ResearchGate

See discussions, stats, and author profiles for this publication at: https://www.researchgate.net/publication/325373910

SMARTS: secure memory assurance of RISC-V trusted SoC

Conference Paper · June 2018

DOI: 10.1145/3214292.3214298

CITATIONS
15

READS
3,647

3 authors, including:

Ming Ming Wong
Nanyang Technological University
41 PUBLICATIONS **331** CITATIONS

SEE PROFILE

Jawad Haj-Yahya
ETH Zurich
46 PUBLICATIONS **352** CITATIONS

SEE PROFILE

Some of the authors of this publication are also working on these related projects:

Stochastic Computation (SC) View project

All content following this page was uploaded by Jawad Haj-Yahya on 07 December 2018.

The user has requested enhancement of the downloaded file.

SMARTS: Secure Memory Assurance of RISC-V Trusted SoC

Ming Ming Wong
Nanyang Technological University (NTU) Singapore
mmwong@ntu.edu.sg

Jawad Haj-Yahya
Nanyang Technological University (NTU) Singapore
jawad@ntu.edu.sg

Anupam Chattopadhyay
Nanyang Technological University (NTU) Singapore
anupam@ntu.edu.sg

ABSTRACT

Security is evolving fast as the prime design concern for modern System-on-Chip (SoC), especially for lightweight design choices. In this manuscript, we study the design of memory protection unit (MPU) that will be integrated in RISC-V trusted SoC, with the intention of achieving lightweight, yet robust countermeasure towards the known attack vectors. The proposed framework provides integrity, confidentiality and also allows the flexibility of partial encryption based on the application requirements. We extensively benchmarked with state-of-the-art works in secure memory design. Our design obtains least storage overhead among the ones reported so far.

CCS CONCEPTS

• **Security and privacy → Hardware security implementation**; • **Computer systems organization → Reduced instruction set computing**;

KEYWORDS

Memory protection unit (MPU), RISC-V SoC, AES-GCM, Bonsai Merkle Tree (BMT), Partial memory encryption (PME)

ACM Reference Format:
Ming Ming Wong, Jawad Haj-Yahya, and Anupam Chattopadhyay. 2018. SMARTS: Secure Memory Assurance of RISC-V Trusted SoC. In *HASP '18: Hardware and Architectural Support for Security and Privacy, June 2, 2018, Los Angeles, CA, USA.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3214292.3214298

1 INTRODUCTION

The emergence of recent attacks such as the Rowhammer RAM attack [26] as well as the new class of timing attacks (Meltdown [21] and Spectre [20]) have imposed a major security threat in computer systems. It has been demonstrated that attacker with suitable computational tool and a substantial physical access to the platform is capable to retrieve secrets reside in the memory storage or to manipulate the platform's security policies.

Memory protection via cryptographic primitives has therefore become an essential ingredient in modern secure processor or System-on-Chip (SoC) designs. This feature enables a closed computing system to execute a software in a trust-worthy manner and to handle sensitive information while the external components is susceptible towards malicious activities such as tampering or eavesdropping. The fundamental assumption is that the off-the-chip region is untrusted where the attacker has full control of the computing system, and the software running on it at any privilege level, and able to read or modify the DRAM content.

Such protection support can be incorporated in the system as an autonomous hardware unit called the Memory Protection Unit (MPU). MPU is designed under a strict engineering constraints and is compliant to the cryptographic primitives requirements so as to be integrated seamlessly with the complex micro-architecture of modern SoC. This study presents a new low-cost, low-overhead, and secure framework for MPU that is practical for RISC-V lightweight SoC.

The main contribution of this study is three-folds. *First*, authenticated encryption, particularly the AES-GCM is implemented in the MPU as it is able to guarantee confidentially and integrity protections with the minimal usage of hardware resources. *Second*, as a measure to preserve the freshness of the memory content and to prevent it from replay attack, a simplified integrity tree using Bonsai Merkle Tree (BMT) structure that imposes less timing and storage overheads is proposed. *Third*, the proposed MPU supports partial memory encryption (PME) of which sensitive data will be identified and stored (and encrypted) in the reconfigurable trusted region of the memory storage.

To this end, we formalize the MPU threat model, evaluate the framework and the rationale behind the design choices, describe the cryptographic properties and elaborate the hardware modification steps necessary for MPU integration in the RISC-V SoC.

2 A PRIMER ON RISC-V SOC

Our project is using the lowRISC [5] open source System-on-Chip (SoC) as baseline shown in Figure 1; lowRISC implements the open RISC-V Instruction Set Architecture (ISA) [31, 32]. We are building a secure system that includes various security features, such as, secure boot, encryption and authentication of off-chip memory, key management and cryptographic accelerators.

Generally, secure SoC incorporates security features in order to protect against the attacks in both hardware and software. The secure SoC implements various features to protect against known attacks on the computing system hardware and the software that is running on it. As shown in Figure 1, these features includes:

- **Secure Debug** (1) and **Secure IO** (2) to protect against various hardware threats such as Key Extraction, Illicit Debugging, Probing and Side-Channel Attacks (SCA).

