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], 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

October 27, 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. It is also a very challenging goal to achieve with reasonable costs in terms of memory overhead and performance penalty. In turn, this usually leads to security posture compromises in products.

In this paper we review 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. We evaluate the storage and performance impacts and, as far as we are aware for the first time, we consider also the impact on performance when the measured benchmarks are the only running tasks or when they are just one task in an environment with heavy additional random traffic, thus simulating a cloud server under full load.

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 to 1/256$^{\text{th}}$ 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 memory bus bandwidth saturated server can thus be contained under 1%.

CCS Concepts: Security and privacy → Hardware-based security protocols.

Keywords: Memory Encryption, Memory Integrity.

Contents

1 Introduction

With the ever growing availability and use of *Computing as a Service* (a.k.a. Cloud Computing), the question of confidentiality and integrity of remotely processed information is becoming paramount. The first means to protect information entrusted to a process or a virtual machine is, classically, *Access Control (AC)*, which is configured by the operating system for the former and by the hypervisor for the latter.

Cloud tenants are becoming increasingly aware that their data can be compromised by various adversaries. The simplest class of adversaries consists in other tenants running unprivileged malicious software on the same hardware, using attacks that can circumvent AC, such as side channel attacks ranging from cache contention [Hu92, Koc96, Ber05a, OST06] to exploitation of micro-architectural features such as speculative execution. (Regarding the latter class of attacks, after the first seminal works on 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 even be targeted by entities having 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 [HSH+09, YADA17, WCJ+21] or even at run-time by chip interposition [Kuh98, LJF+20]. Hardening the operating system and the hypervisor 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, they range from simple memory encryption [Bes80, LTM+00, KFM05] to more advanced techniques to guarantee integrity of memory contents [MVS00, SCG+03, GSC+03, SLGL04, YGZ05, YEP+06, SOD07, RCPS07, CL10, HS10, CRSP11, Gue16a, WUS+17, SNR+18, JLK+23]. The latter range from tables of hashes in memory, each entry associated with a memory region, to structures 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 performance and memory space 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 SGX *enclave page cache* [Gue16a] set the research in motion once again.

Even restricting solely to the cryptographic techniques, as we do in this paper, it is very difficult to compare different technologies since any two papers on the subject will almost never use the same benchmarking suite, memory subsystem, cache sizes, and overall system load. Furthermore, most benchmarks are performed on systems without memory bus contention. This is not realistic, as the main application for these technologies seems to be cloud computing, servers on which hundreds of processes can run concurrently and contend for shared resources.

In this paper we systematise the comparison between various techniques and their combinations, including also some new ideas. We focus only on technologies that require implementation only 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 benchmarking suite. We give here a short summary of our results:

(i) 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.

(ii) We confirm that protection levels that provide temporal uniqueness and integrity can have a lower performance impact than simple full *Memory Encryption (ME)* methods (with or without integrity) because they can use counter modes for encryption, thus removing the cipher from the critical path. This said, a lightweight cipher optimised for memory encryption like `QARMA` [Ava17] still provides better performance than the `AES` [DR02a] even in those modes.

(iii) Even though using longer *Cache Lines (CLs)* (say, 128B instead of 64B) may slightly degrade performance, it is better suited for memory protection because, all other things being equal, storage for *Metadata (MD)* such as MACs for integrity and counters will be halved, easing the pressure on the memory bus and on any MD cache.

(iv) Asynchronous integrity verification brings only a minor performance improvement, so it can be omitted, keeping implementations simpler and thus less bug prone.

(v) For encryption methods based on counters, advanced counter compression makes it viable to store them 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. Our simulations show performance penalties smaller than 2% even with heavy bus contention.

(vi) We 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 on the facing page we define the threat models that memory contents face, the corresponding adversarial capabilities, and accordingly the various levels of protection that are required to thwart these adversaries. Section 3 on page 10 is a brief summary of the background needed for our study (an extended treatment of the background material is given in Appendix B on page 42). In Section 4 on page 13 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 presented and discussed in Section 5 on page 20. In Section 6 on page 27 we make practical recommendations for cloud and client use cases.

Acknowledgements

Parts of Ionuț Mihalcea's work for this paper were performed in partial fulfilment of the requirements for a MSc degree [Mih22]. Ionuț 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].

The authors are grateful to Héctor Montaner for his help in programming and testing, and to Matthias Boettcher, Mike Campbell, Yuval Elad, Wendy Elsasser, Charles Garçia-Tobin, Alexander Klimov, Jason Parker, Prakash Ramrakhyani, Gururaj Saileshwar, Andrew Swaine, Peter Williams and Nicholas Wood for interesting and oftentimes eye opening discussions on memory protection.

2 Systematisation of the problem

2.1 Definitions

The software-accessible volatile memory attached to a memory controller is viewed as an array of 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.

The integrity information computed on a CL's plaintext or ciphertext 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 provides *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 is said to provide *temporal uniqueness* (also known as *freshness*) when repeated writes of the same plaintext to the same location result in different outputs. This is achieved by associating a non-repeating nonce, such as a counter, to each CL and including it in the computation of the function.

In what follows by *mode (of operation)* we understand a general purpose encryption mode of operation, A 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.

We use the terminology *on-chip* to denote components that are 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

First of all, we need to define what we mean by *memory protection* and get beyond the hype that is markets ME as the apparent solution to all security issues – even if they have not been formalised.

Our first observation is 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. A suitable way to characterise the adversaries is by *Adversarial Models (AMs)* that depend on their type of access to the target devices and their resources, i.e. essentially budget. A possible taxonomy is the following:

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 run software on the target. Beside the exploitation of software vulnerabilities, this adversary can mount Rowhammer attacks [KDK+14, Mut19, MK20]. In scope are also any side channel attacks that rely on software-exposed built-in hardware features to perform physical unit and time measurements, or give partial or full read access to memory contents. Access to software-exposed power management control interfaces may enable glitching attacks, similar to those that that exploit a hardware interface (for instance [CVM+21]). Integrity violation is only a partial concern, as some effects of memory corruption are 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 can gain access to exposed interfaces and communication buses but they do not have the capabilities to access on-chip communication interfaces. This adversary will only perform passive attacks, such as:

- Side-channel analysis that requires close proximity, contact or connection with the target device, which includes also recording or measurements of any off-SoC signals;
- Eavesdropping the content of off-SoC memory, either at run-time via memory bus probing, chip or module interposition [Kuh98, LJF+20], abuse of DMA channels [Fri16] or by means of a cold-boot attack [HSH+09, YADA17, WCJ+21].

AM3 This adversary has the same level of access as in the previous model. However, they will also perform *active* attacks, such as blocking memory transactions or signals, replaying them, corrupting them, injecting new ones [KLR+20], performing some form of fault attack. Because of the similarity of the involved technologies, there is little difference in the expertise required beyond the previous adversarial model, whereas resources may need to be have a higher level of precision. The difference in complexity and cost and of the *countermeasures* is also a key factor in distinguishing the two models. Furthermore, since active attacks are more easily detectable, especially if they need to perform several tries triggering repeated failures, an adversary may choose not to mount them even if possessing the required capabilities. Examples of threats mounted by this type of adversary are [BBKN12, BR12, ZDC+12, MDH+13, CVM+21].

Within AM3, following the industry we distinguish two cases:

AM3.(i) Adversaries that limit themselves to corrupt individual memory locations; and

AM3.(ii) Adversaries that replace a memory region together with any associated MD.

AM4 This adversary, in addition to all of the above capabilities, can mount highly invasive attacks at the chip or package level that go far beyond the previous models and require considerable experience, resources, and time to succeed. The attacks this adversary can mount range from micro-probing attacks [Sko17] to actual chip reverse engineering and editing using a Focused Ion Beam Microscope [TJ09, SAFT16, HTLW21].

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 five types of adversaries defined above, and what are their costs in terms of memory overhead and performance?*

Before we establish the defences against each type of adversary, some observations are in order.

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, and therefore it can be argued that this model should be subsumed into AM1.*

Remark 2.2 *AM4 is out of scope for the research described in this paper.*

Remark 2.3 *In what follows we shall assume that appropriate AC policies are in place to prevent unauthorised agents within the SoC from accessing memory.*

2.3 Protection levels

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

L1 A simple encryption of the entire memory to be protected is implemented to defeat adversaries AM1 and AM2, except for: AM1 adversaries with a ciphertext-revealing side channel, or AM2; *and* that exploit memory access patterns as a side channel. The encryption function provides spatial uniqueness to reduce detection of data patterns. Temporal uniqueness and integrity verification are not provided as they are not required against the considered adversaries.

L2 In order to thwart adversaries of type AM3.(i), each CL is encrypted and augmented with an integrity tag. No freshness is provided. This is not sufficient against Adversary AM3.(ii), that calls for either the integrity tags or the freshness information be protected themselves.

L2+ Same as L2 except that freshness information is provided and used as an additional input to the encryption and integrity functions. This serves as partial hardening against adversaries of type AM1 with a ciphertext-revealing side channel or type AM2, that exploit memory access patterns as a side channel.

L3 Against adversaries of type AM3.(ii), additionally, full integrity is provided.

We combine the following types of technologies to implement the above protection levels:

1. ME primitives (Section 3.1 on page 10) and modes (Section 3.3 on page 11);
2. Authentication primitives (Section 3.2 on page 10);
3. Integrity and anti-replay structures (Section 3.4 on page 11) such as tables and trees; and
4. Physical mechanisms to protect memory from tampering, such as including memory 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 2.4 *In this paper, we only consider solutions that need the security perimeter to be no larger than the physical package of the SoC. Hence, out of scope are "smart memory" technologies as those introduced in [AN17]. 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, such smart memories are very expensive. They require the implementation of mutual attestation between memory controllers and memory chips, and duplicate cryptographic engines in each memory chip: We note that putting these security controls only on the on-board controller of the DIMM would not completely remove the risk of interposition. But 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, esp. if the latter is on-chip.*

We now discuss the protection levels offered by some state of the art solutions.

(i) The *Memory Encryption Engine (MEE)* in Intel's SGX [Gue16a] is a L3 solution.

(ii) Memory protection in Intel's TDX is provided by the *Multi-Key Total Memory Engine with Integrity (MKTMEi)* [Int21c, Section 2.A]. It provides encryption without freshness, and integrity with 28b tags. These tags are computed using SHA-3-256, and are stored together with the ciphertext in 28 repurposed ECC bits [Int21a, Section 16.2]. In our terminology the MKTMEi is a L2/MirE solution, where MirE means *MACs in repurposed ECC bits.*

(iii) Amd's SEV [KPW16] uses AES-128 in a construction that gives spatial uniqueness. According to our classification, SEV is a L1 solution.

(iv) SYNERGY [SNR+18] is a ECC memory specific proposal. It is a L3/MirE solution.

(v) CSI:Rowhammer [JLK+23] improves on SYNERGY's usage of the ECC bits by shortening the data MACs to 56 bits to make room for parity bits. It implements a L2/MirE solution.

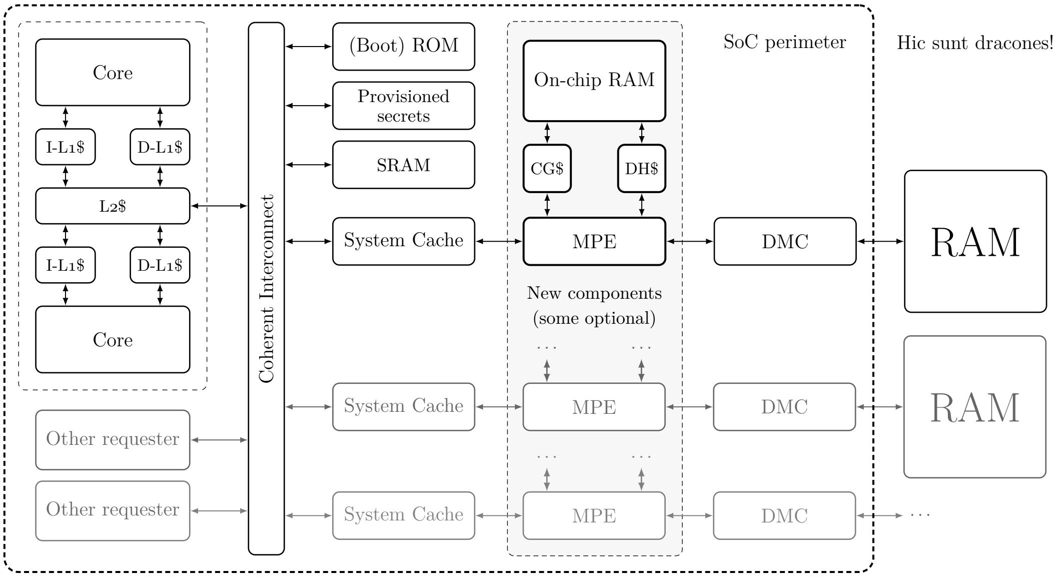

Figure 1: Simplified system level view of a SoC with Memory Protection Engine(s)

2.4 System level view of the technical solution

To answer the question posed in Section 2.2 on page 6 we introduce a set of HW components, of which the central one is called the *Memory Protection Engine (MPE)*. This is not a new concept: all cryptographic memory protection designs cited so far use such a component, usually called somewhat reductively a *Memory Encryption Engine (MEE)* even when it does more than encryption.

Its placement in a SoC system level view is depicted in Fig. 1. 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.

2.5 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. Constraints on area and power will restrict which solutions can be considered for viability, but relaxing these constraints can often 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, and in light of the breadth of our comparisons, we focus mainly on performance penalty and memory overhead.

3 Background

Several technologies have been used to protect memory contents, and many have been cited in the introduction. This is a short summary of the techniques we considered in the development of this paper. An extended version of this summary is given as Appendix B on page 42.

3.1 Memory encryption primitives

Block ciphers are the most common type of functions used to encrypt memory. In *direct encryption* they are 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.

The most important block ciphers for ME are: The standard general purpose memory encryption block cipher `AES` [DR02a, DR02b]; The 64b block cipher `PRINCE` with design goals of low latency and low area [BCG+12] has been designed as a compact and fast ME cipher; and the *Tweakable Block Cipher (TBC)* `QARMA` [Ava17] that has been designed specifically for the ME use case, and includes a third input beside text and key, called tweak, to select the function applied to the text in an agile way in order to facilitate the design of secure modes of operation.

Stream ciphers can also be used, but they lack parallelisability and have an initial high latency, and there are some newer designs like the block cipher `SPEEDY` [LMMR21], which is however difficult to use in some use cases because of its unusual block size of 192 bits.

In this paper therefore we shall use `AES` and `QARMA`. Note that the latter has been used in various papers on cryptographic memory protection [UWM19, JLK+23].

3.2 Authentication primitives

Standard hashing functions such as `SHA-2` [NIS12] or `SHA-3` [NIS15] can be turned into a MAC by means of established constructions such as `NMAC` or `HMAC` [BCK96] and used to verify memory integrity. However, the resulting schemes are very slow and not intrinsically parallelisable.

`SipHash` [AB12] is a cryptographically strong *Pseudo-Random Function (PRF)* that is used as MAC. We do not consider it here because it is based on 64b integer additions, which makes the latency of HW implementations too high, and it is not parallelisable.

Encrypted or hashed *Universal Hash Functions (UHFs)* [CW77, CW79] are a better choice. UHFs admit fully parallelisable constructions such as polynomial or multi-linear functions of the input, computed over a binary Galois field which allows for HW-friendly carry-less arithmetic. Intel uses an encrypted multi-linear hash over $\mathbb{F}_{2^{64}}$ in SGX [Gue16b].

Remark 3.1 *The UHF-based MAC constructions always encrypt the universal hash because an adversary would be otherwise able to successfully manipulate it. However, a universal hash is assumed to be beyond the reach of the adversary if stored in MPE-private on-chip memory. Even though we do not use on-chip RAM for DHs (except for one single test run), this principle applies also to on-chip caches: The DHs are encrypted only when evicted from the latter, and the cached values can be verified more efficiently because an encryption or decryption operation is avoided. This approach also offers reliability and security advantages, as described in Remark 4.1 on page 14.*

TBC-based *Parallel MACs (PMACs)* [Rog04] can also be used. PMACs are slightly more expensive than encrypted UHFs because the text is first processed by encryptions instead of Galois multiplications, but they have the advantage that they can be used for error detection and correction beside integrity, as in IVEC [HS10], SYNERGY [SNR+18], and CSI:Rowhammer [JLK+23].

A checksum of the plaintext can also be used, as in Rogaway's *Offset Codebook mode (OCB)* mode [Rog04] (cf. Fig. 41 on page 49), provided that the encryption offers freshness, block-wise spatial uniqueness, and good per-block diffusion properties. However, for protection levels that provide freshness we use OTP modes to reduce critical path length, and since in this case an encrypted checksum would leave the ciphertext malleable, we need to use a proper MAC.

3.3 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 XEX construction [Rog04]. 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 plaintext and to the ciphertext. 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 $M_i = \gamma^i \cdot M_0$. This results in a variant of Rogaway's OCB mode [Rog04] and is represented graphically in Fig. 35 on page 47. With a TBC such as QARMA, the address of each block can be used directly in the tweak, cf. Fig. 36 on page 47, without the need for Galois multiplications.

Let us now consider OTP encryption,. The AES is used in a counter mode as depicted in Fig. 34 on page 47. Intel's SGX uses a variant of the *Galois/Counter Mode (GCM)* mode, namely it uses a counter-based OTP encryption followed by a *multi-linear (ML)* Carter-Wegman UHF (in place of GCM's polynomial authenticator). QARMA is used in the *CounTeR in Tweak (CTRT)* mode [PS16], depicted in Fig. 37 on page 47. We use a ML UHF, except when we are repurposing the ECC bits to store the MACs, in which case we use a PMAC as mentioned earlier.

3.4 Memory integrity structures

The technically simplest way to ensure full integrity of untrusted memory is to use a *Merkle Tree (MT)* [Mer80], depicted in Fig. 44 on page 50, MTs were used in one of the earliest architecture for memory protection, the XOM system [MVS00]. In [GSC+03] MT nodes are cached, whereby a cached node is treated as trusted, halting the tree verification traversal. This reduced the performance penalty from a factor 10 to just 22%.

In [RCPS07] it is observed that if a counter-based mode is used, the counters are often shorter than the hashes in a common MT (say, 64b vs. 128b). Then, it is proposed to build the MT on the memory region containing the counters, reducing the size of the MT. This smaller tree is called a *Bonsai Merkle Tree (BMT)*. This also allows to reduce the size of the hashes while providing the same level of security.

Hall and Jutla in [HJ05] build on this idea by repeatedly using a counter and a hash to protect a groups of counters, instead of just hashing them recursively into a MT. This way a new type of tree is constructed, the *Parallelisable Authentication Tree (PAT)*, where the nodes are *pairs* (ν, h)

consisting of a nonce ν and a MAC h computed over the children nodes (cf. Fig. 45 on page 52).

Intel's [Gue16a], cf. Fig. 46 on page 52, reorganizes the PAT by grouping all the counters in a set of sibling PAT nodes together with the MAC that in the parent node of the PAT. The nodes of the new resulting *Counter Tree (CT)* are called CGs and in Intel's implementation contain eight 56b counters and a 56b MAC.

A different type of tree is the *Tamper-Evident Counter (TEC)* treen [ECL+07] (cf. Fig. 47 on page 52). In the TEC tree a wide block cipher in a direct mode is used to encrypt data or counter blocks together with a copy of the parent counter and with the address of the block. The correct decryption of the address guarantees authenticity.

An important optimisation consists in using *split counters*. In [YEP+06] a group of a counters is replaced by a group composed of a *major counter* and $a' > a$ smaller, *minor counters*, so that the two types of CGs have the same size. The freshness information associated to a child consists of the concatenation of the common major counter with the minor counter corresponding to the child. 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 major counter in the same group is increased, all minor counters in the group are reset to zero, and the memory blocks corresponding to the minor counters in the same group must be re-encrypted (for leaf nodes), or the hashes recomputed (for non-leaf nodes) in bulk with the updated freshness information. 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.

3.5 Memory overhead of integrity trees

In Table 1 on the next page we compare all the main variants of integrity trees we encountered here. We assume that CGs have the same size of a CL, including also their own tags if embedded, whereas a MAC can cover 1, 2, or 4 CLs.

Remark 3.2 *There are essentially only two ways of reducing memory overheads: (i) compressing the MD or (ii) move it somewhere else ("sweeping-under-the-rug"), for instance on-chip. The latter can improve performance, simplify memory management (no carveout has to be identified and no special AC policies have to be configured and locked) but comes with a high hardware cost. The former approach will be less expensive in terms of physical resources, but it may have a negative impact on performance.*

For counters, compression means the use of split counters, which, as we shall see, improves performance. Sweeping-under-the-rug means storing the split counters on-chip. The two approaches may be combined.

For MACs, compression means either to use shorter MACs or to use longer CLs. The first option needs to be used wisely, otherwise security may be affected too adversely. The second option may considerably worsen performance. Sweeping-under-the-rug can mean storing the MACs on on-chip memory or repurposing the ECC bits for MACs.

