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

Roberto Avanzi¹, Ionuț Mihalcea², David Schall³, Héctor Montaner⁴, and Andreas Sandberg²

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

November 11, 2022

Abstract

For both cloud and client applications, the protection of the confidentiality and integrity of remotely processed information is an increasingly common feature request. To achieve this goal with reasonable costs in terms of memory overhead and performance penalty is also a very challenging endeavour. In turn, this usually leads to security posture compromises in products.

In this paper we review and evaluate the main technologies that have been proposed so far to address this problem, as well as some new techniques and combinations thereof. We systematise the treatment of protecting data in use by starting with models of the adversaries, thus allowing us to define different, yet consistent protection levels. As far as we are aware, for the first time we compare the impact on performance when the measured benchmark is the only running process or when it is just one task in an environment with heavy additional traffic, thus simulating a cloud server under full load.

To make just one example of our results: Using advanced techniques to compress counters can make it viable to store them on-chip – for instance by adding on-chip RAM that can be as small as 1/256th of the off-chip RAM. This allows for implementations of memory protection providing full confidentiality, integrity and anti-replay protection with hitherto unattained penalties, especially in combination with the repurposing of ECC bits to store integrity tags. The performance penalty on a server with a saturated memory subsystem can thus be contained under 2% with a memory overhead of 1/256 and even under 1% with an overhead of 1/128.

Additionally, we discuss various cost/performance tradeoffs for less loaded use cases, such as for protected software modules on client devices.


Keywords: Memory Encryption, Memory Integrity.
Contents

1 Introduction 3

2 Systematisation of the problem 4
   2.1 Definitions 4
   2.2 Problem statement and adversarial models 5
   2.3 System level view of the technical solution 6
   2.4 Protection levels 6
   2.5 Cost indicators 8

3 Background 8

4 Setup and parameters of the study 10
   4.1 Scope of the comparisons 10
   4.2 Technologies used for each level 11
   4.3 Choice of the cryptographic parameters 12
   4.4 Benchmarking environment and methodology 13
   4.5 Description of the plan of simulations 13
   4.6 Unloaded vs. loaded systems 15

5 Results and discussion 16
   5.1 Unloaded system 16
   5.2 Loaded system 19

6 Conclusions 22

References 23

A Selected full benchmark results 30
1 Introduction

With the ever growing availability and use of Computing as a Service (i.e. Cloud Computing) comes also an increased need for guarantees of confidentiality and integrity of remotely processed information. The first mechanism to protect information from unauthorised access is Access Control (AC).

Cloud tenants are becoming increasingly aware that their data can be compromised by attackers that can circumvent basic defenses. The most common class of adversaries consists in other tenants running unprivileged malicious software on the same hardware. These can mount attacks based on software exploitation, and side channels ranging from cache contention [Hu92, Koc96, Ber05, OST06] to micro-architectural features such as speculative execution. (Regarding the latter class of attacks, after the discovery of Meltdown [LSG+18] and Spectre [KHF+19] too many papers followed to reasonably cite, so we refer the reader to the surveys [CBS+19, XS21].) Insider operators running privileged software represent another serious threat. Tenant data may be targeted by actors with access to the actual computing hardware with the capability to perform physical side-channel attacks (see for instance the surveys [FGM+10, CA16, LGG+21]), or to directly compromise the memory contents by means of cold-boot attacks [ISH+09, YADA17, WCJ+21] or even at run-time by chip interposition [Kuh98, LJF+20].

Hardening the system software to prevent privilege escalation attacks is no longer considered sufficient, especially in light of the extreme complexity of modern system software stacks, for which one cannot have absolute reliance on countermeasures against software exploitation.

The same threats apply to client devices, where the compromised party may be the provider of banking, digital IDs, or gaming services. For these use cases, a compromise can lead to economic losses for the device owner or service providers. In the case of gaming, adversaries may be device owners involved in cheating or piracy. Banking applications and digital IDs need to be protected also against adversaries with temporary access to a device (that may have been left unattended).

This implies that steps that go beyond AC need to be put in place to isolate processes, services, or virtual machines from each other and the host environment, including the physical environment. Apart from putting processing elements and memory in the same tamper-proof package, these technologies rely on cryptography. Depending on the adversaries that are considered during their development, these technologies range from memory encryption [Bes80, LTM+00, KFM05] to techniques to guarantee memory integrity [MVS00, SCG+03, GSC+03, SLGL04, YGZ05, YEP+06, SOD07, RCPSo7, CL10, HS10, CRSPl1, Gue16a, WUS+17, SNR+18, JLK+23]. The latter range from tables of hashes of memory regions, to integrity trees that can detect any memory manipulation. These structures can be roughly described as variations on the theme of Merkle trees [Mer87], with the root node protected on-chip.

During the last four decades these technologies have been steadily improved to the point that their performance and memory overheads have become sufficiently acceptable to justify commercial deployment. Still, some more expensive proposals such as SGX [MAB+13] ended up being deprecated on client CPUs because the above mentioned penalties quickly degenerated when used to protect large processes Indeed, after Bastion [CL10], the development of cryptographic isolation methods nearly halted, ushering an era of research in AC based mechanisms, starting with H-SVM [JAS+15] and Hyperwall [SL12] – until the announcement of the cryptographic mechanisms to protect the
Roberto Avanzi, Ionuț Mihalcea, David Schall, Héctor Montaner and Andreas Sandberg

SGX enclave page cache [Gue16a] set the research in motion once again.

Even restricting ourselves to cryptographic techniques, it is often hard to compare different technologies since any two papers on the subject will almost never use the same benchmarking suite, memory subsystem, and cache sizes. Also, most benchmarks are performed on systems without memory bus contention. This is not realistic, as the main application for these technologies is cloud computing, servers with hundreds of concurrent processes that contend for shared resources.

We systematise the comparison between various techniques and their combinations, including also some new ideas. We consider different models of the adversaries, thus allowing us to define multiple, yet consistent protection levels. We focus solely on technologies that only require the implementation of components inside the security perimeter of the System-on-a-Chip (SoC), using external untrusted memory. Our tests consider both unloaded and fully loaded memory subsystems, by running traffic generators alongside the chosen benchmarks.