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
HASP '18, June 2, 2018, Los Angeles, CA, USA
© 2018 Association for Computing Machinery.
ACM ISBN 978-1-4503-6500-0/18/06.
https://doi.org/10.1145/3214292.3214298

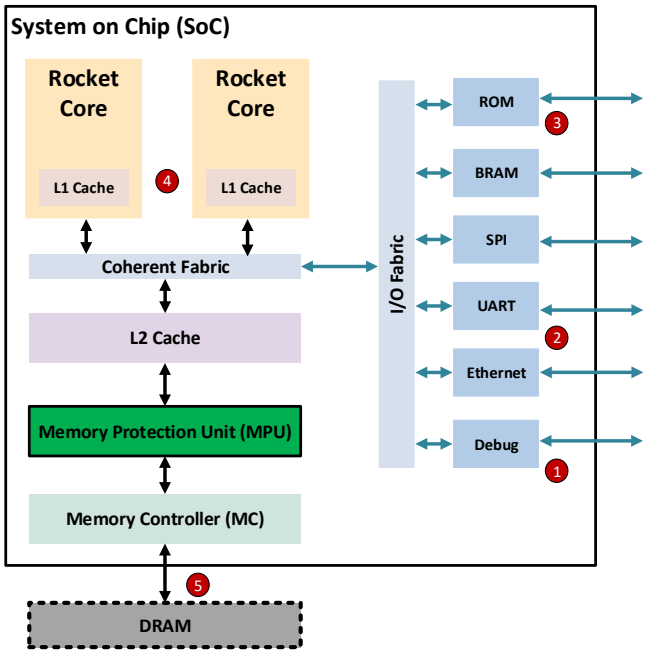

Figure 1: Simplified schematic lowRISC architecture. MPU is added between the L2 cache and the memory controller.

- **Secure Boot** 3 to protect against attacks such as: Image Hacking, Botnet Enrolling and Cold-Boot attack.
- **Trusted Execution Environment (TEE)** 4 which guarantees isolated execution environment for the trusted application, this feature is essential for protecting against attacks such as: Software Exploitation, Privilege Escalation and Botnet Enrolling.
- **Trusted Off-Chip Memory** 5 is an essential feature that protects against Side-Channel Attacks (SCA), Probing and Key Extraction from main memory. In addition, it is used by the TEE to load and execute the trusted applications while protecting the code and data of the running applications.

This study shall focus solely on the **Trusted Off-Chip Memory** architecture.

3 MEMORY PROTECTION UNIT (MPU)

The **memory encryption** and **memory integrity verification** are two main primitives that serve as a measure to prevent an attacker from tampering with the off-chip untrusted memory in a secure processor [27]. They are often realized in a separate entity named as Memory Protection Unit (MPU). MPU operates as an extension to the Memory Controller (MC) and is responsible to provide a protected transaction for the cache-DRAM traffic.

Upon every memory request, the data are routed from the MC to MPU to perform encryption/decryption prior to writing/reading to/from the DRAM. Besides that, MPU also initiates verification (and updating) transaction using metadata. Ideally, both the encryption and verification schemes should not incur excessive hardware cost and performance penalty on the memory access operation.

3.1 Threat Model and Requirements

The current state of the art reported in the literature is mainly compliant to the *non-leaking chip model*. It is based on the assumption that the processor chip (the CPU) is trusted, and the operations of all the other components including off-chip memory are verified by the processor. Hence, no active or passive attacks against the chip can be performed directly.

Aside the CPU, other device components outside the chip are under full control of the adversary. Physical attacker could probe and tamper with buses, exchange peripherals, or turn the whole device on and off [30]. Meanwhile, for off-chip memory, physical attacker is capable to access and modify the memory content.

The general aim of MPU is to provide **confidentiality** and **integrity** to data memory that is written to the DRAM. However, for modern computing, such security level may not be sufficient. During the read operation, should the attacker is able to access to confidential data stored inside the memory, malicious modification that compromise the memory authenticity can be achieved in various way listed below [13].

- **Spoofing attacks**: Existing memory block is replaced with arbitrary fake data.
- **Splicing/Relocation attacks**: Spatial permutation of memory blocks where memory block at address A is replaced with memory block at address B with $A \neq B$.
- **Replay attacks**: Temporal permutation of a memory block where memory block located at a given address is recorded and inserted back to the same address at a later point in time. In doing so, the current block's value is replaced by an older version one.

Therefore the general expectation of the deployed cryptographic schemes should be able to protect the **confidentiality, integrity** and **freshness** of the off-chip memory by following the requirements listed below.

- The only data that an adversary can retrieved from memory is in confidential form (i. e. encrypted as ciphertext).
- The only information an adversary can learn from memory is whether a memory block was changed.
- Prevention of spoofing, splicing and replay attacks.

3.2 Memory Encryption

Memory encryption mainly concerned with the confidentiality of data and code during execution, with the sole purpose to complicate the attackers' attempt to exploit the sensitive information. To be precise, it provides protection active RAM spoofing and splicing attacks. An adversary without the encryption key would be unable to alter encrypted binary, as decryption would result in corrupt code and likely leading to program termination.

Confidentiality with encryption can be deployed using symmetric or public key encryption techniques [19]. To date, memory encryption schemes proposed in the literature are designated for non-leaking chip model and they are such as the tweakable encryption modes XEX [24] and XTS [1], CBC with ESSIV [16], and counter mode encryption [25, 27, 33].

3.3 Memory Authentication (Integrity Verification)

Memory authentication (i. e. integrity verification) on the other hand is needed to prevent hardware attacks that may compromise data integrity and also insusceptible against replay attacks. Several authentication scheme has been proposed for non-leaking chip model. A popular approach is by deploying the keyed Message Authentication Code (MAC) using the block address information. Unfortunately, this scheme is still vulnerable towards replay attacks.

In order to ultimately preserve the **freshness** of the information and to protect against **replay attacks**, certain unique intelligent must be stored in trusted environment such that the attacker could not modify. Therefore, integrity tree implementation is favourable option as the tree's root is stored on the secure on-chip memory storage. This has then achieved the full resilient against spoofing, splicing and replay attacks. Three prominent examples of integrity trees are such as the Merkle trees [23], Parallelizable Authentication Trees (PAT) [18], and Tamper Evident Counter (TEC) [14] trees.

3.4 Authenticated Encryption

Another popular approach is to apply an integrated Authenticated Encryption (AE) [4], which is a shared-key based transformation, to the message to provide both encryption and authentication. In this scheme, ciphertext is derived by encrypting the plaintext with a secret key. On the decryption end, the same secret key and the ciphertext are used to obtain either the original plaintext or an indicator to verify the authenticity of the ciphertext.

