Hardware and Architectural Support for Security and Privacy
(HASP'18), June 2, 2018, Los Angeles, CA, USA

SMARTS: Secure Memory Assurance of RISC-V Trusted SoC

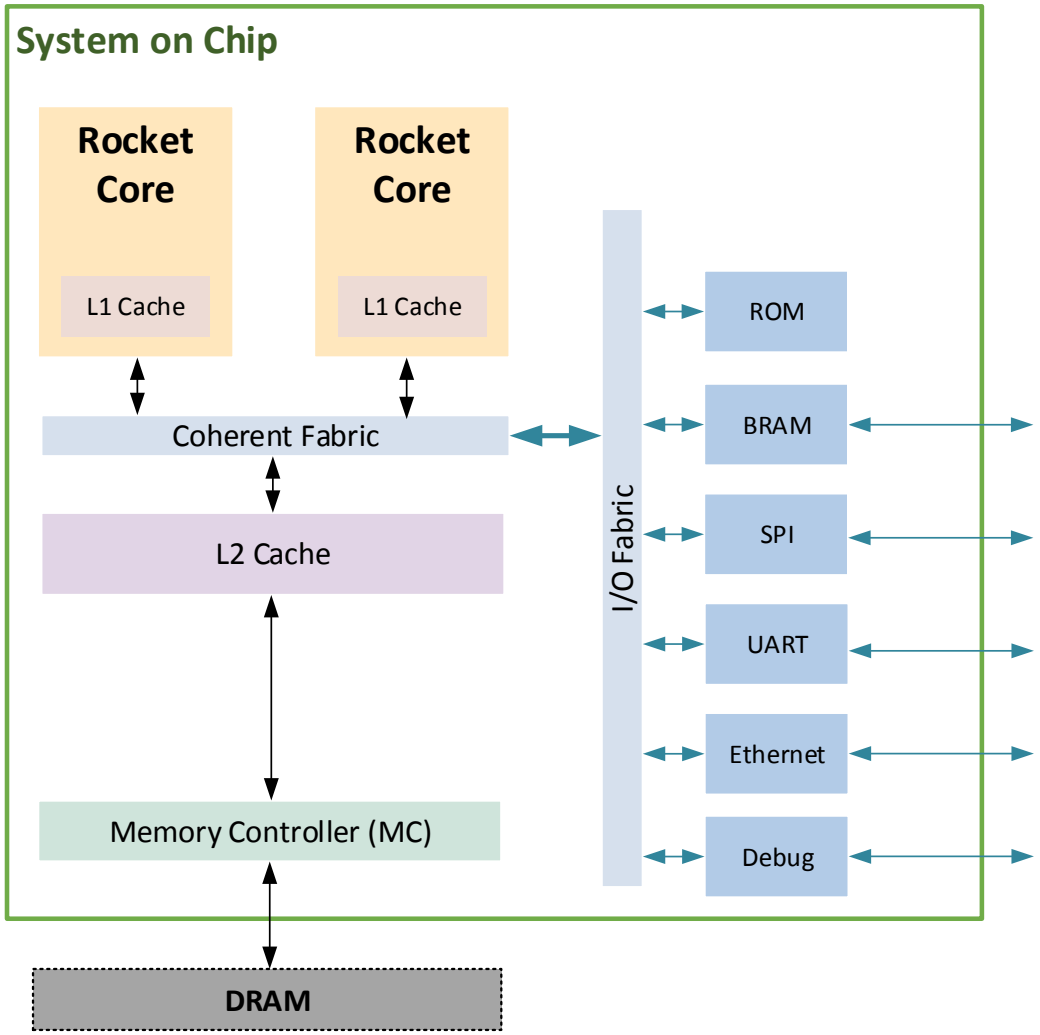
Ming Ming Wong
Jawad Haj-Yahya
Anupam Chattopadhyay

Computing and Engineering (SCSE)
Nanyang Technological University (NTU) Singapore

Introduction

Example Attacks on DRAM
- Side channel attacks (e.g. IO probing)
- Rowhammer
- Buffer overflow