The most striking result is that advanced counter compression makes it viable to store counters on-chip. This allows for implementations of memory protection with anti-replay (i.e. full integrity) with extremely low performance penalties, especially in combination with the repurposing of ECC bits to store integrity tags. Performance penalties smaller than 1%, resp. 2%, with a memory overhead of 1/128, resp. 1/256 can be attained even under heavy bus contention. We conjecture that similar performance penalties are attained if a large system cache can be deployed instead.

We also detail various trade-offs of performance penalty vs. resources if ECC memory is not available or including RAM in the SoC is not feasible.

The structure of the paper is as follows: In Section 2 we model the types of adversaries that want to compromise memory contents, and accordingly we define the levels of protection required to thwart these adversaries. Section 3 contains background material. In Section 4 on page 10 we describe the new technologies that we add to the state to the art, the benchmarking environment, how we select the techniques we test in order to produce a clearly represented and understandable comparison. The results are reported and discussed in Section 5 on page 16. In Section 6 on page 22 we make practical recommendations for cloud and client use cases.

2 Systematisation of the problem

2.1 Definitions

The software-accessible volatile memory attached to a memory controller is viewed as an array of Cache Lines (CLs), i.e. equally sized and contiguous memory ranges adjacent to each other. A CL is the smallest unit that will be encrypted and possibly authenticated by the systems we consider in this study. By CL length we only consider that of a CL in the Last Level Cache (LLC), usually a System Cache. If upstream caches have shorter line lengths, these lengths are ignored.

The integrity information associated with CL is called an integrity tag. It is a Message Authentication Code (MAC).

If a scheme provides integrity it is understood that it simply associates an integrity tag to each CL. A scheme provides full integrity if it also prevents any form of replay attack.

An encryption or authentication function is said to provide spatial uniqueness when, if 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 CL in the computation. An encryption or authentication function provides temporal uniqueness (or: 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 CL and including it in the computation of the function.

In what follows a mode (of operation) is a general purpose encryption mode of operation. A Memory Encryption (ME) mode is understood to be an encryption mode of operation that has fixed input lengths, plaintext and ciphertext having the same size as a CL, and no associated data.

On-chip components are defined to be either part of the same die as the processing cores, or in/on the same package with tamper detection or prevention hardening.

2.2 Problem statement and adversarial models

We aim at rigorously defining what we mean by memory protection, getting beyond the hype that is marketing ME as the apparent solution to all security issues – even if they have not been formalised. First, we assume that appropriate AC policies are in place to stop unauthorised agents within the SoC. We then observe that we cannot define what we mean by protection of an asset without first establishing the adversaries against which we intend to defend the asset. We characterise the adversaries is by defining Adversarial Models (AMs) that depend on their type of access to the target devices and their resources, i.e. essentially budget, as follows:

AM0 The adversary is capable of accessing data that is outside the security perimeter of the complete system that contains the target components and on commonly accessible channels, such as messages in transit or data in storage. This includes network access.

AM1 In addition to the capabilities of AM0, this adversary can only run software on the target and manipulate external interfaces. Beside the exploitation of software vulnerabilities, this adversary can mount Rowhammer attacks \[KDK^{+}14,\ Mut^{19}, MK^{20}\]. Integrity violation is only a partial concern, as it can be arguably made less effective by deploying ME.

AM2 This adversary has physical access to the complete system that contains the target components, including its internals. They have access to exposed interfaces and communication buses but they do not have the capabilities to access on-chip communication interfaces. They will only perform passive attacks, for instance: side-channel analysis that requires close proximity, contact or run-time with the target device, and eavesdropping the content of external RAM, either at run-time via memory bus probing, chip or module interposition \[Kuh98, LJF^{+}20\], abuse of DMA channels \[Fri16\] or via cold-boot attacks \[HS^{+}09, YADA^{17}, WCJ^{+}21\].

AM3 This adversary has the same level of access as AM2, but they will also perform active attacks, such as blocking, corrupting or replaying memory transactions, or injecting new ones \[KLR^{+}20\]. Because of the similarity of the involved technologies, the required expertise beyond AM2 is minor, whereas resources may need higher precision.

The difference in complexity and cost and of the countermeasures is a key factor in distinguishing the two models. We note that active attacks are more easily detectable, as they may trigger repeated failures, so adversaries may just choose not to mount them, even if capable. Examples of threats mounted by this adversary are \[BBKN^{12}, BR^{12}, ZDC^{+}12\].
Within AM3, we distinguish two cases:

AM3.(i) Adversaries limit themselves to corrupt individual memory locations; and
AM3.(ii) Adversaries replace memory contents together with any associated Metadata (MD).

AM4 This adversary, in addition to all of the above capabilities, can mount highly invasive attacks at the chip or package level that require considerable experience, resources, and time to succeed. These attacks this adversary can mount range from micro-probing attacks [Sk07] to actual chip reverse engineering and editing using a Focused Ion Beam Microscope [TJ09, SAFT16, HTLW21]. This adversary is out of scope for the research described in this paper.

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

**Remark 2.1** Against adversaries of type AM0 the usual consensus is that no memory protection is necessary, even though attacks like Nethammer [LSR+20] can corrupt the memory of a target system without a single attacker-controlled line of code on it. Therefore, it can be argued that this model should be subsumed into AM1, and we accordingly shall not consider AM0 separately.

### 2.3 System level view of the technical solution

To answer the above question we introduce a set of HW components, which we call the Memory Protection Engine (MPE). This is not a new concept: all cryptographic memory protection designs cited so far use such a block, usually called a Memory Encryption Engine (MEE).

Its placement in a SoC system level view is depicted in Fig. 1 on the next page. A MPE sits between the interconnect or a System Cache that branches off the interconnect on one side, and a memory controller on the other side, which in turn is connected to RAM. The MPE can optionally have: caches, namely a Counter Group (CG) and a Data Hash (DH) cache; internal buffers (not depicted); and it may have access to a certain amount of on-chip RAM.

*The memory protection technologies that we study in this paper are implemented in the MPE.*

### 2.4 Protection levels

In light of the adversarial models defined above, we define the following protection levels:

**L1** The memory is encrypted to defeat adversaries AM1 and AM2, except for adversaries that exploit memory access patterns, and ciphertexts, as a side channel. The encryption function provides spatial uniqueness to reduce detection of data patterns. Temporal uniqueness and integrity verification are not provided.

