SoK: Cryptographic Protection of Random Access Memory – How Inconspicuous can Hardening Against the most Powerful Adversaries be?

Roberto Avanzi[1], Ionuț Mihalcea[2], David Schall[3], Héctor Montaner[4], and Andreas Sandberg[2]

[1]Arm Germany, GmbH — roberto.avanzi@arm.com, roberto.avanzi@gmail.com
[2]Arm Limited, UK — ionut.mihalcea@arm.com, andreas.sandberg@arm.com
[3]School of Informatics, University of Edinburgh, United Kingdom — david.schall@ed.ac.uk
[4]Graphcore, Cambridge UK — hector.montaner@outlook.com

***Abstract*—There is a trend towards providing stronger isolation between mutually untrusted processes running on the same computer system. For example, Intel SGX, AMD SEV, and Arm CCA all provide access control mechanisms to protect the execution of programs from hostile peer and higher privileged software. Some of these technologies include cryptographic memory protection, such as encryption and integrity checks, to protect against sophisticated physical attacks.**

We review the main technologies, from both academia and industry, to ensure confidentiality and integrity of main memory. We classify these technologies according to models of adversaries with varying capabilities, and group them according to corresponding protection levels. We also provide an extensive benchmarking of their performance penalties and memory overheads both on systems where the benchmark suite is the only running task and on heavily loaded systems.

We additionally propose new solutions to further reduce the performance and memory overheads of such protection. For example, we show that advanced counter compression techniques make it viable to store them in a physically protected memory which is just 1:256 of the total off-chip memory. By repurposing ECC bits to store integrity tags, we can achieve hitherto unattained performance while providing full confidentiality, integrity, and replay protection. In the case of a representative server system, on the industry standard SPEC 2017 benchmark suite we achieve a 1.93% performance penalty if the benchmarks are the only running tasks, and 3.17% on a system if the memory subsystem is fully saturated.

1. Introduction

Cloud computing promises to increase efficiency and drive down cost for users. Such services co-locate multiple mutually untrusted tenants in the same data center and sometimes even the same physical machines. Compared to traditional on-premise solutions, users of cloud computing face two additional threats. First, hostile tenants may try to exploit bugs in the hypervisor or *access control* mechanisms to impact the confidentiality, integrity, or availability of co-located virtual machines. Second, the service provider or its contractors may try to gain access to customer data.

Similar threats exist in client devices such as phones, which have evolved into smart terminals and identity providers. Like in the case of a data center, the adversaries may use co-located untrusted code or even have physical access to the device. Use cases such as secure payments, secure identification, and even gaming need to rely on strong confidentiality and integrity guarantees. Traditional solutions use separate components, such as SIM cards or TPMs, to provide such guarantees. Consolidating this type of functionality onto the main *System-on-a-Chip (SoC)* allows to enable new use cases while reducing overall costs.

Solutions such as AMD SEV [1], Arm CCA [2], and Intel SGX [3] and TDX [4] go towards this goal by providing architectural mechanisms to reduce the impact of hypervisor exploits and malicious operators. Some even include protection against adversaries with physical access to the system. For instance, Intel SGX implements a *Memory Encryption Engine (MEE)* that provides confidentiality by encrypting memory, and integrity and replay protection using an integrity tree. Such strong confidentiality and integrity guarantees can be very costly in terms of performance and storage. Other technologies such as AMD SEV and Intel TDX opt to provide slightly weaker confidentiality and integrity guarantees to offer improved performance.

In this paper, we review the state of the art techniques used to provide confidentiality and integrity guarantees in hardware. We cover the techniques used to protect off-chip memories (e.g., DRAM) from an adversary with physical access to the system. We then propose new solutions to further reduce performance and memory overheads while maintaining the highest level of integrity and confidentiality.

In order to systematically classify and compare existing techniques, we define different protection levels based on a taxonomy of adversaries with varying technical capabilities. Suitable technical mitigations are then deployed to implement each protection level. This lets us reason about the trade-off between security guarantees and performance.

The evaluation uses the industry-standard SPEC 2017 [5] benchmark suites running on the `gem5` simulator [6], [7]. The most striking result is that advanced counter compression makes it viable to store counters in a relatively small physically secure memory. This enables implementations of memory replay protection with very low performance

penalties, especially if some ECC bits are repurposed to store integrity tags: Performance penalties smaller than 4%, with a memory overhead of 1:128 or 1:256, can be attained even under heavy bus contention. On the same system, the older SPEC 2006 benchmark [8] shows performance penalties smaller than 1%, resp. 2%, with a memory overhead of 1/128, resp. 1/256. We conjecture that similar performance penalties can be attained by deploying a suitably large system cache in place of on-chip RAM.

2. Systematisation of the problem

2.1. Definitions

The software-accessible volatile memory attached to a memory controller is viewed as an array of blocks. The size of these blocks corresponds to the cache line size of the *last level cache*, which is usually a system cache. Due to their direct correspondence to cache lines, we call these blocks cache lines even when stored in off-chip memory.

If a scheme provides *integrity*, it associates an *integrity tag* with one or more cache lines. Commonly, the integrity tag is a *Message Authentication Code (MAC)*.

An encryption or authentication function is said to provide *spatial uniqueness* when computed on equal inputs, but written to different locations, it results in different outputs. This is achieved by including the *Physical Address (PA)* of the encrypted or authenticated cache line in the computation.

An encryption or authentication function provides *temporal uniqueness* (*freshness*) when repeated writes of the same plaintext to the same location result in different outputs. This can be achieved by associating a counter with each cache line and including it in the computation of the function.

In what follows a *mode* is a general purpose encryption mode of operation. A *memory encryption mode* is understood to be an encryption mode of operation that has fixed input lengths, plaintext and ciphertext having the same size as a cache line, and no associated data.

Here, an *on-chip* component is defined as a physically secure block in the same package as the processing elements.

2.2. Problem statement and protection levels

The question that we answer in this study is: *What technologies are available to protect the contents of data-in-use in RAM against an adversary, and what are their memory overheads and performance costs?*

Clearly, to properly answer this question we need to group the technologies according to the adversaries they are meant to defend against, avoiding artificial distinctions between adversaries, for instance between adversaries capable of running software on the target device and adversaries that can mount RowHammer attacks [9]. We find that primarily classifying adversaries based on how they access to the target devices and their resources (i.e., essentially budget) results in the cleanest classification. We consider complexity of attack and attack reliability as a secondary concern.

2.2.1. Level 1: Basic memory encryption. This level provides only memory confidentiality. It uses a direct memory encryption method, i.e. the plaintext is passed directly through the encryption function to compute the ciphertext. No metadata is stored. This level provides spatial uniqueness, but neither temporal uniqueness nor integrity verification.

This level is meant to defeat adversaries that can only run software on the target and manipulate external interfaces. Beside software exploitation, the adversaries can also mount RowHammer attacks. However, in general integrity violations are only a partial concern, as they can arguably be made less effective by deploying memory encryption. We exclude access pattern and ciphertext side channels as these variants require more sophisticated techniques.

Regarding RowHammer attacks, it can be argued that delegating access control and memory allocation to a trusted environment could prevent the assignment of adjacent memory rows to mutually untrusted processes. While this can in theory work, it would require extensive re-engineering of existing system software. Such a countermeasure could still be circumvented by deploying an address-remapping interposer between SoC and external memory.

This basic Protection Level is deployed in many commercial systems, a notable example being AMD's SEV [1].

To simplify the treatment, we assume from now on that appropriate access control policies are in place to stop unauthorised agents *within* the SoC, but not to prevent RowHammer attacks.

2.2.2. Level 2: Encryption and integrity verification. This level extends Level 1 with integrity tags, but does not provide any temporal uniqueness.

Level 2 targets adversaries with physical access to the complete system containing the protected components. The adversaries have access to exposed interfaces and communication buses but they do not have the capabilities to access on-chip communication interfaces. They mainly perform passive attacks, e,g,: physical side-channel analysis in close proximity, contact or connection to the target device; eavesdropping the content of external RAM, either at runtime via memory bus probing, chip or module interposition; abuse of DMA channels; or cold-boot attacks.

Because of the similarity of the involved techniques, these countermeasures are also effective against limited active adversaries. For instance, these adversaries may only be able to corrupt individual memory locations. In order to defeat targeted memory replay (payload and integrity tags), more sophisticated countermeasures are required (see Level 3 below). It can be argued that this distinction is arbitrary, but in reality it is made necessary by the difference in complexity and cost of both attacks and *countermeasures*. In other words, the system designer and integrator will decide whether to accept risks arising from specific adversaries following a proper risk assessment. That is, this ends up being a business decision and as such outside the scope of this paper. Still, in general a good reason for an adversary capable of active HW attacks to perform only passive ones is *detectability*.

A passive attack has the advantage of being less likely to trigger errors that may betray that an attack is occurring.

The Intel TDX *Multi-Key Total Memory Engine with Integrity (MKTMEi)* [4] is a L2/MirE solution, where MirE means *MACs in repurposed ECC bits*. We found no documentation on error correction in a TDX system, but the 28b MAC field size suggests the following: Four instances of a $(255, 247)$ Hamming code are used, truncated to $(143, 135)$ to cover 128 bits and 7 bits of the MAC each — with the remaining 4 bits of the effective 576 bits in each cache line used for parity. This very configuration is proposed in [10]. CSI:Rowhammer [11] is a L2/MirE solution as well.

2.2.3. Level 3: Encryption and replay protection. This level includes *Replay protection*, which is any form of integrity protection that is capable of detecting not only memory corruption, but also replay of memory contents including associated metadata.

In addition to the capabilities of the previous adversaries, these adversaries also perform *active* attacks, e.g.: blocking, corrupting, replaying memory transactions, or even injecting new ones [12]. Examples of threats include [13]–[15].

The Intel SGX *Memory Encryption Engine* [3], ELM [16], and the memory protection mechanism of Apple's Secure Enclave [17] (starting with the A11 and S4 SoCs) are L3 solutions. SYNERGY [18] is a L3/MirE solution.

2.2.4. Out of scope adversaries. Out of scope for the research described in this paper are adversaries that can mount highly invasive attacks at the chip or package level that require considerable experience, resources, and time to succeed. Examples of such attacks range from micro-probing attacks [19] to actual chip reverse engineering and editing using a Focused Ion Beam Microscope [20]–[22].

2.3. System level view of the technical solution

To implement the above Protection Levels, we introduce a *Memory Protection Engine (MPE)*. This is not a new idea: all cryptographic memory protection designs cited so far use such a HW block, sometimes also known as a *Memory Encryption Engine (MEE)*. As depicted in Fig. 1, in a SoC the MPE sits between an interconnect (or a system cache) on one side and a memory controller on the other side. It can optionally have: caches, namely a counter group and a hash cache; internal buffers (not depicted); and it may have access to a certain amount of on-chip DRAM. *The memory protection technologies that we study in this paper are implemented in the MPE.*

The fact that we functionally represent the MPE as a separate block sitting between system cache and memory controller does not preclude other designs, such as, for instance, integrating it in the memory controller, or implementing it as a wrapper around the system cache. More in general, a MPE could be associated either to each memory channel, and thus reside between on-chip interconnect and external memory, or to each core or core cluster that needs memory protection, and thus reside between those processing elements and the on-chip interconnect. We do not consider the latter design because it is uncommon in the literature, and it becomes a bottleneck when the associated core or core cluster is generating considerable traffic, whereas MPEs associated to each memory channel profit from large system caches and memory interleaving to reduce bandwidth saturation risks.

This said, we understand that some *secure cores* (such as software defined TPMs) may even need a private MEE. General purpose software solutions can be envisioned as well, for instance delegating memory management to a trusted piece of software which takes care of paging external encrypted memory and mapping a cache of decrypted memory pages in on-chip SRAM for recently used live code and data. In these cases, however, performance is usually not a strong requirement, and is quite poor.

It is likely that the full design space is not knowable, and there may be further meaningful realisations of memory protection that we are not aware of.

