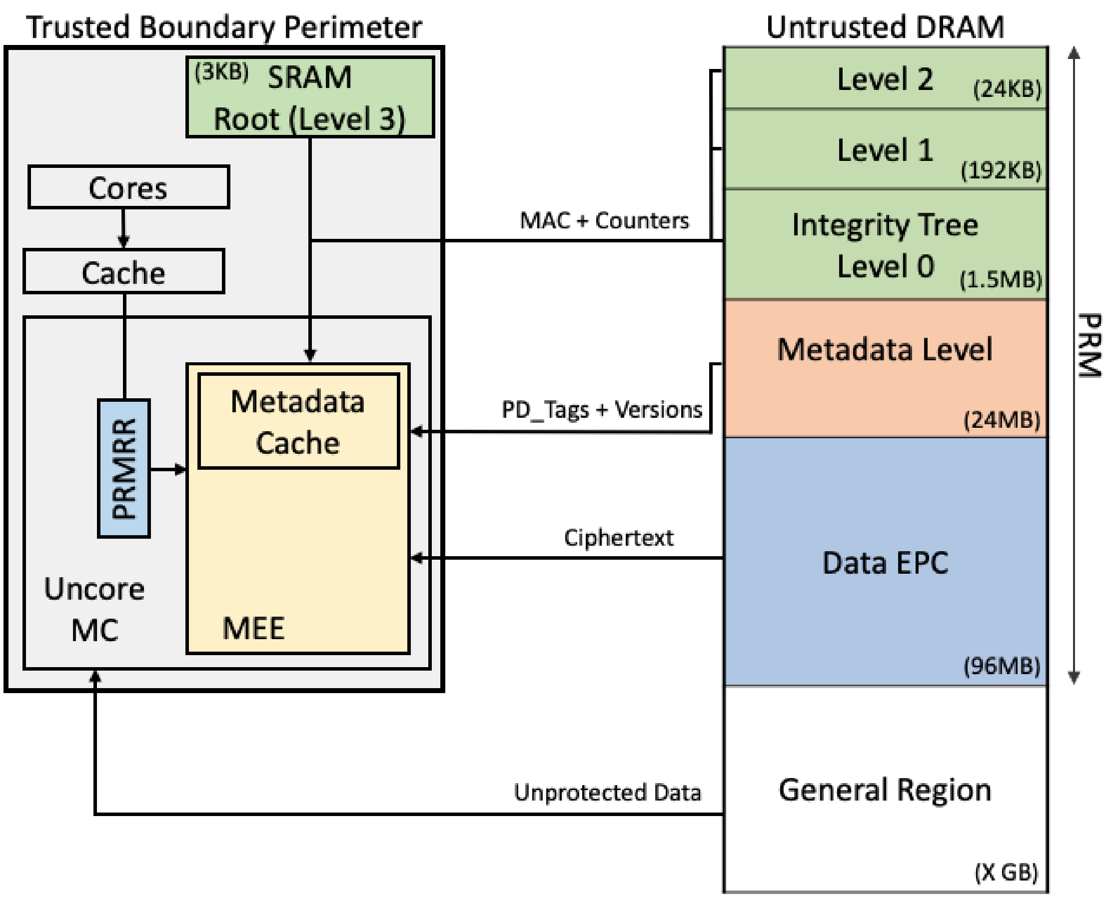

Figure 2.1: Schematic illustration of the MEE operation.

2.1.2 Memory Encryption Engine

Memory Encryption Engine (MEE) is an autonomous hardware that is used to protect EPC against physical attacks. MEE is added to processor's uncore and operates as an extension to memory controller (MC), as it takes over all protected memory traffic between LLC and PRM. Figure 2.1 shows a schematic illustration of MEE operation and PRM organization. All read/write requests to EPC data are routed to MEE, which decrypts data when it is fetched from the DRAM and encrypts when it is written back. MEE does that using a tailored 128-bit AES CTR encryption [23]. Moreover, to provide data integrity, all memory transactions to and from PRM invoke autonomous integrity verification checks. This is done using a modified Merkle Tree of stateful Message Authentication Code (MAC) tags and counters. With the help of these customized cryptographic primitives MEE is able to provide confidentiality, integrity and freshness guarantees to protected data.

2.1.3 Metadata Level and Data Encryption with Versions

Version counters are dedicated per-unit counters that are used as a temporal coordinate in the MEE encryption scheme. To distinguish between version counters in metadata level and counters in the integrity tree, version counters are commonly referred as versions. Simply speaking, versions are write counters, which keep the number of times each data CL was written back to memory. Thus, temporal coordinate ensures that with every new write to memory data CL will always be encrypted to a different ciphertext, even when the plaintext remains the same. Together with data address, which is used as a spatial coordinate, MEE is able to utilize non-repeating location-time based compound nonce for encryption. Spatial coordinate guarantees that two identical plaintext data CLs with different physical addresses will never be encrypted to the same ciphertext. To verify the integrity of data, MEE uses

MAC algorithm over ciphertext and compound nonce to produce short MAC tag called PD_Tag. Every data CL in the protected EPC region has its own PD_Tag. Version counters and PD_Tags are stored in a special PRM region called Metadata.

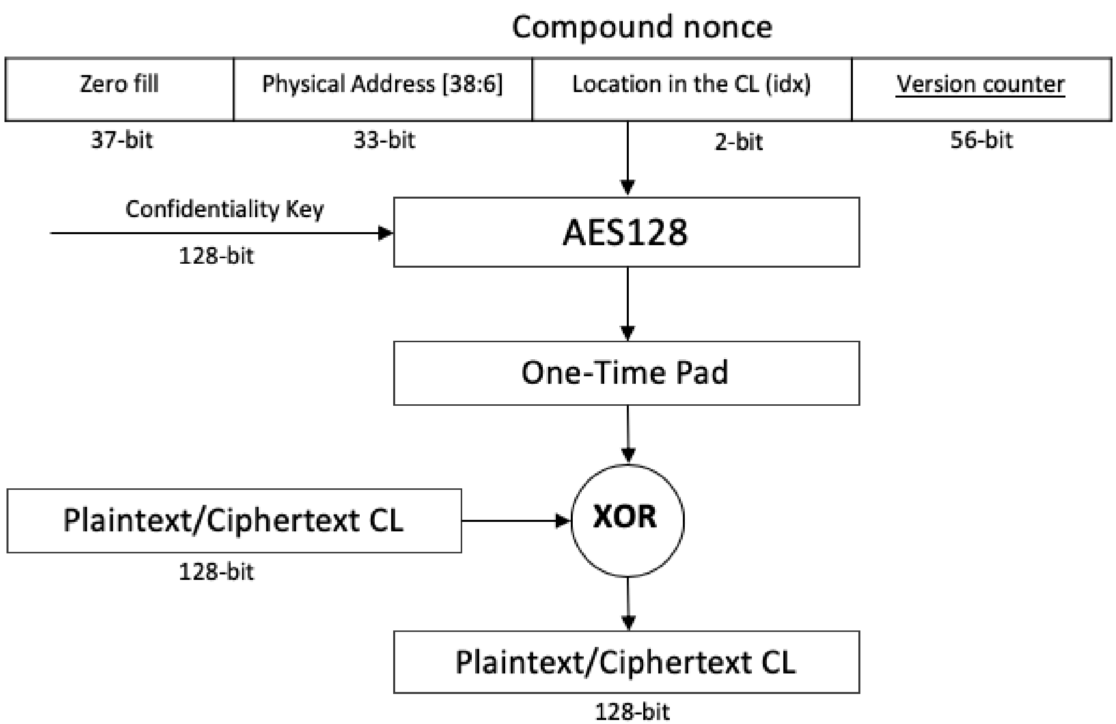

Figure 2.2: Cryptographic primitives used in Counter Mode AES Encryption in SGX MEE.

Encryption over the entire EPC region provides confidentiality guarantee that prevents the adversary from reading protected plaintext data. To encrypt/decrypt the plaintext/ciphertext, data is XORed with a freshly generated secret bit-string called One Time Pad (Figure 2.2). To generate OTP, MEE uses AES, which takes a compound nonce and MEE confidentiality key (generated by MEE at initialization) as inputs [45]. As mentioned previously, non-repeating compound nonces are extremely important to generate a unique OTP. While the spatial coordinate of compound nonce is entirely self-regulated, MEE requires a hardware-assisted mechanism to control temporal coordinates. Temporal nature of non-repeating versions guarantees a non-repeating compound nonce and thus a non-repeating

OTP for the same data address. MEE does that by immediately incrementing the version when the data CL is written back to memory.