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

		Overhead	
Type of Tree	*CL size:*	*64B*	*128B*
Merkle Tree $\ell_H = 128$, $a = 4$, resp. 8		33.3 %	16.7 %
Monolithic CT with embedded MAC and $\ell_c = \ell_h = 56$–64			
• $\ell_H = 64$; $n = 1$; $a = 8$, 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 %
BMT with $a = 8$, resp. $a = 16$, and $\ell_H = \ell_h = 64$		29.2 %	13.4 %
Split CT with embedded MAC and $\ell_c = \ell_h = 56$–64			
• $\ell_H = 64$; $n = 1$; $\ell'_c = 6$, resp. 7; $a = 64$, resp. 128		14.1 %	7.04 %
• $\ell_H = 32$; $n = 1$; $\ell'_c = 6$, resp. 7; $a = 64$, resp. 128		7.84 %	3.91 %
• $\ell_H = 32$; $n = 2$; $\ell'_c = 6$, resp. 7; $a = 64$, resp. 128		4.71 %	2.34 %
• $\ell_H = 32$; $n = 4$; $\ell'_c = 6$, resp. 7; $a = 64$, resp. 128		3.15 %	1.57 %
• $\ell_H = 32$; $n = 1$; $\ell'_c = 3$; $a = 128$, resp. 256		7.04 %	3.52 %
• $\ell_H = 32$; $n = 2$; $\ell'_c = 3$; $a = 128$, resp. 256		3.91 %	1.95 %
• $\ell_H = 32$; $n = 4$; $\ell'_c = 3$; $a = 128$, resp. 256		2.35 %	1.17 %
PAT with $a = 8$, resp. $a = 16$, and $\ell_H = \ell_h = 64$		28.6 %	13.3 %
TEC tree with $a = 8$, resp. $a = 16$, and $\ell_H = \ell_h = 64$		42.9 %	20.0 %

Legend: *$\mathcal{L}_{CL}$, ℓ_H, ℓ_h, ℓ_c, and ℓ'_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. These technologies are summarized in the following list, where the entries marked with † contain technologies that are new contributions in this paper; and those marked with * describe variations not hitherto compared to each other to the best of our knowledge:

- Use of the `AES` or `QARMA` ciphers;
- Size of the MACs (32b or 64b);*
- Type of counter tree: 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
- Runs are measured on two types of systems: systems where only the benchmarking suite is running; and systems where additional traffic is saturating both the MPE and the available memory subsystem bandwidth.*

Remark 4.1 *A new idea we adopt in our implementations consists in evicting DHs that are stored in normal RAM in blocks, 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 increasingly 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, then the four minor counters corresponding to the four DHs that must be directly encrypted together and their common major (and possibly middle) counter are concatenated together. This is used to create a tweak for `QARMA`-128.

Remark 4.2 *Many of the above choices are to some extent independent of each other. However, the selection of the encryption and authentication schemes follow from chosen the level of protection and primitives, as well as on other factors, i.e. whether counters are kept in on-chip memory or ECC bits are repurposed for data MACs. This will be clear in the next section.*

Remark 4.3 *When multi-CL MACs are used, each CL is still encrypted individually and is associated with a monolithic or major counter. Hence, evicting a CL from the LLC will not require the re-encryption of any adjacent CL.*

4.2 Technologies used for each level

We list the technologies used to implement the protection levels defined in Section 2.3 on page 7.

`L1` If `AES`-128 is the chosen encryption primitive, a CL is encrypted using the XEX mode (Fig. 35 on page 47), with the PA as the tweak. If `QARMA`-128 is chosen, it is used in Tweaked *Electronic Codebook (ECB)* mode as in Fig. 36 on page 47, with the PA as tweak.

`L2` The same encryption modes are used for `L1`. Hashing is done by a 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 means that an evicted data hash block is encrypted using its own PA. Since this address is linearly related to the PAs of the corresponding CLs, there is no security concern (all security proofs of the similar *Authenticated Encryption (AE)* modes carry over). This approach has good security and reliability implications, cf. Remark 4.1.

L2+ This level provides freshness over L2. A counter mode is used with both AES and QARMA, where the text input is a concatenation of PA and freshness information for AES-128, and CTRT mode (Fig. 37 on page 47) is used with 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. Hence, there is no need to feed the freshness to the MAC computation function, which has the positive side effects of eliminating a potentially blocking dependency in the hardware. Therefore, we use direct encryption for DHs as in L2.

L3 We use the same encryption mechanism as in L2+. Full integrity is achieved by preventing the adversary from successfully tampering with either the DHs or the CGs *and* including the freshness in the tag computation. Thus, an adversary may still be able to replace an CL together with either its DH or CG, but not both.

Technically, protecting the DHs or the CGs is achieved by either using an integrity tree, or by storing them on-chip: in both cases we consider only the protection of the CGs, as they require less memory.

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 memoryto be non-interposable and MPE private, hence outside adversarial control.

MirE *MACs in repurposed ECC bits.* 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 [SNR+18, JLK+23], the tag is computed using $\texttt{QARMA}_5$-64-σ_0 and is truncated to 56b to reserve eight bits for parity.

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 determine that unusual activity is occurring.

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 the authentication key is 128b.
4. Data MACs should be at least 32b long.

5. Monolithic counters must be at least 64b long. The aggregated length of a major counter with a minor counter (or major plus middle plus minor) should be also at least 64 bits.

A consequence of the above choices, 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

Developing new features for integration into processors and SoCs is an expensive and error-prone endeavour with multiple layers of risk. To provide a comparison of potentially thousands of combinations of techniques it would financially unjustifiable 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 be integrated, tested, and benchmarked on various metrics. This allows us to create very accurate models without having to implement all details. For instance, the latencies of cryptographic primitives are derived from actual implementations, and they are inserted as delays into a simulation framework, simplifying the implementation of the simulation and speeding it up at the same time.

The prototypes used in this paper are built in the gem5 simulator [BBB+11, LAA+20]. 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.

4.5 The simulated system

The simulated system is a single core A72, with a 2GHz CPU frequency and a 1GHz system frequency. The CPU cache hierarchy includes separate L1 instruction (48KiB, LRU replacement policy, 3-way set associative, 1 cycle latency) and data (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 latter plays in our prototyping the role of the System Cache. The memory system consists in 16GiB DRAM as dual-rank DDR4 DIMMs. External memory has a load and access pattern dependent latency on the order of ≈75ns in most cases, and a bandwidth asymptotically reaching of 14GiB/sec at higher latencies (up to 300ns).

The MPE-private caches are 4-way set associative with a LRU replacement policy. The on-chip memory has a access latency of 50ns, and a bandwidth of 20GiB/sec.

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 $\texttt{QARMA}_{11}$-128-σ_1 and 2.2ns for $\texttt{QARMA}_5$-64-σ_0. The latter two implementations are also pipelined, and are included in the Qameleon NIST Lightweight Cryptography Standardization Process submission [ABB+19]. This latency of $\texttt{QARMA}_5$-64-σ_0 is also used in [JLK+23].

4.6 Benchmarking suite and testing methodology

We benchmark the chosen compositions of technologies on the SPEC2006 suite [Hen06].

Simulations of hardware systems via software models such as 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. Our benchmarking strategy relies, instead, on previously characterized SPECint 2006 workloads [SPHC02].

Each SPEC2006 benchmark has been analysed at run-time and split into a set of small workloads that distill the performance characteristics of the initial benchmark. The workloads are weighted to more closely approximate the significance of their content to the entire benchmark. In our case 10 workloads of roughly 30 million instructions each are used to estimate performance for each benchmark in the suite. After each of these is run with our test system, the metrics of interest are combined to produce one representative result for the entire benchmark. The combining step takes into account the values produced by the workloads and their weights.

4.7 Description of the plan of simulations

Comparing thousands of different configurations is not only unfeasible in hardware, but it would require too much time and resources 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 combination, 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.

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 additional technologies may include mono (for monolithic counters), split (counters), oCC, or 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, "{Intel} TDX" is equivalent to L2/AES/MirE, "{Intel} SGX" to L2/AES/mono, and "{AMD} SME" to L1. oCC always implies split. The shorthand L3/oCC is used to denote the combination of L2+ with oCC. We understand L3 *without* oCC as a full integrity capable scheme based an on integrity tree and *neither counters nor hashes on-chip.*

Stage 1 We initially focus on the state of the art and our most basic technologies.

We compare AMD SME (i.e. L1 with AES), L1/QARMA, L2/AES, Intel TDX (i.e. L2/AES/MirE), L2/QARMA, L2/QARMA/MirE, L2+/QARMA with both monolithic and split counters, SGX, L3/QARMA with split counters – all with and without a DH$ if it is not fixed by the manufacturer's architecture. We also compare in some cases the performance with 32b vs. 64b MACs.

For SGX, hash encryption is OTP as described by intel. We use this method also for the split counters variant of it (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. For every other variant counters are split, MACs are 32b, and encrypted in groups, directly.

For levels 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.

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.

Stage 2 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.

Starting with Stage 3, we assume that the MPE has both private caches, where the size of the DH$ is 16KiB, and the size of the CG$ to 256KiB. Similarly, levels L2+ and L3 will use split counters, except when explicitly indicated, or when SGX is benchmarked.

Stages 3 and 4 are intermediate stages towards the more interesting combinations, and we restricted the simulation set to QARMA only for the encryption, as AES results would be substantially equivalent.

Stage 3 Consider 64B and 128B CLs for L2, L2+, and L3. A CG and a CL have the same size.

Stage 4 We compare synchronous to asynchronous verification for for L2, L2+, and L3.

The next few steps will test the impact of the techniques mentioned in Remark 3.2 on page 12.

Stage 5 We analyse the impact of on-chip memory on realisations of full integrity levels.

As the MAC overhead is larger than the CG overhead, we do not consider the case where the MACs (actually, 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-128 and QARMA-128.

Stage 6 We consider here the performance impact of repurposing the ECC bits for tags (see Appendix B.5 on page 53). We compare four schemes: L2, L2+, L3, and L3/oCC, all with and without MirE. To keep the total number of tests manageable, we measure the impact for both 64B and 128B CLs when the counters are off-chip, and consider only 128B CLs oCC.

If we store MACs in the ECC tag bits, we do not need a DH cache, and the MACs are computed as described in Appendix B.2.4 on page 46.

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.

What follows is some off-path branching:

Stage 7 We want to show what can be optimised storage-wise when when cannot repurpose ECC bits, yet we cannot store the MACs in on-chip memory. MACs are thus stored off-chip, but Cached Incremental Hashing for multi-CL MACs can be used to reduce their memory storage requirements [ASC+19]. This makes sense only when we have already chosen to use 128B CLs, as these already lead to a halving of MD storage requirements.

We test levels L2, L2+, L3, and L3/oCC, all with 1,2, or 4 CLs covered by a single MAC.

These runs are performed 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 AES performance will follow the already seen differences for the previous cases.

Stage 8 We select some combinations from the above and show all individual benchmarks in the suite.

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

Stage 9 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.

By doing this can we estimate an upper bound on the actual time spent performing RMW operations, in order to address the concern that these may lead to performance degradation. The selected combinations are the last five of **Stage 8**:

- L3/QARMA/split/128B CLs/32b MACs
- L3/QARMA/oCC 128-ary/32b MACs
- L3/QARMA/oCC 256-ary/32b MACs
- L3/QARMA/oCC 128-ary/MirE
- L3/QARMA/oCC 256-ary/MirE

4.8 ECC-memory vs. non-ECC memory

We are not explicitly considering the 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) but since this is performed in a very quick burst within a single command, the performance loss is much smaller than the overhead. In fact, the usual figures are of penalties smaller than 2% and benchmarks show a loss usually smaller than 0.5% [Bac14]. We thus observe that:

1. If deployed on servers, the schemes that do not repurpose the ECC bits are assumed to be still using ECC bits for error detection and correction; and
2. For all the methods that do not repurpose the ECC bits, the relative performance losses should be nearly identical regardless of whether they run on non-ECC or on ECC memory.

Therefore, we do not consider the impact of ECC memory as a separate parameter.

4.9 Unloaded vs. loaded systems

The various Stages described in Section 4.7 on page 17 are first run on an *unloaded* system, i.e. in a simulated environment where the only task running is benchmarking. The results of these runs are reported and discussed in Section 5.1.

Then, we also want to give an upper bound for the performance penalties that occur in a fully *loaded* system, with potentially hundreds of processes running on dozen of other requesters contending the bandwidth of the memory subsystem, such as in a cloud server. Of course, it is unfeasible to run that many processes in the simulated environment, for the reasons mentioned in Section 4.6 on page 16. We instead inject synthetic traffic upstream of the MPE, but after the L2 cache. The amount of traffic injected is 8GiB/se, which corresponds to the point where the latency offered by the memory subsystem starts to diverge for a SGX-like L3 MPE covering the entire memory, with mostly linear synthetic traffic and with synchronous MAC verification (as it will be seen from the discussion of the benchmark runs in Section 5, the most likely implementation of integrity verification). The value can be inferred from Fig. 2 on the facing page.

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 the runs on a loaded system are reported and discussed in Section 5.2 on page 24.

5 Results and discussion

We now analyse the results of the selected test runs.

5.1 Unloaded system

We first observe that changing CL length from 64B to 128B in our simulated system on the chosen benchmarking suite slows down the system by 1%. Each run is compared to the baseline with the same CL length.

- For **Stage 1** on an *unloaded system* (see Fig. 3 on page 22) we observe 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.
  - For L1 and L2 the latency of AES in XEX mode causes a significant higher slowdown than the use of QARMA-128. Even for L3 the use of QARMA-128 significantly improves the performance with respect to AES, because the generation of the OTP, while it can be

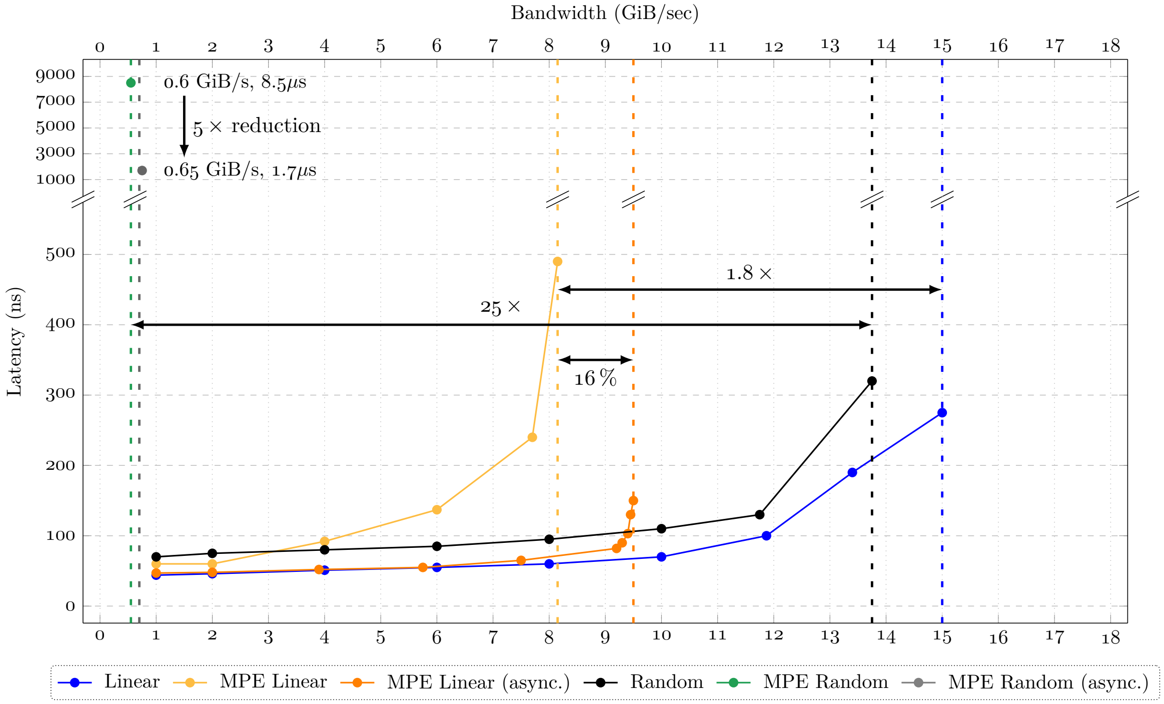

Figure 2: Worst-case bandwidth/latency impact

performed in parallel with a memory fetch, is still increasing memory latency to the point that it has a noticeable effect.

- We confirm that for L3/split counter trees are superior to monolithic trees in both memory overhead (see Table 1 on page 13) and performance.
- A small DH$ does not significantly affect performance on the selected benchmarks.
- As expected, using 64b MACs results in worse performance than using 32b MACs, but the difference is contained.

- **Stage 2** results (see Fig. 4 on the following page) confirm expected significant performance gains with larger MPE caches, the CG$ cache having a higher effect than the DH$.

- **Stage 3** (the results are combined with those of **Stage 4** in Fig. 5 on the next page) proves that the impact of these technologies is comparable across systems with 64B CLs and systems with 128B CLs. Note that using 128B CLs slows down the system by 1% on average, and the data for 128B CL systems is relative to the baseline with a 128B CL and no MPE, so 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 MD memory requirements (cf. Table 1 on page 13).

- **Stage 4** results (see Fig. 5 on the next page) suggest that keeping MAC verification synchronous does not impact performance in a significant way.

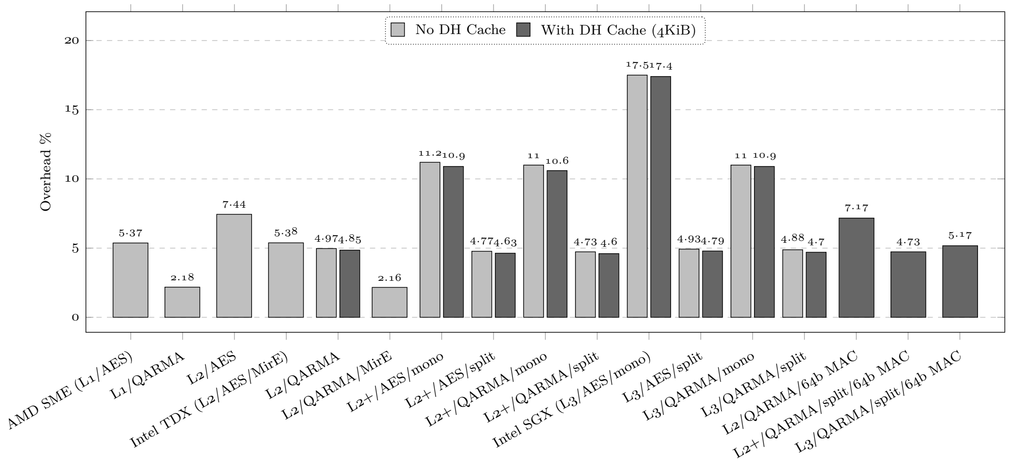

Figure 3: Stage 1/Unloaded system: Comparison of base levels and state of the art; MACs are 32 bits long except for TDX (28 bits), SGX (56 bits) and 64 bits where indicated; The CG$ is 64KiB as in SGX

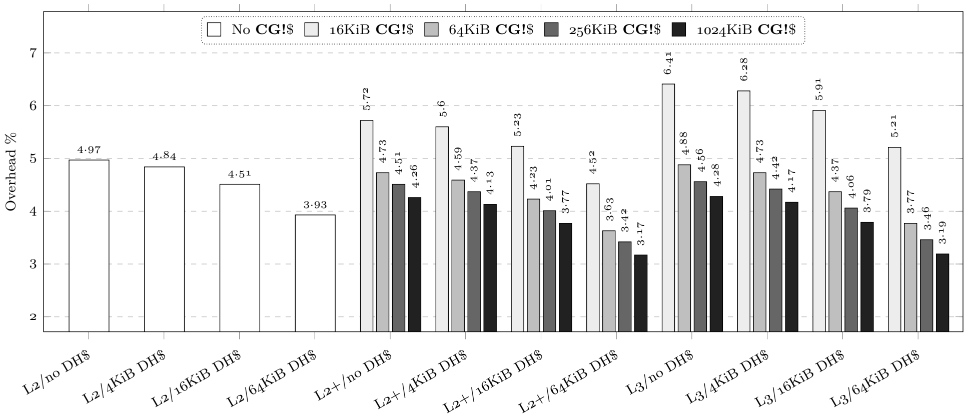

Figure 4: Stage 2/Unloaded system: Impact of MPE cache sizes; ME cipher is QARMA-128; CLs are 64B

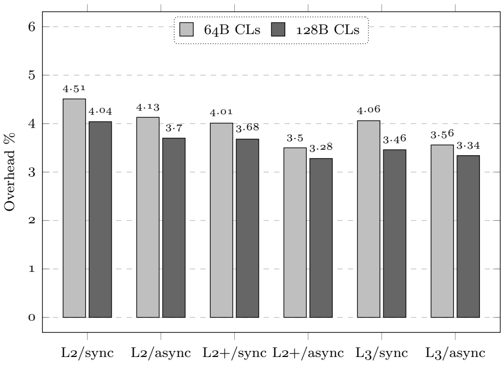

Figure 5: Stages 3 and 4/Unloaded system: Impact of System CL size and asynchronous MAC verification; ME cipher is QARMA-128

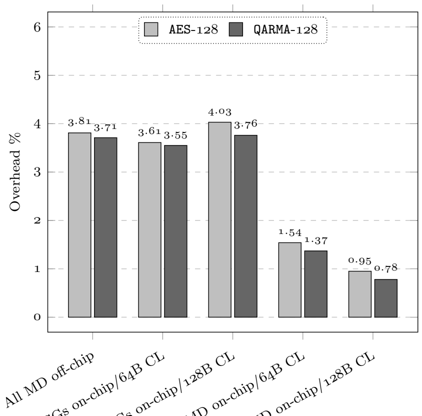

Figure 6: Stage 5/Unloaded system: L3; Impact of storing MD on-chip

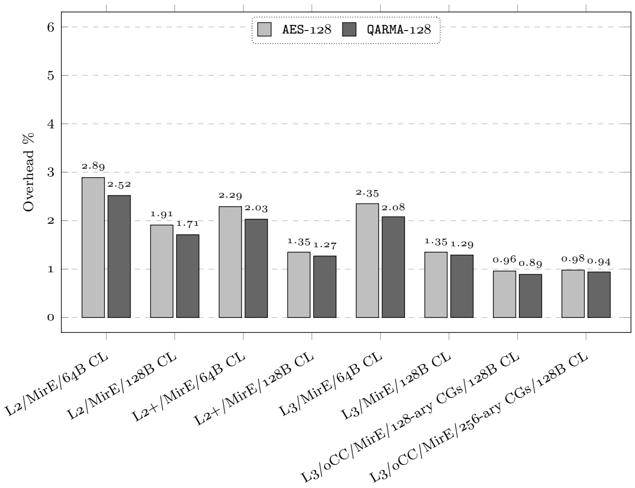

Figure 7: Stage 6/Unloaded system: Impact of repurposing ECC bits for MACs

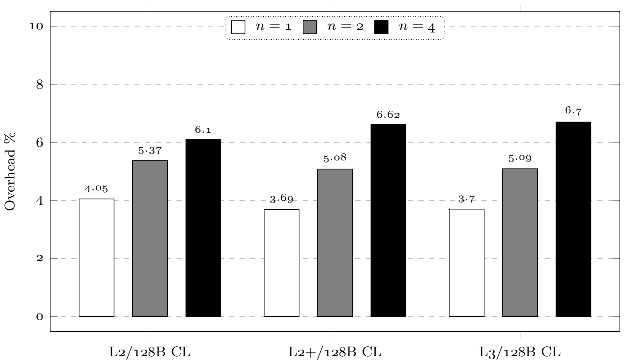

Figure 8: Stage 7/Unloaded system: Impact of using multiple-CL MACs (128B CL)

- **Stage 5** results (see Fig. 6) show, as expected, that relieving the contention on the memory bus between data and MD reduces the performance overhead.