2.4. Building blocks

The following types of technologies are used to implement the various protection levels: (i) Memory encryption primitives and modes; (ii) Authentication primitives; (iii) Integrity and replay protection structures; and (iv) Physical mechanisms to protect *a relatively small amount* of memory from tampering, such as including it in a tamper-proof package. The first groups are reviewed in Section 3.

Note that we exclusively consider solutions that need the security perimeter to be no larger than the physical package of the SoC. Hence, "smart memory" [23] is out of scope.

2.5. Cost indicators

For real-world applications it is very important to know how *expensive* a solution to a problem is. The two principal cost indicators are the performance penalty and the memory overhead. Area and power constraints restrict which solutions can be considered for viability, but relaxing these constraints can sometimes be justified in the presence of a strong market requirement. On the other hand, a solution that impacts performance or memory availability too heavily will face major acceptance hurdles. For this reason, *we focus mainly on performance penalty and memory overhead.*

This said, power consumption is roughly linear in both area and time spent running a circuit. Therefore the area of the MPE and the performance penalty are the main factors determining the energy cost of any solution. The area includes also any cache or on-chip memory to support the MPE.

3. Background

We present here a brief summary of the main technologies that have been considered the development of the technologies discussed in this paper.

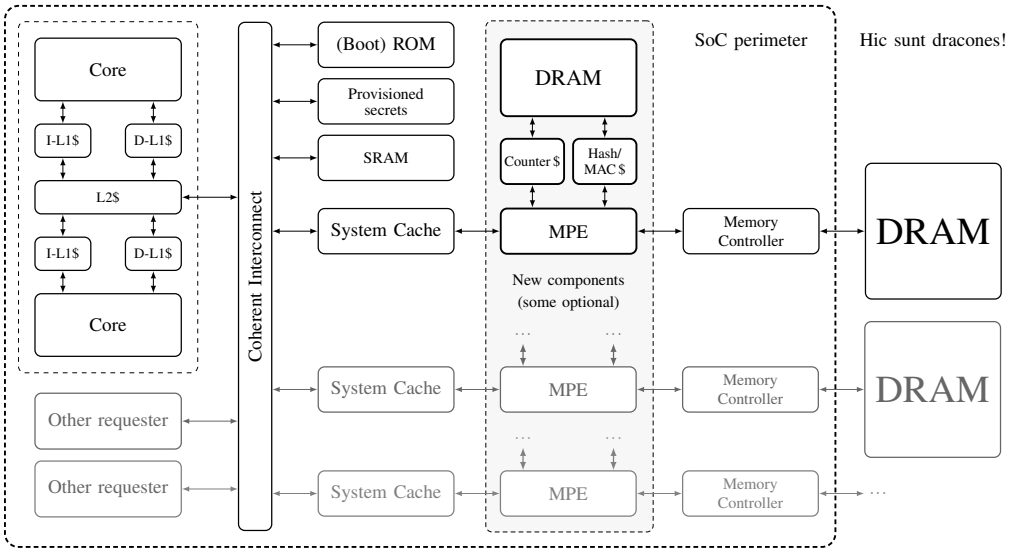

Figure 1: Simplified system level view of a SoC with Memory Protection Engine(s) (DMC is the Dynamic Memory Controller

Memory encryption primitives. Block ciphers are the most commonly used encryption function for memory encryption. Stream ciphers, due to their long initial latency, are considered unsuitable in this context, and almost never used.

In *direct encryption*, the block cipher is applied blockwise to the plaintext to generate the ciphertext. In *One-Time Pad (OTP) encryption*, the encryption of successive values of a counter is used to generate an OTP stream. This stream is then XOR-ed with the plaintext. This is usually known as *CounTeR mode (CTR)* mode in cryptographic literature.

We only consider block ciphers with a block size of 128 bits. The selected block ciphers are the `AES` [24] and the *Tweakable Block Cipher (TBC)* `QARMA` [25]. Other candidates (e.g. `PRINCE` [26]) have either similar latencies, are not tweakable or have a shorter block size.

Authentication primitives. Standard hash functions such as `SHA-2` [27] or `SHA-3` [28] can be turned into MACs but the resulting schemes are very slow and not parallelizable.

Encrypted *Universal Hash Functions (UHFs)* [29] are a better choice. UHFs admit fully parallelisable constructions, such as multi-linear functions of the input computed over a binary Galois field, as used in `SGX` [30]. We note that if a cache is available for UHF-based MACs, then the cached values need not be encrypted: The universal hashes are encrypted only when evicted from the cache, and the cached hashes can be verified more efficiently.

Apple, in their Secure Enclave [17] uses a CMAC [31] to compute integrity tags. This method is not parallelisable and has a high latency, but the use case does not need very high throughput. It would not be suitable for server and advanced client applications. Instead, we evaluate TBC-based *Parallel MACs (PMACs)* [32]. PMACs are more expensive than encrypted UHFs because the text is first processed by encryption instead of Galois multiplications, but they they can be used for error detection and correction beside integrity, cf. [11], [18], [33]. The computation of PMACs is depicted

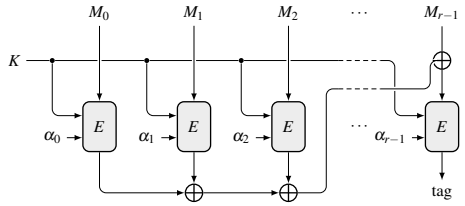

Figure 2: PMAC computed with a TBC for the cases where freshness is not implemented

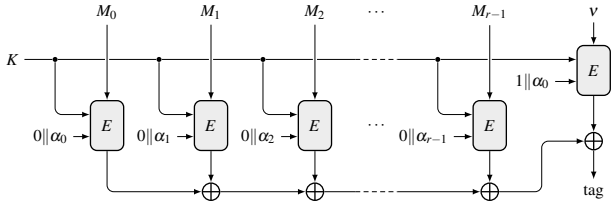

Figure 3: PMAC computed with a TBC for the cases where freshness information is available

in Figs. 2 and 3. Such constructions can easily be made *incremental* where upon a write only the part of the message that has changed needs to be recomputed. A variant for non-TBCs, called PXOR-MAC is detailed in [16].

We do not consider encrypted checksums of the plaintext, even if they are used in Rogaway's *Offset Codebook mode (OCB)* mode [32]. OCB requires freshness to be provided, in which case in all practical systems we are aware of, OTP encryption is used instead, which reduces read latencies.

Modes of operation. For direct encryption, spatial uniqueness is achieved by using the PA as the cipher's tweak. To achieve this with a non-TBC, we use it in Rogaway's *XOR, Encrypt, and XOR (XEX)* construction [32]. XEX is defined as $C_i = E_K(P_i \oplus M_i) \oplus M_i$. In other words, a tweak-derived

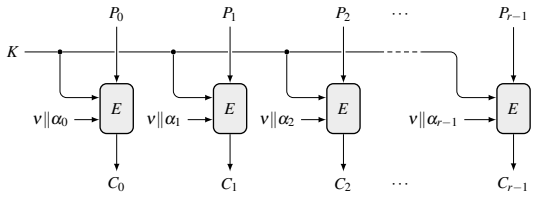

Figure 4: Tweaked ECB mode

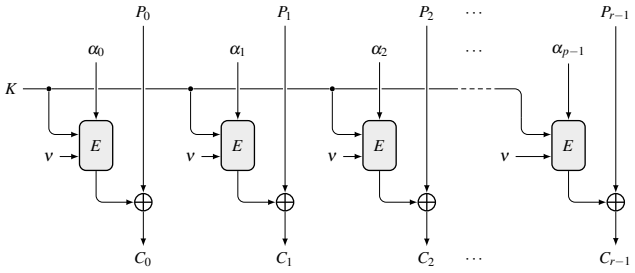

Figure 5: CounTeR in Tweak mode

mask is added to the input and the output of the cipher. The first mask M_0 is derived by encrypting the tweak, and the successive masks M_i for $i \geq 1$ are obtained by multiplying the first mask by a fixed sequence of values. Using a single finite field element γ we can put $M_i = \gamma^i \cdot M_0$. Apart from OCB [32], a similar construction is used in the FLAT-ΘCB mode [16]. There, it is used with a truncated, OTP-encrypted checksum of the plaintext to define an *Authenticated Encryption* mode. PXOR-MAC is used to authenticate counter groups to finally define the L3 scheme *Encryption for Large Memory (ELM)*.

With a TBC, the PA (concatenated with freshness, if provided) of each block is used directly as the tweak, cf. Fig. 4, and the XEX construction is not needed.

In OTP encryption with a TBC, the counter and PA are used as tweak and text respectively (cf. Fig. 5). When not using a TBC, the counter and PA are concatenated concatenated and then encrypted.

Memory integrity structures. A table of hashes or MACs is sufficient to protect against memory corruption.

Protection against replay attacks requires that the table is either protected in an on-chip memory or with a tree structure such as a Merkle Tree [34]. Merkle Tree nodes can be cached [35] to speed up verification.

If a counter-based encryption scheme is used, we can protect memory by recursively protecting just the counters as follows: a set of a of counters and an *embedded* MAC form a node called a counter group, which has the same size as a cache line. Each counter group has a children, which can be either cache lines of data or child counter groups. Each counter in a counter group is associated with one child. The embedded MAC is computed on the a counters and the parent counter. For data (leaf) nodes, the MAC is not embedded. It is instead stored in a separate table. Before a counter group, or a data cache line is evicted, its parent counter is first incremented and the counter group's or cache line's MAC is recomputed. Such a *Counter tree* is used for instance in Intel's SGX [3]. Counter trees are in fact just an in-memory reorganisation of Hall and Jutla's *Parallelisable Authentication Tree (PAT)* [36].

With the *split counters* optimisation [37] a group of a counters is replaced by a group consisting of a single *major counter* and $a' > a$ smaller, *minor counters*, associated with that major counter. A logical counter in this scheme is defined as the concatenation of a minor counter and its associated major counter. Each node (a data cache line or a counter group) is associated with a logical counter. The increased arity (for instance, from $a = 8$ to $a' = 64$) reduces both storage overhead for counters and tree depth. When a minor counter overflows, the common major counter is ticked to ensure that values do not repeat. Since this changes the value of all of the logical counters associated with that major counter, all the sibling nodes need to be refreshed. For data cache lines this means that they are re-encrypted, and for both types of nodes the MACs need to be recomputed. All minor counters in the group are reset to zero at this point to reduce the frequency of minor counter overflows.

Despite these *Read-Modify-Write (RMW)* operations, split counter trees bring a major performance improvement over monolithic counters. As a further optimisation, we introduce in this paper 3-way split counters, which comprise major, middle, and minor counters.

For completeness, we also include the *Tamper-Evident Counter (TEC)* tree [38] It has a large memory overhead, and requires a wide encryption mechanism with a very high latency. This makes it unattractive for practical deployment.

In Table 1 we compare the memory overheads of various integrity tree implementations. We assume that the size of a counter group is a cache line including any embedded tags, and a MAC can cover 1, 2, or 4 cache lines. When multi-cache line MACs are used, each cache line is still encrypted individually and is associated with a monolithic or minor counter. Hence, evicting a cache line from the last level cache will not require the re-encryption of adjacent cache lines.