Essential component in modern **secure processor**
- Memory protection via cryptographic primitives
- Integrated as autonomous hardware

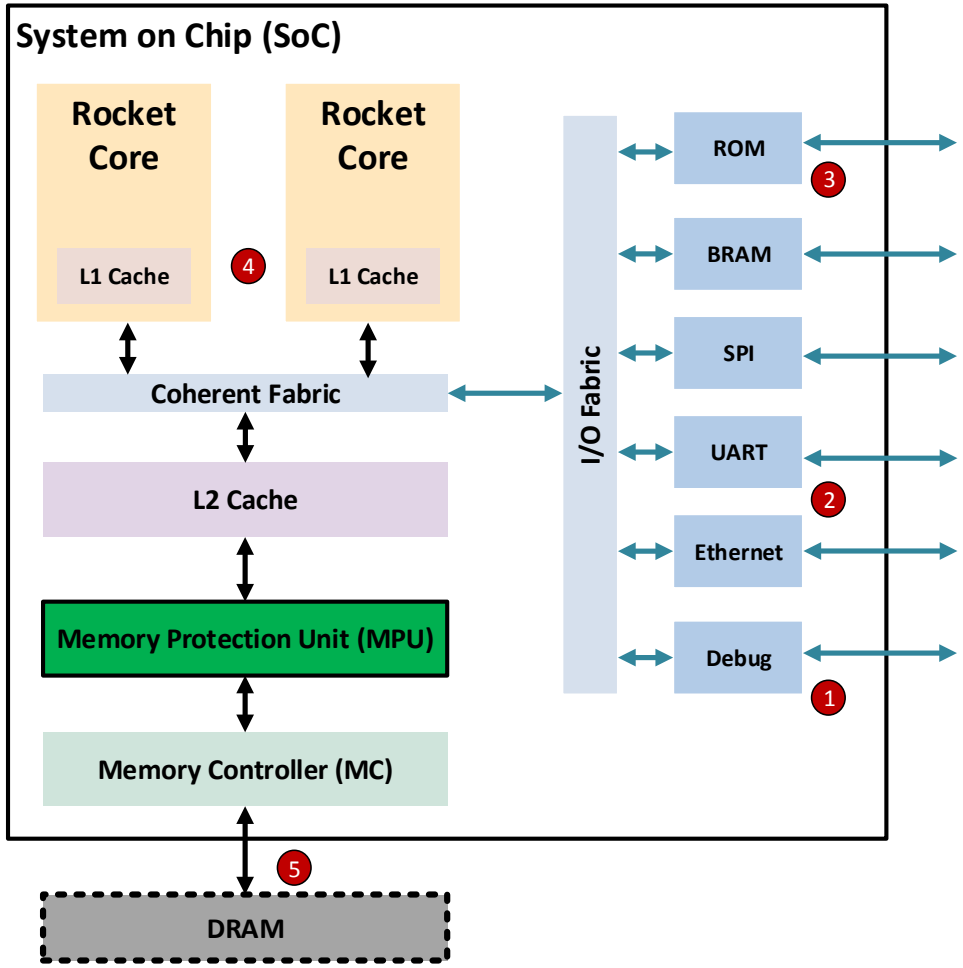


Introduction

- Our team is building a **secure processor based on RISC-V**
- Memory protection unit (MPU):

1. Authenticated Encryption
2. Simplified integrity tree
3. Supports for partial memory encryption (PME)

A primer on our RISC-V secure processor



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).

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.

Threat Model/Requirement

- Malicious modifications that compromise the memory:
  - **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.
  - **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.

Threat Model/Requirement (Cont.)

- Aim of MPU is to provide ***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.