**L2** To thwart adversaries of type AM3.(i), CLs are encrypted and augmented with integrity tags. No freshness is provided. This is not sufficient against Adversary AM3.(ii).

**L2+** CLs are encrypted and authenticated. Freshness information is provided and included in the encryption function.
L3 Additionally, full integrity is provided against AM3.(ii) adversaries.

We combine the following types of technologies to implement the above protection levels: (i) ME primitives and modes; (ii) Authentication primitives; (iii) Integrity and anti-replay structures; and (iv) Physical mechanisms to protect memory from tampering, such as including memory on-chip – however we do not consider putting all the RAM on-chip.

The last solution would in principle work if applied to the entire RAM and without any performance penalty, but it is impractical: for instance, for server applications it is not reasonable to put, say, 512GiB of RAM in the SoC package, for both space and thermal reasons.

Remark 2.2 We only consider solutions that need the security perimeter to be no larger than the physical package of the SoC. Hence, “smart memory” technologies [AN17] are out of scope. These have cryptographic logic in the memory chips to attest themselves to the memory controller – allowing them to communicate on a secured channel only with that memory controller, such as the CXL.memory Integrity and Data Encryption (IDE) scheme [CXL19]. In order to properly address the threats they are meant to defend against, smart memories are very expensive. They must implement mutual attestation with the memory controllers, and include cryptographic engines in each memory chip, as putting the engines only on an on-board controller of the DIMM would not completely remove the risk of interposition. The countermeasures which are the subject of this paper require cryptographic engines only in the SoC.

However, smart memories are suitable for physically remote memories, to implement the communication between the local SoC and the remote storage. This way, the protected address space can be expanded beyond what the local MD would allow.

We now discuss the protection levels in some state of the art solutions. The MEE in Intel’s SGX [Gue16a] is a L3 solution. The Multi-Key Total Memory Engine with Integrity (MKTMEi) [Int21, Section 2.A] in Intel’s TDX is a L2/MirE solution, where MirE means MACs in repurposed
**ECC bits.** Amd’s SEV [KPW16] is a L1 solution. SYNERGY [SNR+18] is a L3/MirE solution. CSI-Rowhammer [JLK+23] is a L2/MirE solution.

### 2.5 Cost indicators

It is not only important to know whether we have a solution to a problem: For real-world applications it is critical to know how expensive is the solution.

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.

### 3 Background

We present here a brief summary of the technologies we considered in the development of this paper.

**Memory encryption primitives.** We use block ciphers for memory encryption – the long initial latency of stream ciphers making them unsuitable. In direct encryption, the block cipher is applied block-wise to the plaintext to generate the ciphertext. In One-Time Pad (OTP) encryption, the encryptions of successive values of a counter are XOR-ed block-wise to the plaintext.

We only use block ciphers with a block size of 128 bits. The selected block ciphers are the AES [DR02] and the Tweakeable Block Cipher (TBC) QARMA [Ava17]. Other candidates have either similar latencies or, in the case of PRINCE [BCG+12], are not tweakable or have a shorter block size.

**Authentication primitives.** Standard hash functions such as SHA-2 [NIS12] or SHA-3 [NIS15] can be turned into MACs but the resulting schemes are very slow and not parallelisable.

Encrypted Universal Hash Functions (UHFs) [CW77, CW79] 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 [Gue16b]. 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.

TBC-based Parallel MACs (PMACs) [Rogo4] can also be used. PMACs are more expensive than encrypted UHFs because the text is first processed by encryptions instead of Galois multiplications, but they can be used for error detection and correction beside integrity. Cf. [HIS10, SNR+18, JLK+23]. The computation of PMACs is depicted in Figs. 2 and 3 on the facing page.

We do not consider encrypted plaintext checksums as in Rogaway’s Offset Codebook mode (OCB) mode [Rogo4]. OCB requires freshness, and if the latter is available we use instead OTP encryption to reduce read latencies. To avoid OTP ciphertext malleability, we must then use a MAC.

**Modes of operation.** For direct encryption, spatial uniqueness is achieved by using the PA as the cipher’s tweak. To achieve this with the AES, a non-tweakable block cipher, we use it in Rogaway’s XOR, Encrypt, and XOR (XEX) construction [Rogo4]. XEX is defined as $C_i = E_K(P_i \oplus M_i) \oplus M_i$. 

In other words, a tweak-derived 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 $\gamma$ we can put $M_i = \gamma^i \cdot M_0$. This results in a variant of Rogaway’s OCB mode [Rog04].

With a TBC, the PA of each block is instead used directly as the tweak in the block cipher.

In OTP encryption, with a non-tweakable block cipher counter and PA are concatenated before encryption. With a TBC, counter and PA are used as tweak and text, respectively.

**Memory integrity structures.** A table of hashes or MACs suffices against memory corruption, but against replay one needs either to protect the table in on-chip memory or with a tree structure such as a Merkle Tree (MT) [Mer80]. MT nodes can be cached [GSC+03] to speed up verification.

With OTP encryption, we can recursively protect the freshness counters as follows: a set of $a$ of counters and an embedded MAC form a node called a Counter Group (CG), which has the same size as a CL. Each CG has $a$ children, which can be either CLs of data in use, or childred CGs. Each counter in a CG 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, and is stored in a table. Before a CG, or a data CL is evicted, its parent counter is first incremented and the CG’s or CL’s MAC is recomputed. Such a Counter Tree (CT) is used for instance in Intel’s SGX [Gue16a].

With the split counters optimisation [YEP+06] a group of $a$ counters is replaced by a group with one major counter and $a' > a$ smaller, minor counters, so that the two types of Counter Groups (CGs) have the same size. Each node (a data CL or a CG) is associated with a minor counter,
and a' sibling nodes share a major counter. A node’s freshness is the concatenation of the associated major and minor counters. 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, all minor counters in the group reset to zero, and all the sibling nodes refreshed: For data CL this means that they are re-encrypted, and for both types of nodes the MACs need to be recomputed. These Read-Modify-Write (RMW) operations may affect performance, but in general split counter trees represent a major performance improvement with respect to non-split, i.e. monolithic, counters.

In Table 1 we compare the memory overheads of various integrity trees. We assume that the size of a CG is a CL; a DH or MAC; of a hash value or MAC embedded in a CG; of a monolithic or major counter; and a minor counter, respectively. $a$ is the arity of a CG, i.e. the number of its monolithic or minor counters; and $n$ is the number of CLs a MAC covers.