- **Stage 6** results (see Fig. 7) prove that combining oCC and MirE provides the highest level of memory protection with extremely low performance penalties.

- The results of **Stage 7** (see Fig. 8) show that multiple-CL MACs effectively reduce memory overheads, but at a significant performance price. Noter that L2+ and L3 performance is virtually identical. This is due to the fact that MAC traffic becomes dominant whereas the need to cover freshness for adjacent CLs profits from counter locality.

  Note, however, we do not implement evicted cache line compression as in [TSB18], which would have allowed to store a MAC in the cache line in external memory if the data section can be sufficiently compressed, thus reducing the amount of memory accesses. To get an idea of the improvements in that case we refer to the paper. Following it, we can estimate that adding compression may approximately halve the performance penalties.

- **Stage 8**: The performance of the individual SPEC2006 benchmarks (Figs. 15 to 22 on pages 37–39) shows a few expected results, namely that some programs suffer significantly more than average under every MPE configuration, and they are the usual suspects, for instance `gcc g23`, `gcc s04`, `mcf`, `libquantum`, and `xalancbmk`. 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 the arity of the integrity trees by split counters is instrumental in bringing down 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 very good performance, esp. if the tasks profit from good spatial locality properties.

- **Stage 9** (see Figs. 20 to 22 on page 39) 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

Our benchmarking suite runs 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 on the loaded system w.r.t the unloaded one. Changing CL length from 64B to 128B in our simulated loaded system on the chosen benchmarking suite makes the system faster by 2.7 %.

- For **Stage 1** on a *loaded system* (see Fig. 9 on the facing 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 however roughly the same as for the unloaded case: A performance penalty reduction by a factor of roughly 3 occurs between the use of monolithic counters and/split counters; and a small DH$ offers only a minimal performance improvement.

- **Stage 2** results (see Fig. 10 on the next page) show a significant difference in behaviour between the unloaded and loaded cases. Whereas in the unloaded case larger MPE caches brought significant improvements, this is not the case for loaded systems, the performance bottleneck at the memory subsystem being dominant.

- **Stage 3** results (combined with those of **Stage 4** in Fig. 11 on the facing page) indicate that in most cases the performance penalty induced by the MPE is higher with 128B CLs.

- **Stage 4** results (see Fig. 11 on the next page) show that in the loaded case asynchronous verification brings a significant speedup. This is due to the fact that memory access latencies increase significantly as the memory bus approaches saturation. Hence, decoupling decryption and MAC verification logics allows a noticeably better use of the MPE resources.

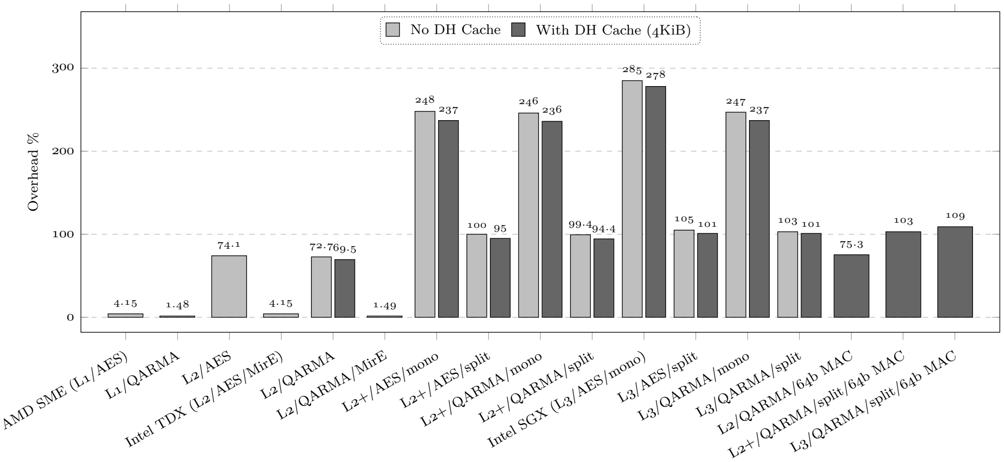

Figure 9: Stage 1/Loaded system: Comparison of base levels and state of the art; MACs are 32 bits long except for TDX (28 bits), SGX (56 bits) and 64 bits where indicated; The CG$ is 64KiB as in SGX

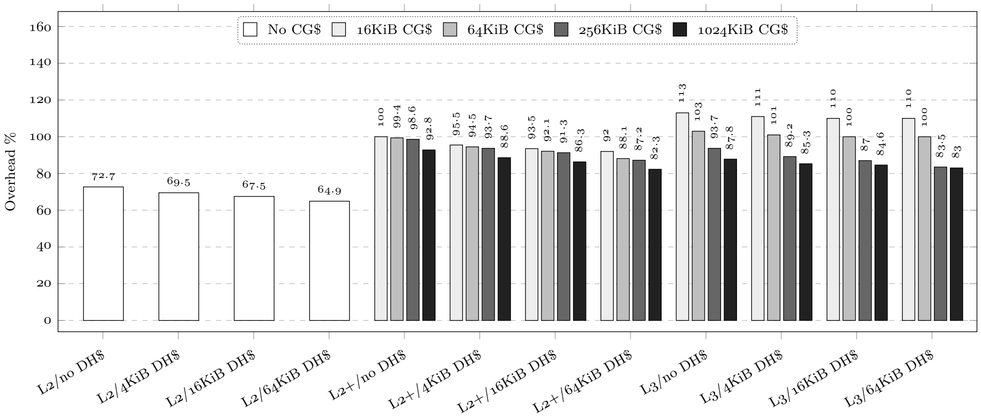

Figure 10: Stage 2/Loaded system: Impact of MPE cache sizes; ME cipher is QARMA-128; CL are 64B

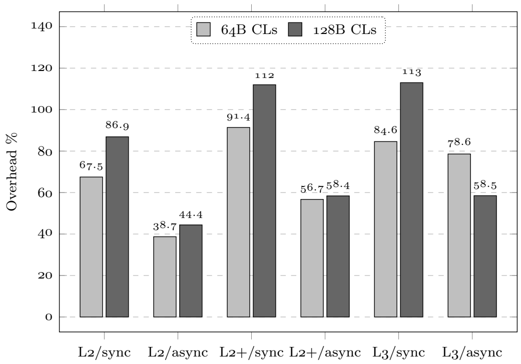

Figure 11: Stages 3 and 4/Loaded system: Impact of System CL size and asynchronous MAC verification; ME cipher is QARMA-128

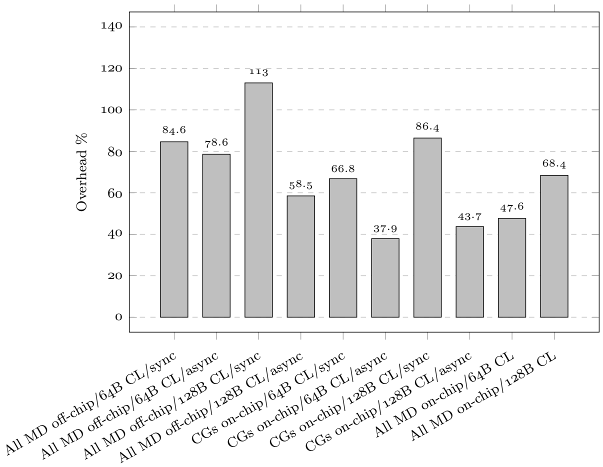

Figure 12: Stage 5/Loaded system: L3; Impact of storing MD on-chip; AES and QARMA results are identical

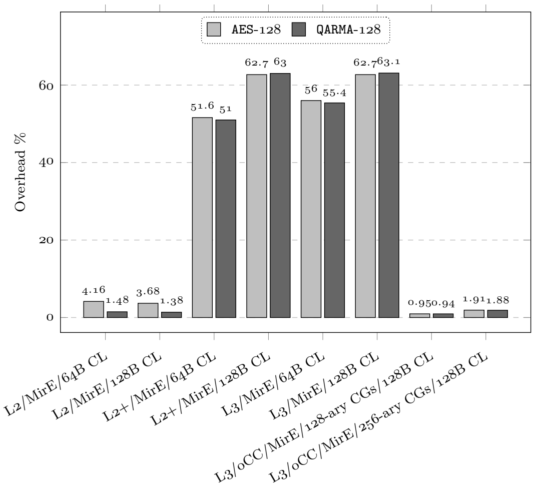

Figure 13: Stage 6/Loaded system: Impact of repurposing ECC bits for MACs

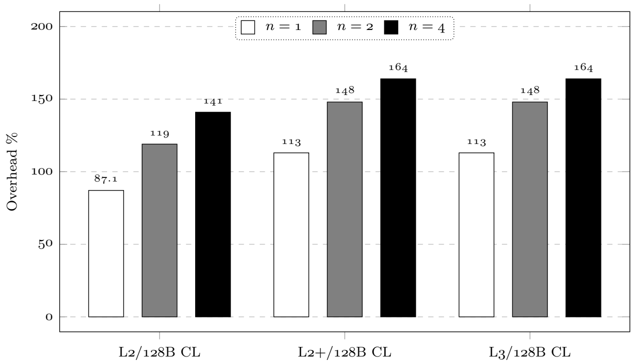

Figure 14: Stage 7/Loaded system: Impact of using multiple-CL MACs (128B CL)

- **Stage 5** results (see Fig. 12) show that the latency difference between ciphers becomes insignificant on a loaded system.

- **Stage 6** results (see Fig. 13) finally bring us that the only way to achieve nearly negligible performance penalties on a loaded system is to use on-chip memory for the CGs and repurpose ECC bits for MAC storage. In fact, by doing this even better performance than with L1 direct encryption can be achieved.

- The results of **Stage 7** (see Fig. 14) show that using multiple-CL MACs offers increasingly worse performance, as expected, also in the loaded case.

- **Stage 8** (see Figs. 23 to 30 on pages 40–42): The results are similar to the unloaded case, however the penalties are larges and in some cases dramatically so, because of the significant amount of usually non-linear traffic expansion. It is only with oCC and MirE that the performance becomes reasonable, with penalties larger than 5 % being the exception – whereas with an Intel SGX-like approach one can nearly reach a 10-fold slowdown. Indeed, our best architecture even outperforms a simple L1/AES, as the effect of the latency of a direct encryption method is amplified when the memory subsystem is saturated.

- **Stage 9** results are displayed in Figs. 28 to 30 on pages 41–42. Note that Figs. 29 and 30 on page 42 have only three results per benchmark because skipping RMWs with 128-ary and 256-ary CGs yields almost perfectly identical results. 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 relatively minor; the reduced performance of the 256-ary trees with respect to the 128-ary is due to the larger amount of RMWs, being the minor counters only 3b as opposed to 7b; and, if counters are stored on-chip and the MACs in the ECC bits, even the worst penalties with 256-ary trees are bounded, and while they are obviously larger than using 128-ary trees (which uses twice as much on-chip memory), they are still lower than the L1/AES/64B CLs/MirE scheme used in TDX.

6 Conclusions

In this work we have performed one of the most thorough and extensive evaluations of techniques for the cryptographic protection of in-use memory contents in the literature. We have considered the state of the art as well as some new technologies, and hitherto not considered combinations thereof.

We have also unified the evaluation of different protection levels, which are to be selected according to a given adversarial model. 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 performance, memory usage, and area in a single rating, makes it challenging to provide recommendations that may be optimal for many use cases.

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 ourselves to confidentiality and full integrity protection, i.e. L3.

Let us start with the use case of cloud computing. The 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 believe there is no reason for not using freshness-based OTP encryption based on AES or QARMA, with on-chip memory for split counters in the amount of 1/128 or 1/256 of the off-chip memory, and repurposing the ECC bits to store MACs for both integrity and bit flip detection and correction. The additional cost for implementing these technologies would be relatively minor, and likely still 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.

For lower lever devices the situation is more nuanced. Whereas L3/oCC/MirE is definitely applicable, many non-server 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. 29 on page 42 indicate, performance can quickly deteriorate if extra accesses are needed for MACs. So the use of ECC-capable memory, or memories with, say, only 32 or 40 extra tag bits for each 128B cache line should be seriously considered (the case of 40 extra bits would include 8 parity bits).

The main takeaway from our study is that nearly-transparent strong memory protection can be realised with current technologies and manufacturing processes, 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 would bypass 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 lower than 2 %.

References

[AB12] Jean-Philippe Aumasson and Daniel J. Bernstein. SipHash: A Fast Short-Input PRF. In Steven D. Galbraith and Mridul Nandi, editors, *Progress in Cryptology - INDOCRYPT 2012, 13th International Conference on Cryptology in India, Kolkata, India, December 9-12, 2012. Proceedings*, volume 7668 of *Lecture Notes in Computer Science*, pages 489–508. Springer, 2012. doi:10.1007/978-3-642-34931-7_28. Cited on pages 10 and 45.

[ABB+19] Roberto Avanzi, Subhadeep Banik, Andrey Bogdanov, Orr Dunkelman, Senyang Huang, and Francesco Regazzoni. Qameleon v.1.0 - A Submission to the NIST Lightweight Cryptography Standardization Process, 2019. Available from: https://csrc.nist.gov/CSRC/media/Projects/Lightweight-Cryptography/documents/round-1/submissions/qameleon.zip. Cited on page 16.

[AKT14] Can Acar, Arvind Krishnaswamy, and Robert Turner. Code pointer authentication for hardware flow control, October 2014. United States Patent US9514305 B2. Assignee: QUALCOMM Incorporated. Available from: http://pdfpiw.uspto.gov/.piw?Docid=09514305. Cited on page 44.

[AMD20] AMD. Secure Encrypted Virtualization API Version 0.24, April 2020. Available from: https://www.amd.com/system/files/TechDocs/55766_SEV-KM_API_Specification.pdf. Cited on page 43.

[AN17] Shaizeen Aga and Satish Narayanasamy. InvisiMem: Smart Memory Defenses for Memory Bus Side Channel. In *Proceedings of the 44th Annual International Symposium on Computer Architecture, ISCA 2017, Toronto, ON, Canada, June 24-28, 2017*, pages 94–106. ACM, 2017. Available from: http://doi.acm.org/10.1145/3079856, doi:10.1145/3079856.3080232. Cited on page 8.

[ARM16] ARM Connected blog. ARMv8-A architecture – 2016 additions. https://www.community.arm.com/processors/b/blog/posts/armv8-a-architecture-2016-additions, October 2016. Cited on page 44.

[ASC+19] Roberto Avanzi, Andreas Sandberg, Michael Andrew Campbell, Matthias Boettcher, and Prakash Ramrakhyani. Cached Incremental Hashing for Reducing Memory Integrity Overhead, 2019. To appear. Cited on page 19.

[Ava17] Roberto 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*, 2017(1):4–44, 2017. Available from: http://ojs.ub.rub.de/index.php/ToSC/article/view/583, doi:10.13154/tosc.v2017.i1.4-44. Cited on pages 4, 10, 16, and 44.

[Bac14] Matt Bach. ECC and REG ECC Memory Performance, May 2014. Available from: https://www.pugetsystems.com/labs/articles/ECC-and-REG-ECC-Memory-Performance-560/. Cited on page 19.

[BBB+11] Nathan L. Binkert, Bradford M. Beckmann, Gabriel Black, Steven K. Reinhardt, Ali G. Saidi, Arkaprava Basu, Joel Hestness, Derek Hower, Tushar Krishna, Somayeh Sardashti, Rathijit Sen, Korey Sewell, Muhammad Shoaib Bin Altaf, Nilay Vaish, Mark D. Hill, and David A. Wood. The gem5 simulator. *SIGARCH Comput. Archit. News*, 39(2):1–7, 2011. doi:10.1145/2024716.2024718. Cited on page 16.

[BBKN12] Alessandro Barenghi, Luca Breveglieri, Israel Koren, and David Naccache. Fault Injection Attacks on Cryptographic Devices: Theory, Practice, and Countermeasures. *Proc. IEEE*, 100(11):3056–3076, 2012. doi:10.1109/JPROC.2012.2188769. Cited on page 6.

[BCG+12] Julia Borghoff, Anne Canteaut, Tim Güneysu, Elif Bilge Kavun, Miroslav Knezevic, Lars R. Knudsen, Gregor Leander, Ventzislav Nikov, Christof Paar, Christian Rechberger, Peter Rombouts, Søren S. Thomsen, and Tolga Yalçin. PRINCE - A Low-Latency Block Cipher for Pervasive Computing Applications - Extended Abstract. In Xiaoyun Wang and Kazue Sako, editors, *ASIACRYPT 2012*, volume 7658 of *Lecture Notes in Computer Science*, pages 208–225. Springer, 2012. doi:10.1007/978-3-642-34961-4_14. Cited on pages 10 and 43.

[BCK96] Mihir Bellare, Ran Canetti, and Hugo Krawczyk. Keying Hash Functions for Message Authentication. In Neal Koblitz, editor, *Advances in Cryptology - CRYPTO '96, 16th Annual International Cryptology Conference, Santa Barbara, California, USA, August 18-22, 1996, Proceedings*, volume 1109 of *Lecture Notes in Computer Science*, pages 1–15. Springer, 1996. doi:10.1007/3-540-68697-5_1. Cited on pages 10 and 45.

[BEK+20] Dusan Bozilov, Maria Eichlseder, Miroslav Knezevic, Baptiste Lambin, Gregor Leander, Thorben Moos, Ventzislav Nikov, Shahram Rasoolzadeh, Yosuke Todo, and Friedrich Wiemer. PRINCEv2 - More Security for (Almost) No Overhead. In Orr Dunkelman, Michael J. Jacobson Jr., and Colin O'Flynn, editors, *Selected Areas in Cryptography - SAC 2020 - 27th International Conference, Halifax, NS, Canada (Virtual Event), October 21-23, 2020, Revised Selected Papers*, volume 12804 of *Lecture Notes in Computer Science*, pages 483–511. Springer, 2020. doi:10.1007/978-3-030-81652-0_19. Cited on page 43.

[Ber05a] Dan Bernstein. Cache-timing attacks on AES, 2005. Available from: http://cr.yp.to/papers.html#cachetiming. Cited on page 3.

[Ber05b] Daniel J. Bernstein. The Poly1305-AES Message-Authentication Code. In Henri Gilbert and Helena Handschuh, editors, *Fast Software Encryption: 12th International Workshop, FSE 2005, Paris, France, February 21-23, 2005, Revised Selected Papers*, volume 3557 of *Lecture Notes in Computer Science*, pages 32–49. Springer, 2005. doi:10.1007/11502760_3. Cited on page 46.