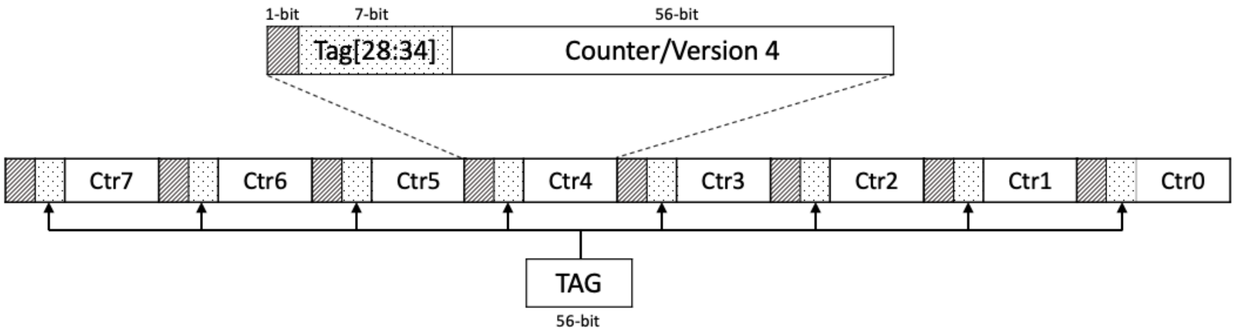

Figure 2.3: Counter block layout in Intel SGX.

2.1.4 Integrity Tree

MEE integrity tree is a tailored Merkle Tree [12] (Bonsai-style tree) that covers the versions. Default SGX design has a 4-level tree with 8-ary layout, where each level ensures the integrity of the level below, and level 0 covers the version counters at the metadata level. 8-ary layout refers to 8 counters per counter block, and thus 8x compression ratio per level. The organization of each tree and version counter block is defined as 512 bits block with 8 56-bit counters, 1 56-bit MAC tag and 8 unused gap bits (Figure 2.3). With the Bonsai-style tree, each counter block at level 0 covers 8 version blocks of metadata level, each counter at level 1 covers 8 counter blocks of level 0 and so on, including the trusted on-die root level 3 [37]. Since all levels of integrity tree and metadata level are dependent on each other, increment of a version due to data writeback will result in autonomous increments of all parent counters at a higher level (propagated all the way to root level).

In order to authenticate a counter block MEE uses Carter-Wegman MAC algorithm [23]. It generates a 56-bit MAC tag for a counter block by applying the algorithm over all counters in that block (as one entity), as well as its parent counter. For example, to compute a MAC tag for a counter block at level 0 the MAC algorithm is applied over all 8 counters within that block and the parent counter from counter block at level 1. Thus, during authentication, MEE recomputes the MAC tags and compares it with the expected value, fetched from the integrity tree. In case the comparison results in mismatch, MEE raises a failure flag, immediately terminates the transaction, and locks the MC, which requires system re-boot to re-generate all MEE keys. Even if the adversary can modify data, versions, counters and MAC tags at all external levels using various physical attacks, counters at on-die root level are out of reach. Therefore, by only trusting the internally stored root level, integrity tree achieves tamper resistance with replay prevention [37].

2.1.5 MEE Counter Cache or Metadata Cache

Fetching the version block together with the traversal of the integrity tree for data authentication, which also includes access to the on-die SRAM root, results in six memory accesses. Write back with updates to the integrity tree requires even more memory accesses. To reduce such performance overhead, MEE has a small internal counter cache, which is dedicated to storing version counters and integrity tree counters. This helps minimize the DRAM access overhead, as the MEE is designed to stop the integrity tree traversal once there is a counter cache hit. For example, a read operation that resulted in cache hit of the version block requires MEE to fetch only data CL and PD_Tag block. Moreover, if the counter is cached, MEE is able to pre-compute the OTP in parallel with data fetch, which hides the decryption latency in memory access latency. While MEE counter cache does not store PD_Tags or any other MACs, it is often referred as metadata cache.

Chapter 3

Metadata Prediction

3.1 Predictions

Prediction of version attempts to achieve the same effect as metadata cache. By successfully predicting the versions, MEE can speculatively precompute OTP used in data decryption. Correct version prediction results in correct pad, which can be immediately used to decrypt newly arrived ciphertext CL, thus hiding a significant part of decryption latency. Therefore, prediction accuracy has a direct impact on decryption overhead and the overall performance of SGX.

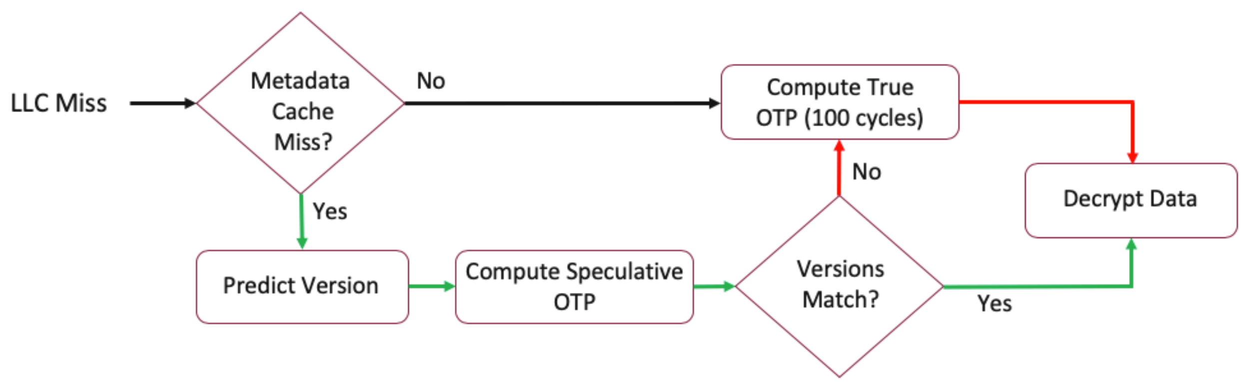

Figure 3.1: Flow chart of version prediction algorithm

As shown in Figure 3.1, the entire process starts with data cache miss – data CL was not found in last level cache, which means it must be fetched from the external memory (DRAM). Since data coming from DRAM is encrypted, MEE will require its version to generate OTP and decrypt it when it arrives. The required version block is also stored in external memory and must be fetched together with data. However, with a metadata cache, there is a chance that the required version block was cached and is already residing on chip. To check that, MEE searches metadata cache. In case of version hit, MEE does not need to fetch the version block from memory and can begin computing OTP right away. Once the data CL arrives, MEE can decrypt it and cache it in LLC. On the other hand, when metadata cache experience a miss, a prediction can be attempted. Predicting a version will allow MEE to compute a speculative OTP. However, since the correctness of prediction is not guaranteed, there is a chance that the attempted prediction is incorrect. To check this, MME must still fetch the actual (”true”) version from the memory. At the arrival, MEE must compare predicted version with the actual version. If two versions mismatch, speculative OTP must be discarded, so MEE can compute true OTP with the actual version and incure the full decryption latency of 100 cycles [30]. If they match, the speculative OTP can be considered correct and therefore, used to successfully decrypt the incoming data without recomputing the OTP. That means that correct version prediction allows MEE to hide the entire OTP generation latency in memory access latency, thus improving the overall performance of fetch and decrypt operations.

3.2 Releveling

The act of setting multiple versions to the same value is called releveling. Constantly releveling all versions to the same exact global value allows to accurately predict every fetched version. Since versions cannot be set to anything below the current value, global value to which all versions are releveled will then be determined by the highest version. The issue with such global approach stems from the natural update of versions, where a version must be incremented whenever data cache line is written back to external memory. Write-heavy data CLs will then have high versions that are frequently updated. With every update of the highest version, the global releveling value will be changed and all versions must be releveled again. This results in huge overhead. Moreover, write-read disparity of data CLs results in uneven version distribution. If global releveling value is based on the version of write-heavy CL, stable read-heavy blocks suffer from unnecessarily frequent and aggressive releveling. Thus, brute-force global releveling of all versions is not viable.

3.3 PC based groups

Using write-read ratio as unique characteristic, data blocks can be divided into separate groups. Moreover, memory blocks accessed by the same static instruction (PC) tend to exhibit the same write-read ratio. Using this information, PC can become the determining characteristic to divide all data blocks into groups where releveling and prediction can be done locally. This eliminates the issue of uneven version distribution, as blocks within the same group will have similar write-read ratio. Aggressive releveling of write-heavy blocks now does not affect stable read-heavy blocks, since they belong to completely separate groups. Another advantage of PC-based grouping is minimal information required to identify the group affiliation. Only PC that is responsible for that access is needed to determine which group a certain fetched data block belongs to.