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

<table>
<thead>
<tr>
<th>Type of Tree</th>
<th>CL size:</th>
<th>64B Overhead</th>
<th>128B Overhead</th>
</tr>
</thead>
<tbody>
<tr>
<td>Merkle Tree with $a = 4$, resp. 8</td>
<td>CL size:</td>
<td>33.3%</td>
<td>16.7%</td>
</tr>
<tr>
<td>Monolithic CT with embedded MAC</td>
<td>$\ell_H = 64$; $n = 1$; $a = 8$, resp. 16</td>
<td>26.8%</td>
<td>12.9%</td>
</tr>
<tr>
<td></td>
<td>$\ell_H = 32$; $n = 1$; $a = 8$, resp. 16</td>
<td>20.5%</td>
<td>9.79%</td>
</tr>
<tr>
<td></td>
<td>$\ell_H = 32$; $n = 2$; $a = 8$, resp. 16</td>
<td>17.4%</td>
<td>8.23%</td>
</tr>
<tr>
<td></td>
<td>$\ell_H = 32$; $n = 4$; $a = 8$, resp. 16</td>
<td>15.8%</td>
<td>7.45%</td>
</tr>
<tr>
<td>Split CT with embedded MAC</td>
<td>$\ell_H = 64$; $n = 1$; $\ell_c' = 6$, resp. 7</td>
<td>14.1%</td>
<td>7.04%</td>
</tr>
<tr>
<td></td>
<td>$\ell_H = 32$; $n = 1$; $\ell_c' = 6$, resp. 7</td>
<td>7.84%</td>
<td>3.91%</td>
</tr>
<tr>
<td></td>
<td>$\ell_H = 32$; $n = 2$; $\ell_c' = 6$, resp. 7</td>
<td>4.71%</td>
<td>2.34%</td>
</tr>
<tr>
<td></td>
<td>$\ell_H = 32$; $n = 4$; $\ell_c' = 6$, resp. 7</td>
<td>3.15%</td>
<td>1.57%</td>
</tr>
<tr>
<td></td>
<td>$\ell_H = 32$; $n = 1$; $\ell_c' = 3$</td>
<td>7.04%</td>
<td>3.52%</td>
</tr>
<tr>
<td></td>
<td>$\ell_H = 32$; $n = 2$; $\ell_c' = 3$</td>
<td>3.91%</td>
<td>1.95%</td>
</tr>
<tr>
<td></td>
<td>$\ell_H = 32$; $n = 4$; $\ell_c' = 3$</td>
<td>2.35%</td>
<td>1.17%</td>
</tr>
</tbody>
</table>

**Legend:** $\ell_{CL}$, $\ell_H$, $\ell_c$, and $\ell_c'$ are the bit lengths of a CL; a DH or MAC; of a hash value or MAC embedded in a CG; of a monolithic or major counter; and a minor counter, respectively. $a$ is the arity of a CG, i.e. the number of its monolithic or minor counters; and $n$ is the number of CLs a MAC covers.

### 4 Setup and parameters of the study

#### 4.1 Scope of the comparisons

Depending on the level, several variants of the involved technologies may be combined, which are summarized in the following list. The entries marked with † contain new contributions in this paper.
Those marked with * describe variations not hitherto compared to each other.

- Use of the AES or QARMA ciphers;
- Size of the MACs (32b or 64b);*
- Counter trees: monolithic, 2-way split or 3-way split;†
- Various choices for the size of CG$ and DH$.
- Use of on-chip memory for hashes and/or counters;‡
- Repurposing of ECC bits for data MAC storage;
- Synchronous or asynchronous integrity checking;
- Use of single MACs covering multiple CLs, with cached incremental hashing; †
- Arity variations in the CGs;
- We consider both 64B and 128B CLs;* and, finally
- We consider systems where only the benchmarking suite is running; and systems with a saturated memory subsystem.*

**Remark 4.1** A new idea we adopt in our implementations applies to the cases where UHF-based MACs that are stored in normal RAM. They are kept as DHs in their cache, and are evicted in groups, which are encrypted directly. We use this technique only if the MACs are 32 bits long. In this case, four DHs are actually encrypted directly as a single 128b block. Any attempt to corrupt one DH will corrupt all four with high probability, vastly increasing the likelihood of detection. By doing this we increase both security and the robustness of the system, and also speed up integrity verification of CLs that are fetched from memory when the hash is already in the DH$, because the latter does not have to be decrypted.

If freshness is available, the four minor counters corresponding to the DHs that are encrypted together and their common major (and possibly middle) counter(s) are concatenated together. This is used to create a tweak for QARMA-128.

### 4.2 Technologies used for each level

We list the technologies used to implement the protection levels defined in Section 2.4 on page 6.

- **L1** If AES-128 is the chosen encryption primitive, a CL is encrypted using the XEX construction, with the PA as the tweak. If QARMA-128 is chosen, it is used in Tweaked ECB mode as in Fig. 4 on page 9, with the PA as tweak.

- **L2** The same encryption modes are used for L1. Hashing is done by a multi-linear (ML) UHF at 32 or 64 bits. The hashes are encrypted block-wise when they are evicted from the DH$ in CL worth groups. This approach has good security and reliability implications, cf. Remark 4.1.

- **L2+** This level provides freshness over L2. A counter based OTP encryption mode is used with both AES and QARMA. We recall that this level does not offer protection against active adversaries with access to the memory bus if both counters or MACs are in off-chip memory. Thus, we do not feed the freshness to the MAC computation function.
L3 CLs are encrypted as in L2+. Full integrity is achieved by including the counters in the tag computation and preventing the adversary from tampering with the CGs. Thus, an adversary may still be able to replace a CL and its MAC, but not its counter(s). This is achieved by either using an integrity tree, or by storing the CGs on-chip.

**oCC on-Chip Counters.** Applied to L2+ it gives an L3 level of protection, provided that the freshness be included in the tag computation. This is due to the fact that we assume counter on-chip memory to be non-interposable and MPE private, hence outside adversarial control.