[Ber05c] Daniel J. Bernstein. Stronger Security Bounds for Wegman-Carter-Shoup Authenticators. In Ronald Cramer, editor, *Advances in Cryptology - EUROCRYPT 2005, 24th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Aarhus, Denmark, May 22-26, 2005, Proceedings*, volume 3494 of *Lecture Notes in Computer Science*, pages 164–180. Springer, 2005. doi:10.1007/11426639_10. Cited on page 46.

[Bes80] Robert Best. Preventing software piracy with crypto-microprocessors. In *Proceedings of the IEEE Spring Compcon*, pages 466–469, 1980. Cited on page 3.

[BKL+07] Andrey Bogdanov, Lars R. Knudsen, Gregor Leander, Christof Paar, Axel Poschmann, Matthew J. B. Robshaw, Yannick Seurin, and C. Vikkelsoe. PRESENT: An Ultra-Lightweight Block Cipher. In Paillier and Verbauwhede [PV07], pages 450–466. Cited on page 43.

[BNS19] Xavier Bonnetain, María Naya-Plasencia, and André Schrottenloher. Quantum Security Analysis of AES. *IACR Trans. Symmetric Cryptol.*, 2019(2):55–93, 2019. doi:10.13154/tosc.v2019.i2.55-93. Cited on page 15.

[BR12] Erik-Oliver Blass and William Robertson. TRESOR-HUNT: attacking CPU-bound encryption. In Robert H'obbes' Zakon, editor, *28th Annual Computer Security Applications Conference, ACSAC 2012, Orlando, FL, USA, 3-7 December 2012*, pages 71–78. ACM, 2012. doi:10.1145/2420950.2420961. Cited on page 6.

[CA16] Brent Carrara and Carlisle Adams. A Survey and Taxonomy Aimed at the Detection and Measurement of Covert Channels. In Fernando Pérez-González, Patrick Bas, Tanya Ignatenko, and François Cayre, editors, *Proceedings of the 4th ACM Workshop on Information Hiding and Multimedia Security, IH&MMSec 2016, Vigo, Galicia, Spain, June 20-22, 2016*, pages 115–126. ACM, 2016. doi:10.1145/2909827.2930800. Cited on page 3.

[CBS⁺19] Claudio Canella, Jo Van Bulck, Michael Schwarz, Moritz Lipp, Benjamin von Berg, Philipp Ortner, Frank Piessens, Dmitry Evtyushkin, and Daniel Gruss. A Systematic Evaluation of Transient Execution Attacks and Defenses. In Nadia Heninger and Patrick Traynor, editors, *28th USENIX Security Symposium, USENIX Security 2019, Santa Clara, CA, USA, August 14-16, 2019*, pages 249–266. USENIX Association, 2019. Available from: https://www.usenix.org/conference/usenixsecurity19/presentation/canella. Cited on page 3.

[CL10] David Champagne and Ruby B. Lee. Scalable architectural support for trusted software. In Matthew T. Jacob, Chita R. Das, and Pradip Bose, editors, *16th International Conference on High-Performance Computer Architecture (HPCA-16 2010), 9-14 January 2010, Bangalore, India*, pages 1–12. IEEE Computer Society, 2010. Available from: http://ieeexplore.ieee.org/xpl/mostRecentIssue.jsp?punumber=5410726, doi:10.1109/HPCA.2010.5416657. Cited on page 3.

[CRSP11] Siddhartha Chhabra, Brian Rogers, Yan Solihin, and Milos Prvulovic. SecureME: a hardware-software approach to full system security. In David K. Lowenthal, Bronis R. de Supinski, and Sally A. McKee, editors, *Proceedings of the 25th International Conference on Supercomputing, 2011, Tucson, AZ, USA, May 31 - June 04, 2011*, pages 108–119. ACM, 2011. Available from: http://doi.acm.org/10.1145/1995896.1995914, doi:10.1145/1995896.1995914. Cited on page 3.

[CVM⁺21] Zitai Chen, Georgios Vasilakis, Kit Murdock, Edward Dean, David F. Oswald, and Flavio D. Garcia. VoltPillager: Hardware-based fault injection attacks against Intel SGX Enclaves using the SVID voltage scaling interface. In Michael Bailey and Rachel Greenstadt, editors, *30th USENIX Security Symposium, USENIX Security 2021, August 11-13, 2021*, pages 699–716. USENIX Association, 2021. Available from: https://www.usenix.org/conference/usenixsecurity21/presentation/chen-zitai. Cited on page 6.

[CW77] Larry Carter and Mark N. Wegman. Universal Classes of Hash Functions (Extended Abstract). In John E. Hopcroft, Emily P. Friedman, and Michael A. Harrison, editors, *Proceedings of the 9th Annual ACM Symposium on Theory of Computing, May 4-6, 1977, Boulder, Colorado, USA*, pages 106–112. ACM, 1977. doi:10.1145/800105.803400. Cited on pages 10 and 45.

[CW79] Larry Carter and Mark N. Wegman. Universal Classes of Hash Functions. *J. Comput. Syst. Sci.*, 18(2):143–154, 1979. doi:10.1016/0022-0000(79)90044-8. Cited on pages 10 and 45.

[CXL19] CXL Consortium. Compute Express Link™ Resource Library, 2019. Available from: https://www.computeexpresslink.org/resource-library. Cited on page 8.

[DR02a] Joan Daemen and Vincent Rijmen. AES and the Wide Trail Design Strategy. In Lars R. Knudsen, editor, *EUROCRYPT 2002*, volume 2332 of *Lecture Notes in Computer Science*, pages 108–109. Springer, 2002. doi:10.1007/3-540-46035-7_7. Cited on pages 4, 10, and 43.

[DR02b] Joan Daemen and Vincent Rijmen. *The Design of Rijndael: AES - The Advanced Encryption Standard*. Information Security and Cryptography. Springer, 2002. doi:10.1007/978-3-662-04722-4. Cited on pages 10 and 43.

[ECL⁺07] Reouven Elbaz, David Champagne, Ruby B. Lee, Lionel Torres, Gilles Sassatelli, and Pierre Guillemin. TEC-Tree: A Low-Cost, Parallelizable Tree for Efficient Defense Against Memory Replay Attacks. In Paillier and Verbauwhede [PV07], pages 289–302. doi:10.1007/978-3-540-74735-2_20. Cited on pages 12 and 51.

[FGM⁺10] Junfeng Fan, Xu Guo, Elke De Mulder, Patrick Schaumont, Bart Preneel, and Ingrid Verbauwhede. State-of-the-art of Secure ECC Implementations: A Survey on Known Side-channel Attacks and Countermeasures. In Jim Plusquellic and Ken Mai, editors, *HOST 2010, Proceedings of the 2010 IEEE International Symposium on Hardware-Oriented Security and Trust (HOST), 13-14 June 2010, Anaheim Convention*

Center, California, USA, pages 76–87. IEEE Computer Society, 2010. doi:10.1109/HST.2010.5513110. Cited on page 3.

[Fri16] Ulf Frisk. macOS FileVault2 Password Retrieval, 12 2016. Available from: https://blog.frizk.net/2016/12/filevault-password-retrieval.html. Cited on page 6.

[GSC+03] Blaise Gassend, G. Edward Suh, Dwaine E. Clarke, Marten van Dijk, and Srinivas Devadas. Caches and Hash Trees for Efficient Memory Integrity Verification. In *Proceedings of the Ninth International Symposium on High-Performance Computer Architecture (HPCA'03), Anaheim, California, USA, February 8-12, 2003*, pages 295–306. IEEE Computer Society, 2003. Available from: http://ieeexplore.ieee.org/xpl/mostRecentIssue.jsp?punumber=8433, doi:10.1109/HPCA.2003.1183547. Cited on pages 3, 11, 48, and 53.

[Gue16a] Shay Gueron. A Memory Encryption Engine Suitable for General Purpose Processors. *IACR Cryptol. ePrint Arch.*, page 204, 2016. Available from: http://eprint.iacr.org/2016/204. Cited on pages 3, 4, 8, 12, 51, and 54.

[Gue16b] Shay Gueron. Memory Encryption for General-Purpose Processors. *IEEE Secur. Priv.*, 14(6):54–62, 2016. doi:10.1109/MSP.2016.124. Cited on pages 10, 43, and 46.

[Hen06] John L. Henning. SPEC CPU2006 benchmark descriptions. *SIGARCH Comput. Archit. News*, 34(4):1–17, 2006. doi:10.1145/1186736.1186737. Cited on page 16.

[HJ05] William Eric Hall and Charanjit S. Jutla. Parallelizable Authentication Trees. In Bart Preneel and Stafford E. Tavares, editors, *Selected Areas in Cryptography, 12th International Workshop, SAC 2005, Kingston, ON, Canada, August 11-12, 2005, Revised Selected Papers*, volume 3897 of *Lecture Notes in Computer Science*, pages 95–109. Springer, 2005. doi:10.1007/11693383_7. Cited on pages 11 and 50.

[HK19] Youngkwang Han and John Kim. A Novel Covert Channel Attack Using Memory Encryption Engine Cache. In *Proceedings of the 56th Annual Design Automation Conference 2019, DAC 2019, Las Vegas, NV, USA, June 02-06, 2019*, page 58. ACM, 2019. doi:10.1145/3316781.3317750. Cited on page 54.

[HS10] Ruirui C. Huang and G. Edward Suh. IVEC: off-chip memory integrity protection for both security and reliability. In André Seznec, Uri C. Weiser, and Ronny Ronen, editors, *37th International Symposium on Computer Architecture (ISCA 2010), June 19-23, 2010, Saint-Malo, France*, pages 395–406. ACM, 2010. doi:10.1145/1815961.1816015. Cited on pages 3, 11, and 53.

[HSH+09] J. Alex Halderman, Seth D. Schoen, Nadia Heninger, William Clarkson, William Paul, Joseph A. Calandrino, Ariel J. Feldman, Jacob Appelbaum, and Edward W. Felten. Lest we remember: cold-boot attacks on encryption keys. *Commun. ACM*, 52(5):91–98, 2009. doi:10.1145/1506409.1506429. Cited on pages 3 and 6.

[HTLW21] Steven Herschbein, Shida Tan, Richard Livengood, and Michael 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*, International Symposium for Testing and Failure Analysis, pages h1–h113, 10 2021. doi:10.31399/asm.cp.istfa2021tph1. Cited on page 7.

[Hu92] Wei-Ming Hu. Lattice scheduling and covert channels. In *1992 IEEE Computer Society Symposium on Research in Security and Privacy, Oakland, CA, USA, May 4-6, 1992*, pages 52–61. IEEE Computer Society, 1992. doi:10.1109/RISP.1992.213271. Cited on page 3.

[Int15] Intel. Intel® Software Guard Extensions, 2015. Available from: https://software.intel.com/en-us/isa-extensions/intel-sgx. Cited on page 43.

[Int21a] Intel. Intel® TDX Module 1.5 Base Architecture Specification, September 2021. Available from: https://www.intel.com/content/www/us/en/developer/articles/technical/intel-trust-domain-extensions.html. Cited on pages 8 and 54.

[Int21b] Intel. Intel® Trust Domain Extensions (Intel® TDX), August 2021. Available from: https://www.intel.com/content/www/us/en/developer/articles/technical/intel-trust-domain-extensions.html. Cited on page 43.

[Int21c] Intel. Intel® Trust Domain Extensions White Paper, August 2021. Available from: https://www.intel.com/content/www/us/en/developer/articles/technical/intel-trust-domain-extensions.html. Cited on pages 8 and 54.

[JAS+15] Seongwook Jin, Jeongseob Ahn, Jinho Seol, Sanghoon Cha, Jaehyuk Huh, and Seungryoul Maeng. H-SVM: Hardware-Assisted Secure Virtual Machines under a Vulnerable Hypervisor. *IEEE Trans. Computers*, 64(10):2833–2846, 2015. doi:10.1109/TC.2015.2389792. Cited on page 3.

[JBS+22] Kyungbae Jang, Anubhab Baksi, Gyeongju Song, Hyunji Kim, Hwajeong Seo, and Anupam Chattopadhyay. Quantum Analysis of AES. *IACR Cryptol. ePrint Arch.*, page 683, 2022. Available from: https://eprint.iacr.org/2022/683. Cited on page 15.

[JLK+23] Jonas Juffinger, Lukas Lamster, Andreas Kogler, Moritz Lipp, Maria Eichlseder, and Daniel Gruss. CSI:Rowhammer – Cryptographic Security and Integrity against Rowhammer. In *Proceedings of the 44th IEEE Symposium on Security and Privacy, S&P'23, San Francisco, California, USA, May 22–26, 2023*, 2023. Cited on pages 3, 8, 10, 11, 15, 16, 44, 46, 48, and 55.

[KDK+14] Yoongu Kim, Ross Daly, Jeremie Kim, Chris Fallin, Ji-Hye Lee, Donghyuk Lee, Chris Wilkerson, Konrad Lai, and Onur Mutlu. Flipping bits in memory without accessing them: An experimental study of DRAM disturbance errors. In *ACM/IEEE 41st International Symposium on Computer Architecture, ISCA 2014, Minneapolis, MN, USA, June 14-18, 2014*, pages 361–372. IEEE Computer Society, 2014. doi:10.1109/ISCA.2014.6853210. Cited on page 6.

[KFM05] Taeho Kgil, Laura Falk, and Trevor N. Mudge. ChipLock: support for secure microarchitectures. *SIGARCH Comput. Archit. News*, 33(1):134–143, 2005. doi:10.1145/1055626.1055644. Cited on page 3.

[KHF+19] Paul Kocher, Jann Horn, Anders Fogh, Daniel Genkin, Daniel Gruss, Werner Haas, Mike Hamburg, Moritz Lipp, Stefan Mangard, Thomas Prescher, Michael Schwarz, and Yuval Yarom. Spectre Attacks: Exploiting Speculative Execution. In *2019 IEEE Symposium on Security and Privacy, SP 2019, San Francisco, CA, USA, May 19-23, 2019*, pages 1–19. IEEE, 2019. doi:10.1109/SP.2019.00002. Cited on page 3.

[KLR+20] Mohamed Amine Khelif, Jordane Lorandel, Olivier Romain, Matthieu Regnery, Denis Baheux, and Guillaume Barbu. Toward a hardware man-in-the-middle attack on PCIe bus. *Microprocess. Microsystems*, 77:103198, 2020. doi:10.1016/j.micpro.2020.103198. Cited on page 6.

[KNR12] Miroslav Knezevic, Ventzislav Nikov, and Peter Rombouts. Low-Latency Encryption - Is "Lightweight = Light + Wait"? In Emmanuel Prouff and Patrick Schaumont, editors, *Cryptographic Hardware and Embedded Systems - CHES 2012 - 14th International Workshop, Leuven, Belgium, September 9-12, 2012. Proceedings*, volume 7428 of *Lecture Notes in Computer Science*, pages 426–446. Springer, 2012. doi:10.1007/978-3-642-33027-8_25. Cited on page 43.

[Koc96] Paul C. Kocher. Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems. In Neal Koblitz, editor, *Advances in Cryptology - CRYPTO '96, 16th Annual International Cryptology Conference, Santa Barbara, California, USA, August 18-22, 1996, Proceedings*, volume 1109 of *Lecture Notes in Computer Science*, pages 104–113. Springer, 1996. doi:10.1007/3-540-68697-5_9. Cited on page 3.

[KPW16] David Kaplan, Jeremy Powell, and Tom Woller. AMD Memory Encryption White Paper, April 2016. Available from: http://amd-dev.wpengine.netdna-cdn.com/wordpress/media/2013/12/AMD_Memory_Encryption_Whitepaper_v7-Public.pdf. Cited on pages 8, 43, and 54.

[KR11] Ted Krovetz and Phillip Rogaway. The Software Performance of Authenticated-Encryption Modes. In Antoine Joux, editor, *Fast Software Encryption - 18th International Workshop, FSE 2011, Lyngby, Denmark, February 13-16, 2011, Revised Selected Papers*, volume 6733 of *Lecture Notes in Computer Science*, pages 306–327. Springer, 2011. doi:10.1007/978-3-642-21702-9_18. Cited on page 48.

[Kuh98] Markus G. Kuhn. Cipher Instruction Search Attack on the Bus-Encryption Security Microcontroller DS5002FP. *IEEE Trans. Computers*, 47(10):1153–1157, 1998. doi:10.1109/12.729797. Cited on pages 3 and 6.

[LAA^{+}20] Jason Lowe-Power, Abdul Mutaal Ahmad, Ayaz Akram, Mohammad Alian, Rico Amslinger, Matteo Andreozzi, Adrià Armejach, Nils Asmussen, Srikant Bharadwaj, Gabe Black, Gedare Bloom, Bobby R. Bruce, Daniel Rodrigues Carvalho, Jerónimo Castrillón, Lizhong Chen, Nicolas Derumigny, Stephan Diestelhorst, Wendy Elsasser, Marjan Fariborz, Amin Farmahini Farahani, Pouya Fotouhi, Ryan Gambord, Jayneel Gandhi, Dibakar Gope, Thomas Grass, Bagus Hanindhito, Andreas Hansson, Swapnil Haria, Austin Harris, Timothy Hayes, Adrian Herrera, Matthew Horsnell, Syed Ali Raza Jafri, Radhika Jagtap, Hanhwi Jang, Reiley Jeyapaul, Timothy M. Jones, Matthias Jung, Subash Kannoth, Hamidreza Khaleghzadeh, Yuetsu Kodama, Tushar Krishna, Tommaso Marinelli, Christian Menard, Andrea Mondelli, Tiago Mück, Omar Naji, Krishnendra Nathella, Hoa Nguyen, Nikos Nikoleris, Lena E. Olson, Marc S. Orr, Binh Pham, Pablo Prieto, Trivikram Reddy, Alec Roelke, Mahyar Samani, Andreas Sandberg, Javier Setoain, Boris Shingarov, Matthew D. Sinclair, Tuan Ta, Rahul Thakur, Giacomo Travaglini, Michael Upton, Nilay Vaish, Ilias Vougioukas, Zhengrong Wang, Norbert Wehn, Christian Weis, David A. Wood, Hongil Yoon, and Éder F. Zulian. The gem5 Simulator: Version 20.0+. *CoRR*, abs/2007.03152, 2020. Available from: `https://arxiv.org/abs/2007.03152`, `arXiv:2007.03152`. Cited on page 16.

[LGG^{+}21] Corentin Lavaud, Robin Gerzaguet, Matthieu Gautier, Olivier Berder, Erwan Nogues, and Stéphane Molton. Whispering Devices: A Survey on How Side-channels Lead to Compromised Information. *J. Hardw. Syst. Secur.*, 5(2):143–168, 2021. `doi:10.1007/s41635-021-00112-6`. Cited on page 3.

[LJF^{+}20] Dayeol Lee, Dongha Jung, Ian T. Fang, Chia-che Tsai, and Raluca Ada Popa. An Off-Chip Attack on Hardware Enclaves via the Memory Bus. In Srdjan Capkun and Franziska Roesner, editors, *29th USENIX Security Symposium, USENIX Security 2020, August 12-14, 2020*, pages 487–504. USENIX Association, 2020. Available from: `https://www.usenix.org/conference/usenixsecurity20/presentation/lee-dayeol`. Cited on pages 3 and 6.

[LMMR21] Gregor Leander, Thorben Moos, Amir Moradi, and Shahram Rasoolzadeh. The SPEEDY Family of Block Ciphers Engineering an Ultra Low-Latency Cipher from Gate Level for Secure Processor Architectures. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2021(4):510–545, 2021. `doi:10.46586/tches.v2021.i4.510-545`. Cited on pages 10 and 44.

[LRW02] Moses Liskov, Ronald L. Rivest, and David Wagner. Tweakable Block Ciphers. In Moti Yung, editor, *CRYPTO 2002*, volume 2442 of *Lecture Notes in Computer Science*, pages 31–46. Springer, 2002. `doi:10.1007/3-540-45708-9`. Cited on pages 43 and 48.

[LRW11] Moses Liskov, Ronald L. Rivest, and David Wagner. Tweakable Block Ciphers. *Journal of Cryptology*, 24(3):588–613, 2011. `doi:10.1007/s00145-010-9073-y`. Cited on page 43.

[LSG^{+}18] Moritz Lipp, Michael Schwarz, Daniel Gruss, Thomas Prescher, Werner Haas, Anders Fogh, Jann Horn, Stefan Mangard, Paul Kocher, Daniel Genkin, Yuval Yarom, and Mike Hamburg. Meltdown: Reading Kernel Memory from User Space. In William Enck and Adrienne Porter Felt, editors, *27th USENIX Security Symposium, USENIX Security 2018, Baltimore, MD, USA, August 15-17, 2018*, pages 973–990. USENIX Association, 2018. Available from: `https://www.usenix.org/conference/usenixsecurity18/presentation/lipp`. Cited on page 3.