3.4 Metadata Prediction Extension

Metadata Prediction Extension is a hardware extension to MEE that performs releveling and prediction of versions that are not covered by metadata cache. It relies on PC based groups, where both stages are performed locally within each group. To perform local relevelings and predictions metadata prediction extension requires the following components (Figure 3.2):

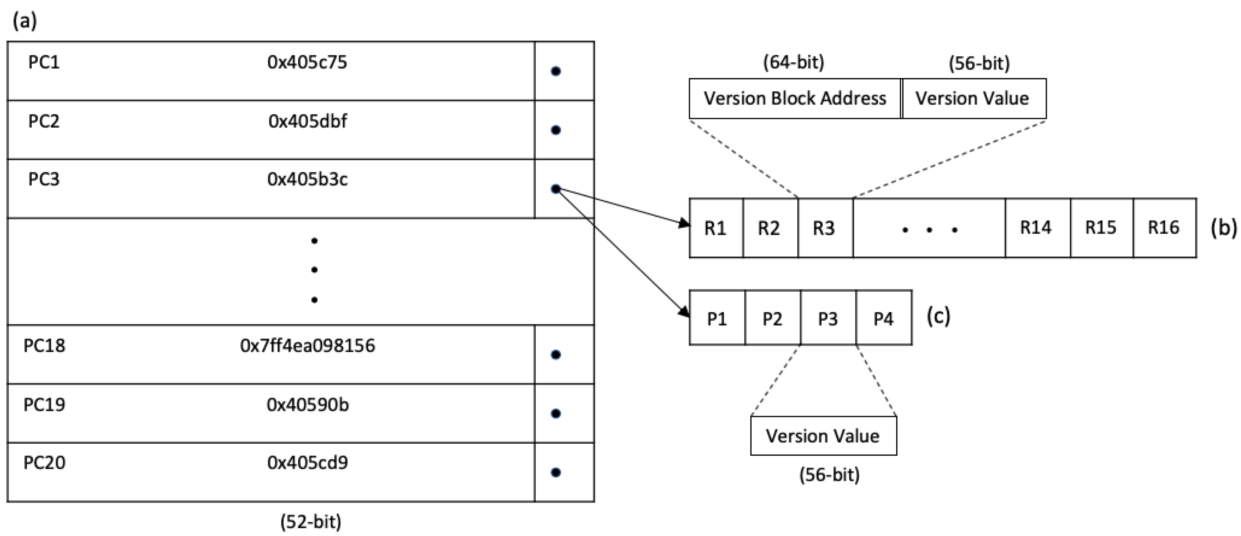

Figure 3.2: Metadata prediction extension organization: (a) Program Counter Table, (b) Relevel Queue and (c) Prediction Queue.

- Program Counter Table (PCT) is a fixed size priority queue of most frequently used unique PCs. Each of the unique PCs in PCT is recognized as a separate group, where all fetched data cachelines associated with a certain PC (brought to data cache by that PC) will belong to that PC's group. Both prediction and releveling stages are performed within those groups.

- Relevel Queue (RQ) is a fixed size FIFO queue of 16 entries, where each entry stores the physical addresses of fetched version blocks. Each PC group from PCT has a its own unique RQ.

- Prediction Queue (PQ) is a 4-entry fixed size FIFO queue that stores the versions of the last 4 recently fetched data cachelines. Similarly, each PC group from PCT has its own unique PQ.

Before any version can be releveled and predicted, MEE must identify most frequently used PCs in the application. This can be done during a simple PC learning warm-up, where top PCs are identified and stored in PCT. Depending on the length and the nature of the running application, MEE can repeat PC learning warm-up at a regular interval in order to keep PCT up to date. Figure 3.3 shows that 100 most frequent PCs are responsible for more than 95% of all memory traffic. To minimize storage overhead, PCT size can be limited to top 20 unique PCs, which captures almost 80% of memory traffic. Every PC that is responsible for a data cache miss, will be checked against PCT. If found, the PC and data CL associated with it can participate in the metadata prediction algorithm. Otherwise, both are ignored, as their low frequency rate does not justify any overhead from prediction or releveling.

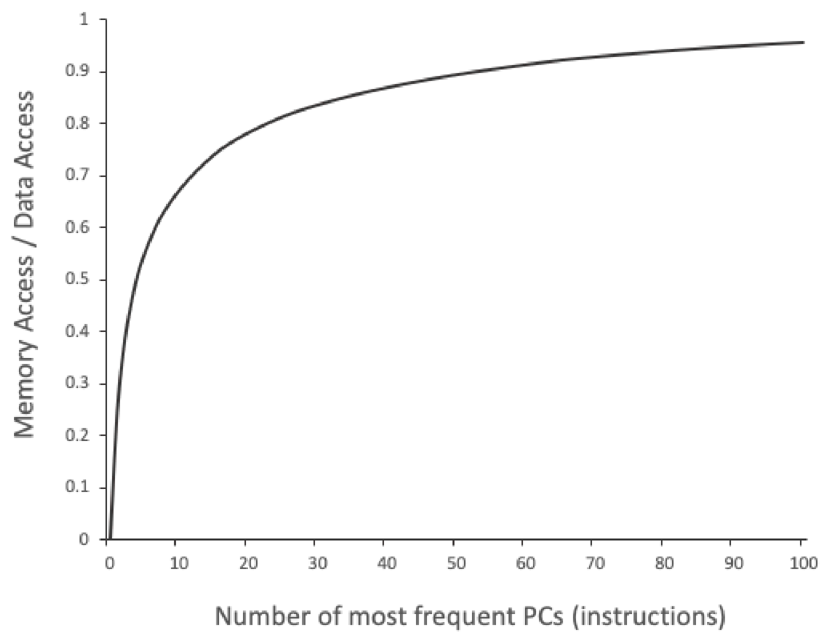

Figure 3.3: Number of most frequent unique PCs vs. Total memory traffic.

3.4.1 Prediction Stage

An incoming data block must be decrypted with an OTP decryption scheme before it can be cached. Predicting the version allows MEE to speculatively precompute the OTP before the data and its version blocks arrive from the memory. Version prediction is done using the local group Prediction Queue. If the local PQ is empty, version prediction is unfeasible – OTP will be computed with actual version from the newly arrived version block. On the other hand, if the local PQ is full with 4 most recent version entries, the prediction algorithm can predict a version to precompute a speculative OTP for the incoming data CL. This is done by identifying the most recent R and the most frequent F entries within the local PQ. If the most recent and most frequent versions are not the same, MEE can use both to precompute two separate speculative OTPs. Correct prediction means that the precomputed OTP can be safely used to decrypt the arrived data block and hide the decryption latency. In the case of incorrect prediction (versions mismatch), OTP must be recomputed using the actual version and thus incur the full decryption latency. Once data and version blocks are cached, the version counter is pushed into local PQ becoming the most recent entry. Versions of data CLs with very few writes and low write-read ratios can be accurately predicted with almost no releveling, given that the versions of such blocks remain very stable. However, high write-read ratio of data CLs result in highly volatile versions. This means the version predictions of such unstable blocks are highly ineffective without frequent aggressive releveling.

3.4.2 Releveling Stage

Releveling is the process of equalizing the versions of multiple data CLs that belong to the same local group, which can then be effectively predicted during the prediction stage. Just like the prediction stage, releveling stage relies on local groups based on most frequent PCs in PC Table. Since each of such PC has its own RQ, the algorithm performs local releveling on the blocks associated with a specific PC. The releveling itself is a relatively straightforward process. Once the incoming data CL and its version block are cached, counter and version block address will be pushed to a local RQ as a new entry. When the RQ is full and the local group of 16 entries is formed, the releveling stage begins. First, the highest counter within the group is found to be used as a relevel value for the entire group. As mentioned previously, due to security constraints of counters in SGX, the version can never be reused and so releveling can never be done to a lower value – chosen relevel value must always be higher or equal to the current version of any block within the local group. One by one the entries are popped from the RQ and the corresponding versions are releveled. Due to the relatively small size of RQs, corresponding data CLs still remain in cache during the releveling stage. In the rare case when the data is evicted from LLC before the local releveling is complete, the releveling for that block is simply skipped, so as to avoid extra memory bandwidth overhead. Once the RQ is empty and all versions in that group are releveled, the cycle restarts, as RQ is repopulated with a new local group of 16 new entries.

3.4.3 Memory Bandwidth Overhead Control

Unlike clean evicts, dirty data CLs, which when evicted are written to memory, require update of versions and all associated integrity counters. After the version is incremented, the data CL must be encrypted using this new version before it is written back to DRAM. Moreover, all updated integrity tree counter blocks become dirty and will generate additional memory traffic when evicted from metadata cache.

Since releveling explicitly updates the versions in spite of data block dirty status, it causes memory bandwidth overhead. However, this overhead only stems from releveling clean data blocks. Because dirty-evicts from on-chip cache already require version increment, releveling of dirty blocks does not generate any additional traffic, as it efficiently takes advantage of both data and counters dirty status. On the other hand, releveling of clean data blocks makes them dirty. Making all clean evicts dirty can cause a substantial memory traffic overhead. Releveling stage includes two mechanisms that allow the system to control the memory bandwidth overhead.