Cryptographic parameters. In the choice of parameters such as key and MAC lengths, the fundamental difference between encryption and authentication is that the encryption parameters must provide long term confidentiality whereas authentication only needs to deter an adversary. A system can monitor unrecoverable integrity violations and take corrective actions (e.g., destroy sensitive state and shut down) if such events are detected. Hence, we recommended that:

- Encryption keys should be at least 128b long. We note that even on a quantum computer, the complexity for a quantum computer assisted key search against AES-128 has a hardware cost of 2^{84} quantum gates and a time complexity of $1.09 \cdot 2^{75}$ quantum cycles [39]. This makes AES-128 sufficiently strong even against adversaries with access to a large-scale quantum computer. Intel's SGX and TDX, and AMD's SEV use AES-128 in modes that need two independent 128-bit keys.
- Encryption block sizes must be at least 128b;

Table 1: Memory Overhead of Various Types of Integrity Trees at 32b and 64b security levels

Type of Tree *cache line size:*	Overhead *64B*	*128B*
Merkle Tree with $a = 4$, resp. 8	33.3 %	16.7 %
Monolithic Counter Tree with embedded MAC, $\ell_c = \ell_h = 56$		
• $\ell_H = 64$; $n = 1$; $a = 8$, resp. 16	26.8 %	12.9 %
• $\ell_H = 32$; $n = 1$; $a = 8$, resp. 16	20.5 %	9.79 %
• $\ell_H = 32$; $n = 2$; $a = 8$, resp. 16	17.4 %	8.23 %
• $\ell_H = 32$; $n = 4$; $a = 8$, resp. 16	15.8 %	7.45 %
Split Counter Tree with embedded MAC, $\ell_c = \ell_h = 56$		
• $\ell_H = 64$; $n = 1$; $\ell'_c = 6$, resp. 7	14.1 %	7.04 %
• $\ell_H = 32$; $n = 1$; $\ell'_c = 6$, resp. 7	7.84 %	3.91 %
• $\ell_H = 32$; $n = 2$; $\ell'_c = 6$, resp. 7	4.71 %	2.34 %
• $\ell_H = 32$; $n = 4$; $\ell'_c = 6$, resp. 7	3.15 %	1.57 %
• $\ell_H = 32$; $n = 1$; $\ell'_c = 3$	7.04 %	3.52 %
• $\ell_H = 32$; $n = 2$; $\ell'_c = 3$	3.91 %	1.95 %
• $\ell_H = 32$; $n = 4$; $\ell'_c = 3$	2.35 %	1.17 %
PAT with $a = 8$, resp. $a = 16$	28.6 %	13.3 %
TEC tree with $a = 8$, resp. $a = 16$	42.9 %	20.0 %

Legend: *$\mathscr{L}_{CL}$, ℓ_H, ℓ_h, ℓ_c, and ℓ'_c are the bit lengths of a cache line; a data hash or MAC; of a embedded hash value or MAC; of a monolithic or major counter; and a minor counter, respectively. a is the arity of a counter group, i.e. the number of its monolithic or minor counters; and n is the number of cache lines a MAC covers.*

- For a traditional hash-based Merkle Tree the required hash length is 128b;
- Authentication keys should be at least 128b long;
- Data MACs should be at least 32b long (28b in a MirE configuration); and
- Monolithic counters must be at least 56b long. The minimal aggregated length of a major and a minor counter (or major plus middle plus minor) is also 56b.

Intel's SGX use 56b MACs and 56b counters. With these parameters, a successful replay attack on its memory of would need both the counter and the MAC to be repeated, with time $2^{56} \times O(2^{56/2}) = O(2^{84})$. Since, however, the purpose of memory integrity is to make the attack non-viable, we believe that halving the data MACs memory requirement (and using, say, 32b MACs) is sufficient: with 64b worth of counters (which is achievable with split counters, as we shall see) and 32b data MACs, corruption has likelihood 2^{-32} and replay needs time $2^{64} \times O(2^{32/2}) = O(2^{80})$.

4. Setup of the study

4.1. Scope of the comparisons

Depending on the level, several variants of the involved technologies can be combined. We summarize the variants we compare in the following list. The entries marked with † contain new contributions in this paper, while ⋆ denote variations not hitherto compared to each other.

1) Use of the AES-128 or QARMA-128 ciphers;
2) Size of the MACs (32b or 64b);⋆
3) Counter trees: monolithic, or split;
4) Various choices for the size of counter and hash caches.
5) Use of on-chip memory for hashes and/or counters;†
6) Repurposing of ECC bits for data MAC storage;
7) Synchronous or asynchronous integrity checking;
8) Use of single MACs covering multiple cache lines, with cached incremental hashing;†
9) Arity variations in the counter groups;
10) We consider both 64B and 128B cache line sizes;⋆ and
11) We run the benchmarks as the only running tasks, i.e. on an *unloaded* system, and also under extreme memory bus contention, i.e. on a *loaded* system.⋆

We assume that all algorithms are parallelised wherever possible (i.e., there are sufficient instances of the relevant building blocks to attain the lowest possible latency of the whole scheme). This is not a significant restriction in our study. As we shall see, the fastest L2 and L3 schemes use OTP encryption and have low sensitivity to the latency of the encryption and authentication primitives (for instance, the performances of the variants with AES and QARMA are quite close). Hence, their performance even on pipelined implementations would be similar.

4.2. Technologies used for each level

We list the technologies used to implement the protection levels defined in Section 2.4.

L1 If AES-128 is the chosen encryption primitive, a cache line is encrypted using the XEX construction, with the PA as the tweak. If QARMA-128 is chosen, it is used in Tweaked *Electronic Codebook (ECB)* mode as in Fig. 4, with the PA as tweak.

L2 The same encryption modes are used as for L1. Hashing is done by a *multi-linear* UHF [29] at 32 or 64 bits. The hashes are encrypted block-wise when they are evicted from the hash cache in cache line-sized groups. For the security and reliability implications, cf. Remark 4.1.

L3 First of all, this level provides freshness over L2. The freshness information must be included in the tag computation. A counter based OTP encryption mode is used with both AES and QARMA, except for ELM, which uses FLAT-OCB. Replay protection is provided as well, by including the counters in the tag computation and preventing the adversary from tampering with the counter groups. Thus, an adversary may still be able to replace a cache line and its MAC, but not its counter(s). This is commonly achieved by using an integrity tree, but there are other options, as follows:

LoC One such option is storing the counters in a in-package tamper proof DRAM (an SRAM would be too large), where the counters are assumed to be MPE private, i.e. outside adversarial control. The latter solution is denoted by LoC which stands for *Leaves-on-Chip*, denoting the

fact that, of the original tree, if we store the leaf nodes on chip, then we do not need to keep any other node.

BoC A less expensive version of the LoC solution consists in keeping the leaf nodes in external memory, and store on chip the level of nodes immediately above. We call this BoC, i.e. *Branches-on-Chip* – again, once the level immediately above the leaf one is on chip, the system need no other node levels above.

MirE *MACs in repurposed ECC bits*. This eliminates the need to reserve memory for the MACs, and only memory for counters needs to be allocated, if not on chip as well. This also reduces memory bus traffic. Note that MACs are still accessible to a HW capable adversary. Hence, freshness information, if available, *must* still enter the MAC computation. Following [11], the tag is computed using $\texttt{QARMA}_5$-64-σ_0. *Note that not all the ECC bits need to be repurposed for a MAC: these bits may contain both a shorter ECC and a MAC.*

MirE raises the question of the performance impact of using ECC memory. Because of expanded traffic and extra processing in the DRAM controller, there are penalties which have been reported as smaller than 0.5% [40]. On servers, schemes that do not repurpose the ECC bits will still be using them for error detection and correction, so memory access time will not vary. In all other cases, we consider the impact of ECC memory to be so small as to not significant change the performance relative to baseline. Hence, we do not take into account ECC memory as a separate configuration.

Remark 4.1. A new idea we adopt here applies to the cases where UHF-based MACs are stored in external RAM. We keep them as *hashes* in their cache, speeding up verification w.r.t. caching MACs. Also, *they are evicted in groups, which are encrypted directly*. For instance, four 32b hashs are encrypted as a single 128b block. Corrupting one hash will corrupt all hashes in the group with high probability, increasing the detectability of any corruption, and with it both security and robustness of the system. If freshness is available, the minor counters associated with the hashs in the same block are grouped and concatenated to their common major counter (and, if implemented, also the middle one) to form a tweak.

4.3. Benchmarking environment and methodology

It would be impractical to implement several thousands of combinations of technologies in silicon for the purpose of evaluating them. A solution to this problem lies in prototyping, i.e. the creation of an approximate implementation of the desired features that can thus be tested, and benchmarked. Very accurate models can be created even without implementing all details. For instance, the latencies of cryptographic primitives can be derived from actual implementations and inserted as delays into the simulation.

The prototypes used in this paper are built in the gem5 simulator [6], [7]. gem5 allows engineers to build software versions of hardware components typically included in computer systems. gem5 also helps abstract away the interfaces between components, which can thus be combined programmatically and configured at run-time. It includes approximate timing models for several processor cores.

The processor is modelled as an approximated Arm Cortex A72-like core, with a 2GHz frequency and a 1GHz system frequency. The cache hierarchy includes L1-I (48KiB, LRU replacement policy, 3-way set associative, 1 cycle latency) and L1-D (32KiB, LRU replacement policy, 2-way, 1 cycle latency) caches, and a unified L2 cache (1MiB, tree-PLRU replacement policy, 16-way, 5 cycles latency). The memory is 16GiB DRAM in a dual-rank DDR4 DIMMs. The MPE-private caches are 4-way set associative with an LRU replacement policy.

We assume that the SoC is implemented in a 7nm process. Thus, we can re-use the latencies from [25], for instance a latency of 15.76ns for a pipelined implementation of AES-128, of 4.8ns for $\texttt{QARMA}_{11}$-128-σ_1 and 2.2ns for $\texttt{QARMA}_5$-64-σ_0. This latency of $\texttt{QARMA}_5$-64-σ_0 is also used in [11].

Our evaluation uses the SPEC 2017 [5] benchmark suite. Detailed software models such as gem5 typically increase benchmark execution time by several orders of magnitude. As shown in [41], a typical SPEC benchmark could take around a month to run. To make rapid prototyping and analysis feasible, we use the SimPoint [42] methodology. This methodology uses clustering to find a number of representative regions that serve as a proxy for the whole application. After simulating these regions, their results are combined using a set of weights that signify how important a region is to the application as a whole. Instead of sequentially simulating several billions of instructions per benchmark, we simulate 10 SimPoints of 30 million instructions from each benchmark. This both decreases the number of instructions we need to simulate and improves parallelism.

An alternative approach could have been to run the entire benchmarks programs, as opposed to SimPoints, in parallel on a large distributed cloud. Even in this case, a single program could have taken several days to run, providing no practical advantages and only significantly increasing the cost of the simulations. A quicker turnaround also allows to more effectively identify and correct bugs in the code.

In both cases of full or SimPoints based simulations, we may ask ourselves about the impact on systems that include context switches, virtual memory swap, any I/O including network communications. These are aspects that only a fully implemented system deployed in a real world production environment can answer, and are very difficult to emulate. In fact, benchmarking in such a context is nearly absent from the literature. We make the following observations:

1) The increased memory footprints of the solution with smallest memory overhead (1:128 and less) should not have a noticeable effect on paging even if the metadata is pinned in physical memory;
2) It can be argued that context switches, paging, and general I/O are affected by the performance penalties on memory accesses only in a minor way: context switches can operate on pinned memory, and the timing of disk, network operations is dominated by media which are orders of magnitude slower than physical RAM.

Therefore, any performance penalty we discuss here is most likely an upper bound to real-world penalties.

5. Benchmarking Plan, Results, and Discussion

The space of all possible MPE configurations spans a vast multi-dimensional space. Exhaustively evaluating all of the possible configurations is clearly infeasible, not to speak of the difficulties of properly presenting the data. For this reason, we have planned a tour through the design space consisting in various stations, each consisting in a *set* of benchmark runs. Each set builds on some previous configuration by expanding the parameter space or introducing some new technique.

We use shorthands to describe the various configurations:

Level / {additional technologies} / Cipher / / cache line length / MAC length .

The optional "additional technologies" may include: monolithic counters (mono), split counters (split), Leaves or Branches on Chip (LoC or BoC), or the use of MACs in Repurposed ECC bits (MirE).

The default cache line length is 64B, unless the counter groups are on chip, in which case it is 128B. The default MAC length is 56–64b.