AE is deemed to be hardware cost saving since the encryption and authentication can share a part of the computation. Besides, the key exchange and storage issues are better managed compared to using two separated algorithms. The recommended modes of operation for authenticated encryption are being the Counter with Cipher Block Chaining Mode (CCM) [11] and Galois Counter Mode (GCM) [22]. The latter has became a favourable choice and has earned NIST's recommendation when combined with AES cipher to form AES-GCM.

4 RELATED WORKS AND CHALLENGES

This section describes the broad landscape of the the current issues in trusted hardware projects. We highlight the recent challenges architectural design as well as the resulting security properties in the latest secure processors.

4.1 Overhead and Latency

Choosing a highly secured, efficient and cost-effective memory authentication is probably much more challenging than that of the memory encryption for several reasons. First of all, well-known authentication algorithms such as MD-5, SHA-1, or CBC-MAC have **long authentication latencies** [13]. Consequently, this imposes timing overhead where the effective memory access latency significantly increased when data brought into the processor chip and can not be used before it is authenticated.

Second, RAM authentication with replay protection that relies on authentication trees causes **significant area overheads** [27]. Not only that, in the occasion of data cache miss will result in misses at all levels of the authentication tree. Thus, one block from each level must be brought from memory sequentially and authenticated in order to complete authentication of the data block. The scenario will be worst when both data and authentication codes are cached together and this leads to the increase in cache miss rates for data accesses.

Third, the increasing size of the authentication code (hash) reduces the arity of the integrity tree and increase both **storage and performance overheads** [33]. However, it is also a concern that the probability of an undetected data modification decreases in exponential proportion to the authentication code's size.

4.2 Attacks

Attack techniques that are used to compromise computer systems can be broadly classified into two which are namely the **physical attacks** and **software attacks**. In the prior case, attacker exploits the physical implementations details to perform an operation that bypasses the limitations set by the computer system's software abstraction layers. On the hand, the attack in the latter cases are performed solely by software on the victim computer.

Though the fact that vast majority of software attack is performed via software components, there are few attack classes that exploit information in architectural level. For instance, untrusted and malicious system software may potentially lead towards memory mapping attacks on system hardware architecture (such as the Spectre or Meltdown). Furthermore, cache timing attack exploits the micro-architectural behaviour that are observable in software level. These software-based side-channel attacks are often dismissed by security analysis of most systems.

4.3 Software Attacks in Intel SGX

Intel's Software Guard Extensions (SGX) [2, 3] is a set of extensions to the Intel architecture that aims to provide integrity and confidentiality guarantees to security sensitive computation performed on a computer where all the privileged software (kernel, hypervisor, etc) is potentially malicious. The main protection mechanism in SGX is attained by Memory Encryption Engine (MEE) described in [2, 17], as it encrypts and MACs the DRAM's contents. The SGX's threat model considers DRAM and the bus connecting it to the CPU chip to be untrusted. Therefore, SGX's Memory Encryption Engine (MEE) provides confidentiality, integrity and freshness guarantees to the Enclave Page Cache (EPC) data while it is stored in DRAM.

However, MEE does not protect the addresses of the DRAM locations accessed when cache lines holding EPC data are evicted or loaded. Such drawback in SGX architecture gives way to malicious software to learn an enclave's memory access patterns and leading to cache timing attacks. To be precise, the DRAM address line bus tap can be combined with carefully crafted system software that creates artificial pressure on the last level cache (LLC) lines that hold the enclave's EPC pages [8]. This is also applicable to other processors such as Aegis [28] and Bastion [6], the predecessors of SGX.

4.4 Physical Attacks in Sanctum

Meanwhile, Sanctum [9] processor has a software/hardware co-design that offers the same promise as SGX, which is strong provable isolation of software modules running concurrently and sharing resources. Not only that, Sanctum provides the protection against software side-channel attacks, including cache timing attacks and passive address translation attacks. This additional resilience is achieved through a software isolation scheme, a simple cache partitioning scheme, where the DRAM is split into equally sized continuous region that uses distinct sets in the shared LLC. Each region is allocated to exactly one container in such a way that it is isolated in both DRAM and LLC. Similar as containers are isolated from one another, they are protected from untrusted outside software in the same manner.

However, the Sanctum design solely focus on software attacks and does not offer protection from any physical attack like SGX, Aegis and Bastion or processor that implemented Oblivious RAM (ORAM) in the MC, such as Ascend [15], that incurred a significant amount hardware's cost and performance overhead.

On the other hand, this study will present a memory protection framework equipped with customized cryptographic primitives that is both secure and practical for hardware implementation.

5 MPU FRAMEWORK FOR LIGHTWEIGHT SOC

We propose a lightweight MPU framework to provide secure memory assurance that is suitable for lowRISC. Taking into detailed consideration of the lightweight nature of RISC-V SoC, as well as the outlined model threat and requirements, and reasoning from security challenges in the existing technologies, a feasible memory access protection scheme is described in following subsections.

5.1 The AES-GCM Scheme

Technically, AES-GCM is a counter-based encryption scheme which also provides data authentication (refer to Figure 2). The encryption portion operates as a standard counter mode where sequence of pads is generated from a nonce and XORed with plaintext to produce the ciphertext. Decryption is identical, except that the plaintext and ciphertext are swapped. On the other hand, the overheads and latencies issues highlighted in the Section 4.1 can be best avoided by using the Galois Counter Mode (GCM) [12, 22] for memory authentication.

Hence, in this study, AES-GCM is chosen as the basis for RISC-V MPU framework due to several unique benefits. First, the underlying GCM authentication portion has been proven to be as secure as the deployed cipher, which is the AES. Second, unlike any existing authentication mechanisms, GCM authentication can be largely overlapped with memory latency. This low authentication latency is desirable as it allows authentication to be execute at the the soonest the data arrived after it is decrypted. With that the program performance is not severely affected by delaying instruction commit (or even data use) until authentication is complete.