The first control mechanism is fairly straightforward and relies entirely on prediction accuracy of versions in a local group. Every RQ is fitted with a 4-bit counter that keeps track of local prediction accuracy. When the fetched data CL and its version block arrive, actual and predicted versions are compared. Correct prediction increments that counter. Once RQ is filled with 16 new entries, the prediction accuracy counter is checked. If all 16 versions were correctly predicted before they were added to RQ, counter value should be 16. 100% prediction accuracy means that releveling of this local RQ can be skipped, as those versions will most likely be correctly predicted when they are fetched again next time. This is especially true for read-heavy applications, where stable read-heavy blocks only require occasional releveling to maintain high prediction accuracy. Threshold parameter for prediction accuracy counter controls the aggressiveness level of the above mechanism.

Second bandwidth control mechanism is based on explicit limitation of overhead memory traffic. It sets a pre-defined periodic budget for all memory traffic caused by releveling. When overhead traffic exceeds the budget, MEE can limit or completely disable any future relevels until the next period. The period can be based on either CPU time or number of memory instructions. At the begging of a new period the budget is replenished in full. The budget value can be dynamically adjusted depending on how memory bandwidth heavy the application is. Moreover, if necessary, unused budget can be carried over into the next period. One possible scenario where carryover budget can be extremely beneficial is write-heavy period followed by read-heavy period. Releveling during write-heavy period should not result in budget depletion since most releveling is done on already dirty blocks. Adding that extra carryover budget to read-heavy period budget allows more clean blocks to enjoy releveling and thus, future accurate predictions.

3.4.4 Decryption Overhead Control

In order to generate OTP for encryption and decryption, MEE uses custom AES units. This includes OTPs that were speculatively generated during the prediction stage. Correct prediction of the version results in a correct speculative OTP, which can be successfully used to decrypt the incoming data CL. However, any incorrect prediction means an incorrect and discarded speculative OTP, as the correct OTP has to be recomputed with the actual version, when the version block arrives from the memory. Therefore, lower prediction accuracy causes higher decryption overhead. To partially minimize this overhead, prediction stage can limit the number of predicted versions and thus limit the number of speculatively generated OTPs. As mentioned previously, current implementation of prediction stage uses the most frequent F and the most recent R version entries from local PQ. Moreover, to increase the probability of correct predictions, prediction algorithm can extend the number of predictions to 4/6/8 by using R+1/2/3 and/or F+1/2/3 to generate even more speculative OTPs. If both versions are different, N OTPs are precomputed, which can potentially result in Nx overhead when all predictions are incorrect. Using a simple throttling mechanism with predefined budget, the system can regulate the number of attempted predictions and, thus minimize the number of speculatively generated OTPs. In the worst-case scenario, when regular AES operations are critical and must be prioritized, prediction algorithm can cut all attempted predictions down to 1 and use only F or R for speculative pads.

Chapter 4

Experimental Evaluation

4.1 Simulation Framework

Intel Pin was used to simulate and evaluate the performance of metadata prediction extension. Pin is a dynamic binary instrumentation tool that allows engineers to create dynamic program analysis tools (Pin tools) [31, 36]. It utilizes just-in-time (JIT) compiler to insert code, thus performing efficient instrumentation on compiled binary file at run time without any need for source code re-compilation. High-level software simulation model was implemented using C/C++ based Pin instrumentation tool [6, 7, 20]. Simulation model includes two-level hierarchical data cache model, Intel SGX MEE components (MEE cache, integrity tree), simplified model of PRM and prediction algorithm extension. Table 4.1 shows system configurations used in Pin simulation.

Table 4.1: Simulated System Configuration

Number of cores	1
Last Level Cache (L2) Metadata (MEE) Cache	2MB, 8-Way, 64B CLs 64KB, 8-Way, 64B CLs
Counters per Counter Block	8
PC Table Size	20
Relevel Queue Size Prediction Queue Size	16 4

To evaluate the metadata prediction performance, numerous workloads from SPEC2006 [8], SPEC2016 [9], NASA [4], GAP [1] and PARSEC [5] benchmark suites were used. In order to skip the initialization phase in a workload and start useful instrumentation at the computing phase, our Pintool fast-forwards (skips) N billion instructions, which was experimentally found for each benchmark. For the actual simulation, warm-up of 25 billion memory instructions are used before evaluating performance over 5 billion memory instructions. Warm-up phase of the simulation is divided into two warm-up stages – PC learning and releveling. PC learning stage is the first 5 billion memory instructions of the warm-up phase, which is used to identify top most frequent workload PCs. Releveling stage is the next 20 billion memory instructions, used to perform only the releveling stage of the prediction algorithm. This helps relevel and stabilize as many version blocks as possible, before evaluation stage begins. At the start of the simulation all versions were initialized with a random value between 1 and 2^{14}.

4.2 Prediction Accuracy with No Overhead Control

Prediction accuracy of versions is the main statistic used to experimentally estimate the performance of metadata prediction extensions in high-level Pin simulation. Higher prediction accuracy means higher number of correctly predicted versions, which in turn result in higher number of correctly precomputed OTPs. Correct speculative OTPs allows MEE to immediately decrypt arrived data, thus hide the decryption latency in memory access latency and improve the overall performance of SGX.

First set of experiments was done with all overhead control mechanisms disabled. This allows both releveling and prediction stages of metadata prediction extension to operate without any limitations. With no memory bandwidth overhead control, releveling stage is not constrained by any budget and thus, it does not skip releveling of any versions. Similarly, prediction stage attempts every prediction when it is not controlled by decryption overhead control. Figure 4.1 shows version prediction accuracy for previously accessed data cache lines. This means that the CL was already fetched, cached and evicted in the past, before it was accessed again and prediction was attempted on its version. While it is not always guaranteed, there is a high probability that the version was releveled at the previous access, thus increasing the chance of correct version prediction. The average prediction accuracy across all benchmarks is 75%, with highest accuracy of 99% (pt_twitter) and lowest at 36% (sssp_web).

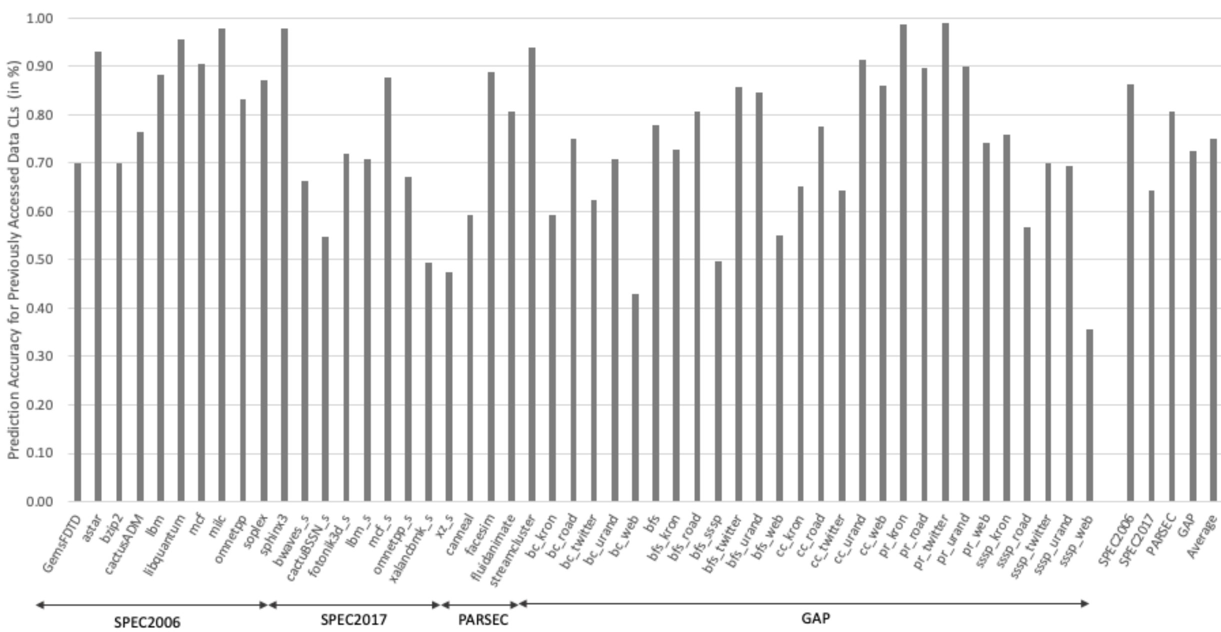

Figure 4.1: Version prediction accuracy for previously accessed data CLs.

Figure 4.2 shows the total version coverage achieved by metadata cache and metadata prediction. While their operations are entirely different, both aim to minimize decryption latency by precomputing OTPs and having them ready before the data CLs arrive. Metadata prediction thus complements metadata cache, as it covers all versions that were not covered by metadata cache (cache miss). The total version coverage is then calculated as the total metadata cache hit rate plus total version coverage of metadata prediction. We see that the total coverage is quite high, varying between 50% and 99% with an average of 83%. This means that for more than 80% of all fetched data cache lines OTP was precomputed and MEE was able to decrypt them immediately on arrival — decryption latency for more than 80% of all fetched traffic was hidden in memory access latency.