[LSR^{+}20] Moritz Lipp, Michael Schwarz, Lukas Raab, Lukas Lamster, Misiker Tadesse Aga, Clémentine Maurice, and Daniel Gruss. Nethammer: Inducing Rowhammer Faults through Network Requests. In *IEEE European Symposium on Security and Privacy Workshops, EuroS&P Workshops 2020, Genoa, Italy, September 7-11, 2020*, pages 710–719. IEEE, 2020. `doi:10.1109/EuroSPW51379.2020.00102`. Cited on page 7.

[LTM^{+}00] David Lie, Chandramohan A. Thekkath, Mark Mitchell, Patrick Lincoln, Dan Boneh, John C. Mitchell, and Mark Horowitz. Architectural Support for Copy and Tamper Resistant Software. In Larry Rudolph and Anoop Gupta, editors, *ASPLOS-IX Proceedings of the 9th International Conference on Architectural Support for Programming Languages and Operating Systems, Cambridge, MA, USA, November 12-15, 2000.*, pages 168–177. ACM Press, 2000. Available from: `http://doi.acm.org/10.1145/356989.357005`, `doi:10.1145/356989.357005`. Cited on page 3.

[LZW^{+}21] Mengyuan Li, Yinqian Zhang, Huibo Wang, Kang Li, and Yueqiang Cheng. CIPHERLEAKS: Breaking Constant-time Cryptography on AMD SEV via the Ciphertext Side Channel. In Michael Bailey and

Rachel Greenstadt, editors, *30th USENIX Security Symposium, USENIX Security 2021, August 11-13, 2021*, pages 717–732. USENIX Association, 2021. Cited on page 54.

[MAB$^+$13] Frank McKeen, Ilya Alexandrovich, Alex Berenzon, Carlos V. Rozas, Hisham Shafi, Vedvyas Shanbhogue, and Uday R. Savagaonkar. Innovative instructions and software model for isolated execution. In Ruby B. Lee and Weidong Shi, editors, *HASP 2013, The Second Workshop on Hardware and Architectural Support for Security and Privacy, Tel-Aviv, Israel, June 23-24, 2013*, page 10. ACM, 2013. Available from: `http://dl.acm.org/citation.cfm?id=2487726`, `doi:10.1145/2487726.2488368`. Cited on page 3.

[MDH$^+$13] Nicolas Moro, Amine Dehbaoui, Karine Heydemann, Bruno Robisson, and Emmanuelle Encrenaz. Electromagnetic Fault Injection: Towards a Fault Model on a 32-bit Microcontroller. In Wieland Fischer and Jörn-Marc Schmidt, editors, *2013 Workshop on Fault Diagnosis and Tolerance in Cryptography, Los Alamitos, CA, USA, August 20, 2013*, pages 77–88. IEEE Computer Society, 2013. `doi:10.1109/FDTC.2013.9`. Cited on page 6.

[Mer80] Ralph C. Merkle. Protocols for Public Key Cryptosystems. In *Proceedings of the 1980 IEEE Symposium on Security and Privacy, Oakland, California, USA, April 14-16, 1980*, pages 122–134. IEEE Computer Society, 1980. Available from: `http://ieeexplore.ieee.org/xpl/mostRecentIssue.jsp?punumber=6233684`, `doi:10.1109/SP.1980.10006`. Cited on pages 11 and 48.

[Mer87] Ralph C. Merkle. A Digital Signature Based on a Conventional Encryption Function. In Carl Pomerance, editor, *Advances in Cryptology - CRYPTO '87, A Conference on the Theory and Applications of Cryptographic Techniques, Santa Barbara, California, USA, August 16-20, 1987, Proceedings*, volume 293 of *Lecture Notes in Computer Science*, pages 369–378. Springer, 1987. `doi:10.1007/3-540-48184-2\_32`. Cited on page 3.

[Mih22] Ionuț Mihalcea. Prototyping Memory Integrity Tree Algorithms for Internet of Things Devices. Master's thesis, Information Security Group, Royal Holloway University of London, Egham, Surrey, UK, 2022. Cited on page 5.

[MK20] Onur Mutlu and Jeremie S. Kim. RowHammer: A Retrospective. *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.*, 39(8):1555–1571, 2020. `doi:10.1109/TCAD.2019.2915318`. Cited on page 6.

[Mut19] Onur Mutlu. RowHammer and Beyond. In Ilia Polian and Marc Stöttinger, editors, *Constructive Side-Channel Analysis and Secure Design - 10th International Workshop, COSADE 2019, Darmstadt, Germany, April 3-5, 2019, Proceedings*, volume 11421 of *Lecture Notes in Computer Science*, pages 3–12. Springer, 2019. `doi:10.1007/978-3-030-16350-1\_1`. Cited on page 6.

[MV04] David A. McGrew and John Viega. The Security and Performance of the Galois/Counter Mode (GCM) of Operation. In Anne Canteaut and Kapalee Viswanathan, editors, *Progress in Cryptology - INDOCRYPT 2004, 5th International Conference on Cryptology in India, Chennai, India, December 20-22, 2004, Proceedings*, volume 3348 of *Lecture Notes in Computer Science*, pages 343–355. Springer, 2004. `doi:10.1007/978-3-540-30556-9\_27`. Cited on page 46.

[MVS00] Umesh Maheshwari, Radek Vingralek, and William Shapiro. How to Build a Trusted Database System on Untrusted Storage. In Michael B. Jones and M. Frans Kaashoek, editors, *4th Symposium on Operating System Design and Implementation (OSDI 2000), San Diego, California, USA, October 23-25, 2000*, pages 135–150. USENIX Association, 2000. Available from: `http://dl.acm.org/citation.cfm?id=1251239`. Cited on pages 3, 11, and 48.

[NIS12] NIST. FIPS PUB 180-4 – Secure Hash Standard. Technical report, National Institute of Standards and Technology, Gaithersburg, MD, United States, March 2012. Cited on pages 10 and 45.

[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, August 2015. Available from: `https://csrc.nist.gov/publications/detail/fips/202/final`. Cited on pages 10, 45, and 54.

[OST06] Dag Arne Osvik, Adi Shamir, and Eran Tromer. Cache Attacks and Countermeasures: The Case of AES. In David Pointcheval, editor, *Topics in Cryptology - CT-RSA 2006, The Cryptographers' Track at the RSA Conference 2006, San Jose, CA, USA, February 13-17, 2006, Proceedings*, volume 3860 of *Lecture Notes in Computer Science*, pages 1–20. Springer, 2006. `doi:10.1007/11605805\_1`. Cited on page 3.

[PS16] Thomas Peyrin and Yannick Seurin. Counter-in-Tweak: Authenticated Encryption Modes for Tweakable Block Ciphers. In Matthew Robshaw and Jonathan Katz, editors, *Advances in Cryptology - CRYPTO 2016 - 36th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2016, Proceedings, Part I*, volume 9814 of *Lecture Notes in Computer Science*, pages 33–63. Springer, 2016. doi:10.1007/978-3-662-53018-4_2. Cited on pages 11 and 48.

[PV07] Pascal Paillier and Ingrid Verbauwhede, editors. *Cryptographic Hardware and Embedded Systems - CHES 2007, 9th International Workshop, Vienna, Austria, September 10-13, 2007, Proceedings*, volume 4727 of *Lecture Notes in Computer Science*. Springer, 2007. Cited on pages 29 and 30.

[QPS17] Qualcomm Product Security. Pointer Authentication on ARMv8.3 – Design and Analysis of the New Software Security Instructions. https://www.qualcomm.com/documents/whitepaper-pointer-authentication-armv83, January 2017. Cited on page 44.

[RBB03] Phillip Rogaway, Mihir Bellare, and John Black. OCB: A block-cipher mode of operation for efficient authenticated encryption. *ACM Trans. Inf. Syst. Secur.*, 6(3):365–403, 2003. Available from: http://doi.acm.org/10.1145/937527.937529, doi:10.1145/937527.937529. Cited on page 48.

[RCPS07] Brian Rogers, Siddhartha Chhabra, Milos Prvulovic, and Yan Solihin. Using Address Independent Seed Encryption and Bonsai Merkle Trees to Make Secure Processors OS- and Performance-Friendly. In *40th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO-40 2007), 1-5 December 2007, Chicago, Illinois, USA*, pages 183–196. IEEE Computer Society, 2007. Available from: http://ieeexplore.ieee.org/xpl/mostRecentIssue.jsp?punumber=4408231, doi:10.1109/MICRO.2007.44. Cited on pages 3, 11, and 50.

[Rog04] Phillip Rogaway. Efficient Instantiations of Tweakable Blockciphers and Refinements to Modes OCB and PMAC. In Pil Joong Lee, editor, *Advances in Cryptology - ASIACRYPT 2004, 10th International Conference on the Theory and Application of Cryptology and Information Security, Jeju Island, Korea, December 5-9, 2004, Proceedings*, volume 3329 of *Lecture Notes in Computer Science*, pages 16–31. Springer, 2004. doi:10.1007/978-3-540-30539-2_2. Cited on pages 11, 43, and 46.

[SAFT16] Bicky Shakya, Navid Asadizanjani, Domenic Forte, and Mark M. Tehranipoor. Chip editor: leveraging circuit edit for logic obfuscation and trusted fabrication. In Frank Liu, editor, *Proceedings of the 35th International Conference on Computer-Aided Design, ICCAD 2016, Austin, TX, USA, November 7-10, 2016*, page 30. ACM, 2016. doi:10.1145/2966986.2967014. Cited on page 7.

[San14] Andreas Sandberg. *Understanding Multicore Performance: Efficient Memory System Modeling andSimulation*. PhD thesis, Uppsala University, Disciplinary Domain of Science and Technology, Mathematics and Computer Science, Department of Information Technology, Division of Computer Systems, Uppsala, Sweden, 2014. Cited on page 17.

[SCG$^+$03] G. Edward Suh, Dwaine E. Clarke, Blaise Gassend, Marten van Dijk, and Srinivas Devadas. AEGIS: architecture for tamper-evident and tamper-resistant processing. In Utpal Banerjee, Kyle Gallivan, and Antonio González, editors, *Proceedings of the 17th Annual International Conference on Supercomputing, ICS 2003, San Francisco, CA, USA, June 23-26, 2003*, pages 160–171. ACM, 2003. doi:10.1145/782814.782838. Cited on pages 3, 43, and 53.

[Sch19] David H. Schall. Evaluation and Optimization of Memory Encryption and Integrity Protection. Master's thesis, University of Kaiserslautern, Department of Electrical Engineering and Information Technology, Microelectronic Systems Design Research Group, 2019. Cited on page 5.

[Sko17] Sergei Skorobogatov. How Microprobing Can Attack Encrypted Memory. In Hana Kubátová, Martin Novotný, and Amund Skavhaug, editors, *Euromicro Conference on Digital System Design, DSD 2017, Vienna, Austria, August 30 - Sept. 1, 2017*, pages 244–251. IEEE Computer Society, 2017. doi:10.1109/DSD.2017.69. Cited on page 7.

[SL12] Jakub Szefer and Ruby B. Lee. Architectural support for hypervisor-secure virtualization. In Tim Harris and Michael L. Scott, editors, *Proceedings of the 17th International Conference on Architectural Support for Programming Languages and Operating Systems, ASPLOS 2012, London, UK, March 3-7, 2012*, pages 437–450. ACM, 2012. doi:10.1145/2150976.2151022. Cited on page 4.

[SLGL04] Weidong Shi, Hsien-Hsin S. Lee, Mrinmoy Ghosh, and Chenghuai Lu. Architectural Support for High Speed Protection of Memory Integrity and Confidentiality in Multiprocessor Systems. In *13th International Conference on Parallel Architectures and Compilation Techniques (PACT 2004), 29 September - 3 October 2004, Antibes Juan-les-Pins, France*, pages 123–134. IEEE Computer Society, 2004. doi:10.1109/PACT.2004.10025. Cited on page 3.

[SNR+18] Gururaj Saileshwar, Prashant J. Nair, Prakash Ramrakhyani, Wendy Elsasser, and Moinuddin K. Qureshi. SYNERGY: Rethinking Secure-Memory Design for Error-Correcting Memories. In *IEEE International Symposium on High Performance Computer Architecture, HPCA 2018, Vienna, Austria, February 24-28, 2018*, pages 454–465. IEEE Computer Society, 2018. doi:10.1109/HPCA.2018.00046. Cited on pages 3, 8, 11, 15, 46, 48, and 54.

[SOD07] G. Edward Suh, Charles W. O'Donnell, and Srinivas Devadas. Aegis: A Single-Chip Secure Processor. *IEEE Des. Test Comput.*, 24(6):570–580, 2007. doi:10.1109/MDT.2007.179. Cited on page 3.

[SPHC02] Timothy Sherwood, Erez Perelman, Greg Hamerly, and Brad Calder. Automatically characterizing large scale program behavior. In Kourosh Gharachorloo and David A. Wood, editors, *ACM SIGPLAN Notices, vol. 37 (Proceedings of the 10th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS-X), San Jose, California, USA, October 5-9, 2002)*, pages 45–57. ACM Press, 2002. doi:10.1145/605397.605403. Cited on page 17.

[TJ09] Randy Torrance and Dick James. The State-of-the-Art in IC Reverse Engineering. In Christophe Clavier and Kris Gaj, editors, *Cryptographic Hardware and Embedded Systems - CHES 2009, 11th International Workshop, Lausanne, Switzerland, September 6-9, 2009, Proceedings*, volume 5747 of *Lecture Notes in Computer Science*, pages 363–381. Springer, 2009. doi:10.1007/978-3-642-04138-9_26. Cited on page 7.

[TSB18] Meysam Taassori, Ali Shafiee, and Rajeev Balasubramonian. VAULT: Reducing Paging Overheads in SGX with Efficient Integrity Verification Structures. In Xipeng Shen, James Tuck, Ricardo Bianchini, and Vivek Sarkar, editors, *Proceedings of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems, ASPLOS 2018, Williamsburg, VA, USA, March 24-28, 2018*, pages 665–678. ACM, 2018. doi:10.1145/3173162.3177155. Cited on page 23.

[UWM19] Thomas Unterluggauer, Mario Werner, and Stefan Mangard. MEAS: memory encryption and authentication secure against side-channel attacks. *J. Cryptogr. Eng.*, 9(2):137–158, 2019. doi:10.1007/s13389-018-0180-2. Cited on pages 10 and 44.

[WCJ+21] Yoo-Seung Won, Soham Chatterjee, Dirmanto Jap, Arindam Basu, and Shivam Bhasin. DeepFreeze: Cold Boot Attacks and High Fidelity Model Recovery on Commercial EdgeML Device. In *IEEE/ACM International Conference On Computer Aided Design, ICCAD 2021, Munich, Germany, November 1-4, 2021*, pages 1–9. IEEE, 2021. doi:10.1109/ICCAD51958.2021.9643512. Cited on pages 3 and 6.

[WUS+17] Mario Werner, Thomas Unterluggauer, Robert Schilling, David Schaffenrath, and Stefan Mangard. Transparent memory encryption and authentication. In Marco D. Santambrogio, Diana Göhringer, Dirk Stroobandt, Nele Mentens, and Jari Nurmi, editors, *27th International Conference on Field Programmable Logic and Applications, FPL 2017, Ghent, Belgium, September 4-8, 2017*, pages 1–6. IEEE, 2017. doi:10.23919/FPL.2017.8056797. Cited on page 3.

[XS21] Wenjie Xiong and Jakub Szefer. Survey of Transient Execution Attacks and Their Mitigations. *ACM Comput. Surv.*, 54(3):54:1–54:36, 2021. doi:10.1145/3442479. Cited on page 3.

[YADA17] Salessawi Ferede Yitbarek, Misiker Tadesse Aga, Reetuparna Das, and Todd M. Austin. Cold Boot Attacks are Still Hot: Security Analysis of Memory Scramblers in Modern Processors. In *2017 IEEE International Symposium on High Performance Computer Architecture, HPCA 2017, Austin, TX, USA, February 4-8, 2017*, pages 313–324. IEEE Computer Society, 2017. doi:10.1109/HPCA.2017.10. Cited on pages 3 and 6.

[YEP+06] Chenyu Yan, Daniel Englender, Milos Prvulovic, Brian Rogers, and Yan Solihin. Improving Cost, Performance, and Security of Memory Encryption and Authentication. In *33rd International Symposium on Computer Architecture (ISCA 2006), June 17-21, 2006, Boston, MA, USA*, pages 179–190. IEEE

Computer Society, 2006. Available from: http://ieeexplore.ieee.org/xpl/mostRecentIssue.jsp?punumber=10899, doi:10.1109/ISCA.2006.22. Cited on pages 3, 12, 51, and 53.

[YGZ05] Jun Yang, Lan Gao, and Youtao Zhang. Improving Memory Encryption Performance in Secure Processors. *IEEE Trans. Computers*, 54(5):630–640, 2005. doi:10.1109/TC.2005.80. Cited on page 3.

[ZDC+12] Loic Zussa, Jean-Max Dutertre, Jessy Clédiere, Bruno Robisson, and Assia Tria. Investigation of timing constraints violation as a fault injection means. In *27th Conference on Design of Circuits and Integrated Systems (DCIS), Avignon, France*, pages 1–6, 11 2012. Cited on page 6.

A Selected full benchmark results

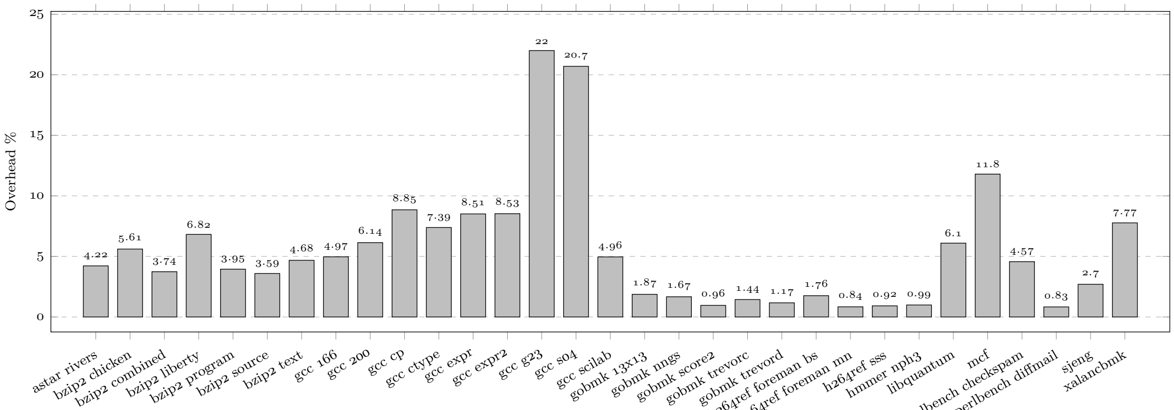

Figure 15: Stage 8/Unloaded system: AMD SEV (i.e. L1/AES/64B CLs)