Third, both the encryption and authentication portions shares the same AES hardware and this offer an optimal cost-effective solution. Besides, the GCM authentication only adds a few cycles of latency on top of AES encryption and hence high performance

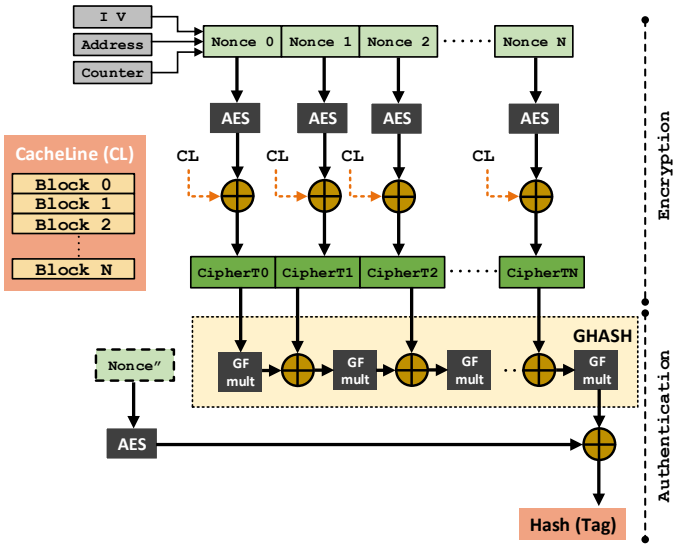

Figure 2: Encryption and authentication with AES-GCM.

efficiency. Ascon [10] is also a good alternative for authenticated encryption designs. However, it has a dedicated design and hence harder to customize in RISC-V architecture and it is not parallelizable on a message block level.

5.2 Dynamic and Reconfigurable Trusted Memory Region

The proposed MPU supports partial memory encryption (PME) mechanism with the additional support of dynamic and reconfigurable secure and trusted region. Upon system boots up, DRAM allocation is freed and repartitioned into finer granularity regions where each can be assigned to : **trusted region**, **non-trusted region** or **medata region** (as depicted in Figure 3). In compliant with the secure and trusted boot up sequence, these regions allocation is protected from interception and modification by software.

Therefore, data memory will be mapped to the allocated region according to their memory address. Data that are designated in the trusted region will then be sent forward to the MPU for authenticated encryption prior to and after the encapsulation. Such feature preserves the confidentiality of sensitive data in the DRAM and at the same time guarantees minimal leakage of the address/location mapping of the memory data. In addition to that, PME in general, is efficient in keeping the memory overhead to the minimal. As other software is blocked from altering the region allocation, this also protects the sensitive data to be maliciously written/read to untrusted zone.

Within the trusted region, the deployed cryptographic primitives is authorized to further partition the data memory address spaces into blocks of predefined size based on the deployed cipher scheme as well as the hardware specifications (such as the cache line size). The chosen block size essentially affects the trade-off in relation to metadata overhead, access granularity, and speed [30].

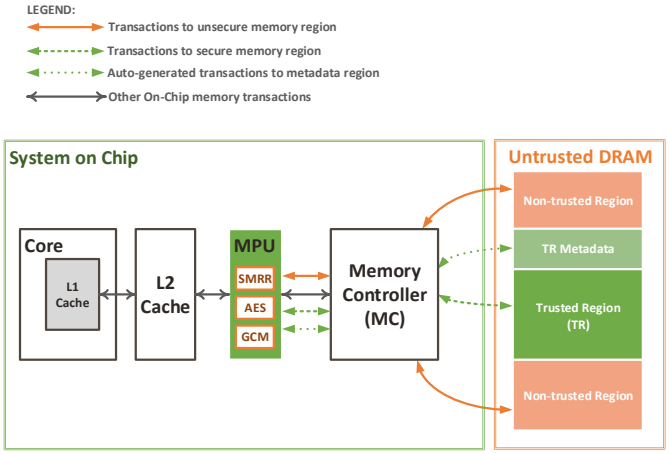

Figure 3: Dynamic and reconfigurable DRAM memory region allocation.

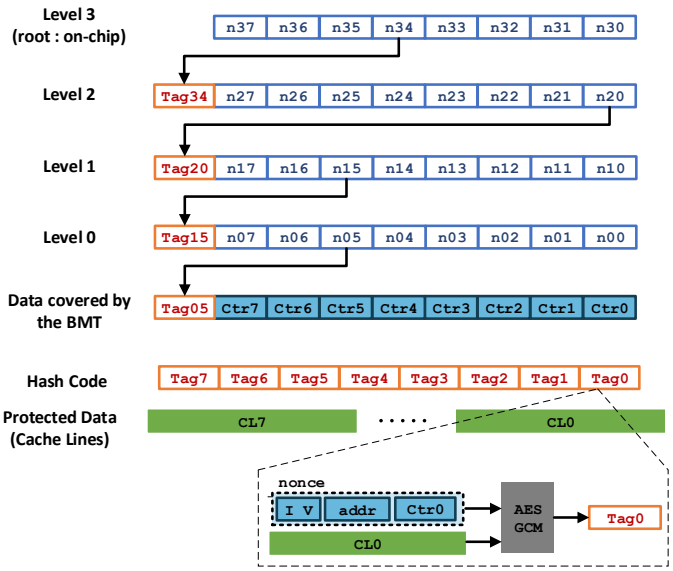

Figure 4: Branch of the Bonsai Merkle Tree (BMT) with arity of 8 with 4 levels which covers the *Counters* (Ctr) used for the authentication of the secure memory's cache lines (CL). Each *Counter* is 56 bits, *Tag* is 64 bits and cache line is 512 bits respectively.