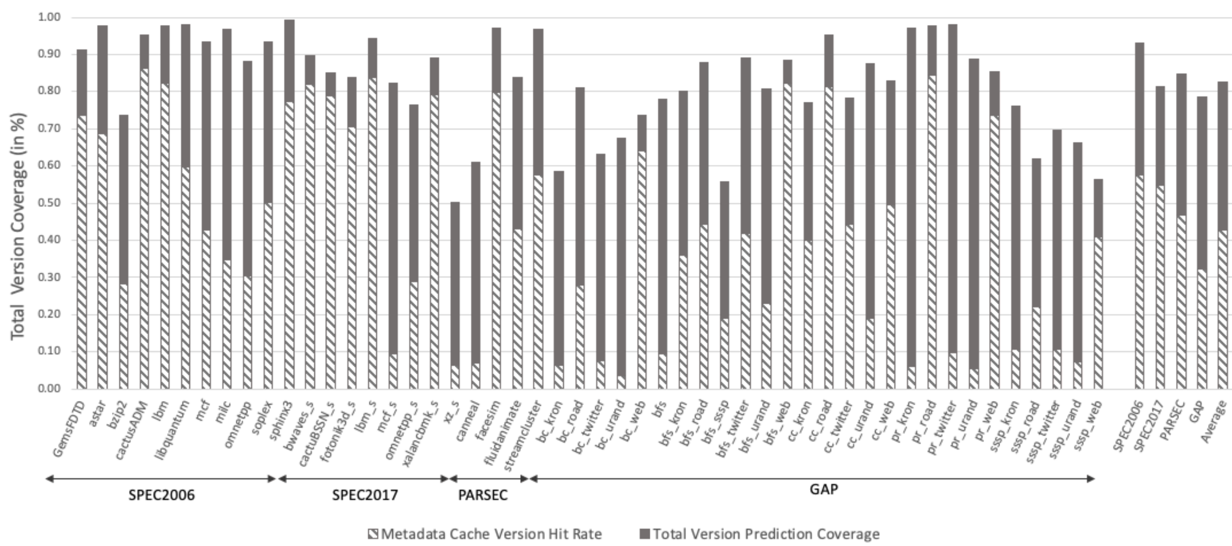

Figure 4.2: Total version coverage by metadata cache and metadata prediction extension.

As discussed in previous chapter, both releveling and prediction generate their respective overheads. Releveling stage results in memory bandwidth overhead when it updates the versions of clean data cache lines. In worst case scenario, such update results in three overhead memory operations: version block must be fetched if not already in metadata cache, version block is written back due to version value update and data cache line has to be re-encrypted with the new version and written back to memory. This is not the case for already dirty data cache lines, whose version will be updated naturally, thus, releveling can be done without any memory bandwidth overhead. Figure 4.3 shows memory traffic overhead caused due to version releveling normalized to regular memory traffic. The average memory bandwidth overhead is 40%, with few extreme ones at more than 200%.

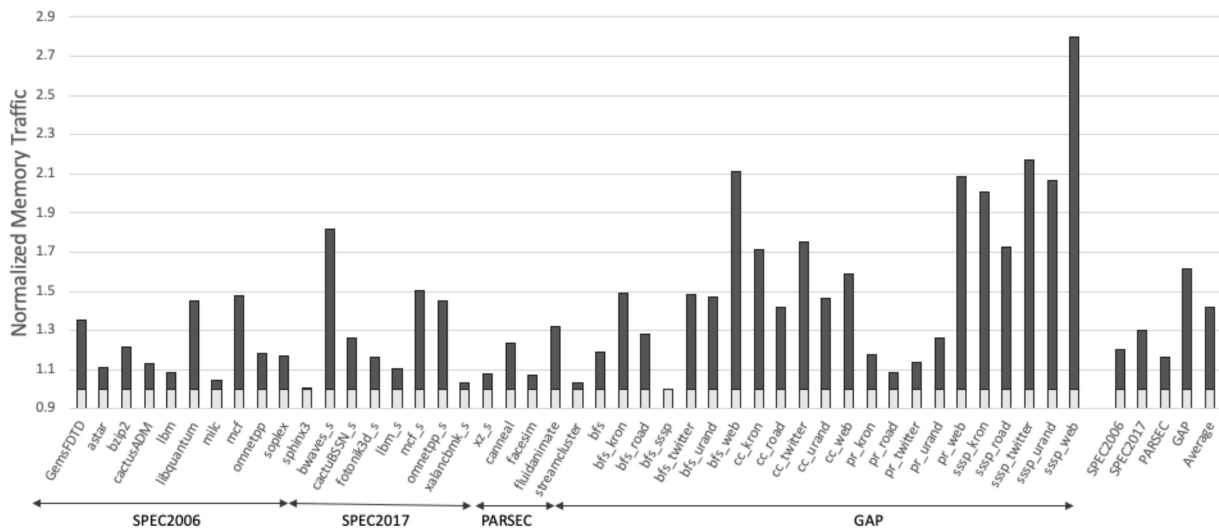

Figure 4.3: Memory bandwidth overhead due to version releveling.

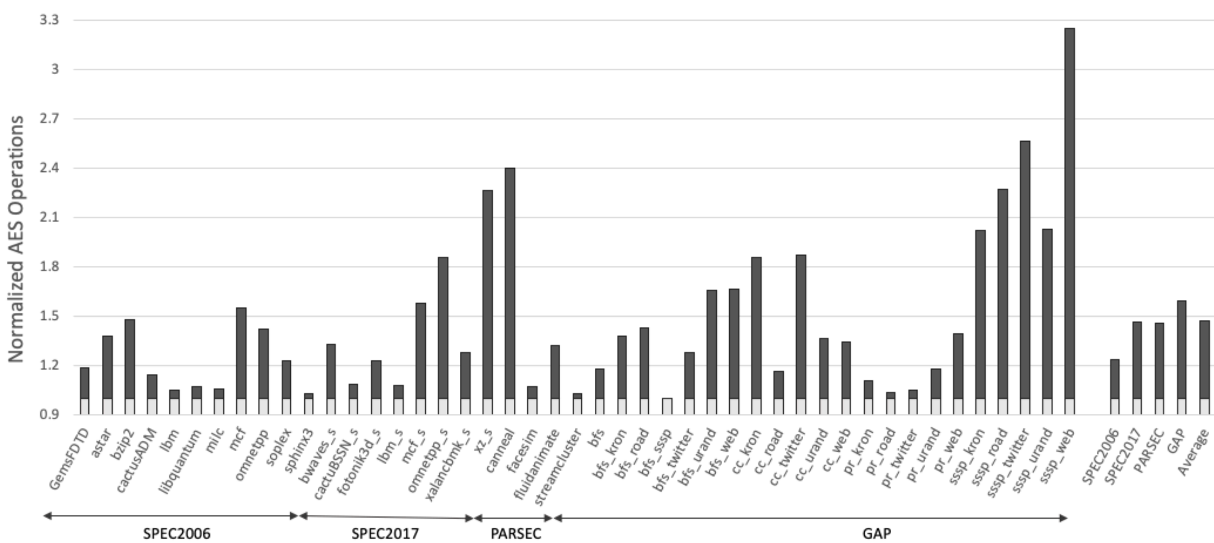

Figure 4.4: Decryption overhead due to version prediction.

Due to extremely limited on-chip area, MEE has a finite number of AES blocks, which are used both for encryption and decryption of data. Every attempted prediction during prediction stage will use AES block to generate speculative OTP. All incorrect predictions result in incorrect speculative OTPs and will require MEE to recompute ”true” OTP with the actual version. Correct prediction on the other hand, results in correct OTP and will not require any additional AES operations. Therefore, all incorrect predictions that are used to generate incorrect OTPs can be considered a decryption overhead. Figure 4.4 shows total decryption overhead due to version predictions normalized to regular AES operations. The average across all benchmarks is less than 50%, with lowest overhead at less than 5% and highest at more than 300%.

4.3 Prediction Accuracy with Overhead Control

Second set of experiments was done with overhead control mechanisms enabled. For memory bandwidth overhead control both methods described in section 3.4.3 were enabled. First, a threshold of 93% was chosen to skip releveling for highly predictable groups. This means that when 15 out of 16 local group versions were correctly predicted it can be considered a highly predictable group. At this point of time, versions in such group are already uniform, thus they can still be accurately predicted in the near future, even if the current releveling is skipped. For second mechanism, which relies on throttling, 30% of total memory traffic was set as a throttling budget. The distribution of budget was implemented without any prioritization and was done at ”first come first served” basis. With a defined budget and a simple proportional controller releveling stage is able actively manage memory bandwidth overhead. As shown in Figure 4.5, with bandwidth overhead control the memory traffic overhead is always kept under 30% with average of 16% across all benchmarks, which is a 26% drop compared to overhead with all control mechanisms disabled.