**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 (which, with oCC, is on-chip), at the price of a slightly higher latency for writes to reach the memory controller (but less writes overall) and for processing reads. MACs are still accessible to a HW capable adversary. Hence freshness information, if available, must enter the MAC computation, otherwise birthday bound replay attacks apply. Following [JLK+23], the tag is computed using QARMA$_{5}$-64-$\sigma_{0}$.

The MirE technology raises the question of the performance impact of using ECC vs non-ECC memory. ECC memory needs to store and retrieve 9/8 of the data with respect to non-ECC memory, newer DDR5 memories even 5/4. This is performed in a single burst within a transaction, so the performance loss is much smaller than the overhead. In fact, usual penalties are reported as less than 2%, and in actual benchmarks they are usually smaller than 0.5% [Bac14]. We also note that on servers, schemes that do not repurpose the ECC bits are assumed to be still using them for error detection and correction; and for all the methods that do not repurpose the ECC bits, we assume that the relative performance losses should be similar regardless of whether they run on non-ECC or on ECC memory. Hence, we do not take into account ECC memory as a separate configuration.

### 4.3 Choice of the cryptographic parameters

In the choice of parameters such as lengths of keys and MAC, the fundamental difference between encryption and authentication is that the encryption parameters must provide long term confidentiality, whereas authentication needs only to deter an adversary, since a system that can monitor unrecoverable integrity violations may detect unusual activity.

As a result, the following parameters are recommended:

1. Encryption keys should be at least 128 bits long. A single key may be used for the AES in the XEX construction. We note that even on a quantum computer, the complexity for a key search attack on AES-128 given as the product of total number of decomposed gates and full depth required is around $2^{160}$ [BNS19, JBS+22]. The block size must also be at least 128 bits. For this reason from the QARMA family we choose QARMA-128 over QARMA-64 for encryption.
2. For a MT the required hash length is 128b to practically prevent replay attacks.
3. The recommended length for authentication keys is 128b.
4. Data MACs should be at least 32b long.
5. Monolithic counters must be at least 64b long. The minimal aggregated length of a major and a minor counter (or major plus middle plus minor) is also 64b.
With the above parameters, a successful replay attack on the memory of a L3 system would require both the counter and the MAC to be repeated, with complexity \(2^{64} \times O(2^{64/2}) = O(2^{96})\).

### 4.4 Benchmarking environment and methodology

To provide a comparison of potentially thousands of combinations of techniques, it would be impractical to implement each variant in silicon. 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 are derived from actual implementations, and they are inserted as delays into the simulation.

The prototypes used in this paper are built in the \texttt{gem5} simulator [BBB+11, LAA+20]. \texttt{gem5} allows engineers to build software versions of hardware components typically included in computer systems. The framework also helps abstract away the interfaces between components. The components can thus be combined programmatically and configured at run-time. It includes very precise models for several common CPU cores.

The prototypes used in this paper are built in \texttt{gem5} [BBB+11]. The simulated CPU is modelled around an Arm Cortex A72 core, with a 2GHz CPU frequency and a 1GHz system frequency. The CPU cache hierarchy includes L1-I (4KiB, 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 L2 unified cache (1MiB, tree-PLRU replacement policy, 16-way, 5 cycles latency). The memory is 16GiB DRAM as dual-rank DDR4 DIMMs. The MPE-private caches are 4-way set associative with a LRU replacement policy.

We also assume that the SoC is implemented in a 7nm process, in order to re-use the information about latencies from [Ava17], for instance a latency of 15.76ns for a pipelined implementation of AES-128, of 4.8ns for QARMA\textsubscript{11}-128-\textsubscript{\sigma}, and 2.2ns for QARMA\textsubscript{5}-64-\textsubscript{\sigma}. The latter two implementations are also pipelined, and are included in the Qameleon NIST Lightweight Cryptography Standardization Process submission [ABB+19]. This latency of QARMA\textsubscript{5}-64-\textsubscript{\sigma} is also used in [JLK+23].

We benchmark using the SPEC2006 suite [Hen06]. Simulations of hardware systems via software models such as \texttt{gem5} have lengthy execution times even for short workloads. As shown in [San14], a typical SPEC benchmark could take around a month to run, making it infeasible for rapid prototyping and analysis. We rely, instead, on previously characterised SPECint 2006 workloads [SPHC02], with each benchmark distilled to a set of 10 workloads of roughly 30 million instructions each, which are then weighted and combined.

### 4.5 Description of the plan of simulations

Comparing thousands of different configurations is not only unfeasible in HW, but it would take too long also in a simulated environment, not to speak of the difficulties of properly presenting the data. For this reason as have planned a tour through the jungle of combinations, in various stages, each stage resulting in a selection of cases to be compared in the successive ones with added variability in only a few parameters. Stage number \(n\) is abbreviated as \(S_n\).

We use shorthands to describe the various configurations. They have the following form:
Level / \{additional technologies\} / Cipher / CL length / MAC length

where the optional field \textit{additional technologies} may include \textit{mono} (for monolithic counters), \textit{split} (counters), \textit{oCC}, or \textit{MirE}. The default CL length is 64B, except when indicated or when the CGs are on chip, in which case it is always 128B. The default MAC length is 56–64B. Furthermore, \textit{"{Intel} TDX"} is equivalent to L2/AES/MirE, \textit{"{Intel} SGX"} to L2/AES/mono, and \textit{"{AMD} SME"} to L1. \textit{oCC} always implies split. The shorthand \textit{L3/oCC} is used to denote the combination of L2 with \textit{oCC}. We understand \textit{L3} without \textit{oCC} as a full integrity capable scheme based on an integrity tree and \textit{neither counters nor hashes on-chip}, unless specified otherwise.

**S1** We initially focus on the state of the art and our most basic technologies.

We compare AMD SME (i.e. L1/AES), L1/QARMA, L2/AES, Intel TDX (i.e. L2/AES/MirE), L2/QARMA, L2/QARMA/MirE, L2+/QARMA with split counters – all with and without both monolithic and split counters, SGX, L3/QARMA with split counters variant (L3/AES/split), and in any case where counters are monolithic or MACs are 64 bits long, such as L2/QARMA/64b MACs, L2+/QARMA/split/64b MACs, and L3/QARMA/split/64b MACs. In all other cases counters are split, MACs are 32b long, and directly encrypted in groups.