"{Intel} TDX" is equivalent to L2/AES/MirE, "{Intel} SGX" to L2/AES/mono, and "{AMD} SME" to L1/AES. oCC always implies split. The shorthand L3/LoC is used to denote the version of L3 that uses LoC, and thus no integrity tree. Similarly, L3/BoC is a L3 solution with the leaf counters off chip and the next level on chip, also without a full tree. L3 *without* BoC or LoC denotes by default a replay protection capable scheme based an on integrity tree and *neither counters nor hashes on-chip*.

General Remarks. Without memory protection, our benchmarks run 13.3% slower on a loaded system with 64B cache lines than on an unloaded system. Changing cache line length from 64B to 128B slows down our simulations by roughly 1% in unloaded systems and roughly 2.7% on loaded systems. In all cases, runs are always compared to the baseline with the same cache line.

Unloaded vs. Hald-loaded vs. Loaded Systems. All benchmark runs are first run on an *unloaded* system, where the current benchmark is the only running task.

We then want an upper bound for the performance degradation in a fully *loaded* system, with up to hundreds of processes running on dozens of processing elements, all sharing the bandwidth of the memory subsystem, such as in a cloud server. Directly simulating such a system is very complex and impractical. We instead inject synthetic traffic upstream of the MPE, but after the L2 cache. This traffic amounts to 8 GiB/s. It is obtained from the measurements reported in Fig. 6 and it corresponds to the point where the latency of the memory subsystem just starts to diverge for a SGX-like L3 MPE covering the entire memory, while handling mostly linear traffic. We assume that MAC verification is synchronous because, following the discussion of the benchmark runs in the next section, this will be the most likely implementation. The simulated traffic is a mix of linear and random accesses. We do not add a L3 cache to the system, in order to simulate the extreme situation where the latter has been completely swamped by traffic coming from other requesters or clusters of requesters.

Fig. 6 suggests that there one can expect some interpolation of the performance penalties between the unloaded and loaded cases depending on the load of the system. This will be considered by taking into account also half-loaded systems. Beyond that we expect a catastrophic deterioration of performance, which occurs also in systems without cryptographic memory protection hardware, at which point a, say, cloud provider would migrate VMs in order to balance the bandwidths and thus minimise performance degradation. This would bring also the MPE penalties under control.

For this reason, we also run the benchmarks with an additional traffic on only 4 GiB/s, to give an idea of what happens with in-between loads ("half-loaded" system).

A note on short minor counters. In order to make our simulations as realistic as possible, in all split counter runs we initialise the counters to random values. This way the benchmarks will not be advantaged w.r.t. real world software, or other benchmarks that write less often to the same locations, by the fact that they start with zero counters that are not expected to overflow for a longer time. In a real world setting, the user cannot expect that the integrity trees will be initialised in a favourable way. In fact, this configuration choice magnified the performance difference between 3-way and 2-way split counters of the same arity, highlighting the advantage of 3-way split counters.

We now report and discuss the results of all the runs.

5.1. Set 1: State of the Art

We start with the state of the art, as well as some simple variations thereof in order to guid us in the next stges.

We compare L1/AES (e.g., AMD SME), L1/QARMA, L2/AES, L2/AES/MirE (e.g., Intel TDX), L2/QARMA, L2/QARMA/MirE, and ELM with both monolithic and split counters, SGX, L3/QARMA/split – all with and without a hash cache if not fixed by the manufacturer's architecture.

We also compare 32b and 64b MACs in selected cases.

For SGX, hash encryption is OTP as described by Intel [3]. We use this method for the SGX's split counters variant (L3/AES/split) as well, and in any L3 scheme where the published architecture prescribes its use. In all other cases, data MACs are replaced by 32b long hashes which are directly encrypted in groups of four upon eviction.

The ELM method follows [16] except when QARMA is used, in which case the *XOR and Encrypt (XE)* constructions are replaced by simply feeding nonces and separation fields as the tweak to QARMA, as well as using $\texttt{QARMA}_5$-64-σ_0 to generate the OTPs to encrypt the tags.

For schemes with freshness, the counter cache is 64KiB as in SGX to level the comparisons.

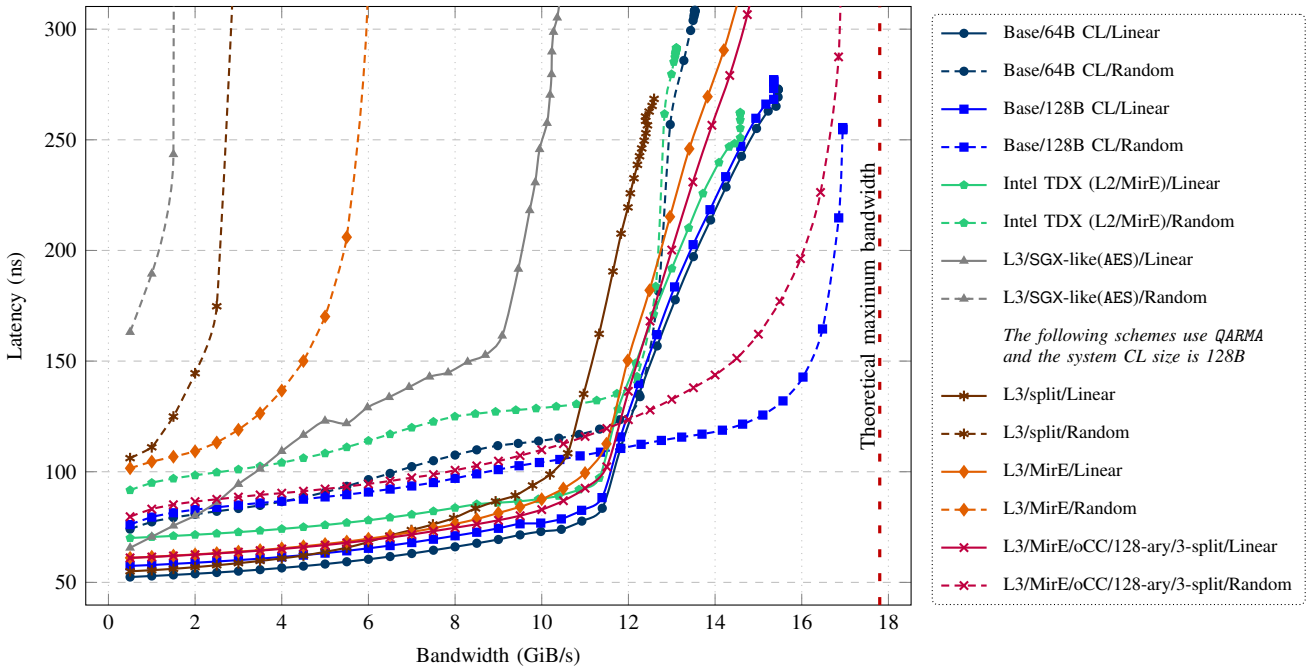

Figure 6: Bandwidth/latency plot with various MPEs and without, for linear or random synthetic traffic

These principles apply to every successive set as well, except where explicitly indicated otherwise.

A first look at the results in Fig. 7 shows that:

- In a plain implementation, the performance penalty increases with the protection levels.
- The performance of L1 and L2 schemes improves if we replace the AES with QARMA because of the latter's lower latency. This also holds for L3 because the OTP generation, while it can be performed in parallel with a memory fetch, still affects write latency to the point that it bears noticeable effect.
- Split counter trees are superior to monolithic trees in memory overhead (see Table 1) and performance.
- A small hash cache has a minor, yet not always negligible, effect on performance. This is perhaps the first third-party confirmation of the wisdom of not including one in SGX since it was meant for isolated tasks and running out of a restricted memory range.
- ELM has a higher performance penalty than SGX, having the encryption primitive on the critical path.
- As expected, using 64b MACs results in slightly worse performance than using 32b MACs.

For the remainder of the evaluation, we assume that MACs are 32 bits long and directly encrypted in groups of four with the following exceptions: when SGX is benchmarked, MirE is used, or otherwise explicitly indicated.

5.2. Set 2: Impact of MPE Cache Size

We study the impact of the sizes of the two MPE caches, the hash cache and the counter group cache. Since L1 does not need caches, we only consider L2 and L3.

The possible hash cache sizes are 4KiB, 16KiB, and 64KiB. The possible counter cache sizes are 16KiB, 64KiB, 256KiB, and 1MiB. The presented results use QARMA for encryption as the AES results display an identical pattern.

We confirm the expected significant performance gains with larger MPE caches, the counter cache having a higher effect than the hash cache. The improvement gets more significant as the load of the system increases (see Fig. 9).

Starting with Set 3, the MPE has a 16KiB hash cache and a 256KiB counter cache. Level L3 uses split counters, unless explicitly indicated otherwise, or with SGX.

5.3. Set 3: Impact of the Cache Line Length

We consider 64B and 128B cache lines for L2 and L3. It is assumed that the counter group and cache line sizes are the same. Note that the results of Sets 3 and 4 are combined in Fig. 10. We see that the relative impact of memory protection is comparable across systems with 64B cache lines and systems with 128B cache lines.

5.4. Set 4: Asynchronous MAC Verification

We compare synchronous to asynchronous verification. L1 is out of scope and we consider only L2 and L3. The results are displayed in Fig. 10. Whereas in the unloaded case asynchronous MAC verification does not significantly improve performance, in the loaded case the speedup is substantial. Indeed, as the memory bus approaches saturation, decoupling decryption and MAC verification logics allows for better scheduling of otherwise idle MPE resources

In the following, we run only benchmarks with synchronous verification.

5.5. Set 5: Use of on-Chip Memory for L3

We analyse the use of on-chip memory in integrity and replay protection schemes. As MACs/hashes have a larger memory overhead than counter groups, we do not consider the case where the hashes are on-chip and counters off-chip.

The results Fig. 11 confirm that relieving the contention on the memory bus between data and metadata reduces performance penalties in the unloaded and half-loaded cases. However, the loaded system results go against the intuition that using longer cache lines should *always* perform better because of reduced metadata traffic. On the contrary, the less fine-grained caching with 128B cache lines increases the overall data traffic. With all metadata on chip, performance is close to the baseline, as expected. AES and QARMA results are very close being the cipher not on the critical path.

An important observation here is that L3/BoC performs always better than L3 even if not as good as L3/LoC. This is because in L3, the leaf counter group level has non optimal locality properties. On a half-loaded system BoC performance penalty may be acceptable if restricted only to some services.

5.6. Set 6: Impact of Repurposing ECC Bits, 3-way Split Counters, and Large Counter Caches

The deployment of Intel TDX MKTMEi [4] and [10] prompt us to study how using ECC bits for tags affects performance. We run only L3 schemes L3/LoC, and L3/TBoC, with and without MirE, since L2 schemes with MirE are reported in Fig. 7. We consider 128B cache lines only. With the MirE option a hash cache is not needed, and MACs are computed as PMACs.

In this set we also introduce 3-way split high-arity 128B counter groups. These configurations are all tested with MirE and for 128B cache lines. The minor counters in the 256-ary counter groups cannot be longer than 3b, and the major counter does not need to be larger than 64b, so this allows us to fit 32 6b middle counters. The purpose of fitting middle counters is to keep the amount of RMW operations under control. To quantify the impact of this optimization, we also run the tests without middle counters. The 3-way split counter groups we use are:

- 128B cache lines and counter groups with: 128 $\times$ 7b minor, 8 $\times$ 8b middle, and 1 $\times$ 64b major counters; This results in a memory overhead of 1:128.
- 128B cache lines and counter groups with: 256 $\times$ 3b minor, 32 $\times$ 6b middle, and 1 $\times$ 64b major counters; This results in a memory overhead of 1:256.

In [10] and [43] "delta encoded" split counters with rebasing are used to reduce the amount of RMWs, together with methods to accommodate a limited number of larger minor counters in a counter group. In both papers, the reduction in RMW overhead seems smaller than when using our 3-way split counter groups (cf. Set 9), so we do not consider these optimisations.