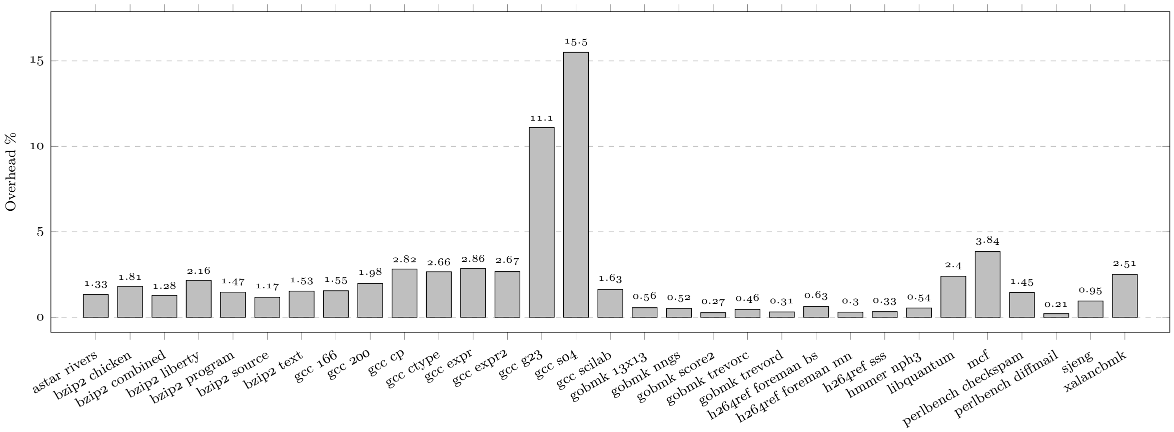

Figure 16: Stage 8/Unloaded system: L1/QARMA/64B CLs

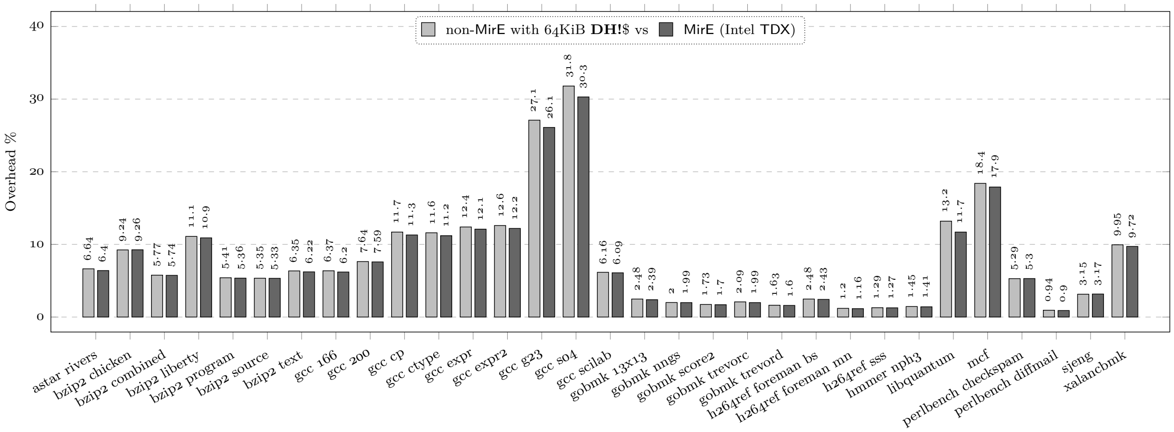

Figure 17: Stages 8 and 9/Unloaded system: `L2`/`AES`/28-32b MACs/64B CLs

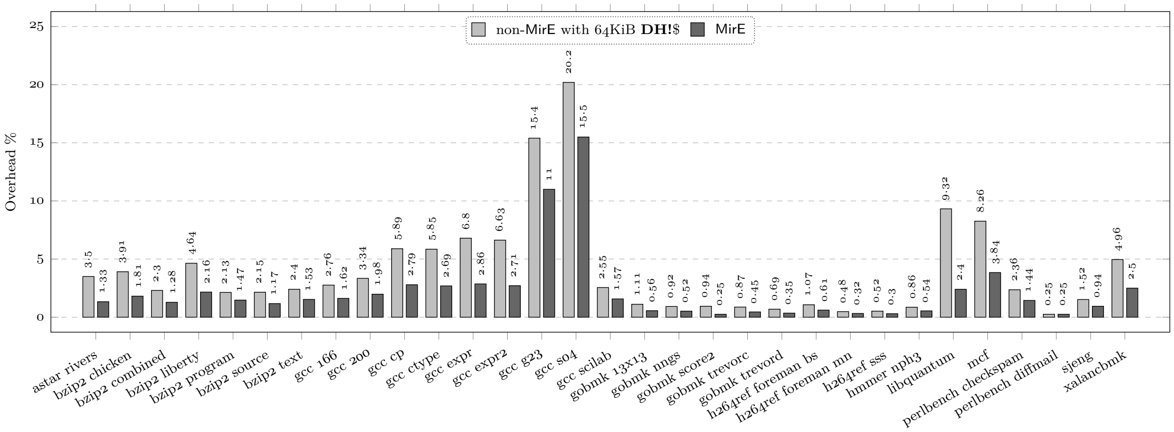

Figure 18: Stages 8 and 9/Unloaded system: `L2`/`QARMA`/64B CLs/32b MAC

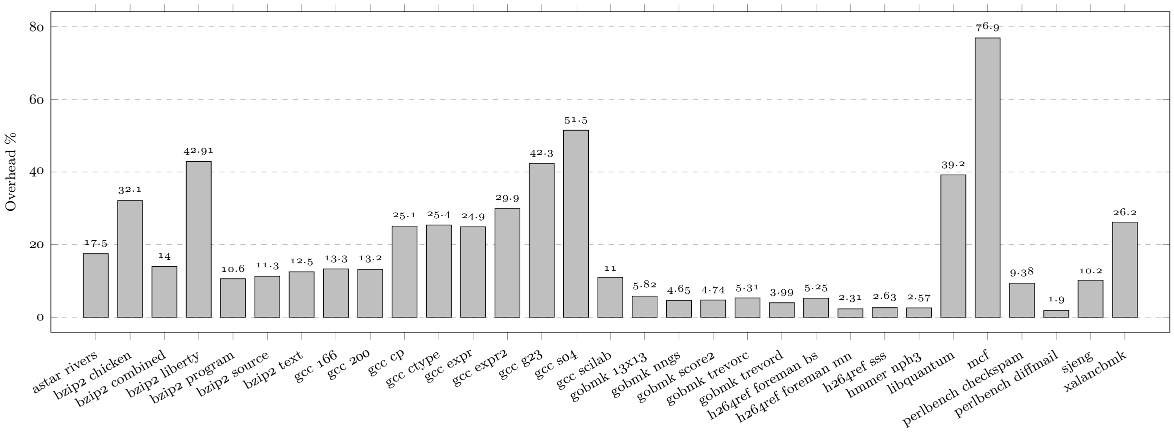

Figure 19: Stage 8/Unloaded system: Intel `SGX`/64B CLs (`L3`/`AES`/56b MACs/64B CLs)

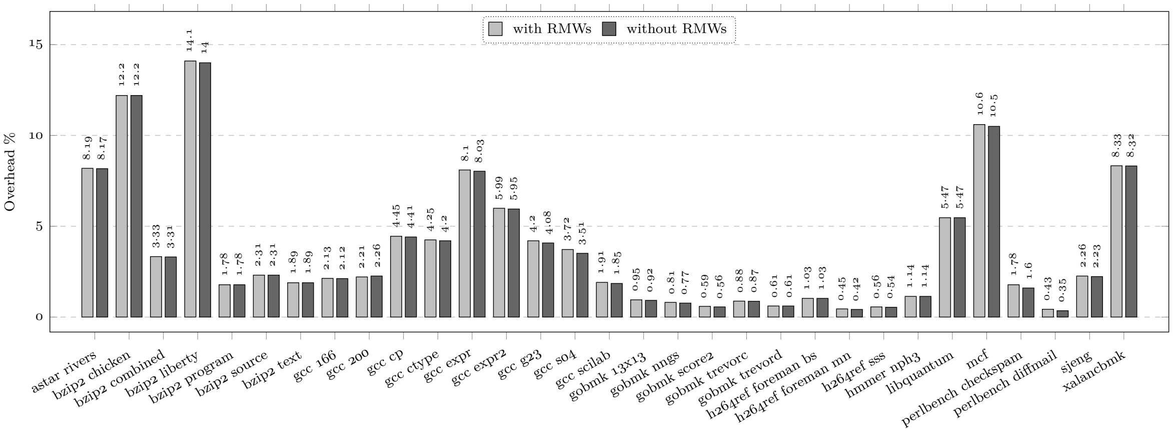

Figure 20: Stages 8 and 9/Unloaded system: L3/QARMA/split/128B CLs/32b MACs – runs with and without RMWs

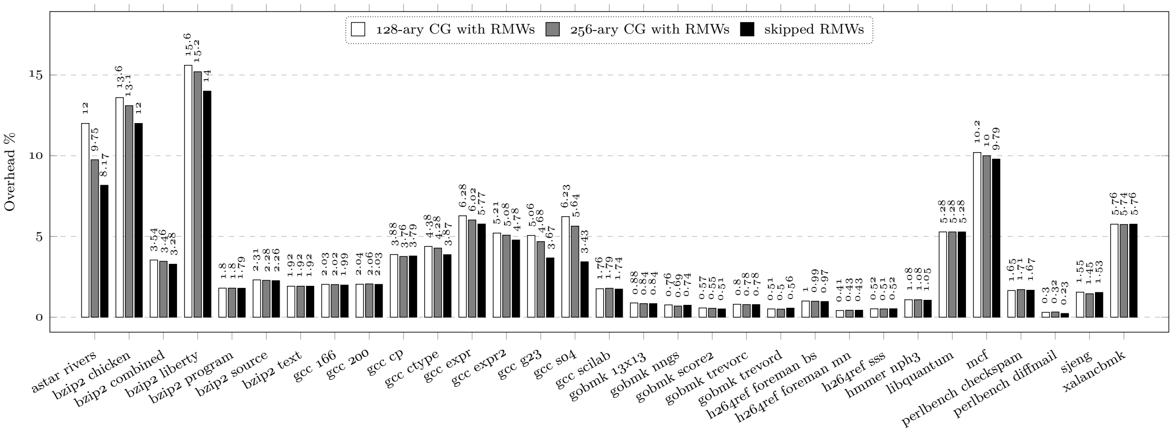

Figure 21: Stages 8 and 9/Unloaded system: L3/QARMA/oCC/32b MACs – runs with and without RMWs

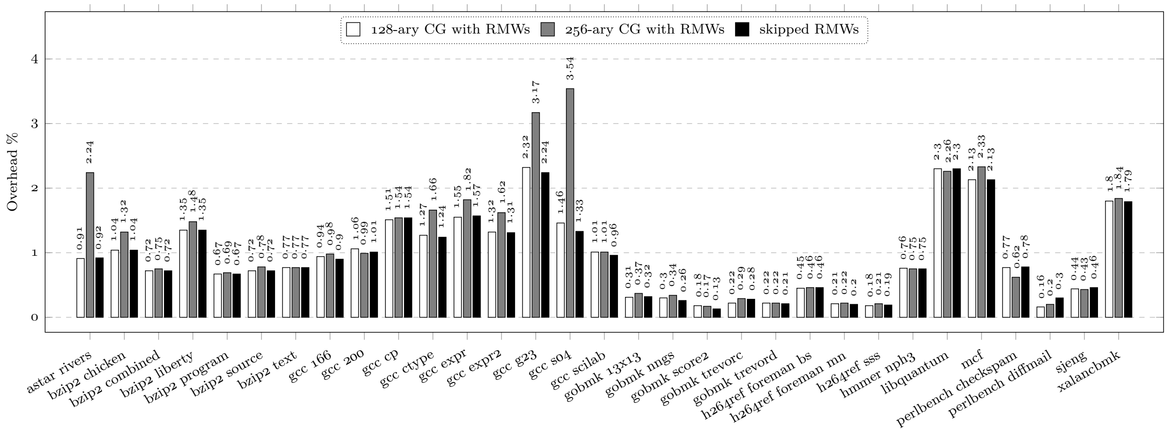

Figure 22: Stages 8 and 9/Unloaded system: L3/QARMA/oCC/MirE – runs with and without RMWs

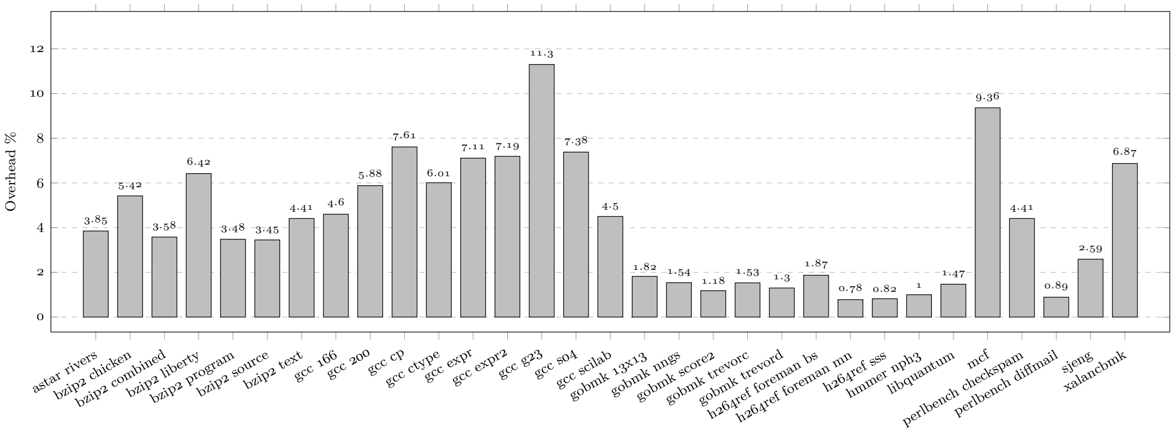

Figure 23: Stage 8/Loaded system: AMD SEV (i.e. L1/AES/64B CLs)

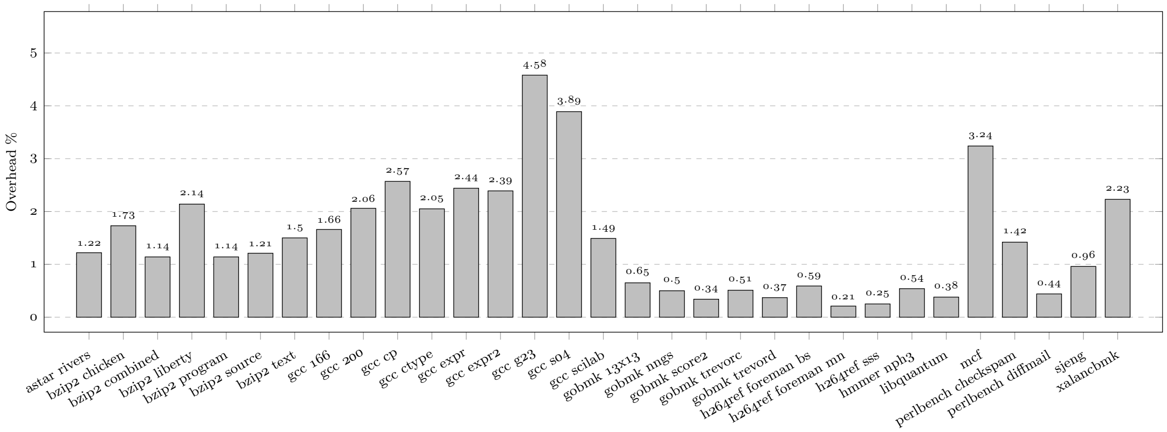

Figure 24: Stage 8/Loaded system: L1/QARMA/64B CLs

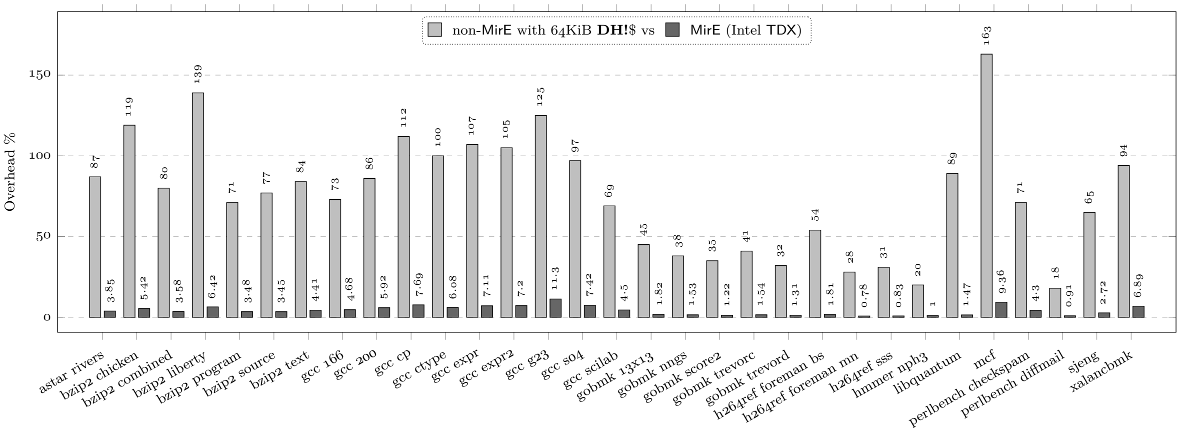

Figure 25: Stages 8 and 9/Loaded system: L2/AES/28-32b MACs/64B CLs

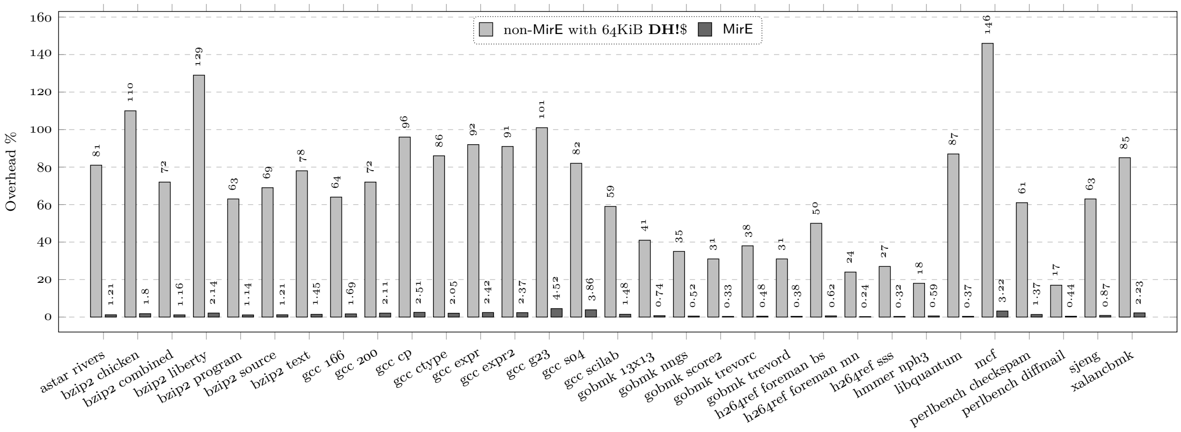

Figure 26: Stages 8 and 9/Loaded system: L2/QARMA/64B CLs/32b MAC

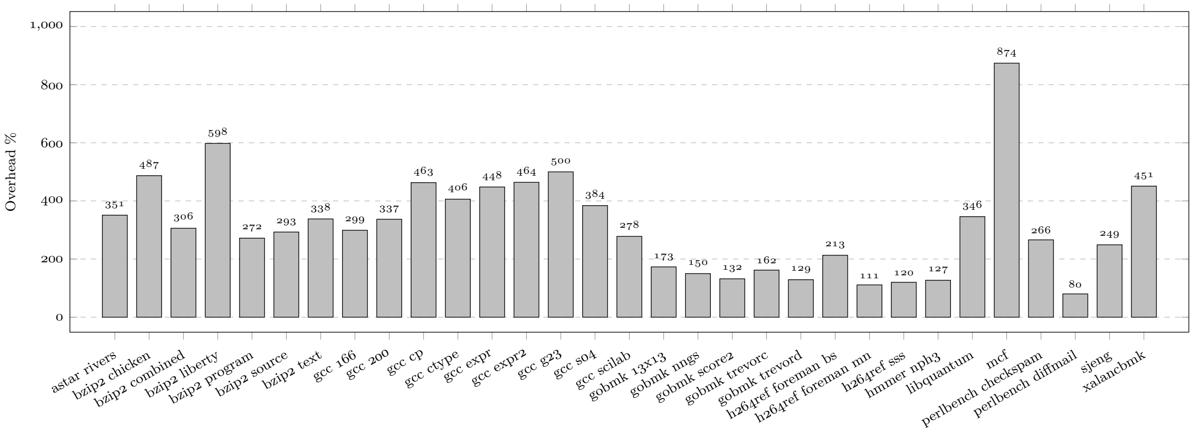

Figure 27: Stage 8/Loaded system: Intel SGX/64B CLs (L3/AES/56b MACs)

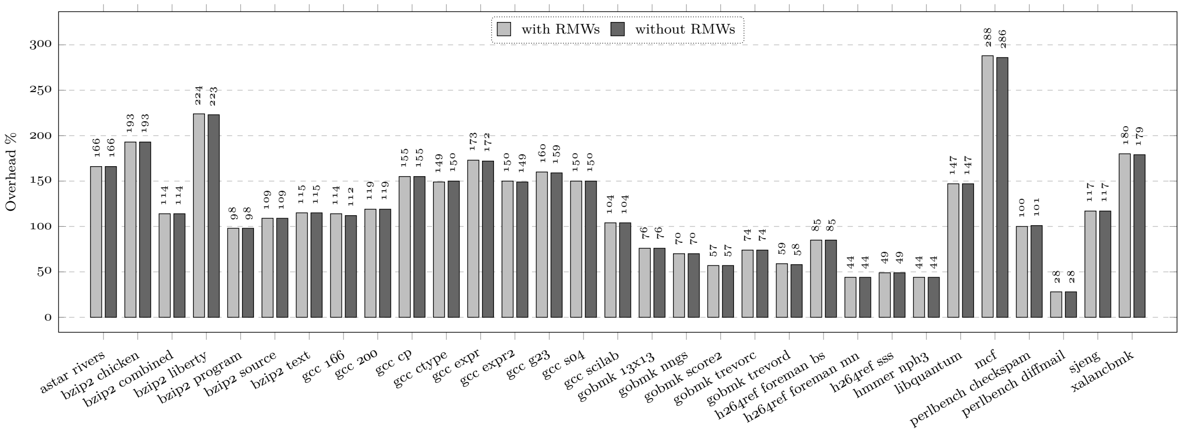

Figure 28: Stages 8 and 9/Loaded system: L3/QARMA/split/128B CLs/32b MACs – runs with and without RMWs