To mitigate some decryption overhead from prediction stage, a similar throttling mechanism was used, where prediction stage can reduce range of attempted predictions whenever decryption overhead exceeds the budget. The budget for decryption overhead mechanism was chosen to be 50%. Due to some issues in implemented closed-loop controller, the overall over-

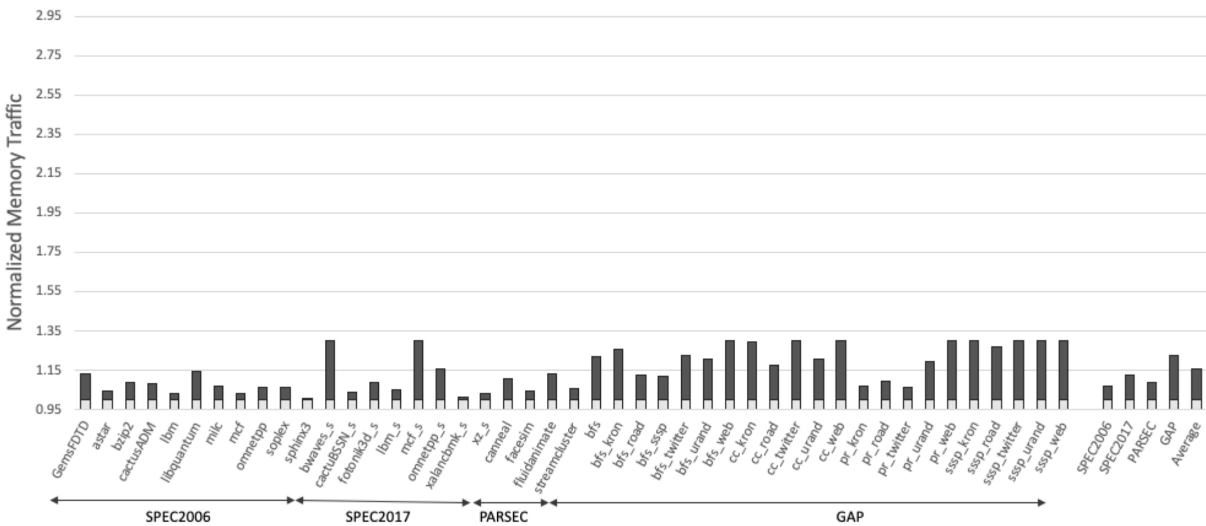

Figure 4.5: Memory bandwidth overhead due to version releveling with memory traffic overhead control enabled.

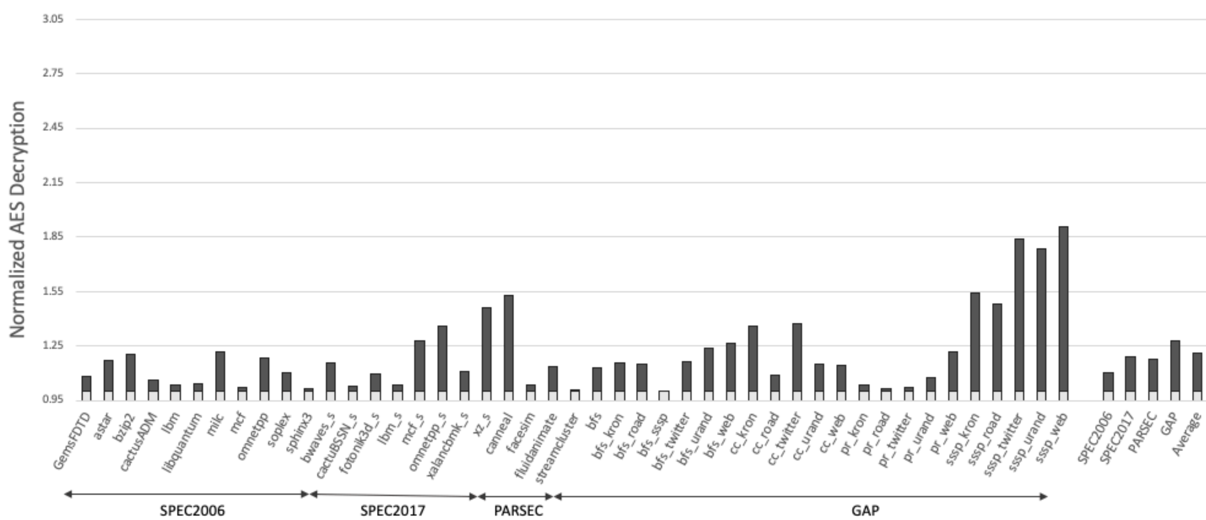

Figure 4.6: Decryption overhead due to version prediction with decryption overhead control enabled.

head frequently overshot above the budget. Despite that, decryption overhead across tested benchmarks was still significantly reduced. As shown in Figure 4.6, the average decryption overhead with decryption control is almost 20%, a drop of 25% compared to overhead without throttling mechanism enabled.

Reducing the number of releveled versions and attempted predictions directly affect the prediction accuracy of metadata prediction algorithm. Figure 4.7 shows the total version coverage achieved by metadata cache and metadata prediction extension with both overhead control mechanisms enabled. The total average coverage is 79%, which is a 4% decrease compared to total coverage with all overhead controls disabled. Despite the fact that the budget parameters were not excessively aggressive, these results show that both control mechanisms can be effectively used to mitigate generated overhead while maintaining a high prediction accuracy.

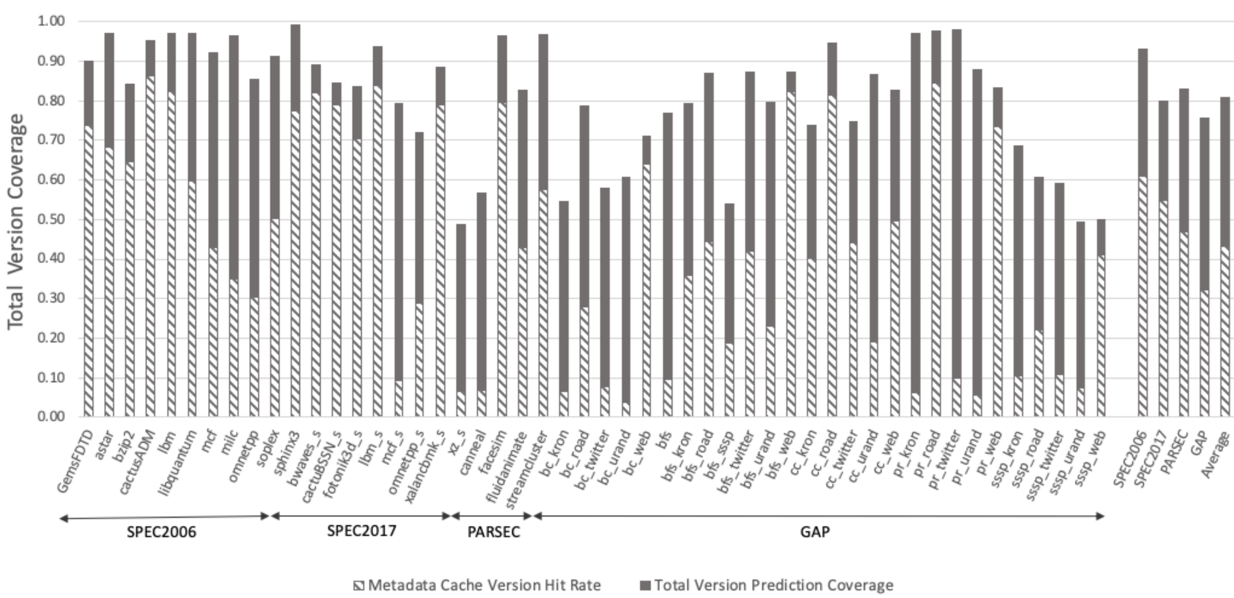

Figure 4.7: Total version coverage by metadata cache and metadata prediction extension with overhead control mechanisms enabled. Memory bandwidth control budget: 30% of total regular traffic. Decryption overhead control budget: 50% of total AES operations.

4.4 Storage Overhead

Given limited internal (on-die) storage any storage overhead is very expensive. If the storage required for effective operation is too high, the entire design can be infeasible, despite high performance improvements. Thus, it is extremely important to understand where the storage overhead comes from and minimize it. Using the configuration parameters shown in Figure 4.1 for PC Table, Prediction Queues and Releveling Queues, the total storage overhead of metadata prediction extension is about 5.5 KB. This is less than 1% of the LLC used in the simulation. This shows that the current storage overhead is already extremely small and can be considered negligible.

Chapter 5

Related Work

This chapter provides an overview of the recent work aimed at improving performance of Intel SGX, as well as other related work that indirectly mitigate certain apsects of SGX performance degradation.