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

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

\textit{From now on, MACs are 32 bits long, directly encrypted in groups of four, except where SGX is benchmarked, the MirE technology is used, or where explicitly indicated.}

**S2** For L2, L2+, and L3 only, we study the impact of the sizes of the two MPE caches. The possible sizes of the DH$ are 4KiB, 16KiB, and 64KiB. The possible sizes of the CG$ are 16KiB, 64KiB, 256KiB, and 1MiB. This simulation set is restricted to QARMA only for encryption, as the AES results would show similar relative performances.

\textit{Starting with Stage 3, the MPE has a 16KiB DH$ and a 256KiB CG$. Also, levels L2+ and L3 will use split counters, except when explicitly indicated otherwise, or with SGX.}

**S3** Consider 64B and 128B CLs for L2, L2+, and L3. A CG and a CL have the same size.

**S4** We compare synchronous to asynchronous verification for for L2, L2+, and L3.

**S5** We analyse the use of on-chip memory in full integrity schemes. As the MAC memory overhead is larger than that of the CGs, we do not consider the case where the MACs (actually, the data hashes) are on-chip and the counters off-chip.

Since these variants together with the ones in the next stage are amongst the most promising ones in terms of performance, we run them with both AES and QARMA.
We consider here the impact that repurposing the ECC bits for tags bears on performance. We compare \( L_2, L_2 +, L_3, \) and \( L_3/oCC \) schemes with and without \( \text{MirE} \). We measure the impact for both 64B and 128B CLs if counters are off-chip, and only 128B CLs with \( oCC \). If MACs are stored in the ECC tag bits, we need no \( \text{DH} \)$, and the MACs are computed as PMACs.

The types of high-arity CGs on-chip we consider are:

- 128B CLs and CGs with: 128 7b minor, 8 8b middle, and 1 64b (49b) major counters; This results in a memory overhead of \( 1/128 \).
- 128B CLs and CGs with: 256 3b minor, 32 6b middle, and 1 64b (55b) major counters; This results in a memory overhead of \( 1/256 \).

The last three stages are some off-path branching to verify missing and corner cases.

We want to show what can be optimised storage-wise when when cannot store MACs in the ECC bits or on-chip. MACs are thus stored off-chip, but MACs computed incrementally over multiple CL can be used to reduce their memory footprint [ASC+19]. This makes sense only when we have already chosen to use 128B CLs, as these already halve MD storage requirements.

We test \( L_2 \), \( L_2 + \), \( L_3 \), and \( L_3/oCC \), with a MAC covering 1, 2, or 4 CLs. These runs are performed only with \( \text{QARMA-128} \) as the encryption cipher, since the performance differences are caused only by the increased memory traffic, and therefore we can expect that AES performance will follow the same pattern.

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

- AMD SEV (L1/AES), and L1/QARMA, with 64B CLs;
- Intel TDX/64B CLs (i.e. L2/AES/MirE);
- L2/QARMA/64B CLs/64b MACs and L2/QARMA/64B CLs/MirE;
- Intel SGX (i.e. L3/AES/56b MACs);
- L3/QARMA/split/128B CLs/32b MACs;
- L3/QARMA/oCC/32b MACs, with 128- and 256-ary CGs;
- and L3/QARMA/oCC/MirE, with 128- and 256-ary CGs.

In this stage we compare the performance of an MPE with a hypothetical one where the RMW operations have zero costs, i.e. they are instantaneous. This is achieved by simply skipping them. Such an experiment is possible because the simulated MPE does not actually perform the cryptographic operations on any data passing through, simulating instead the timing delays involved in the processing steps. This gives an upper bound on the actual time spent performing RMW operations. The selected combinations are the last five of S8, which are L3 schemes with split counters, i.e. the only ones in S8 with RMWs.

4.6 Unloaded vs. loaded systems

All stages described above are first run on an unloaded system, where benchmarking is the only running task. The results are reported and discussed in Section 5.1 on the next page.

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. It is very lengthy to run that many processes in a simulation. We instead inject synthetic traffic upstream of the MPE, but after the L2 cache. The amount of traffic injected is 8GiB/s. This corresponds to the point where the latency of the memory subsystem starts to diverge for a SGX-like L3 MPE covering the entire memory, with mostly linear synthetic traffic, and synchronous MAC verification. We assume that MAC verification is synchronous because, following the discussion of the benchmark runs, this will be the most likely implementation. The bandwidth of 8GiB/s is derived from the measurements reported in Fig. 6. 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 the additional traffic. The results of these runs are reported and discussed in Section 5.2 on page 19.

5 Results and discussion

We now analyse the results of the selected test runs.

5.1 Unloaded system

Changing CL length from 64B to 128B in our simulated system slows down the system by 1% on our benchmarks. Each run is compared to the baseline with the same CL length.

For $S_1$ on an unloaded system (see Fig. 7 on the facing page) we note that:

- If implemented in a plain way, the performance penalty of lower levels of protection is smaller than the performance penalty of the higher ones.
L1 and L2 schemes have worse performance with the AES w.r.t. QARMA because of the higher latency of the former cipher. This holds also for L3 because the OTP generation, while it can be performed in parallel with a memory fetch, still increases write latency to the point that its effect becomes noticeable.

Split counter trees are superior to monolithic trees in memory overhead (see Table 1 on page 10) and performance.

A small DH$ does not significantly affect performance.

As expected, using 64b MACs results in slightly worse performance than using 32b MACs.

S2 results (see Fig. 8) confirm the expected significant performance gains with larger MPE caches. The CG$ cache having a higher effect than the DH$.

S3 (the results are combined with those of S4 in Fig. 9 on the next page) proves that the impact of memory protection is comparable across systems with 64B CLs and systems with 128B CLs.
Figure 9: S3 and S4/unloaded: Impact of CL size and asynchronous MAC verification; ME cipher is QARMA-128

Figure 10: S5/unloaded: L3; Impact of storing MD on-chip

Figure 11: S6/unloaded: Impact of repurposing ECC bits for MACs

Figure 12: S7/unloaded: Impact of using multiple-CL MACs (128B CL)

Note that taking into account the effect of 128B CLs, the performance of a memory protected 128B CL system is slightly slower than that of a memory protected 64B CL system. However, switching to 128B CLs halves the footprint of MD (cf. Table 1 on page 10).

S4 results (see Fig. 9) suggest that asynchronous MAC verification does not significantly improve performance.