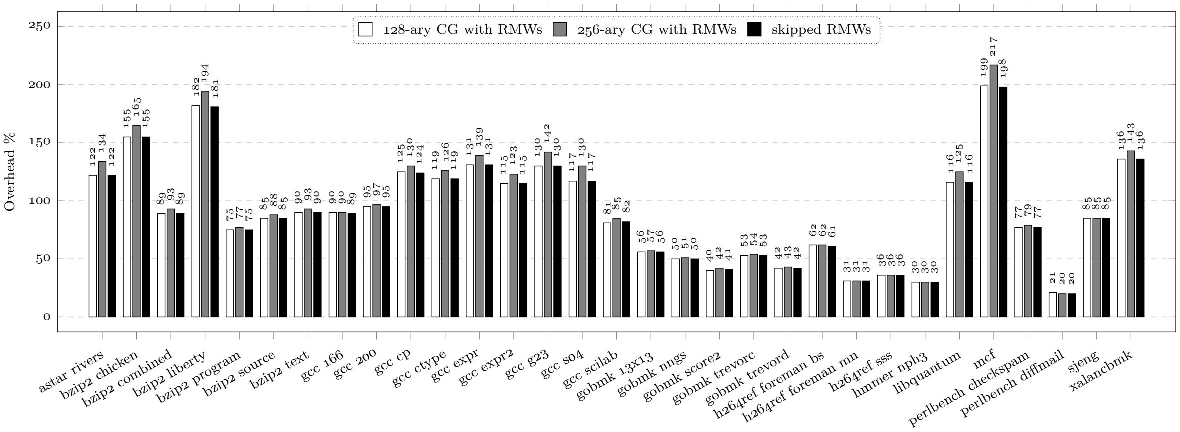

Figure 29: Stages 8 and 9/Loaded system: L3/QARMA/oCC/32b MACs – runs with and without RMWs

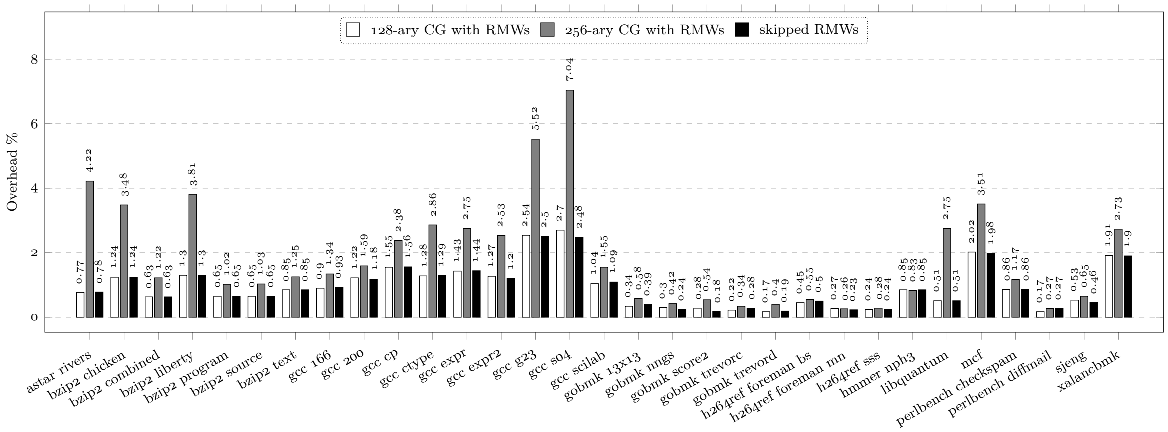

Figure 30: Stages 8 and 9/Loaded system: L3/QARMA/oCC/MirE – runs with and without RMWs

B Extended background material

Here we recall some of the most important technologies that can be deployed for memory protection. This overview does not attempt to be exhaustive. Rather, it serves to provide background reference to the rest of the paper and context for our choices of technologies.

B.1 Memory encryption primitives

B.1.1 Block ciphers and tweakable block ciphers

Block ciphers are the most common type of functions used to encrypt memory. In *direct encryption* they are applied block-wise to the plaintext to generate the ciphertext. In *OTP encryption*, the encryptions of successive values of a nonce are XOR-ed block-wise to the plaintext.

Classical block ciphers accept two inputs: a key and a text. To include the physical memory address and freshness information in the encryption algorithm it is useful to use a special type of block cipher that accepts a third input, which called the *tweak*. Key and tweak together select the function applied to the text. Such a primitive is called a TBC.

Liskov, Rivest and Wagner in [LRW02, LRW11] introduced TBCs and two methods to construct them from classical block ciphers: LRW1 and LRW2. Rogaway's XEX construction [Rog04] is an extension of LWR2 to encrypt several blocks with the same tweak. XEX is defined as

$$C_i = E_K(P_i \oplus M_i) \oplus M_i \ \ ,$$

where 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 have $M_i = \gamma^i \cdot M_0$. This is represented graphically in Fig. 35 on page 47.

B.1.2 The Advanced Encryption Standard (AES)

The block cipher most commonly used for ME is the AES [DR02a, DR02b]. When it was made into a standard in 2001 by the U.S. NIST, ME was not considered as part of the requirements motivating the selection process. This does not mean that the cipher cannot be used for the purpose, and in fact it has been employed, just to mention a few examples, in AEGIS [SCG+03], in Intel's SGX [Int15, Gue16b] and TDX [Int21b], and in the AMD SEV [AMD20] encryption engine [KPW16] but area, power consumption, and latency are not optimal. The AES is a classical block cipher and therefore requires an additional construction in order to be used as a tweakable block cipher, which further increases latency and area requirements.

B.1.3 PRESENT

PRESENT is a 64b block cipher developed by Andrey Bogdanov et al. and introduced at CHES 2007 [BKL+07]. There are two versions, with key lengths of 80 and 128 bits, called PRESENT-80 and PRESENT-128, respectively. It was designed for low-cost devices like RFID-tags, and it has been standardised in ISO/IEC 29192 Part 2: Block ciphers.

NXP revealed at the LightSec workshop in Cannes, on November 10th, 2016, that one of their customers has used PRESENT to implement memory encryption in a secure smart phone.

We do not consider PRESENT in this study because, whereas it is optimised for extremely compact HW implementations, and the individual rounds are very fast, its overall latency is comparable to that of the AES [KNR12]. Thus, it does not offer a different proposition in our context.

B.1.4 PRINCE

PRINCE is a 64b block cipher with design goals of low latency and low area [BCG+12]. It supports the single key size of 128 bits. ME is an explicit goal of this cipher. By design, decryption and encryption differ only in the key schedule, and therefore both operations can be provided with a very small overhead with respect to an encryption-only implementation. Similarly to the AES, PRINCE requires an additional construction to be used as a tweakable block cipher.

A revised version of PRINCE, called PRINCEv2 with an improved key schedule has been recently introduced [BEK+20]. Like PRINCE, PRINCEv2 is not tweakable.

B.1.5 QARMA

QARMA [Ava17] is a TBC designed to meet the needs of specific use cases such as: memory encryption; the generation and verification of short tags in pointers for hardware assisted prevention of software exploitation [AKT14, ARM16, QPS17]; and the construction of keyed hash functions. It supports two block sizes of 64b and 128b, with 128b and 256b keys, respectively, and it aims at providing conservative security margins within the constraints determined by the mentioned applications, while still achieving best-in-class latency. It has been considered as a primitive in various papers on cryptographic memory protection [UWM19, JLK+23].

We consider QARMA as a primitive in our study because QARMA-128 offers latency advantages over AES at similar or smaller area, and QARMA-64 can be used in the implementation of PMACs. PRINCE on the other hand does not have a 128-bit block version and is not tweakable, making it less interesting for high-performance applications.

B.1.6 SPEEDY

SPEEDY [LMMR21] is a new and original design for ME. Block size and key size are both 192 bits. It is not a tweakable design. It provides extremely fast encryption, and somewhat slower decryption, thanks to a high-speed 6b substitution box directly optimised for performance in CMOS hardware – but whose inverse is considerably more complex. For this reason its designers argue that the cipher is ideally used to create a bitstream for OTP encryption. Furthermore, the unusual block size makes it unsuitable for use it with current DDR4 and DDR5 memories in direct encryption modes.

B.1.7 Other block ciphers

Several states have introduced their own equivalent of the AES. Some of these ciphers are:

- The Chinese block cipher SM4, a generalised Feistel Network with high latency;
- The AES-like Russian block cipher Кузнечик (Kuznechik, i.e. “Grasshopper” in Russian);
- The Ukrainian block cipher Калина (Kalyna, the ukrainian name of the national flower of Ukraine, i.e. the Guelder rose), another AES-like design; and
- BelT, the Belarusian encryption standard, a complex 8-round Massey-Lai/Feistel hybrid design with high latency in HW because of the use of modular additions and subtractions.

The arguments about the AES apply to these ciphers as well: these are generic purpose block that are targeted at applications that do not require low latency. Their performance is similar to that of the AES, hence considering them here would essentially be a duplication of work.

B.1.8 Stream ciphers

Stream ciphers can be used to generate OTPs for ME, however because of their initial latency and the lack of parallelisability we do not consider them in this study.

B.2 Authentication primitives

B.2.1 General purpose hash functions and Message Authentication Codes

Similarly to the AES, there are general purpose standards for hashing functions, the SHA series: SHA-2 [NIS12] and SHA-3 [NIS15]. However, these functions have significant latency.

Prepending or appending a secret key string to the message turns a hash function into a keyed hash function, i.e. a MAC, however this may not be a secure construction as it might expose to extension attacks. This is not a concern if the use case requires only the hashing of messages of *fixed size*, such as CLs, eliminating the need for expensive constructions such as NMAC or HMAC [BCK96].

B.2.2 SipHash

SipHash [AB12] is a cryptographically strong PRF that is used as MAC. We do not consider it here because it is based on 64b integer additions, which makes the latency of HW implementations too high, and it is not parallelizable.

B.2.3 Universal Hashing

A UHF [CW77, CW79] is a parametrised family $\mathcal{H} = \{h_\varkappa(x) : \varkappa \in \mathcal{K}\}$ of functions from a set $\mathbb{U}$ (the *universe*) to a second set $\mathbb{V} = [0 \,..\, m-1] \subset \mathbb{N}$ (the set of *values*) with the following property: for all $x \neq y \in \mathbb{U}$, the probability that for a uniformly chosen at random $\varkappa \in \mathcal{K}$ there is a collision for x and y, i.e. $h_\varkappa(x) = h_\varkappa(y)$, is at most $1/m$. For $\epsilon \geq 1$, a *ϵ-almost UHF* satisfies the weaker condition that this probability is at most ϵ/m.

The parametrised function $h_\varkappa(x) : \mathcal{K} \times \mathbb{U} \to \mathbb{V}$ is also called a UHF. *Universal hashing* consists then in picking a random $\varkappa$ and using the function $h_\varkappa$ to hash from $\mathbb{U}$ to $\mathbb{V}$. The *hash functions* $h_\varkappa$ do not need to be cryptographically secure: The result of universal hashing can be fed into a cryptographically strong hash function (or encrypted) to obtain a MAC, where the secret key is $\varkappa$.

Another property is *uniformity*. A UHF is uniform if all hash values are equally likely: $\Pr(h(x) = z) = 1/m$ for any value $z \in \mathbb{V}$.

A further desirable property is *pairwise independence*, which means that $\forall x, y \in \mathbb{U}$, $x \neq y$ the probability that (x, y) hashes to any pair of hash values $(w, z) \in \mathbb{V}^2$ is as if they were perfectly random: $\Pr(h(x) = w \wedge h(y) = z) = 1/m^2$. Pairwise independence is also called *strong universality*.

Universality does not imply uniformity, whereas strong universality does.

There are several constructions for UHFs in the literature. One of Carter and Wegman's original proposals consists in splitting a message M which is rt bits long in r chunks of t bits each as in

$$M = m_0 \| m_1 \| \ldots \| m_{r-1}$$

and computing its hash as

$$h_\varkappa(x) = \tau(\kappa_{-1} + \kappa_0 \cdot m_0 + \kappa_1 \cdot m_1 + \cdots + \kappa_{r-1} \cdot m_{r-1})$$

in some ring $\mathcal{R}$, where:

1. The elements of $\mathcal{R}$ are represented by strings of at least t bits;

2. All t-bit strings map to different elements of $\mathcal{R}$;
3. The values $\varkappa = [\kappa_{-1}, \kappa_0, \kappa_1, \dots, \kappa_{r-1}]$, are generated (pseudo-)randomly and independently – resulting in a ML hash – or they can be of the form $\kappa_i = \kappa_0^{i+1}$ (for $i \geq 0$), i.e. be successive powers of a single element of $\mathcal{R}$ – giving a *polynomial hash*; and
4. The function τ maps the result to t-bit strings, usually by truncation of a canonical internal representation of the elements of $\mathcal{R}$.

Usual choices for the ring $\mathcal{R}$ are the integers modulo a prime $p > 2^t$, or a Galois field $\mathbb{F}_{2^t}$.

Polynomial or ML Carter-Wegman UHFs over a binary field are among the algorithms of choice to hash memory CLs because of their intrinsic parallelism and much shorter critical path than general purpose hash functions or UHFs based on modular arithmetic. Intel uses an encrypted multi-linear hash over $\mathbb{F}_{2^{64}}$ in SGX [Gue16b].

Daniel Bernstein's Poly1305-AES [Ber05b, Ber05c] is a MAC based on a Carter-Wegman polynomial UHF, computed over the ring of integers modulo $2^{130} - 5$. The message is divided into 128-bit coefficients and the secret point at which the polynomial is evaluated is also 128 bits long. The result is then added to the AES encryption of a nonce under a second secret key and truncated to 128 bits. Because of its use of modular arithmetic it is not suitable for memory authentication.

B.2.4 Using a TBC-based PMAC for authentication

A TBC-based *Parallel MAC (PMAC)* [Rog04] for authentication is defined as in Fig. 31 on the facing page where α_i is the PA of block i for all i, $0 \leq i < r$, when there is no freshness, otherwise it is defined as in Fig. 32 on the next page where all blocks are encrypted using their addresses as tweak and the sum of these results is encrypted by adding it to an encryption of the nonce. The nonce is encrypted using the same primitive and the same key, but with a domain separated version of the CL's PA as tweak. This construction allows to use MACs for error correction [SNR+18, JLK+23].

B.3 Modes of operation

B.3.1 Electronic Codebook Mode and tweaked variants

For a non-tweakable block cipher E the use of ECB mode is discouraged as it does not hide spatial data patterns. For a tweakable block cipher E, however, a Tweakable ECB mode can be used, where each block is encrypted using a common key and the PA of the CL enters the tweak, as in Fig. 36 on the facing page as part of the nonce ν. Thus, spatial patterns are hidden, but temporal patterns are not. These methods are trivially parallelisable.

B.3.2 Authenticated counter-based modes

GCM [MV04] is a mode based on OTP encryption and polynomial authentication. It is depicted in Fig. 38 on page 49 in a variant simplified for ME. There H is a secret hashing key.

For ME the mode can be straightforwardly modified by replacing the polynomial UHF by a ML UHF for the authentication part, as done by Intel in SGX [Gue16b].

Partial taxonomy of memory encryption modes and integrity algorithms

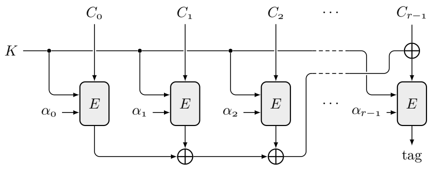

Figure 31: PMAC computed with a TBC for the cases where freshness is not implemented

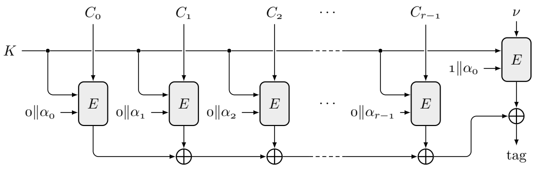

Figure 32: PMAC computed with a TBC for the cases where freshness information is available

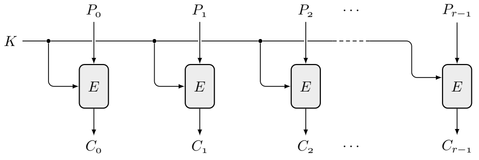

Figure 33: *Electronic Codebook (ECB)* mode

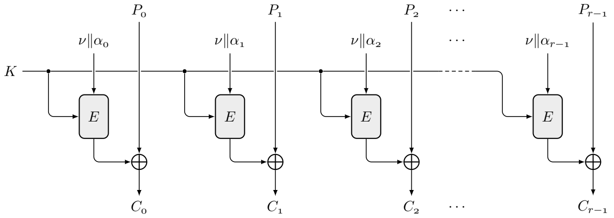

Figure 34: Counter mode

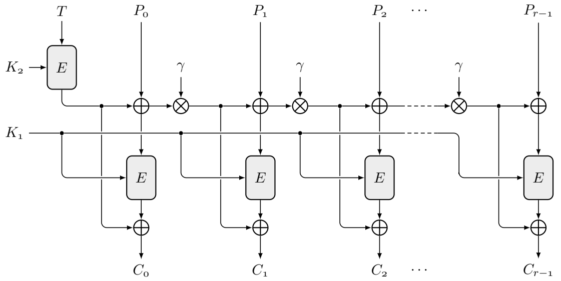

Figure 35: *XOR, Encrypt, and XOR (XEX)* mode

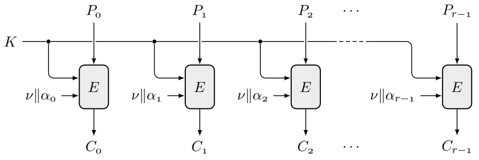

Figure 36: Tweaked ECB mode

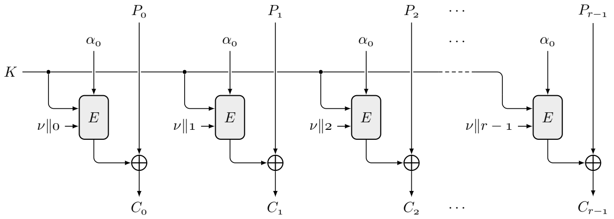

Figure 37: CounTeR in Tweak mode

B.3.3 XEX and OCB modes

XEX (Fig. 35 on the preceding page) can be turned into an AE mode by applying to the ciphertext a Carter-Wegman UHF based on Galois multiplication, using already available hardware. This is represented in Fig. 40 on the next page in the case where the plaintext has fixed length and there is no *Associated Data (AD)*. H is a secret hashing key (the parameter selecting the polynomial UHF). There γ is a fixed value, as mentioned in Appendix B.1.1 on page 42: it can be a "small" element – i.e. be represented by an element of small degree and implemented by a simple LFSR. A variant with a ML UHF is straightforward to realise.

XEX was originally devised to construct efficient instantiations of Liskov, Rivest and Wagner's *Tweakable Authenticated Encryption (TAE)* mode [LRW02, Section 4.3], resulting in Rogaway's OCB modes [RBB03]. The version depicted in Fig. 41 on the next page for ME is a particular case of OCB version 3 [KR11, Section 4.2].

B.3.4 Using Tweakable Block Ciphers

The CTRT mode, introduced by Peyrin and Seurin in [PS16], depicted in Fig. 37 on the preceding page, is a counter mode designed around a TBC. Note that all calls to the TBC use the same base address of the CL as the text input, and only the nonce is concatenated with a local counter. In the cited paper CTRT is used as a building block, and does not provide integrity per se. To provide a tag, we present two choices:

- If the MAC is not used to provide error detection and correction, then the best solution, area- and latency-wise, is a ML UHF (cf. Appendix B.2.3 on page 45), and a realisation with 64-bit tags is represented in Fig. 39 on the facing page (note that the encryption of the hash is depicted only as an example, as in practice we encrypt the tags together with direct encryption as explained in Remark 4.1 on page 14).
- If the MAC provides both integrity and error correction [SNR$^+$18, JLK$^+$23], then a PMAC construction or similar, as described in Appendix B.2.4 on page 46, must be used.

B.4 Memory integrity structures

B.4.1 Merkle trees

In a MT [Mer80] each CL, i.e. $\mathcal{L}$ bits, is hashed to ℓ_h bits. The hashes of $a = \mathcal{L}/\ell_h$ blocks thus form another CL, that is hashed as well, and so on, until we obtain a single value. By connecting a data CL or a group of hashes that fit into a CL to its hash we obtain an a-ary tree. The root of the tree is kept in protected memory, usually on-chip.

A MT is depicted in Fig. 44 on page 50, where H is the hash function and several inputs to it are simply concatenated. MTs have been used to protect data first in the XOM system [MVS00].

Full MT traversal for each verification can be very expensive. In [GSC$^+$03] MT nodes are cached, whereby a cached node is treated as trusted, halting the traversal. This reduced the performance penalty from a factor 10 to just 22%.