5.1 Data Prefetching

There have been numerous proposed schemes for data prefetching that help CPU mitigate performance impact of off-chip accesses. Prefetching relies on fetching the data before it is actually needed by CPU. Simple stride prefetchers [28, 39], which are usually based on PC or cache line address, are effective for scientific and desktop applications. Temporal address-correlating prefetching [16, 33, 42, 47], where CPU attempts to predict repeated sequences of misses, can be quite effective for commercial workloads. Spatial correlating prefetcheing relies on prediction of recurring spatial patterns over contiguous memory space and is particularly effective for application with high rate of compulsory misses [15, 29, 43]. All these prefetchers aim to hide memory latency, where successful prefetching can help improve SGX performance by hiding both decryption and memory access latency.

5.2 Counter Block Optimization

Research work aimed at optimizing counter block organization improves the performance by making counter blocks more amenable to caching. Yan et. al. [48] proposed split counters that allow 64B counter blocks to accommodate a much higher number of counters. By splitting the block into 64 minor counters and 1 large shared major counter, SC-64 design can effectively cover 64 counters in one counter block. The effective counter is then calculated by adding minor counter to the major counter. Major counter is only incremented when any of the minor counters overflows. By fitting 64 effective counters, in the best-case scenario, caching the same 64B version block in metadata cache can now result in 63 hits compared to 7 in default SGX. A recent work by Saileshwar et al. [38] improves on SC-64 design and proposes Morphable Counters (MorphCtr) with an even higher arity of 128. Similar to SC-64, MorphCtr has a large major counter, but accommodates a total of 128 minor counters. To mitigate frequent overflows of minor counters, MorphCtrs can dynamically change their representation to account for current usage pattern. Such high-arity overflow-tolerant counters significantly improve performance while minimizing storage overhead.

Chapter 6

Conclusion

Encryption of external memory is an essential component of any hardware-assisted TEE technology, like Intel SGX. Unfortunately, to provide modern security guarantees, which require numerous cryptographic primitives and control mechanisms, SGX suffers a significant performance degradation. Part of performance loss comes from high data decryption latency. This thesis present metadata prediction extension that effectively relevels and predicts encryption versions. Current implementation achieves 75% prediction accuracy and a total version coverage of 80% together with metadata cache. Correct predictions of versions directly affect the performance of SGX, as it allows to hide decryption latency associated with expensive generation of decryption one time pad. This can considerably improve the performance of applications with high cache pollution running under SGX. Moreover, metadata prediction extension does not modify any existing components of Memory Encryption Engine. This achieves two major goals. First, it does not interfere with any existing mechanism that also aims to improve performance, like metadata cache. In fact, prediction algorithm complements it by attempting predictions on transactions that are not covered by the metadata cache. This is especially beneficial in cache-sensitive applications. Second, and most importantly, it does not compromise any original security guarantees, as it follows all SGX established security rules (ex. non-repeat property of counters). Thus, MEE with metadata prediction can still guarantee data confidentiality, integrity and replay prevention.

6.1 Future Work

The are numerous avenues for further research that can help make this project stronger. These can range from reworking overhead control mechanisms in order to minimize prediction accuracy degradation to adapting the current design for implementations with higher arity counter blocks. However, the most important next step that must be researched to better understand the strengths and limitations of current design is to use architectural simulator for more accurate system performance evaluation. Current system evaluation described in this thesis was done using Pin, where high-level simulation models were created in Pin tool. While Pin experimental evaluation shows that the current design is working and can be effective for cache-sensitive applications, system performance has to be inferred entirely from prediction accuracy. Architecture simulator, on the other hand, allows researchers to evaluate an actual system speedup. Using simulation framework, like gem5 [13], can help us accurately simulate CPU model, memory system and metadata prediction extension, in order to measure speedup and quantify that performance improvement from version predictions.

Bibliography

[1] Gap benchmark suite. http://gap.cs.berkeley.edu/benchmark.html.

[2] Gdb: The gnu project debugger. https://www.gnu.org/software/gdb/.

[3] Graphene. https://oscarlab.github.io/projects/graphene/.

[4] Nas parallel benchmarks. https://www.nas.nasa.gov/publications/npb.html.

[5] Parsec benchmark suite. https://parsec.cs.princeton.edu/.

[6] Pin 3.6 user guide. https://software.intel.com/sites/landingpage/pintool/docs/97554/Pin/html/.

[7] Intel® software guard extensions developer guide. https://download.01.org/intel-sgx/linux-1.7/docs/Intel_SGX_Developer_Guide.pdf.

[8] Spec cpu2006 benchmark suite. https://www.spec.org/cpu2006/CINT2006/, .

[9] Spec cpu2017 benchmark suite. https://www.spec.org/cpu2017/, .

[10] Ittai Anati, Shay Gueron, Simon Paul Johnson, and Vincent Scarlata. Innovative technology for cpu based attestation and sealing. 2013.

[11] N. Asokan. Hardware-assisted trusted execution environments: Look back, look ahead. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, CCS '19, pages 1687–1687, New York, NY, USA, 2019. ACM. ISBN 978-1-4503-6747-9. doi: 10.1145/3319535.3364969. URL http://doi.acm.org/10.1145/3319535.3364969.

[12] Georg Becker and Ruhr universität Bochum. Merkle signature schemes, merkle trees and their cryptanalysis, 2008.

[13] Nathan Binkert, Bradford Beckmann, Gabriel Black, Steven K. Reinhardt, Ali Saidi, Arkaprava Basu, Joel Hestness, Derek R. Hower, Tushar Krishna, Somayeh Sardashti, Rathijit Sen, Korey Sewell, Muhammad Shoaib, Nilay Vaish, Mark D. Hill, and David A. Wood. The gem5 simulator. *SIGARCH Comput. Archit. News*, 39(2):1–7, August 2011. ISSN 0163-5964. doi: 10.1145/2024716.2024718. URL http://doi.acm.org.ezproxy.lib.vt.edu/10.1145/2024716.2024718.

[14] Stefan Brenner, Michael Behlendorf, and Rüdiger Kapitza. Trusted execution, and the impact of security on performance. In *Proceedings of the 3rd Workshop on System Software for Trusted Execution*, SysTEX '18, pages 28–33, New York, NY, USA, 2018. ACM. ISBN 978-1-4503-5998-6. doi: 10.1145/3268935.3268943. URL http://doi.acm.org/10.1145/3268935.3268943.

[15] C. F. Chen, S. . Yang, B. Falsafi, and A. Moshovos. Accurate and complexity-effective spatial pattern prediction. In *10th International Symposium on High Performance Computer Architecture (HPCA'04)*, pages 276–287, Feb 2004. doi: 10.1109/HPCA.2004.10010.

[16] Y. Chou. Low-cost epoch-based correlation prefetching for commercial applications. In *40th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO 2007)*, pages 301–313, Dec 2007. doi: 10.1109/MICRO.2007.39.

[17] V. Costan, I. Lebedev, and S. Devadas. *Secure Processors Part II: Intel SGX Security Analysis and MIT Sanctum Architecture*. now, 2017. ISBN null. URL https://ieeexplore.ieee.org/document/8187110.

[18] Victor Costan and Srinivas Devadas. Intel sgx explained. *IACR Cryptology ePrint Archive*, 2016:86, 2016.

[19] J. Powell D. Kaplan and T. Woller. Amd memory encryption, white paper. https://developer.amd.com/wordpress/media/2013/12/AMD_Memory_Encryption_Whitepaper_v7-Public.pdf, 2016.

[20] Tevi Devor. Pin: Intel's dynamic binary instrumentation engine: Pin tutorial. https://software.intel.com/sites/default/files/managed/62/f4/cgo2013.pdf, 2013.

[21] Tu Dinh Ngoc, Bao Bui, Stella Bitchebe, Alain Tchana, Valerio Schiavoni, Pascal Felber, and Daniel Hagimont. Everything you should know about intel sgx performance on virtualized systems. *Proc. ACM Meas. Anal. Comput. Syst.*, 3(1):5:1–5:21, March 2019. ISSN 2476-1249. doi: 10.1145/3322205.3311076. URL http://doi.acm.org/10.1145/3322205.3311076.

[22] M. Eldred, A. Good, and C. Adams. A case study on data protection and security decisions in cloud hpc. In *2015 IEEE 7th International Conference on Cloud Computing Technology and Science (CloudCom)*, pages 564–568, Nov 2015. doi: 10.1109/CloudCom.2015.114.

[23] S. Gueron. Intel® software guard extensions (intel® sgx). https://rwc.iacr.org/2016/Slides/RWC_2016_Gueron_2016_01_08.pdf, 2016. Accessed: 2019-10-29.

[24] Shay Gueron. A memory encryption engine suitable for general purpose processors. *IACR Cryptology ePrint Archive*, 2016:204, 2016.