Hence, the above-mentioned properties have in fact imposed restriction upon software access towards the DRAM and this indirectly avoids side-channel software leakage.

5.3 Integrity Tree for Memory Authentication

Authentication primitives like cryptographic hash functions and Message Authentication Code (MAC) functions can be effectively used to verify the integrity of memory data. These functions are applied to every cache line and their respective nonces. While, the produced hashes can be stored in the on-chip memory, this leads to excessive storage overheads on the secure processor. Tree-based structures called Integrity Trees were proposed to eliminate this storage overhead. One of these schemes is called **Merkle Tree (MT)**, where tree splits the memory space into M equal size blocks which form the leaf nodes of an integrity tree. The remaining tree levels are created by recursively applying a hash over the children of that node. This recursive application of the authentication primitive yields a single Integrity Tree Root (ITR) node that is stored on the secure tamper-resistant processor. The root reflects the current state of the entire memory. The cache lines and intermediate tree nodes are stored in the main memory. The number of checks required to verify the integrity of a leaf node depends on the number of memory blocks. The number of checks corresponds to the number of tree levels is given by $\log_a M$ where a is the arity of the tree and M is the number of memory blocks.

In our MPU framework, we proposed the usage of efficient implementation of Merkle Tree, called the Bonsai Merkle Tree (BMT)[25]. BMT reduces the amount of data to authenticate with a Merkle Tree in order to decrease the tree height. To do so, BMT compute a MAC over every memory cache line (CL) with a nonce - a *Counter* (Ctr) concatenated with the data address ($addr$) and Initialization Vector (IV) as extra input of the MAC function: $MAC = MAC_K(CL, \{IV||addr||Ctr\})$. To be precise, rather than having a Merkle Tree over the CL data, BMT is applied to build the integrity tree over the *Counters* instead. The MPU uses a compound nonce where the spatial coordinate of a unit is its address, and its temporal coordinate is a dedicated per memory block counter.

The specific instantiation of the MPU data structure is defined by setting equal size data units of CL size, which is in 512 bits in our case, *Counter* with 56 bits, and *Tag* (hash code) with 64 bits. We built an 8-ary tree over the cache lines' *Counters*; for each cache line, eight *Counters* and one *Tag* are stored. The infrastructure supports 128MB of secure memory region. Therefore, we have a tree of 6 levels, the root (top level) will be securely stored in the on-chip memory and the other levels of the tree are stored inside the main memory at the metadata section.

Therefore, each CL in the MPU region are stacked in groups of 8, in order to leverage the 8-ary layout. To read and verify a CL at a given address, the MPU derives the respective sub-address (pointers) to the CL that hold the corresponding *Tag*, *Counter* of each node in the tree from the leave until the root and recalculates the *Tag* over the path where finally the calculated root is compared to the on-chip root. For simplicity, the base address of the region is naturally aligned (to its total size). The overview of the proposed integrity tree is as depicted in Figure 4.

5.4 Chain of Trust

Our SoC maintains a chain of trust starting from the early stage of boot sequence and maintained during the OS and applications run. A hardware component called Code Authentication Unit (CAU) cryptographically verifies the signature of each trusted code that will be running on the processor using Elliptic Curve Digital Signature Algorithm (ECDSA). CAU verifies each software code that is to be executed starting from the first boot up to the applications.

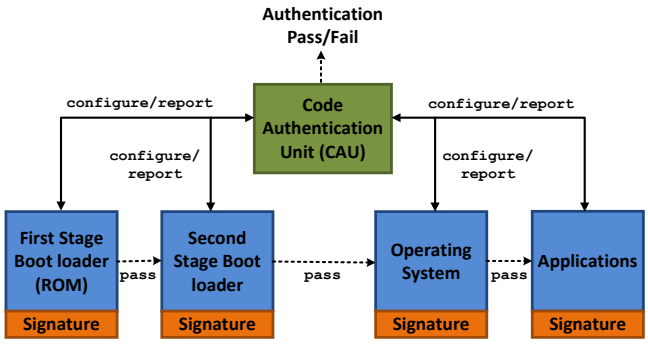

Figure 5: Chain of Trust. Starting from the fist bootloader up to the applications

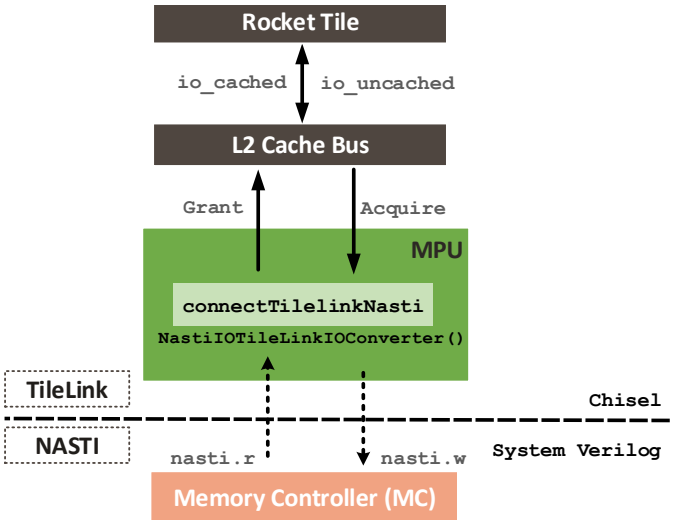

Figure 6: Bus Interface for MPU in RISC-V Rocket.

Such mechanism effectively prevents unauthorized or modified code from being executed in our secure SoC.

As shown in Figure 5, the chain starts with the first stage where the immutable bootloader is run from the read-only-memory (ROM). This bootloader cryptographically verifies the signature of the second stage bootloader in the chain and then, this subsequent bootloader again cryptographically verifies the signature of the next software image(s) and so forth. At the end of the chain, a trusted application is verified before it is running on the trusted execution environment (TEE) of the SoC. A Key Management Unit (KMU) is used to store the public key (used for signature verification) in the non-volatile memory inside the SoC. The KMU also provides secure ports to derive and to load the keys to other security subsystems on the SoC, among which the MPU.