Partial taxonomy of authenticated modes suitable for memory encryption

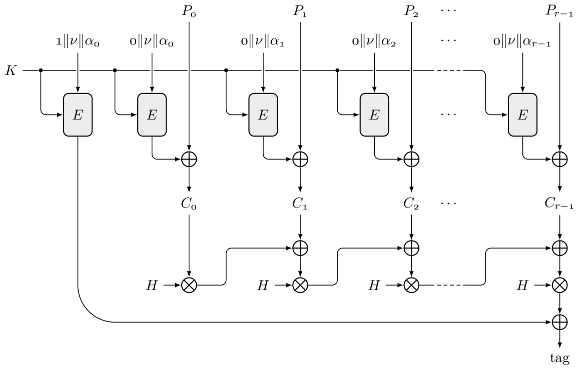

Figure 38: GCM mode

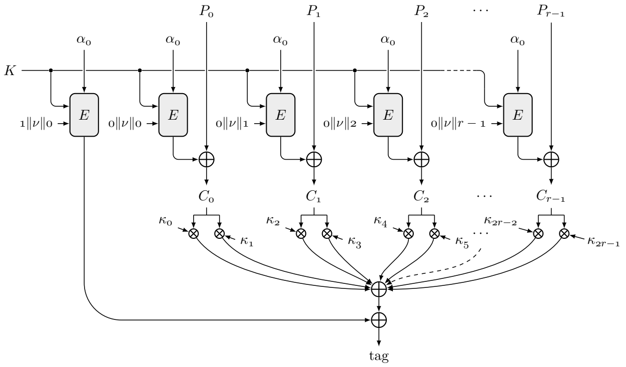

Figure 39: CounTeR in Tweak ME mode with half-block encrypted ML UHF for authentication

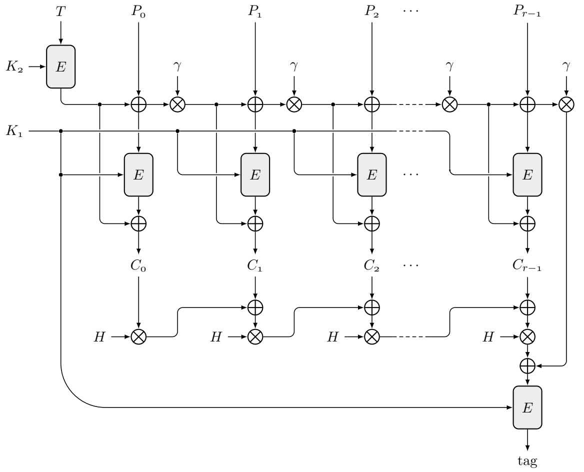

Figure 40: XEX ME mode with polynomial authentication

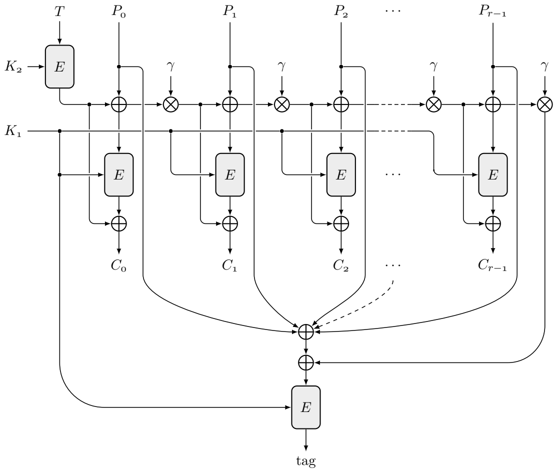

Figure 41: Modified ME OCB mode with XEX construction

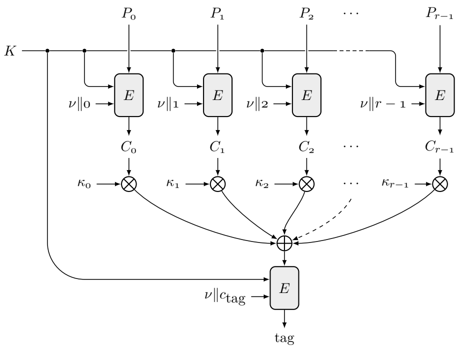

Figure 42: Tweaked Codebook ME mode with encrypted multi-linear UHF for authentication

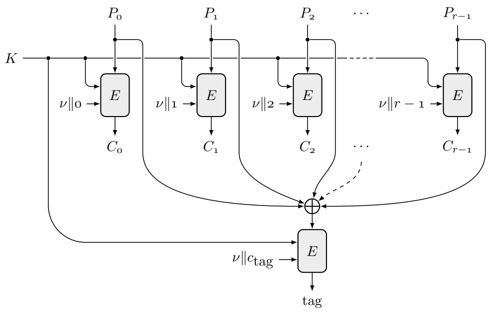

Figure 43: Tweaked Codebook ME mode with checksum based authentication

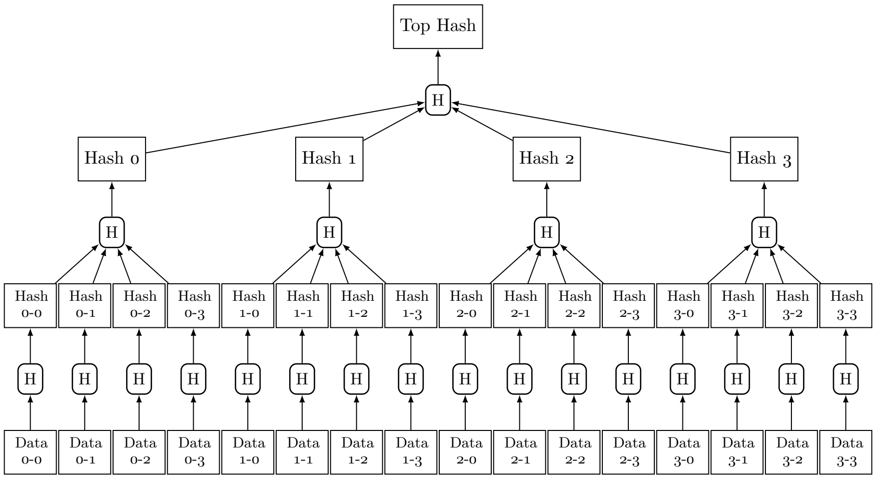

Figure 44: A Merkle Tree for Memory Integrity

B.4.2 Counter-based encryption and the Bonsai Merkle Tree

The authors of [RCPS07] observed that the counters are often shorter than the hashes in a common MT (say, 64b vs. 128b), and that *in order to protect the integrity of the text it suffices to include the counters in the computation of the hashes and to build a MT over the memory region containing the counters.* A smaller MT is obtained, called a BMT.

Note that if the counters were not included in the hash computation, any known *ciphertext* could be replayed together with its hash (even if the hash function is keyed) and would be decrypted as pseudo-random data, corrupting the memory.

Since hashes are not protected themselves by an integrity structure, they shall not be easily recomputed by an attacker. This is achieved by using a keyed hash, where the key is at least as large as the hash. The fact that the counters are used in the hash computation will allow also to use shorter hashes, as in order to successfully replay data the adversary would have to wait that the counter repeats as well. The nodes of BMT can be cached as well, since it is just a MT.

Finally, we remark that the authors of [RCPS07] use a split counter approach (described in Appendix B.4.5 on the facing page) but for simplicity of exposition, we separate the two concepts.

B.4.3 Counter Trees

Hall and Jutla's PAT [HJ05] builds on the BMT idea by repeatedly using counter/hash pairs to protect groups of counters, instead of using a MT. This way a new type of tree is constructed, where the nodes are *pairs* (ν, h) consisting of a nonce ν and a MAC h computed over the children nodes. An important of this structure over the MT is that it not only verification, but also the update operation is parallelisable. Hall and Jutla's PAT is depicted in Fig. 45 on page 52.

In a CT the PAT is reorganised to make memory accesses more efficient. First of all, a counters are grouped together so that they fit in a CL. Each such group is called a CG, and has an associated MAC. A CL worth of counters is protected itself by a counter and a MAC, instead of just a hash, and so on. All the counters in a CG provide freshness either to the encryption and hashing of

a adjacent data CLs, or to the computation of the MACs of a child CGs. The MAC of a CG is computed on the a counters and the corresponding counter in the parent node. Therefore the CGs build an a-ary *integrity tree*. The top node of this tree is just a counter. In the simplest version of a CT, all hashes are stored in their own memory region.

Intel's [Gue16a], cf. Fig. 46 on the following page regroups all the counters in a set of sibling PAT nodes together with the MAC that is otherwise stored in the parent node. The nodes of this new CT are called CGs and in Intel's implementation contain 8 56b counters and a 56b MAC.

B.4.4 TEC-Tree

In [ECL+07] an alternative to the Hall-Jutla PAT is described, which is called the TEC tree. Each node of this tree contains either a CG or a data CL, and a nonce. The nonce is concatenation of the address of the chunk and copy of the parent counter. Each node, which is called a *chunk* by the authors, is encrypted with a "block encryption" primitive called *Added Redundancy Explicit Authentication (AREA)* scheme, that can encrypt an entire chunk with full diffusion. Upon decryption of a chunk, the nonce can be immediately checked. The TEC tree is represented in Fig. 47 on the next page: the leaf nodes are data chunks, the intermediate nodes are counter chunks, and the top node is stored in secure memory.

The TEC tree's main disadvantages, i.e. the large memory storage overhead, and that it requires a wide encryption mechanism (with 64 byte CLs and 64 bit addresses and counters, this amounts to a 640 bit block cipher) with a very high latency, make it unattractive for practical deployment.

B.4.5 Split Counters for encryption and integrity

Counters and hashes use considerable space. The size of the hashes cannot be reduced without reducing security, but *split counters* can reduce the memory overhead of a BMT.

In [YEP+06] the group of a counters is replaced by a group composed of a *major counter* and $a' > a$ smaller, *minor counters*, so that the two types of CGs have the same size. The freshness information associated to a child (which can be another counter group or a data CL) consists of the concatenation of the common major counter with the minor counter corresponding to the child. The increased arity (for instance, from $a = 8$ to $a' = 64$) reduces both storage overhead for counters and tree depth. This comes however at a price in complexity and the need for potentially expensive RMW operations: Each time the a dirty cache line is evicted, the corresponding minor counter is increased first. Similarly, when a CG is evicted from the CG$, the corresponding minor counter in its parent node is increased. When a minor counter overflows, the major counter in its CG is increased, all minor counters in the CG are reset to zero, and the CLs corresponding to the minor counters in the group must be re-encrypted (for leaf nodes), or the hashes recomputed (for non-leaf nodes) in bulk, using the updated counters.

Due to the longer times of atomic re-encryption, this scheme may not be ideal for systems for which the memory latency must be kept strictly bounded. Still, if parameters are chosen carefully, reduced memory overhead and diminished cache pressure will amply more than offset the cost of re-encryptions, resulting in a significant overall performance gain.

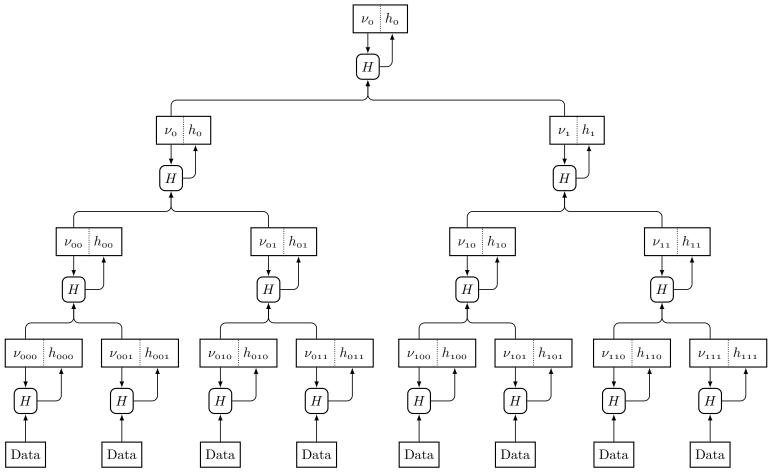

Figure 45: Hall and Jutla's Parallelisable Authentication Tree

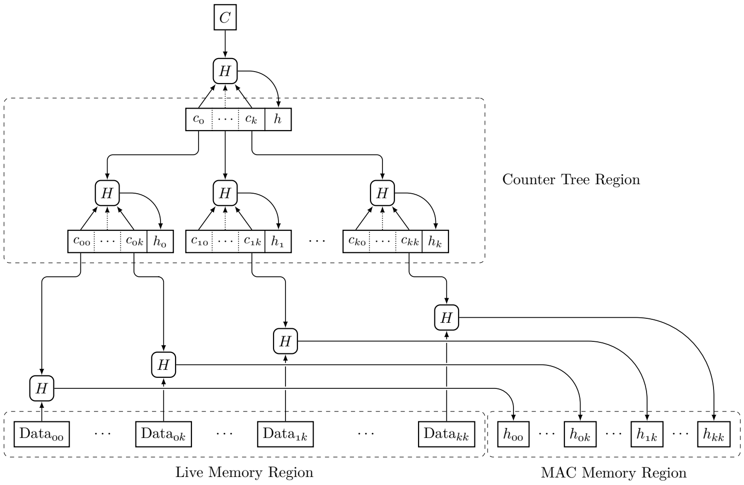
C
H
c_0 $\cdots$ c_k h
Counter Tree Region
c_{00} $\cdots$ c_{0k} h_0
c_{10} $\cdots$ c_{1k} h_1
$\cdots$
c_{k0} $\cdots$ c_{kk} h_k
Data$_{00}$ $\cdots$ Data$_{0k}$ $\cdots$ Data$_{1k}$ $\cdots$ Data$_{kk}$
h_{00} $\cdots$ h_{0k} $\cdots$ h_{1k} $\cdots$ h_{kk}
Live Memory Region
MAC Memory Region

Figure 46: Counter Tree

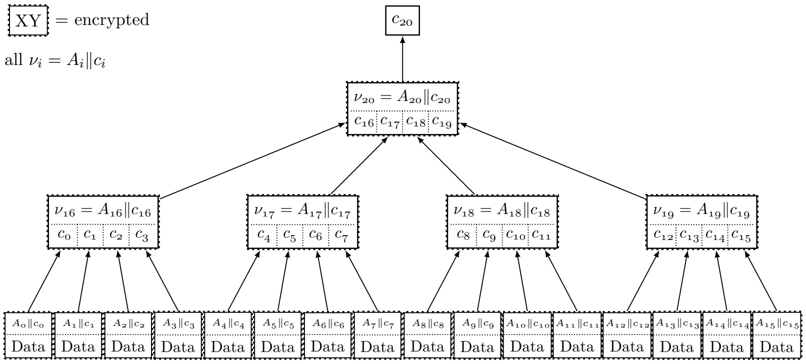
XY = encrypted
all $\nu_i = A_i \| c_i$
c_{20}
$\nu_{20} = A_{20} \| c_{20}$
c_{16} c_{17} c_{18} c_{19}
$\nu_{16} = A_{16} \| c_{16}$
c_0 c_1 c_2 c_3
$\nu_{17} = A_{17} \| c_{17}$
c_4 c_5 c_6 c_7
$\nu_{18} = A_{18} \| c_{18}$
c_8 c_9 c_{10} c_{11}
$\nu_{19} = A_{19} \| c_{19}$
c_{12} c_{13} c_{14} c_{15}
$A_0 \| c_0$ $A_1 \| c_1$ $A_2 \| c_2$ $A_3 \| c_3$ $A_4 \| c_4$ $A_5 \| c_5$ $A_6 \| c_6$ $A_7 \| c_7$ $A_8 \| c_8$ $A_9 \| c_9$ $A_{10} \| c_{10}$ $A_{11} \| c_{11}$ $A_{12} \| c_{12}$ $A_{13} \| c_{13}$ $A_{14} \| c_{14}$ $A_{15} \| c_{15}$
Data

Figure 47: TEC Tree

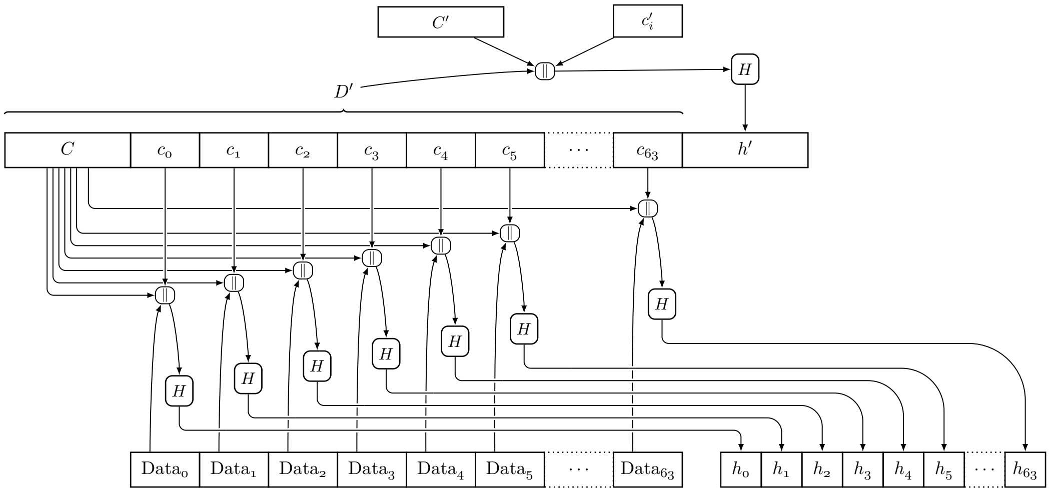

Figure 48: Split Counters

B.5 Selection of the state of the art

We briefly recall here some milestones in the area of cryptographic memory protection, the most important industrial deployments, and a few new developments.

B.5.1 AEGIS

AEGIS [SCG+03], revised in [GSC+03], is one of the most significant architectures for memory protection with confidentiality and full integrity to be found in the scientific literature. It uses a cached MT for full integrity and counter based encryption. The authors for the first time also compare the use of 64B and 128B CLs, as well as the use of multi-CL MACs.

In our terminology AEGIS is a L3 MPE.

B.5.2 Yan et al., ISCA 2006

Yan et al. [YEP+06] were the first to propose split counters. They use a cached split CT for full integrity and for ME they use the AES in a simplified GCM mode. The computation of CT node MACs reuses the same function to compute the GCM tag. Their architecture is a L3 MPE.

B.5.3 Integrity Verification with Error Correction

Huang and Suh's *Integrity Verification with Error Correction (IVEC)* [HS10] improves on the split counter based L3 scheme by Yan et al. [YEP+06] by using the MACs to perform error detection and correction. The basic idea is to use the MAC to detect and correct up to a very small number of bit errors. Suppose that for a given CL M – which can be either be a data CL or CG – the verification of its MAC m fails. Then, all texts M' are enumerated which are close to M in the Hamming metric, i,e, they differ in at most Δ bits. For each such M' it is checked whether it hashes onto the given MAC. Similarly, bit flips in the MAC m can be addressed, resulting in a corrected

MAC m', as well as bit flips in both data and MACs. If a match is found, then error correction is performed, i.e. M and m are replaced by M' and m' in memory, otherwise the error is considered unrecoverable. In order to significantly narrow and accelerate the search, parity bits are used.

IVEC is a L3 MPE.

B.5.4 Intel SGX

According to [Gue16a] the SGX MEE encrypts cache lines with a modified counter mode. The 56b integrity MACs are computed as a truncated encrypted ML UHF over a 64b Galois field. Full integrity is guaranteed by an 8-ary counter tree, where each 512b node contains eight monolithic 56b counters and a 56b MAC to authenticate the counter group using the associated counter in the parent node (8 bits are unused). It does not have a MAC or DH cache. It has a CG cache: through reverse engineering it has been established [HK19] that it is is a 64KiB, 8-way set associative cache.

In our terminology the SGX MEE is a L3 MPE.

B.5.5 SYNERGY

SYNERGY [SNR+18] provides confidentiality and full integrity protection of memory by implementing a CT with monolithic counters, like SGX, but storing the 64b data MACs in the repurposed ECC bits. The MACs are used for error correction as in IVEC. Parity bits are stored in a special region of the RAM.

SYNERGY has a L3/MirE MPE, where MirE means *MACs in repurposed ECC bits.*

B.5.6 Intel TDX

According to [Int21c, Section 2.A] the MKTMEi computes the tags by applying SHA-3-256 [NIS15] to the “cache line” (according to the documentation) and then truncating the 256b output to just 28 bits. These tags are stored together with the ciphertext by repurposing 28 of the ECC bits [Int21a, Section 16.2]. We are going to assume that SHA-3 is applied to the plaintext, so that it can be computed in parallel with the encryption.

Intel does not document error correction.

In our terminology the MKTMEi is a L2 MPE.

B.5.7 Amd SEV

The ME in Amd SEV is controlled by an *Instruction Set Architecture (ISA)* extension called *Secure Memory Encryption (SME)* [KPW16]. It uses AES-128 in a XEX construction where the physical address is used to give a different permutation for each location, and each protected virtual machine has its own encryption key. The selected encryption algorithm and the fact that the hypervisor has read access allows adversaries to use the ciphertext as a side-channel to break the constant-time implementations of RSA and ECDSA from OpenSSL [LZW+21].

Amd SEV has a L1 MPE.

B.5.8 CSI:Rowhammer

CSI:Rowhammer [JLK+23] improves on SYNERGY by truncating the data MACs to 56 bits and thus storing the eight parity bits in the repurposed ECC bits next to the truncated MAC. The CSI:Rowhammer paper does not mention freshness or anti-replay trees, which is in line with the stated goal to *protect the integrity of all data in the RAM with a cryptographic MAC to mitigate software-based RAM fault attacks like Rowhammer but also randomly occurring bit-flips.*

CSI:Rowhammer implements a L1/MirE MPE.