S5 results (see Fig. 10) show, as expected, that relieving the contention on the memory bus between data and MD reduces performance penalties.

S6 results (see Fig. 11) prove that combining oCC and MirE provides the highest level of memory protection at very low performance penalties.
The results of S7 (see Fig. 12 on the facing page) show that while multiple-CL MACs effectively reduce memory overheads, this comes at a significant performance price. L2+ and L3 performance is virtually identical, due to the fact that data MAC traffic becomes dominant whereas, for L3, the rest of the tree profits from spatial locality.

Note, however, that we do not implement evicted cache line compression as in [TSB18], which would have allowed to store a MAC in the CL with the data if the latter can be sufficiently compressed, reducing the amount of memory accesses. Following the paper, we estimate that compression may halve the performance penalties.

S8: The performance of the individual SPEC2006 benchmarks (Figs. 19 to 26 on pages 30–32) shows a few expected results, namely that some programs such as gcc g23, gcc s04, mcf, libquantum, and xalancbmk suffer significantly more than average under every MPE configuration. Other programs are affected in a significant way only when there is traffic expansion, such as the remaining gcc programs, bzip2 chicken, and bzip2 liberty. Increasing integrity tree arity by split counters is instrumental in reducing the penalties, but it is only with the compressed OCC and the MirE that all penalties are consistently brought down to less than 5%. For client and edge applications, L3/QARMA/split/128B CLs/32b MACs provides excellent performance, esp. if the tasks profit from good spatial locality properties.

S9 (see Figs. 24 to 26 on page 32) shows that the impact of RMWs is small and often negligible. This signifies that techniques for reducing RMWs, such as snooping the CLs to be re-encrypted in the LLC in order to skip RMWs for data still in the LLC and marked as dirty, will not give a significant performance boost. Hence, they could be omitted in order to keep state machines simple, at least if we only consider unloaded systems.

5.2 Loaded system

Without memory protection, our benchmarks run on a loaded system with 64B CLs 16.5% slower than on an unloaded system; If CLs are 128B long, the benchmarks run 12.2% slower than on the unloaded system. Changing CL length from 64B to 128B makes the loaded system faster by 2.7%.

For S1 on a loaded system (see Fig. 13 on the following page) we observe that L1 always performs better than all other levels, as expected since the expanded traffic will face extreme contention on the memory bus. Two observations are similar to the unloaded case: The use of split counters reduces performance penalty by a factor of roughly 3 with respect to monolithic counters; and a small DH$ offers only a minimal improvement.

S2 (see Fig. 14 on the next page) shows a significant difference between the unloaded and loaded cases. Whereas in the unloaded case larger MPE caches brought significant improvements, on a loaded system they are less effective.

S3 results (combined with S4 in Fig. 15 on page 21) indicate that the MPE performance is often worse with 128B CLs.

S4 (see Fig. 15 on page 21) shows that in the loaded case asynchronous verification brings a significant speedup. Indeed, as the memory bus approaches saturation, decoupling decryption and MAC verification logics allows better for scheduling of otherwise idle MPE resources.
S5 results (see Fig. 16 on the facing page) are interesting since they go against the intuition that using longer CLs should perform better because of reduced MD traffic. In fact, the increased data traffic with 128B CLs is a noticeable penalty. As expected, with all MD on chip performance is close to the baseline.

S6 results (see Fig. 17 on the next page) finally show that nearly negligible performance penalties on a loaded system can be achieved by using on-chip memory for the CGs and repurpose ECC bits for MAC storage. In fact, this has even better performance than with L1 direct encryption – because the latter places the cipher on the critical path, and the effect of any additional latency is amplified when the memory subsystem is saturated.

S7 (see Fig. 18 on the facing page) shows that multiple-CL MACs offers increasingly worse performance also in the loaded case.

S8 (see Figs. 27 to 34 on pages 33–35) results are similar to the unloaded case, however the
penalties are often much larger, because of the significant amount of scattered traffic expansion. It is only with oCC and MirE that the performance becomes reasonable across the board.

**Figure 15:** S3 and S4/loaded: Impact of CL size and asynchronous MAC verification; ME cipher is QARMA-128

**Figure 16:** S5/loaded: L3; Impact of storing MD on-chip; AES and QARMA results are identical

**Figure 17:** S6/loaded: Impact of repurposing ECC bits for MACs

**Figure 18:** S7/loaded: Impact of using multiple-CL MACs (128B CL)

S9 results are given in Figs. 32 to 34 on pages 34–35. Note that Figs. 33 and 34 on page 35 have bars for 128-ary and 256-ary CGs, but a single bar for overheads while skipping RMWs because in this case CG arity is unimportant. We observe that: The impact of RMWs on a loaded system is still small; if the counters are off-chip or the MACs require their own addressable storage, the impact of the RMWs is minor, and is negligible with longer (7b) minor counters; with oCC and MirE the reduced performance of the 256-ary CGs with respect to the 128-ary CGs (which uses twice as much on-chip memory) due to the larger amount of RMWs becomes noticeable, but even in this
case the worst penalties with 256-ary CGs are still lower than with a direct encryption L2/AES/64B CLs/MirE scheme as in TDX.

6 Conclusions

We have performed a very thorough evaluations of techniques for the cryptographic protection of in-use memory contents. We have considered the state of the art, some new technologies, and hitherto not considered combinations thereof.

We have 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.

Let us start with the use case of cloud computing. There are two possible scenarios.

The first one starts with the observation that SoCs for cloud servers are expensive, may contain several dozens of cores, have multiple memory channels and can easily address hundreds of GiBs of physical memory. Because of the very high HW costs, we have a strong argument for using freshness-based OTP encryption based on AES or QARMA, with oCC in the amount of 1/128 or 1/256 of the off-chip memory, and storing MACs in the ECC bits for both integrity and error correction. The additional cost for implementing these technologies would be relatively minor. It would be likely less expensive than basing the protection of local memory on the CXL.memory IDE. This would enable the highest level of memory protection at a performance cost which is actually lower than some currently deployed schemes that provide encryption and, optionally also integrity – but not anti-replay.

However, it can also be argued that the budget for a large amount of memory on-chip should rather be spent on a large system cache, from which the whole system profits, with CGs getting a similar cache hit rate as in the unloaded case. The evaluation of this approach is an open question.