The MPU plays substantial role in maintaining the chain of trust; while boot and applications authentication ensures that program code is not changed before it is loaded into memory; the MPU guarantees memory authentication, ensuring that program code and data is not changed or compromised while they are executed.

6 MPU INTEGRATION IN RISC-V ROCKET

This section describes the modifications needed to integrate a MPU framework in RISC-V Rocket. DRAM is exposed to the rest of the system via memory controller (MC) which feature a specific hardware interface. Cryptographic primitives, on the other hand are stringent with their respective computational cycles and block size requirements. Besides, they are also utilized with metadata which needs to be computed accordingly in the correct timing. Therefore, these hardware compatibility and computational requirements need to be aligned. A basic analysis of the presented framework is summarized at the end of the section.

6.1 Bus Interface Conversion

Rocket core uses TileLink internal buses to connect tiles, caches and the rest of the core components. TileLink uses only few basic signals for interfacing and allow easy exchanging of tiles. In Rocket tile default configuration, there are two TileLink interfaces namely the `io_cached` and `io_uncached`. It is the cache coherence protocol which connects different agents in the memory coherent model.

Meanwhile, a custom bus interface, NASTI/NASTI-Lite is deployed (UC Berkeley implementation of AXI-Lite) for the external buses and as Rocket interface for all the IO devices. Therefore, the interconnection between the cache, MC and DRAM involves interface convertor `connectTilelinkNasti` that requires `NastiIOTileLinkIOConverter()`. For effective alignment and efficient performance, the cryptographic primitives are place in between or within the connector. From the unencrypted side, translation of the memory request, decoding write request and encoding read request are performed. While on the encrypted side, support for memory read/write operations that is integrated with authenticated encryption schemes is provided. The overview of the interface connections and detail is illustrated in Figure 6.

6.2 Memory Request Translation

The memory write/read requests are incorporated with the authenticated encryption (AE) operation simultaneously and hence invisible at the higher level interface. With AES-GCM, every memory request from TileLink `io.tl.acquire.valid` will invoke AE computation and metadata generation before storing/fetching to DRAM via NASTI buses. For memory write request, the outgoing TileLink `Acquires` invokes AE computation and followed by decomposition into NASTI address and data channels to DRAM (`io.nasti.w.bits`). For memory read request, the incoming NASTI responses (`io.nasti.ar.bits`) go through AE and aggregated into TileLink `Grants`. The transaction's flow for memory write/read requests is as illustrated in Figure 7.

6.3 Data and Metadata Storage Allocation

Rocket core supports the extension for *Tagged memory* [5] which associates metadata with each memory location by extending on-chip caches to hold tags and by adding a tag cache. Interleaving

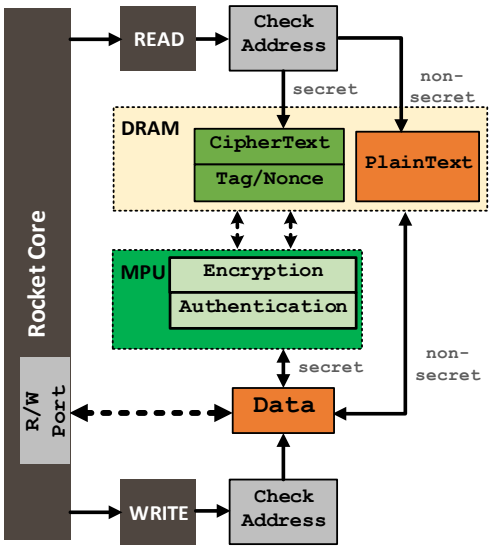

Figure 7: Write/Read memory request flow.

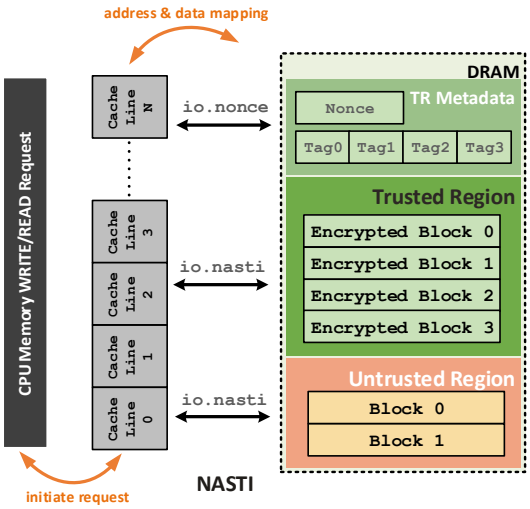

Figure 8: Data and metadata storage in allocated DRAM regions.

the metadata with the ciphertext keeps the latency low as the corresponding metadata arrives at the AE module exactly in the timing it is actually needed. However, the default size of the cache data arrays is increased to accommodate the additional metadata bits. A better alternative will be storing/fetching both the metadata and ciphertext via NASTI buses (`io.nasti` and `io.nonce`) to the separate location of the DRAM regions simultaneously (refer to Figure 8).

6.4 Characteristic Analysis and Overview

An overview of the proposed MPU framework as well as some of the existing technologies are deduced in Table 1. The analysis presented here highlights the overall overheads resulted from cryptographic computations in terms of the execution time as well as the memory storage. In addition to that, general security observation based on several perspectives such as protection coverage, mode of operation, cipher used and region of coverage are also summarized. Note that, maturity indicates how the framework was evaluated and tested.

Execution overhead which resulted from the timing latencies is analyzed in terms of total number of cycles required to perform encryption and authentication of a cache line of 64 bytes. Meanwhile, storage overheads are observed in terms of the additional memory allocated for the metadata in both the internal cache and the external RAM.