[25] Mohamed Hanini and Said El Kafhali. Cloud computing performance evaluation under dynamic resource utilization and traffic control. In *Proceedings of the 2Nd International Conference on Big Data, Cloud and Applications*, BDCA'17, pages 5:1–5:6, New York,

NY, USA, 2017. ACM. ISBN 978-1-4503-4852-2. doi: 10.1145/3090354.3090359. URL http://doi.acm.org/10.1145/3090354.3090359.

[26] Matthew Hoekstra, Reshma Lal, Pradeep Pappachan, Vinay Phegade, and Juan del Cuvillo. Using innovative instructions to create trustworthy software solutions. 06 2013. doi: 10.1145/2487726.2488370.

[27] Aamer Jaleel, Kevin B. Theobald, Simon C. Steely, Jr., and Joel Emer. High performance cache replacement using re-reference interval prediction (rrip). In *Proceedings of the 37th Annual International Symposium on Computer Architecture*, ISCA '10, pages 60–71, New York, NY, USA, 2010. ACM. ISBN 978-1-4503-0053-7. doi: 10.1145/1815961.1815971. URL http://doi.acm.org/10.1145/1815961.1815971.

[28] N. P. Jouppi. Improving direct-mapped cache performance by the addition of a small fully-associative cache and prefetch buffers. In *[1990] Proceedings. The 17th Annual International Symposium on Computer Architecture*, pages 364–373, May 1990. doi: 10.1109/ISCA.1990.134547.

[29] S. Kumar and C. Wilkerson. Exploiting spatial locality in data caches using spatial footprints. In *Proceedings. 25th Annual International Symposium on Computer Architecture (Cat. No.98CB36235)*, pages 357–368, July 1998. doi: 10.1109/ISCA.1998.694794.

[30] T. S. Lehman, A. D. Hilton, and B. C. Lee. Poisonivy: Safe speculation for secure memory. In *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 1–13, Oct 2016. doi: 10.1109/MICRO.2016.7783741.

[31] Chi-Keung Luk, Robert Cohn, Robert Muth, Harish Patil, Artur Klauser, Geoff Lowney, Steven Wallace, Vijay Janapa Reddi, and Kim Hazelwood. Pin: Building customized program analysis tools with dynamic instrumentation. In *Proceedings of the 2005 ACM*

SIGPLAN Conference on Programming Language Design and Implementation, PLDI '05, pages 190–200, New York, NY, USA, 2005. ACM. ISBN 1-59593-056-6. doi: 10.1145/1065010.1065034. URL http://doi.acm.org/10.1145/1065010.1065034.

[32] Frank McKeen, Ilya Alexandrovich, Alex Berenzon, Carlos V. Rozas, Hisham Shafi, Vedvyas Shanbhogue, and Uday R. Savagaonkar. Innovative instructions and software model for isolated execution. In *Proceedings of the 2Nd International Workshop on Hardware and Architectural Support for Security and Privacy*, HASP '13, pages 10:1–10:1, New York, NY, USA, 2013. ACM. ISBN 978-1-4503-2118-1. doi: 10.1145/2487726.2488368. URL http://doi.acm.org/10.1145/2487726.2488368.

[33] K. J. Nesbit and J. E. Smith. Data cache prefetching using a global history buffer. In *10th International Symposium on High Performance Computer Architecture (HPCA'04)*, pages 96–96, Feb 2004. doi: 10.1109/HPCA.2004.10030.

[34] D. J. Palframan, N. S. Kim, and M. H. Lipasti. Cop: To compress and protect main memory. In *2015 ACM/IEEE 42nd Annual International Symposium on Computer Architecture (ISCA)*, pages 682–693, June 2015. doi: 10.1145/2749469.2750377.

[35] Moinuddin K. Qureshi, Aamer Jaleel, Yale N. Patt, Simon C. Steely, and Joel Emer. Adaptive insertion policies for high performance caching. In *Proceedings of the 34th Annual International Symposium on Computer Architecture*, ISCA '07, pages 381–391, New York, NY, USA, 2007. ACM. ISBN 978-1-59593-706-3. doi: 10.1145/1250662.1250709. URL http://doi.acm.org/10.1145/1250662.1250709.

[36] Vijay Janapa Reddi, Alex Settle, Daniel A. Connors, and Robert S. Cohn. Pin: A binary instrumentation tool for computer architecture research and education. In *Proceedings of the 2004 Workshop on Computer Architecture Education: Held in Conjunction with the 31st International Symposium on Computer Architecture*, WCAE '04, New York,

NY, USA, 2004. ACM. ISBN 978-1-4503-4733-4. doi: 10.1145/1275571.1275600. URL http://doi.acm.org/10.1145/1275571.1275600.

[37] Brian Rogers, Siddhartha Chhabra, Milos Prvulovic, and Yan Solihin. Using address independent seed encryption and bonsai merkle trees to make secure processors os- and performance-friendly. pages 183–196, 01 2008. ISBN 978-0-7695-3047-5. doi: 10.1109/MICRO.2007.16.

[38] G. Saileshwar, P. Nair, P. Ramrakhyani, W. Elsasser, J. Joao, and M. Qureshi. Morphable counters: Enabling compact integrity trees for low-overhead secure memories. In *2018 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 416–427, Oct 2018. doi: 10.1109/MICRO.2018.00041.

[39] T. Sherwood, S. Sair, and B. Calder. Predictor-directed stream buffers. In *Proceedings 33rd Annual IEEE/ACM International Symposium on Microarchitecture. MICRO-33 2000*, pages 42–53, Dec 2000. doi: 10.1109/MICRO.2000.898057.

[40] T. Silbergleit Lehman, A. D. Hilton, and B. C. Lee. Maps: Understanding metadata access patterns in secure memory. In *2018 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, pages 33–43, April 2018. doi: 10.1109/ISPASS.2018.00012.

[41] J. Sobchuk, S. O'Melia, D. Utin, and R. Khazan. Leveraging intel sgx technology to protect security-sensitive applications. In *2018 IEEE 17th International Symposium on Network Computing and Applications (NCA)*, pages 1–5, Nov 2018. doi: 10.1109/NCA.2018.8548184.

[42] Yan Solihin, Jaejin Lee, and Josep Torrellas. Using a user-level memory thread for correlation prefetching. In *Proceedings of the 29th Annual International Symposium on*

Computer Architecture, ISCA '02, pages 171–182, Washington, DC, USA, 2002. IEEE Computer Society. ISBN 0-7695-1605-X. URL http://dl.acm.org/citation.cfm?id=545215.545235.

[43] S. Somogyi, T. F. Wenisch, A. Ailamaki, B. Falsafi, and A. Moshovos. Spatial memory streaming. In *33rd International Symposium on Computer Architecture (ISCA'06)*, pages 252–263, June 2006. doi: 10.1109/ISCA.2006.38.

[44] Stephen Somogyi, Thomas F. Wenisch, Anastasia Ailamaki, and Babak Falsafi. Spatio-temporal memory streaming. In *Proceedings of the 36th Annual International Symposium on Computer Architecture*, ISCA '09, pages 69–80, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-526-0. doi: 10.1145/1555754.1555766. URL http://doi.acm.org/10.1145/1555754.1555766.

[45] G. E. Suh, D. Clarke, B. Gasend, M. van Dijk, and S. Devadas. Efficient memory integrity verification and encryption for secure processors. In *Proceedings. 36th Annual IEEE/ACM International Symposium on Microarchitecture, 2003. MICRO-36.*, pages 339–350, Dec 2003. doi: 10.1109/MICRO.2003.1253207.

[46] Meysam Taassori, Ali Shafiee, and Rajeev Balasubramonian. Vault: Reducing paging overheads in sgx with efficient integrity verification structures. In *Proceedings of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '18, pages 665–678, New York, NY, USA, 2018. ACM. ISBN 978-1-4503-4911-6. doi: 10.1145/3173162.3177155. URL http://doi.acm.org/10.1145/3173162.3177155.

[47] Thomas F. Wenisch, Stephen Somogyi, Nikolaos Hardavellas, Jangwoo Kim, Anastassia Ailamaki, and Babak Falsafi. Temporal streaming of shared memory. In *Proceedings of the 32Nd Annual International Symposium on Computer Architecture*, ISCA '05, pages

222–233, Washington, DC, USA, 2005. IEEE Computer Society. ISBN 0-7695-2270-X. doi: 10.1109/ISCA.2005.50. URL https://doi.org/10.1109/ISCA.2005.50.

[48] Chenyu Yan, Daniel Englender, Milos Prvulovic, Brian Rogers, and Yan Solihin. Improving cost, performance, and security of memory encryption and authentication. In *Proceedings of the 33rd Annual International Symposium on Computer Architecture*, ISCA '06, pages 179–190, Washington, DC, USA, 2006. IEEE Computer Society. ISBN 0-7695-2608-X. doi: 10.1109/ISCA.2006.22. URL https://doi.org/10.1109/ISCA.2006.22.