For lower end devices the situation is more nuanced. L3/oCC/MirE is not applicable if the devices lack ECC bits. If the use case considers memory bus saturation as an exceptional event, which is often the case for edge applications or client applications limited to business oriented virtual machines and special secure modules, then high arity split counter trees with counters either on-chip or in a dynamically allocated carveout together with the MACs can be considered. However, as the results in Fig. 33 on page 35 indicate, performance can quickly deteriorate if extra accesses are needed for MACs. So, the use of ECC-capable memory should be seriously considered also for client use cases.

The main takeaway from our study is that nearly-transparent strong memory protection is possible with current technology, especially if we consider recent developments in 2.5D chip manufacturing. It is also achievable for use cases where only a few processes need to be protected, such as banking, content delivery, and software licensing modules, whereas the rest of the traffic bypasses the MPE.
Future work includes upstreaming our MPE framework into gem5. This will allow interested parties to perform simulations tailored to their specific use cases. A further promising research direction is the development of strategies to reduce the impact of RMWs in some schemes, such as L3/oCC/MirE with 256-ary CGs, where in corner cases the performance penalty can exceed 5%, even thought the weighted average of all benchmarks remains under 2%. Finally, it should be confirmed whether the use of very large system caches or MPE private caches could bring performance penalties on a loaded system down to unloaded system levels.

Acknowledgements

Parts of Ionut Mihalcea’s work for this paper were performed in partial fulfilment of the requirements for a MSc degree [Mih22]. Ionut wishes to thank his academic supervisor Prof. Konstantinos Markantonakis, and his line manager at Arm, Paul Howard, for their unwavering support. The work by David Schall described herein was done during two internships at Arm Research and Arm’s Architecture and Technology Group, respectively. Part of the work performed during the first internship was documented in his Master’s Thesis [Sch19]. Héctor Montaner’s work was performed while he was an Arm employee.

The authors are grateful to Matthias Boettcher, Mike Campbell, Yuval Elad, Wendy Elsasser, Charles García-Tobin, Alexander Klimov, Jason Parker, Prakash Ramakryhani, Gururaj Sailswhar, Andrew Swaine, Peter Williams and Nicholas Wood for interesting and oftentimes eye opening discussions on memory protection.

References


[NIS15] NIST. FIPS PUB 202 – SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions. Technical report, National Institute of Standards and Technology, Gaithersburg, MD, United States,


A Selected full benchmark results

We collect here the detailed benchmarking results for the unloaded and loaded $S_8$ and $S_9$ runs.
<table>
<thead>
<tr>
<th>Program</th>
<th>Overhead % non-MirE</th>
<th>Overhead % MirE</th>
</tr>
</thead>
<tbody>
<tr>
<td>astar</td>
<td>13.2</td>
<td>20</td>
</tr>
<tr>
<td>bzip</td>
<td>2</td>
<td>10</td>
</tr>
<tr>
<td>gcc</td>
<td>10.3</td>
<td>40</td>
</tr>
<tr>
<td>chicken</td>
<td>15.7</td>
<td>80</td>
</tr>
<tr>
<td>combined</td>
<td>2</td>
<td>20</td>
</tr>
<tr>
<td>xalancbmk</td>
<td>9.26</td>
<td>4.64</td>
</tr>
<tr>
<td>sjeng</td>
<td>5.74</td>
<td>1.28</td>
</tr>
<tr>
<td>program</td>
<td>11.1</td>
<td>4.6</td>
</tr>
<tr>
<td>source</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>text</td>
<td>1.17</td>
<td>5.33</td>
</tr>
<tr>
<td>perlbench</td>
<td>1.33</td>
<td>6.4</td>
</tr>
<tr>
<td>checkspam</td>
<td>2.88</td>
<td>7.64</td>
</tr>
<tr>
<td>mcf</td>
<td>42.91</td>
<td>11.1</td>
</tr>
<tr>
<td>liberty</td>
<td>14.0</td>
<td>2.3</td>
</tr>
<tr>
<td>libquantum</td>
<td>10.9</td>
<td>2.16</td>
</tr>
<tr>
<td>cryptocompression</td>
<td>23.2</td>
<td>11.7</td>
</tr>
<tr>
<td>gobmk</td>
<td>3.34</td>
<td>7.64</td>
</tr>
<tr>
<td>hobmark</td>
<td>13.3</td>
<td>11.7</td>
</tr>
<tr>
<td>gobmk score</td>
<td>1.47</td>
<td>5.36</td>
</tr>
<tr>
<td>figure</td>
<td>12.5</td>
<td>2.4</td>
</tr>
<tr>
<td>program</td>
<td>11.2</td>
<td>2.69</td>
</tr>
<tr>
<td>source</td>
<td>2</td>
<td>2</td>
</tr>
</tbody>
</table>

**Figure 21:** S8/unloaded: L2/AES/28-32b MACs/64B CLs

**Figure 22:** S8/unloaded: L2/QARMA/64B CLs/32b MACs

**Figure 23:** S8/unloaded: Intel SGX/64B CLs (L3/AES/56b MACs/64B CLs)
**Figure 24:** S8 and S9/unloaded: L3/QARMA/split/128B CLs/32b MACs – runs with and without RMWs

**Figure 25:** S8 and S9/unloaded: L3/QARMA/oCC/32b MACs – runs with and without RMWs

**Figure 26:** S8 and S9/unloaded: L3/QARMA/oCC/MirE – runs with and without RMWs
Overhead %

Figure 27: S8/loaded: AMD SEV (i.e. L1/AES/64B CLs)

Overhead %

Figure 28: S8/loaded: L1/QARMA/64B CLs

Overhead %

Figure 29: S8/loaded: L2/AES/28-32b MACs/64B CLs
Figure 30: S8/loaded: L2/QARMA/64B CLs/32b MACs

Figure 31: S8/loaded: Intel SGX/64B CLs (L3/AES/56b MACs)

Figure 32: S8 and S9/loaded: L3/QARMA/split/128B CLs/32b MACs – runs with and without RMWs
Cryptographic Protection of Random Access Memory

**Figure 33:** S8 and S9/loaded: L3/QARMA/oCC/32b MACs – runs with and without RMWs

**Figure 34:** S8 and S9/loaded: L3/QARMA/oCC/MirE – runs with and without RMWs