The results (Fig. 12) prove that combining LoC and MirE provides the highest protection level at very low performance. In fact, this is the only combination of techniques that can yield nearly negligible performance penalties on a loaded or half-loaded system. Middle counters are instrumental in getting the best performance out of the high-arity counter groups, which otherwise would would incur in very large RMWs overheads. The resulting schemes perform even better than L1 direct encryption – because the effect of latency of the cipher on the critical path is amplified with a saturated memory subsystem, but L3 and variants have no cipher block on the critical path. If the memory subsystem is expected to be fully saturated only in some circumstances, in a L3/MirE system BoC can be considered in place of LoC, as the performance penalty can still be acceptable, and the hardware cost is significantly smaller.

The use of a large cache for the counters shows interesting behaviours. On a fully loaded system, LoC performance is reached in practice only when the cache is large enough be able contain large parts of the tree. In our example, with 16GiB of RAM, this seems to be achieved between 1MiB and 2MiB, and only then the performance penalty suddenly drops to LoC levels. This would be impractical on a large server – such machines can be configured today with up to 8TiB of RAM. However, when the load on the bus is moderate, performance seems to improve with the cache size

We note that the speed of the counter cache is not critical: the counters just need to be available to the MPE before the data from RAM. So, a slower, smaller, and less expensive DRAM can be used to implement this cache (the same holds for hash caches).

5.7. Set 7: Impact of incremental MACs

If we cannot store MACs in the ECC bits or on-chip, there is another option for reducing their external memory footprint: to compute them incrementally over multiple cache lines [44]. This makes sense only if we are using 128B cache lines, as these already reduce metadata storage requirements by a factor of two. We test L2 and L3, L3/LoC, and L3/BoC, with a MAC covering 1, 2, or 4 cache lines. These runs are reported only with QARMA-128 as the encryption cipher, since the performance differences are caused only by the increased memory traffic, and therefore we can expect that performance with the AES will follow the same pattern. Multiple-cache line MACs effectively reduce memory overheads, but at a significant performance price, as shows in Fig. 8.

Plaintext compression can be used to store a MAC in the corresponding cache line together with the data if the latter can be sufficiently compressed, thus reducing the amount of memory accesses. Indeed, the performance of an incremental hashing scheme may be improved somewhat [45]. However, an attacker capable of monitoring memory subsystem transactions would be able to infer properties of the data just by its compressibility [46], [47], defeating the purpose of using memory encryption in the first place – all while adding significant complexity, such as maintaining compressibility information. Thus, we have decided not to considering it.

5.8. Set 8: Detailed Breakdown of the Performance in Selected Configurations

We select some combinations from the above and show all individual benchmarks in the suite:

- AMD SEV and L1/QARMA, with 64B cache lines;
- Intel TDX/64B CLs (i.e. L2/AES/MirE);
- L2/QARMA/64B CLs, with off-chip 64b MACs and MirE;
- Intel SGX (i.e. L3/AES/56b MACs);
- L3/QARMA/oCC/MirE, with 128- and 256-ary 3-way split counter groups.

The performance of the individual SPEC2017 benchmarks (cf. Figs. 13 to 18) shows a few expected results, namely that some programs such as `omnetpp`, `mcf`, and `bwaves` suffer significantly more than average under most MPE configurations. Increasing integrity tree arity by means of split counters is key for an initial reduction of the penalties, but it is only with 3-way oCC and MirE that L3 penalties can be pushed to be smaller than 5% for most benchmarks. The only difference between unloaded and loaded systems is the degree of amplification of the performance penalties.

5.9. Set 9: Impact of RMW Operations

Here we compare the performance of an MPE with a hypothetical one where the RMW operations have zero cost, i.e. are instantaneous. This is achieved by simply skipping them. Such an experiment is possible because the simulated MPE does not actually perform cryptographic operations, simulating instead their timing delays. This gives an upper bound on the actual time spent performing RMW operations. We select the last five combinations of S8, L3 schemes with split counters, i.e. the only ones with RMWs. For the schemes with 3-way split counters, we also report the performance with 2-way split counters by omitting the middle counters, to show the gains brought by the 3-way splitting (which, as far as we are aware, have not been reported before).

The results are in Figs. 17 and 18. We notice that the impact of RMWs is not always negligible. Using 2-way split counters with 3b minors (L3/QARMA/oCC/MirE with 256-ary counter groups) carries a significant performance penalty, but the use of middle counters brings the performance close to the ideal one with "free" RMWs.

Note that the performance penalties and the proportion of time spent doing RMWs increase with the laod. This suggests that further research to RMWs may benefit loaded systems. However, even in this case the penalties with 256-ary, 3-way split counter groups are smaller than with a direct encryption L2/AES/64B CLs/MirE scheme as in TDX.

5.10. SPEC 2006

As mentioned in the introduction, we ran also the SPEC 2006 benchmark [8]. The results display the same behaviour as the SPEC 2017 ones, with the penalties being roughly the same, except for the most extreme technology combination. For the L3/*/oCC/MirE group, performance penalties are smaller than 1%, resp. 2%, with a memory overhead of 1/128, resp. 1/256. This result is more interesting for hardened client er edge devices, whereas the SPEC 2017 better reflects the costs to be expected on a server.

5.11. A remark on area and power

The area of the MPE mostly comprises cryptographic circuits, caches, and any internal DRAM for counter storage, if present. The control circuitry, and possibly Galois multipliers, which are small in comparison to the block ciphers.

There is extreme variability in the parameters. For instance, in order to squeeze the maximum performance from schemes based on direct encryption, such as L1 and L2 schemes, the implementer may use several encryption blocks in parallel for encryption and for the integrity PMAC. Sacrificing some latency, pipelined implementations can be used to save area. Following [25], also the implementations of the AES and of QARMA may vary a lot, but for a single MPE we can estimate between ≈50KGE (gate equivalents) for a pipelined encryption circuit based on QARMA, optimised for area, and ≈ 800K GE for 8 parallel instances if optimised for latency in order to encrypt a whole 64B cache line at the same time. AES areas are much bigger, with optimised implementations exceeding 17K GE *per round* [48], so a full latency optimised implementation can be ≈ 170KGE, and 8 such blocks in parallel would use an area of ≈ 1.3MGE. The pipelined QARMA circuit and the full parallelised AES circuit would hovever have comparable latency.

Integrity tag computation and verification can re-use the same blocks used for encryption, or much smaller ones based on $\texttt{QARMA}_5$-64-σ_0, as in [11].

Most moderately sized caches are built from SRAM and the area is roughly 3 GE per cache bit (recall that 1 GE is the area of a NAND gate, i.e. two transistors), so a 4KiB cache is about 100KGE, and a 256KiB cache is roughly 6MGE. We can expect that if resources are balanced, the caches will be between 2 and 6 times the area of the encryption blocks.

Even one large MPE per memory channel represents a minor, but not negligible, amount of area for a modern SoC, that can include a few to several billion transistors. Architects and implementers need to carefully consider the various tradeoffs. The cost of DRAM memory included in the package or module to store entire counter tables is minor with respect to the total memory of the whole system, but it cannot be ignored, especially since a tamper proof or detecting multi-chiplet design bears its own additional costs.

Besides these considerations, it is unfeasible to provide area estimates for all configurations. In general, the energy consumption of an MPE seems to be a minor contribution to the total power envelope. Still, designs like QARMA help bring the latter further down, as the most energy consuming components in the MPE are the encryption primitives: a single pipelined implementation of QARMA-128 can process eight blocks in the same time as eight parallel AES-128 blocks, and this can make a significant difference in OTP based systems which are less sensitive to cipher latency.

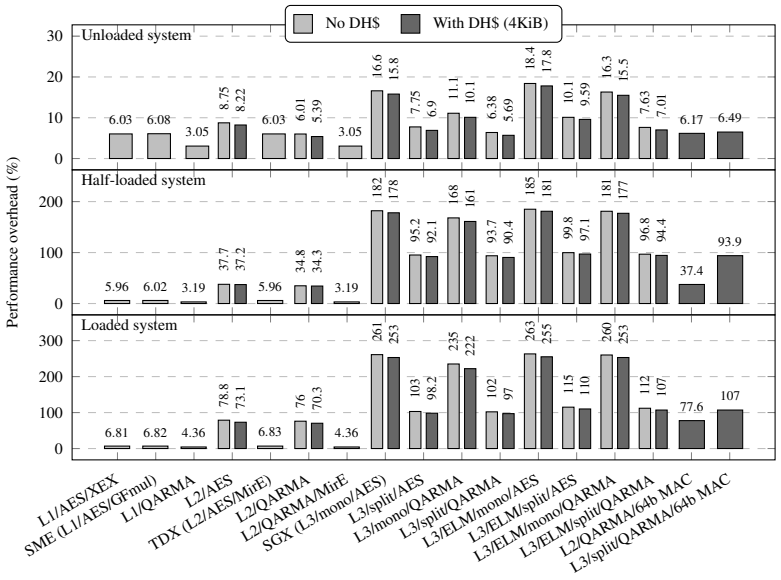

Figure 7: Set 1: Comparison of base levels and state of the art

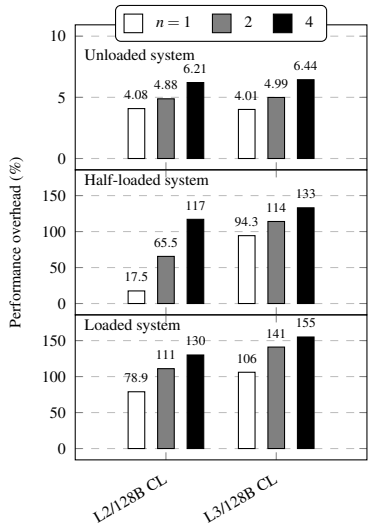

Figure 8: Set 7: Impact of incremental MACs

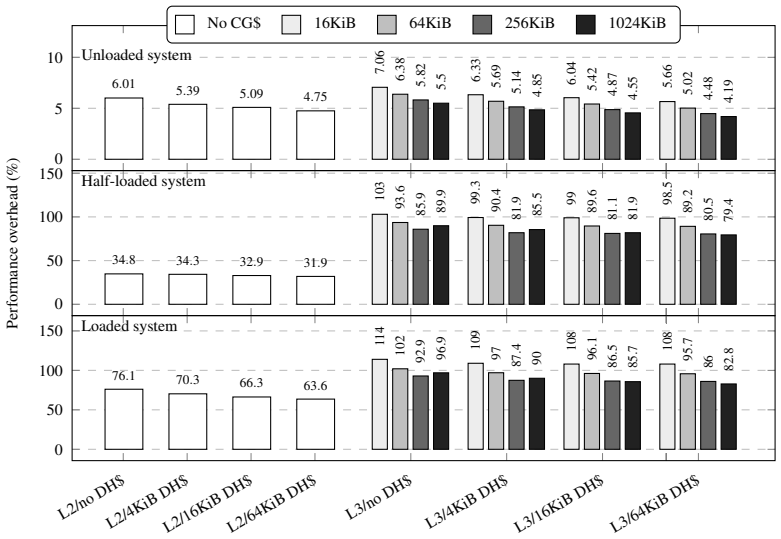

Figure 9: Set 2: Impact of MPE cache sizes;
the memory encryption cipher is `QARMA`-128; cache lines are 64B

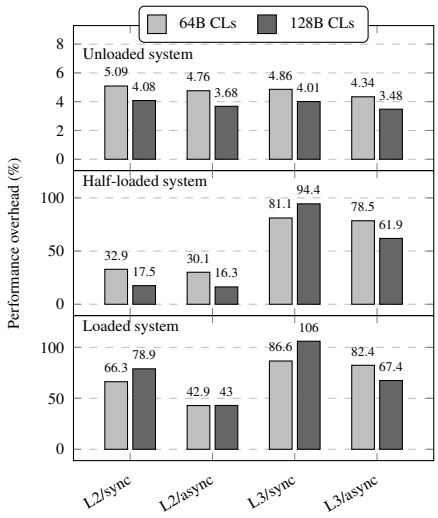

Figure 10: Set 3 and **Set 4**: Impact of cache line size and asynchronous MAC verification

6. Conclusions

We performed a thorough survey and evaluation of techniques for the cryptographic protection of in-use memory contents. This included the state of the art, some new technologies, and hitherto not considered combinations thereof. By doing this, we also answer two implicit open questions in [16, VII.A] regarding the performance of ELM with more lightweight primitives in place of the `AES`.