Based on the result analysis, it is evident that the proposed MPU framework is more hardware cost effective as compared to the related works. To be precise, the customized cryptographic primitives in our MPU only requires 64 bytes of the internal cache to store the root of the integrity tree. Meanwhile, the rest of the metadata which are stored in the external RAM only imposes an additional of 0.43% of the total memory storage. Besides, GCM gives the flexibility to clip the resulting Tag (hash) to a smaller size (64 bits) and is sufficiently obscured. This is still provably secured as our MPU fulfilled the conditions where AES block cipher (strong cipher) is used and that the nonces are non-repeating (composed of an incrementing counter and the block address).

Not only that, as our MPU supports reconfigurable protected memory region, the storage overhead presented is calculated based on the worst case scenario where full memory encryption is performed. Besides, the main computational module in our MPU is the AES-GCM, which comprised of `AESEnc` and `GHASH` functions. In terms FPGA implementation, the functions consumed a total of 3,943 slices and 262 slices in Xilinx platform. As AES-GCM is operated on counter mode, this contributed further saving as both the encryption and decryption processes have the identical computational flow.

7 CONCLUSION

We have presented the memory protection unit (MPU) which was implemented into our RISC-V lightweight SoC. The framework uses AES-GCM for authenticated encryption and alongside with the lightweight integrity tree, the customized cryptographic primitives incurred the least storage overhead in comparison to the existing technologies. Besides, our MPU also features partial memory encryption where sensitive software, data or application programming interfaces (APIs) will be diligently identified, encrypted and stored in the reconfigurable trusted region of the DRAM. From security perspective, our MPU fulfilled the conditions required to preserve the confidentiality, integrity and freshness of data memory that is essential for secure SoC.

ACKNOWLEDGMENTS

This research is supported by NRF-BICSAF project (Project ID: NRF2016NCR-NCR001-006).

Table 1: Characteristic analysis and overview for the proposed MPU framework

Characteristic	Our Work	XOM [29]	AEGIS [28]	Split Counter [33]	SecureMe [7]	AISE [25]
Processor Category	Mono/Counter	Mono/Direct	Mono/Counter	Mono/Counter	Multi/Counter	Multi/Counter
Execution Overhead (Average)	3.0%	50%	4.5%	2.0%	5.2%	1.6%
Storage Overhead: (I)nternal, (R)am	64B(I) 0.43%(R)	PrivateMem & XVMM (I)	12KB(I) 6%(R)	32KB(I) 1.5%(R)	32KB(I) 1.6%(R)	32KB(I) 1.6%(R)
Maturity	Simulation	Math	P-FPGA	Simulation	Simulation	Simulation
(C)onfidentiality (I)ntegrity	C + I	C + I	C + I	C + I	C + I	C + I
Encryption Algorithm	AES-GCM	3DES	AES	AES-GCM	AES	AES
Full/Partial Mem Encryption	PME	PME	FME	FME	FME	FME

REFERENCES

[1] 2008. IEEE Standard for Cryptographic Protection of Data on Block-Oriented Storage Devices. *IEEE Std 1619-2007* (April 2008), c1–32. https://doi.org/10.1109/IEEESTD.2008.4493450

[2] 2013. Intel Corporation. Software Guard Extensions Programming Reference. *Reference no. 329298-001US* (2013).

[3] 2014. Intel Corporation. Software Guard Extensions Programming Reference. *Reference no. 329298-002US* (2014).

[4] Mihir Bellare and Chanathip Namprempre. 2000. Authenticated Ecryption: Relations among Notions and Analysis of the Generic Composition Paradigm. In *Advances in Cryptology — ASIACRYPT 2000*, Tatsuaki Okamoto (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 531–545.

[5] Alex Bradbury, Gavin Ferris, and Robert Mullins. 2014. Tagged memory and minion cores in the lowRISC SoC. *Memo, University of Cambridge* (2014).

[6] D. Champagne and R. B. Lee. 2010. Scalable architectural support for trusted software. In *HPCA - 16 2010 The Sixteenth International Symposium on High-Performance Computer Architecture*. 1–12. https://doi.org/10.1109/HPCA.2010.5416657

[7] Siddhartha Chhabra, Brian Rogers, Yan Solihin, and Milos Prvulovic. 2011. SecureME: A Hardware-software Approach to Full System Security. In *Proceedings of the International Conference on Supercomputing (ICS '11)*. ACM, New York, NY, USA, 108–119. https://doi.org/10.1145/1995896.1995914

[8] Victor Costan and Srinivas Devadas. 2016. Intel SGX Explained. *IACR Cryptology ePrint Archive* 2016 (2016), 86. http://eprint.iacr.org/2016/086

[9] Victor Costan, Ilia A. Lebedev, and Srinivas Devadas. 2016. Sanctum: Minimal Hardware Extensions for Strong Software Isolation. In *25th USENIX Security Symposium, USENIX Security 16, Austin, TX, USA, August 10-12, 2016*. 857–874. https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/costan

[10] C. Dobraunig, M. Eichlseder, F. Mendel, and M. Schlaffer. 2016. Ascon v1.2. In *Submission to the CAESAR Competition*.

[11] Morris J. Dworkin. 2004. *SP800-38C. Recommendation for Block Cipher Modes of Operation: The CCM Mode for Authentication and Confidentiality*. Technical Report. Gaithersburg, MD, United States.

[12] Morris J. Dworkin. 2007. *SP 800-38D. Recommendation for Block Cipher Modes of Operation: Galois / Counter Mode (GCM) and GMAC*. Technical Report. Gaithersburg, MD, United States.