We also unified the evaluation of different protection levels, selected according to adversarial models.

This results in a vast set of mutually independent choices, for each of which different types of hardening may be deployed, with correspondingly different prices in term of performance penalty, memory overhead, and hardware cost. The lack of an absolute metric to combine these three costs in a single rating makes it challenging to provide recommendations that may be suitable for different applications. Therefore, the extensive set of benchmarking runs we document should be used as a guidance for further investigations. This said, we can provide rough indications for some use cases.

For simplicity, let us restrict to `L3` memory protection.

We first consider the use case of cloud computing. Server SoCs are expensive: they may contain dozens of cores, have multiple memory channels and can easily address hundreds of GiBs of physical memory. Because of the very high total system costs, we have an argument for OTP encryption

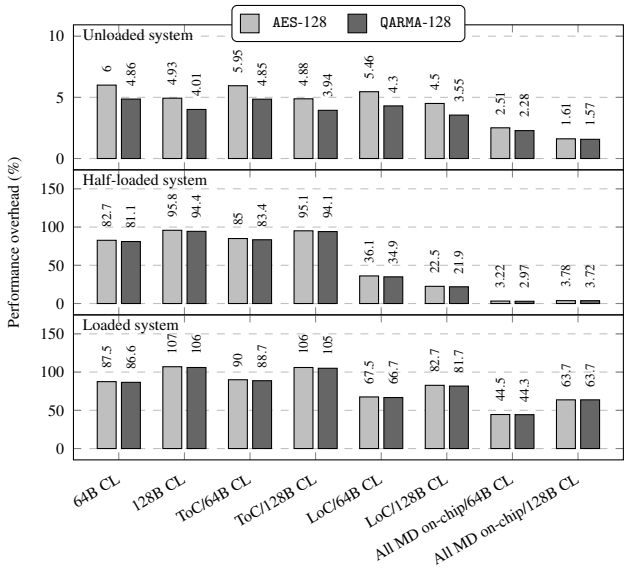

Figure 11: Set 5: L3; Impact of storing metadata on-chip

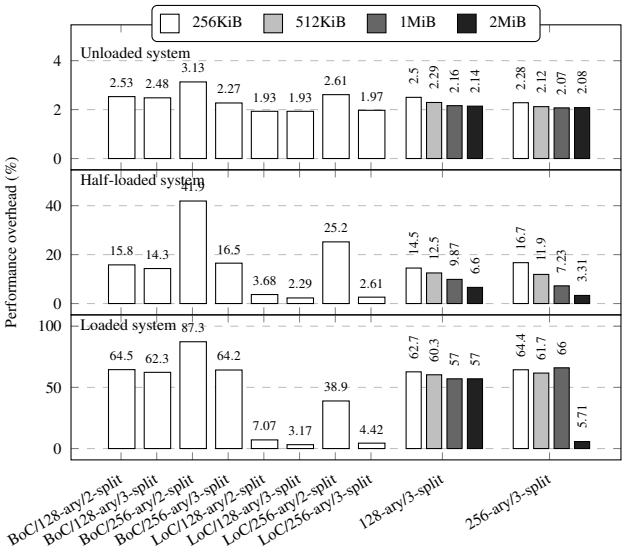

Figure 12: Set 6: Impact of repurposing ECC bits for MACs and of large counter caches on L3/MirE schemes

with counter groups in on-chip DRAM in the amount of 1:128 or 1:256 of the external memory. The additional cost for counter group storage would be *relatively* minor, but, combined with the re-use of ECC bits for MAC storage, it would enable the highest level of memory protection at a lower performance impact than currently deployed schemes without replay protection. It would also be likely less expensive than basing the protection of local memory on the `CXL.memory` *Integrity and Data Encryption (IDE)*.

It can be argued that the budget for a large amount of on-chip memory should rather be spent on large system caches, from which the whole system benefits, *with counter groups getting a similar cache hit rate as in the unloaded case*. It appears reasonable that with suitably scaled-up system caches, and multiple interleaved memory channels, a L3/MirE system running the various SPEC2017 tasks concurrently on separate cores should show the overall performance of an unloaded system running a single task. Even for HPC jobs such as large linear algebra problems, with vectors or matrices that do not fit in, say, 32MiB worth of system cache, in many cases the corresponding counter groups could still fit in the system cache, suggesting for, instance, that dynamically partition the system cache between system and MPE may be a good option.

But one can also go further: if placing 64GiB or more of DRAM in a module close to the main SoC, on a common substrate (which could be made tamper proof), is today feasible for a client device, one can easily imagine to use such a large DRAM based as a cache in a system with several TiBs of memory. For all but some exceptional loads, cache evictions would be a rare event and therefore also a simpler MPE with MirE and an integrity tree off-chip would probably have a minor impact. The cache could be encrypted to reduce side-channels, whereas full integrity would only be added on eviction.

For client devices, MirE is often not applicable, as their memories usually lack ECC bits. For use cases such as security modules and business oriented containers, memory bus saturation is an exceptional event, often caused by a single application. In this case, we expect performance penalties in line with the most demanding tasks benchmarked on unloaded systems, and we recommend the use of high arity split counter trees with counters either on-chip or in a dynamically allocated carveout together with the MACs.

The main takeaway from our study is that nearly-transparent strong memory protection is possible with current technology.

We further observe that data structures and the organisation of integrity trees play a larger role in determining overall performance than the chosen cryptographic primitives – including the case of a bandwidth bound system, where the data structures have a direct impact on the traffic expansion.

Future work includes upstreaming our MPE framework into `gem5` – to allow interested parties to perform simulations tailored to their specific needs –, and research into further reducing the impact of RMWs.

References

[1] D. Kaplan, J. Powell, and T. Woller, "Amd memory encryption white paper," April 2016. [Online]. Available: http://amd-dev.wpengine.netdna-cdn.com/wordpress/media/2013/12/AMD_Memory_Encryption_Whitepaper_v7-Public.pdf

[2] D. P. Mulligan, G. Petri, N. Spinale, G. Stockwell, and H. J. M. Vincent, "Confidential computing - a brave new world," in *2021 International Symposium on Secure and Private Execution Environment Design (SEED), Washington, DC, USA, September 20-21, 2021*. IEEE, 2021, pp. 132–138. [Online]. Available: https://doi.org/10.1109/SEED51797.2021.00025

[3] S. Gueron, "A memory encryption engine suitable for general purpose processors," *IACR Cryptol. ePrint Arch.*, p. 204, 2016. [Online]. Available: http://eprint.iacr.org/2016/204

[4] Intel, "Intel® trust domain extensions white paper," August 2021. [Online]. Available: https://www.intel.com/content/www/us/en/developer/articles/technical/intel-trust-domain-extensions.html

[5] J. Bucek, K. Lange, and J. von Kistowski, "SPEC CPU2017: next-generation compute benchmark," in *Companion of the 2018 ACM/SPEC International Conference on Performance Engineering, ICPE 2018, Berlin, Germany, April 09-13, 2018*, K. Wolter, W. J. Knottenbelt, A. van Hoorn, and M. Nambiar, Eds. ACM, 2018, pp. 41–42. [Online]. Available: https://doi.org/10.1145/3185768.3185771

[6] N. L. Binkert, B. M. Beckmann, G. Black, S. K. Reinhardt, A. G. Saidi, A. Basu, J. Hestness, D. Hower, T. Krishna, S. Sardashti, R. Sen, K. Sewell, M. S. B. Altaf, N. Vaish, M. D. Hill, and D. A. Wood, "The gem5 simulator," *SIGARCH Comput. Archit. News*, vol. 39, no. 2, pp. 1–7, 2011.

[7] J. Lowe-Power, A. M. Ahmad, A. Akram, M. Alian, R. Amslinger, M. Andreozzi, A. Armejach, N. Asmussen, S. Bharadwaj, G. Black, G. Bloom, B. R. Bruce, D. R. Carvalho, J. Castrillón, L. Chen, N. Derumigny, S. Diestelhorst, W. Elsasser, M. Fariborz, A. F. Farahani, P. Fotouhi, R. Gambord, J. Gandhi, D. Gope, T. Grass, B. Hanindhito, A. Hansson, S. Haria, A. Harris, T. Hayes, A. Herrera, M. Horsnell, S. A. R. Jafri, R. Jagtap, H. Jang, R. Jeyapaul, T. M. Jones, M. Jung, S. Kannoth, H. Khaleghzadeh, Y. Kodama, T. Krishna, T. Marinelli, C. Menard, A. Mondelli, T. Mück, O. Naji, K. Nathella, H. Nguyen, N. Nikoleris, L. E. Olson, M. S. Orr, B. Pham, P. Prieto, T. Reddy, A. Roelke, M. Samani, A. Sandberg, J. Setoain, B. Shingarov, M. D. Sinclair, T. Ta, R. Thakur, G. Travaglini, M. Upton, N. Vaish, I. Vougioukas, Z. Wang, N. Wehn, C. Weis, D. A. Wood, H. Yoon, and É. F. Zulian, "The gem5 simulator: Version 20.0+," *CoRR*, vol. abs/2007.03152, 2020. [Online]. Available: https://arxiv.org/abs/2007.03152

[8] J. L. Henning, "SPEC CPU2006 benchmark descriptions," *SIGARCH Comput. Archit. News*, vol. 34, no. 4, pp. 1–17, 2006.

[9] M. Seaborn and T. Dullien, "Exploiting the DRAM RowHammer bug to gain kernel privileges," Talk at Black Hat 2015, 2016, https://www.blackhat.com/us-15/briefings.html.

[10] S. F. Yitbarek and T. M. Austin, "Reducing the overhead of authenticated memory encryption using delta encoding and ECC memory," in *Proceedings of the 55th Annual Design Automation Conference, DAC 2018*. ACM, 2018, pp. 1–35. [Online]. Available: https://doi.org/10.1145/3195970.3196102

[11] J. Juffinger, L. Lamster, A. Kogler, M. Lipp, M. Eichlseder, and D. Gruss, "CSI:Rowhammer – Cryptographic Security and Integrity against Rowhammer," in *Proceedings of the 44th IEEE Symposium on Security and Privacy, S&P'23*, 2023.

[12] M. A. Khelif, J. Lorandel, O. Romain, M. Regnery, D. Baheux, and G. Barbu, "Toward a hardware man-in-the-middle attack on pcie bus," *Microprocess. Microsystems*, vol. 77, p. 103198, 2020. [Online]. Available: https://doi.org/10.1016/j.micpro.2020.103198

[13] A. Barenghi, L. Breveglieri, I. Koren, and D. Naccache, "Fault injection attacks on cryptographic devices: Theory, practice, and countermeasures," *Proc. IEEE*, vol. 100, no. 11, pp. 3056–3076, 2012. [Online]. Available: https://doi.org/10.1109/JPROC.2012.2188769

[14] E. Blass and W. Robertson, "TRESOR-HUNT: attacking cpu-bound encryption," in *28th Annual Computer Security Applications Conference, ACSAC 2012*, R. H. Zakon, Ed. ACM, 2012, pp. 71–78.

[15] L. Zussa, J.-M. Dutertre, J. Clédiere, B. Robisson, and A. Tria, "Investigation of timing constraints violation as a fault injection means," in *27th Conference on Design of Circuits and Integrated Systems (DCIS), Avignon, France*, November 2012, pp. 1–6.

[16] A. Inoue, K. Minematsu, M. Oda, R. Ueno, and N. Homma, "ELM: A low-latency and scalable memory encryption scheme," *IEEE Trans. Inf. Forensics Secur.*, vol. 17, pp. 2628–2643, 2022. [Online]. Available: https://doi.org/10.1109/TIFS.2022.3188146

[17] Apple Inc., "Secure Enclave," 2020. [Online]. Available: https://support.apple.com/en-gb/guide/security/sec59b0b31ff/web

[18] G. Saileshwar, P. J. Nair, P. Ramrakhyani, W. Elsasser, and M. K. Qureshi, "SYNERGY: rethinking secure-memory design for error-correcting memories," in *IEEE International Symposium on High Performance Computer Architecture, HPCA 2018*. IEEE Computer Society, 2018, pp. 454–465.

[19] S. Skorobogatov, "How microprobing can attack encrypted memory," in *Euromicro Conference on Digital System Design, DSD 2017*, H. Kubátová, M. Novotný, and A. Skavhaug, Eds. IEEE Computer Society, 2017, pp. 244–251. [Online]. Available: https://doi.org/10.1109/DSD.2017.69

[20] R. Torrance and D. James, "The state-of-the-art in IC reverse engineering," in *Cryptographic Hardware and Embedded Systems - CHES 2009, 11th International Workshop, Proceedings*, ser. Lecture Notes in Computer Science, C. Clavier and K. Gaj, Eds., vol. 5747. Springer, 2009, pp. 363–381. [Online]. Available: https://doi.org/10.1007/978-3-642-04138-9_26

[21] B. Shakya, N. Asadizanjani, D. Forte, and M. M. Tehranipoor, "Chip editor: leveraging circuit edit for logic obfuscation and trusted fabrication," in *Proceedings of the 35th International Conference on Computer-Aided Design, ICCAD 2016*, F. Liu, Ed. ACM, 2016, p. 30. [Online]. Available: https://doi.org/10.1145/2966986.2967014

[22] S. Herschbein, S. Tan, R. Livengood, and M. Wong, "Focused ion beam (fib) for chip circuit edit and fault isolation," in *ISTFA 2021: Tutorial Presentations from the 47th International Symposium for Testing and Failure Analysis*, ser. International Symposium for Testing and Failure Analysis, November 2021, pp. h1–h113.

[23] S. Aga and S. Narayanasamy, "Invisimem: Smart memory defenses for memory bus side channel," in *Proceedings of the 44th Annual International Symposium on Computer Architecture, ISCA 2017*. ACM, 2017, pp. 94–106. [Online]. Available: http://doi.acm.org/10.1145/3079856

[24] J. Daemen and V. Rijmen, "Aes and the wide trail design strategy," in *EUROCRYPT 2002*, ser. Lecture Notes in Computer Science, L. R. Knudsen, Ed., vol. 2332. Springer, 2002, pp. 108–109.

[25] R. Avanzi, "The QARMA Block Cipher Family – Almost MDS Matrices over Rings with Zero Divisors, Nearly Symmetric Even-Mansour Constructions with Non-Involutory Central Rounds, and Search Heuristics for Low-Latency S-Boxes," *IACR Transactions on Symmetric Cryptology*, vol. 2017, no. 1, pp. 4–44, 2017. [Online]. Available: http://ojs.ub.rub.de/index.php/ToSC/article/view/583

[26] J. Borghoff, A. Canteaut, T. Güneysu, E. B. Kavun, M. Knezevic, L. R. Knudsen, G. Leander, V. Nikov, C. Paar, C. Rechberger, P. Rombouts, S. S. Thomsen, and T. Yalçin, "Prince - a low-latency block cipher for pervasive computing applications - extended abstract," in *ASIACRYPT 2012*, ser. Lecture Notes in Computer Science, X. Wang and K. Sako, Eds., vol. 7658. Springer, 2012, pp. 208–225.

[27] NIST, "FIPS PUB 180-4 – Secure Hash Standard," National Institute of Standards and Technology, Gaithersburg, MD, United States, Tech. Rep., Mar. 2012.

[28] ——, "FIPS PUB 202 – SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions," National Institute of Standards and Technology, Gaithersburg, MD, United States, Tech. Rep., Aug. 2015. [Online]. Available: https://csrc.nist.gov/publications/detail/fips/202/final

[29] L. Carter and M. N. Wegman, "Universal classes of hash functions," *J. Comput. Syst. Sci.*, vol. 18, no. 2, pp. 143–154, 1979.

[30] S. Gueron, "Memory encryption for general-purpose processors," *IEEE Secur. Priv.*, vol. 14, no. 6, pp. 54–62, 2016.

[31] T. Iwata and K. Kurosawa, "OMAC: one-key CBC MAC," in *Fast Software Encryption, 10th International Workshop, FSE 2003, Lund, Sweden, February 24-26, 2003, Revised Papers*, ser. Lecture Notes in Computer Science, T. Johansson, Ed., vol. 2887. Springer, 2003, pp. 129–153. [Online]. Available: https://doi.org/10.1007/978-3-540-39887-5_11

[32] P. Rogaway, "Efficient Instantiations of Tweakable Blockciphers and Refinements to Modes OCB and PMAC," in *Advances in Cryptology - ASIACRYPT 2004, 10th International Conference on the Theory and Application of Cryptology and Information Security*, ser. Lecture Notes in Computer Science, P. J. Lee, Ed., vol. 3329. Springer, 2004, pp. 16–31.

[33] R. C. Huang and G. E. Suh, "IVEC: off-chip memory integrity protection for both security and reliability," in *37th International Symposium on Computer Architecture (ISCA 2010)*, A. Seznec, U. C. Weiser, and R. Ronen, Eds. ACM, 2010, pp. 395–406.

[34] R. C. Merkle, "Protocols for public key cryptosystems," in *Proceedings of the 1980 IEEE Symposium on Security and Privacy*. IEEE Computer Society, 1980, pp. 122–134. [Online]. Available: http://ieeexplore.ieee.org/xpl/mostRecentIssue.jsp?punumber=6233684

[35] B. Gassend, G. E. Suh, D. E. Clarke, M. van Dijk, and S. Devadas, "Caches and hash trees for efficient memory integrity verification," in *Proceedings of the Ninth International Symposium on High-Performance Computer Architecture (HPCA'03)*. IEEE Computer Society, 2003, pp. 295–306. [Online]. Available: http://ieeexplore.ieee.org/xpl/mostRecentIssue.jsp?punumber=8433

[36] W. E. Hall and C. S. Jutla, "Parallelizable authentication trees," in *Selected Areas in Cryptography, 12th International Workshop, SAC 2005, Revised Selected Papers*, ser. Lecture Notes in Computer Science, B. Preneel and S. E. Tavares, Eds., vol. 3897. Springer, 2005, pp. 95–109.

[37] C. Yan, D. Englender, M. Prvulovic, B. Rogers, and Y. Solihin, "Improving cost, performance, and security of memory encryption and authentication," in *33rd International Symposium on Computer Architecture (ISCA 2006)*. IEEE Computer Society, 2006, pp. 179–190. [Online]. Available: http://ieeexplore.ieee.org/xpl/mostRecentIssue.jsp?punumber=10899

[38] R. Elbaz, D. Champagne, R. B. Lee, L. Torres, G. Sassatelli, and P. Guillemin, "Tec-tree: A low-cost, parallelizable tree for efficient defense against memory replay attacks," in *Cryptographic Hardware and Embedded Systems - CHES 2007, 9th International Workshop, Proceedings*, ser. Lecture Notes in Computer Science, P. Paillier and I. Verbauwhede, Eds., vol. 4727. Springer, 2007, pp. 289–302.

[39] S. Jaques, M. Naehrig, M. Roetteler, and F. Virdia, "Implementing grover oracles for quantum key search on AES and lowmc,"

[43] G. Saileshwar, P. J. Nair, P. Ramrakhyani, W. Elsasser, J. A. Joao, and M. K. Qureshi, "Morphable counters: Enabling compact integrity trees for low-overhead secure memories," in *51st Annual IEEE/ACM International Symposium on Microarchitecture, MICRO 2018*. IEEE Computer Society, 2018, pp. 416–427. [Online]. Available: https://doi.org/10.1109/MICRO.2018.00041

in *Advances in Cryptology - EUROCRYPT 2020 - 39th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Proceedings, Part II*, ser. Lecture Notes in Computer Science, A. Canteaut and Y. Ishai, Eds., vol. 12106. Springer, 2020, pp. 280–310. [Online]. Available: https://doi.org/10.1007/978-3-030-45724-2_10

[40] M. Bach, "Ecc and reg ecc memory performance," May 2014. [Online]. Available: https://www.pugetsystems.com/labs/articles/ECC-and-REG-ECC-Memory-Performance-560/

[41] A. Sandberg, "Understanding Multicore Performance: Efficient Memory System Modeling and Simulation," Ph.D. dissertation, Uppsala University, Disciplinary Domain of Science and Technology, Mathematics and Computer Science, Department of Information Technology, Division of Computer Systems, Uppsala, Sweden, 2014.

[42] T. Sherwood, E. Perelman, G. Hamerly, and B. Calder, "Automatically characterizing large scale program behavior," in *ACM SIGPLAN Notices, vol. 37 (Proceedings of the 10th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS-X)*, K. Gharachorloo and D. A. Wood, Eds. ACM Press, 2002, pp. 45–57.

[44] R. Avanzi, A. Sandberg, M. A. Campbell, M. Boettcher, and P. Ramrakhyani, "Cached incremental hashing for reducing memory integrity overhead," 2019, to appear.

[45] M. Taassori, A. Shafiee, and R. 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 2018*, X. Shen, J. Tuck, R. Bianchini, and V. Sarkar, Eds. ACM, 2018, pp. 665–678.

[46] J. Kelsey, "Compression and information leakage of plaintext," in *Fast Software Encryption, 9th International Workshop, FSE 2002, Leuven, Belgium, February 4-6, 2002, Revised Papers*, ser. Lecture Notes in Computer Science, J. Daemen and V. Rijmen, Eds., vol. 2365. Springer, 2002, pp. 263–276. [Online]. Available: https://doi.org/10.1007/3-540-45661-9_21

[47] M. Schwarzl, P. Borrello, G. Saileshwar, H. Müller, M. Schwarz, and D. Gruss, "Practical timing side channel attacks on memory compression," *CoRR*, vol. abs/2111.08404, 2021. [Online]. Available: https://arxiv.org/abs/2111.08404

[48] R. Ueno, N. Homma, S. Morioka, N. Miura, K. Matsuda, M. Nagata, S. Bhasin, Y. Mathieu, T. Graba, and J. Danger, "High throughput/gate AES hardware architectures based on datapath compression," *IEEE Trans. Computers*, vol. 69, no. 4, pp. 534–548, 2020. [Online]. Available: https://doi.org/10.1109/TC.2019.2957355