[13] Reouven Elbaz, David Champagne, Catherine Gebotys, Ruby B. Lee, Nachiketh Potlapally, and Lionel Torres. 2009. *Hardware Mechanisms for Memory Authentication: A Survey of Existing Techniques and Engines*. Springer Berlin Heidelberg, Berlin, Heidelberg, 1–22. https://doi.org/10.1007/978-3-642-01004-0_1

[14] Reouven Elbaz, David Champagne, Ruby B. Lee, Lionel Torres, Gilles Sassatelli, and Pierre Guillemin. 2007. TEC-Tree: A Low-Cost, Parallelizable Tree for Efficient Defense Against Memory Replay Attacks. In *Cryptographic Hardware and Embedded Systems - CHES 2007*, Pascal Paillier and Ingrid Verbauwhede (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 289–302.

[15] Christopher W. Fletcher, Marten van Dijk, and Srinivas Devadas. 2012. A Secure Processor Architecture for Encrypted Computation on Untrusted Programs. In *Proceedings of the Seventh ACM Workshop on Scalable Trusted Computing (STC '12)*. ACM, New York, NY, USA, 3–8. https://doi.org/10.1145/2382536.2382540

[16] Clemens Fruhwirth. 2005. *New methods in hard disk encryption*. http://clemens.endorphin.org/nmihde/nmihde-A4-ds.pdf.

[17] Shay Gueron. 2016. A Memory Encryption Engine Suitable for General Purpose Processors. *IACR Cryptology ePrint Archive* 2016 (2016), 204.

[18] W. Eric Hall and Charanjit S. Jutla. 2006. Parallelizable Authentication Trees. In *Selected Areas in Cryptography*, Bart Preneel and Stafford Tavares (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 95–109.

[19] Michael Henson and Stephen Taylor. 2014. Memory Encryption: A Survey of Existing Techniques. *ACM Comput. Surv.* 46, 4, Article 53 (March 2014), 26 pages. https://doi.org/10.1145/2566673

[20] Paul Kocher, Daniel Genkin, Daniel Gruss, Werner Haas, Mike Hamburg, Moritz Lipp, Stefan Mangard, Thomas Prescher, Michael Schwarz, and Yuval Yarom. 2018. Spectre Attacks: Exploiting Speculative Execution. *arXiv preprint arXiv:1801.01203* (2018).

[21] Moritz Lipp, Michael Schwarz, Daniel Gruss, Thomas Prescher, Werner Haas, Stefan Mangard, Paul Kocher, Daniel Genkin, Yuval Yarom, and Mike Hamburg. 2018. Meltdown. *arXiv preprint arXiv:1801.01207* (2018).

[22] David A. McGrew and John Viega. 2004. The Security and Performance of the Galois / Counter Mode (GCM) of Operation. In *Proceedings of the 5th International Conference on Cryptology in India (INDOCRYPT'04)*. Springer-Verlag, Berlin, Heidelberg, 343–355. https://doi.org/10.1007/978-3-540-30556-9_27

[23] R. C. Merkle. 1980. Protocols for Public Key Cryptosystems. In *1980 IEEE Symposium on Security and Privacy*. 122–122. https://doi.org/10.1109/SP.1980.10006

[24] Phillip Rogaway. 2004. Efficient Instantiations of Tweakable Blockciphers and Refinements to Modes OCB and PMAC. In *Advances in Cryptology - ASIACRYPT 2004*, Pil Joong Lee (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 16–31.

[25] B. Rogers, S. Chhabra, M. Prvulovic, and Y. Solihin. 2007. 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 2007)*. 183–196. https://doi.org/10.1109/MICRO.2007.16

[26] Mark Seaborn and Thomas Dullien. 2015. Exploiting the DRAM Rowhammer bug to gain kernel privileges. *Black Hat* (2015), 7–9.

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

[28] G. Edward Suh, Dwaine Clarke, Blaise Gassend, Marten van Dijk, and Srinivas Devadas. 2003. AEGIS: Architecture for Tamper-evident and Tamper-resistant Processing. In *Proceedings of the 17th Annual International Conference on Supercomputing (ICS '03)*. ACM, New York, NY, USA, 160–171. https://doi.org/10.1145/782814.782838

[29] David Lie Chandramohan Thekkath, Mark Mitchell, Patrick Lincoln, Dan Boneh, John Mitchell, and Mark Horowitz. 2000. Architectural Support for Copy and Tamper Resistant Software. *SIGARCH Comput. Archit. News* 28, 5 (Nov. 2000), 168–177. https://doi.org/10.1145/378995.379237

[30] Thomas Unterluggauer, Mario Werner, and Stefan Mangard. 2018. MEAS: memory encryption and authentication secure against side-channel attacks. *Journal of Cryptographic Engineering* (25 Jan 2018). https://doi.org/10.1007/s13389-018-0180-2

[31] Andrew Waterman, Yunsup Lee, Rimas Avizienis, David A. Patterson, and Krste Asanovic. 2016. *The RISC-V Instruction Set Manual Volume II: Privileged Architecture Version 1.9*. Technical Report UCB/EECS-2016-129. EECS Department, University of California, Berkeley. http://www2.eecs.berkeley.edu/Pubs/TechRpts/2016/EECS-2016-129.html

[32] Andrew Waterman, Yunsup Lee, David A. Patterson, and Krste Asanovic. 2014. *The RISC-V Instruction Set Manual, Volume I: User-Level ISA, Version 2.0*. Technical Report UCB/EECS-2014-54. EECS Department, University of California, Berkeley. http://www2.eecs.berkeley.edu/Pubs/TechRpts/2014/EECS-2014-54.html

[33] Chenyu Yan, Daniel Englender, Milos Prvulovic, Brian Rogers, and Yan Solihin. 2006. Improving Cost, Performance, and Security of Memory Encryption and Authentication. In *Proceedings of the 33rd Annual International Symposium on Computer Architecture (ISCA '06)*. IEEE Computer Society, Washington, DC, USA, 179–190. https://doi.org/10.1109/ISCA.2006.22

View publication stats