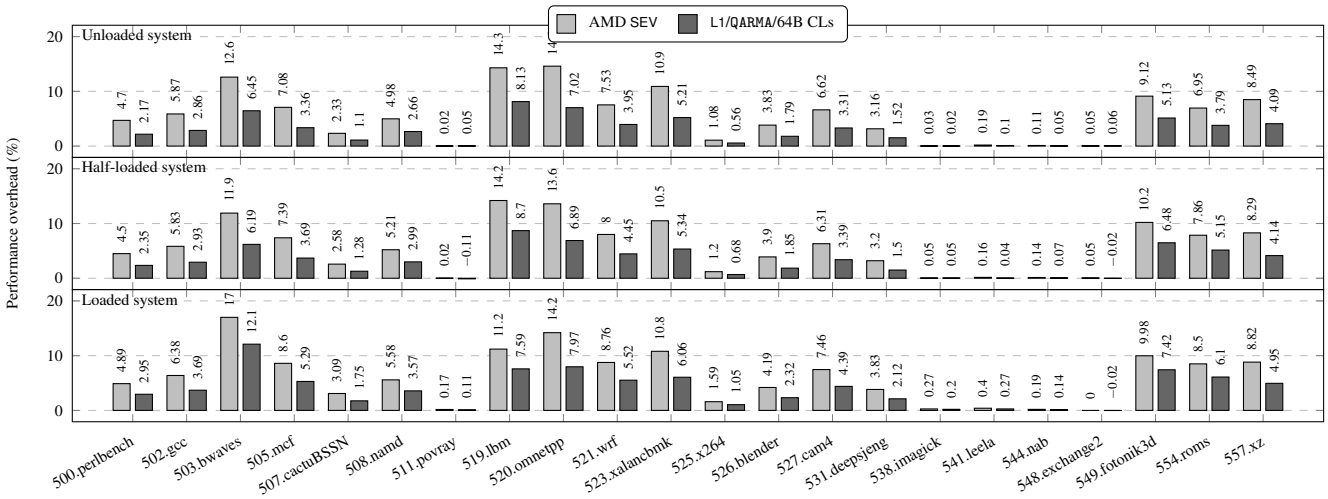

Figure 13: Set 8: AMD SEV (i.e. L1/AES/64B CLs) vs. L1/QARMA/64B CLs

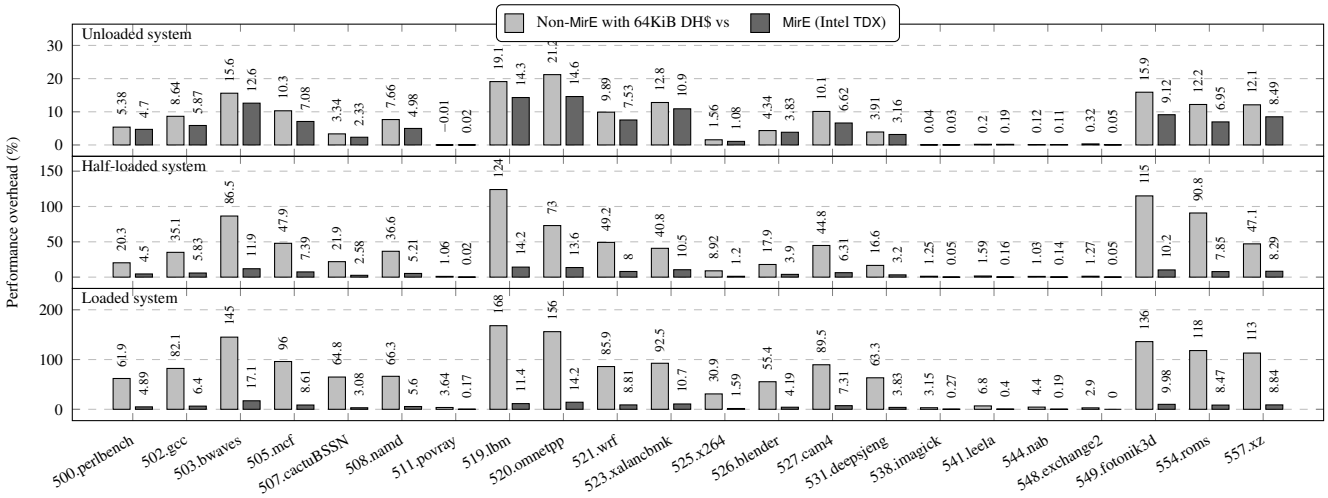

Figure 14: Set 8: L2/AES with and without MirE (64B cache lines)

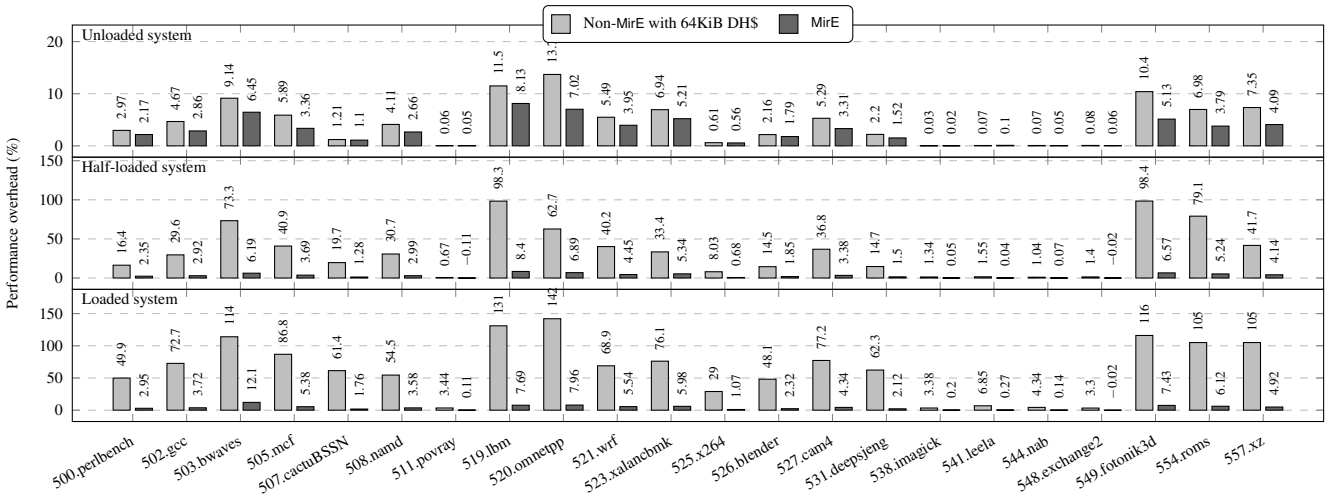

Figure 15: Set 8: L2/QARMA with and without MirE (64B cache lines)

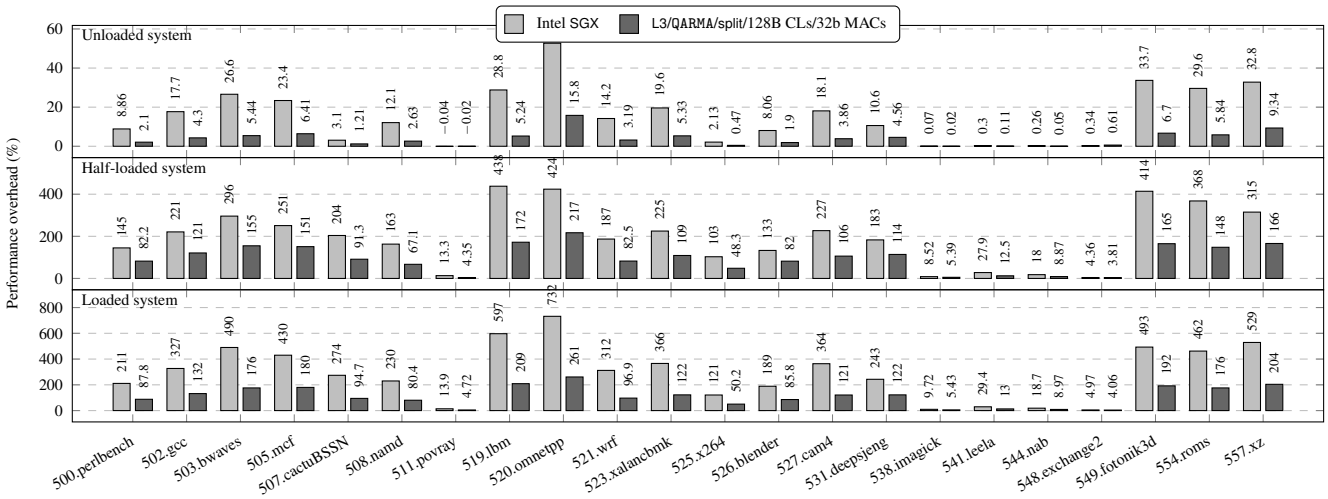

Figure 16: Set 8: Intel SGX/64B CLs (L3/AES/56b MACs/64B CLs) vs. L3/QARMA/split/128B CLs/32b MACs

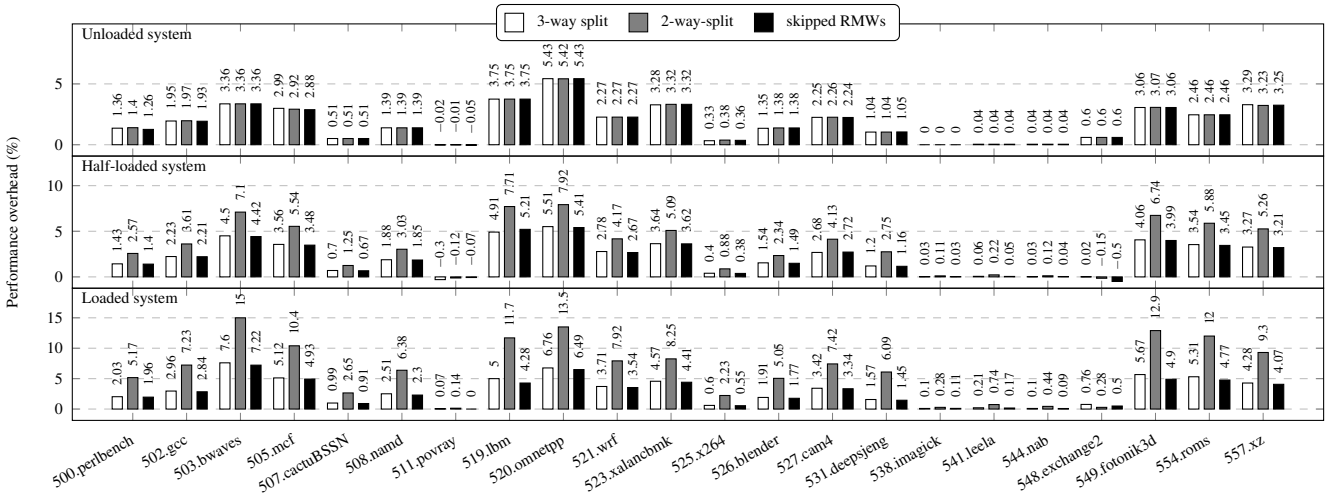

Figure 17: Set 8 and **Set 9**: L3/MirE/QARMA/oCC/128-ary – runs with 3-way and 2-way split counters, and with no RMWs

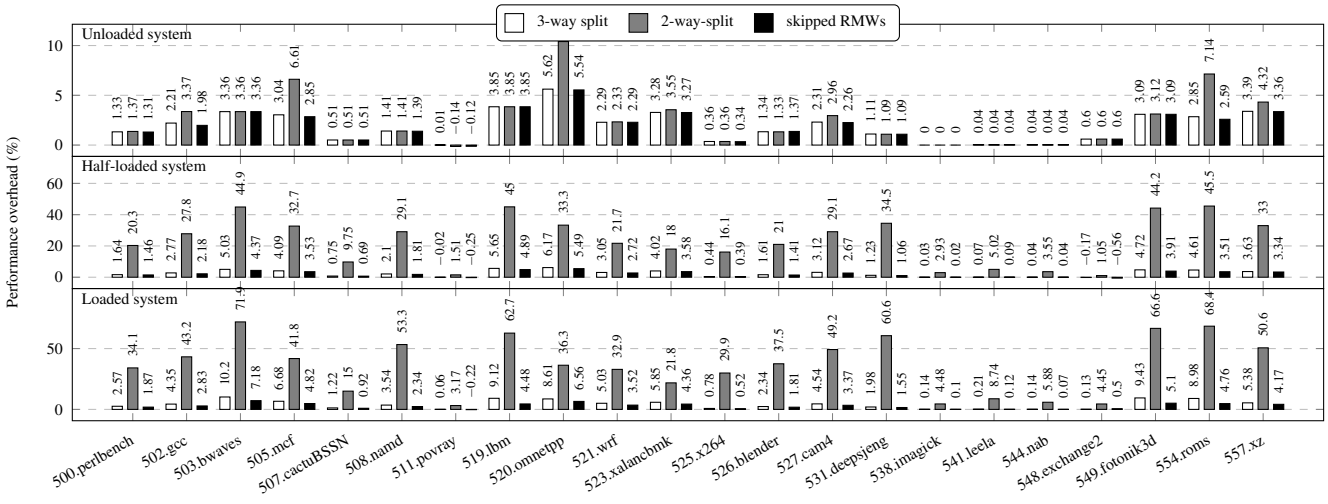

Figure 18: Set 8 and **Set 9**: L3/MirE/QARMA/oCC/256-ary – runs with 3-way and 2-way split counters, and with no RMWs