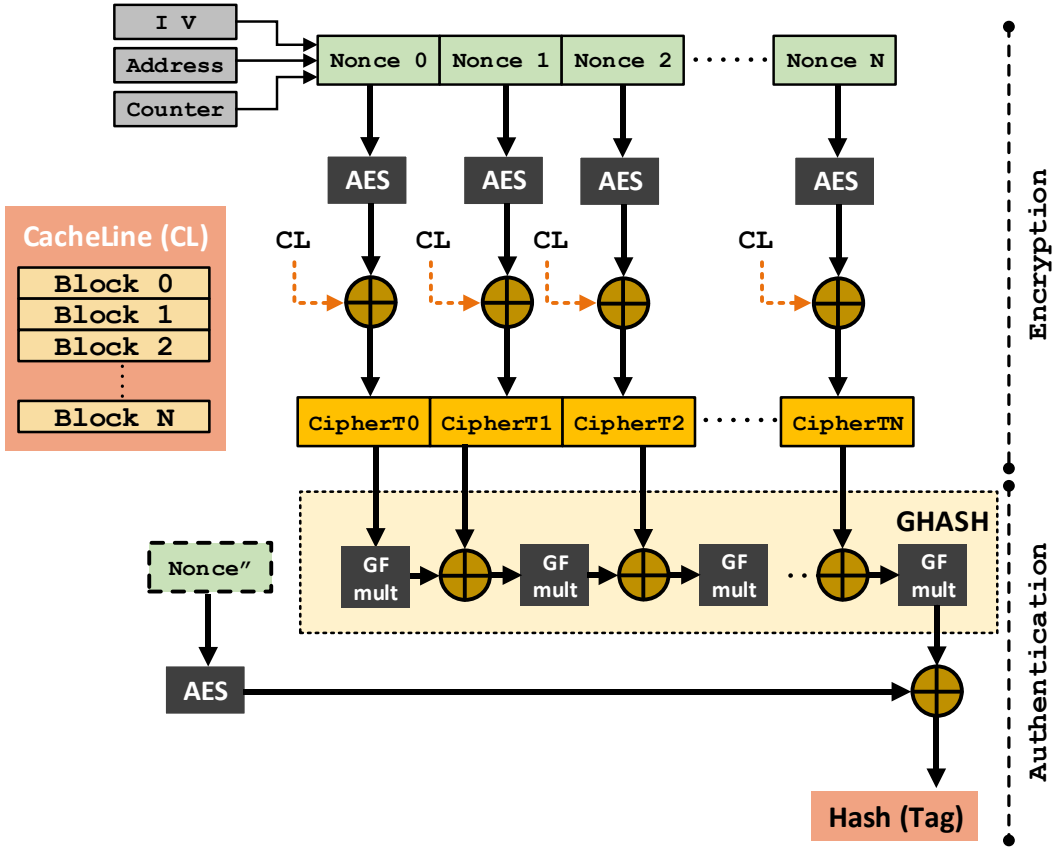
Memory Encryption and Integrity Verification

- **Memory encryption**
  - mainly concerned with the confidentiality of data and code during execution
  - deployed using ***symmetric key encryption***
- **Memory authentication (i.e. integrity verification)**
  - needed to prevent hardware attacks that may compromise data integrity and also insusceptible against replay attacks.
  - deployed the ***keyed Message Authentication Code (MAC)*** using the block address information and counter (prevent replay attack).
  - deployed ***integrity tree*** is favourable option as the tree's root is stored on the secure on-chip memory storage

Authenticated Encryption

- A shared-key based transformation, to the message to provide both encryption and authentication.
- **[Encryption]** ***ciphertext*** is derived by encrypting the plaintext with a secret key
- **[Decryption]** same secret key and the ciphertext are used to obtain either the *original plaintext* or an indicator to verify the *authenticity* of the ciphertext.
- Hardware cost saving since the encryption and authentication can share a part of the computation
- Recommended modes for AE:
  - Counter with Cipher Block Chaining Mode (CCM)
  - *Galois Counter Mode (GCM)*
- NIST's recommendation: AES cipher combined GCM to form **AES-GCM**

AES GCM used at MPU



Encryption and authentication with AES-GCM

AES-GCM is a ***counter-based encryption scheme*** which also provides data authentication.

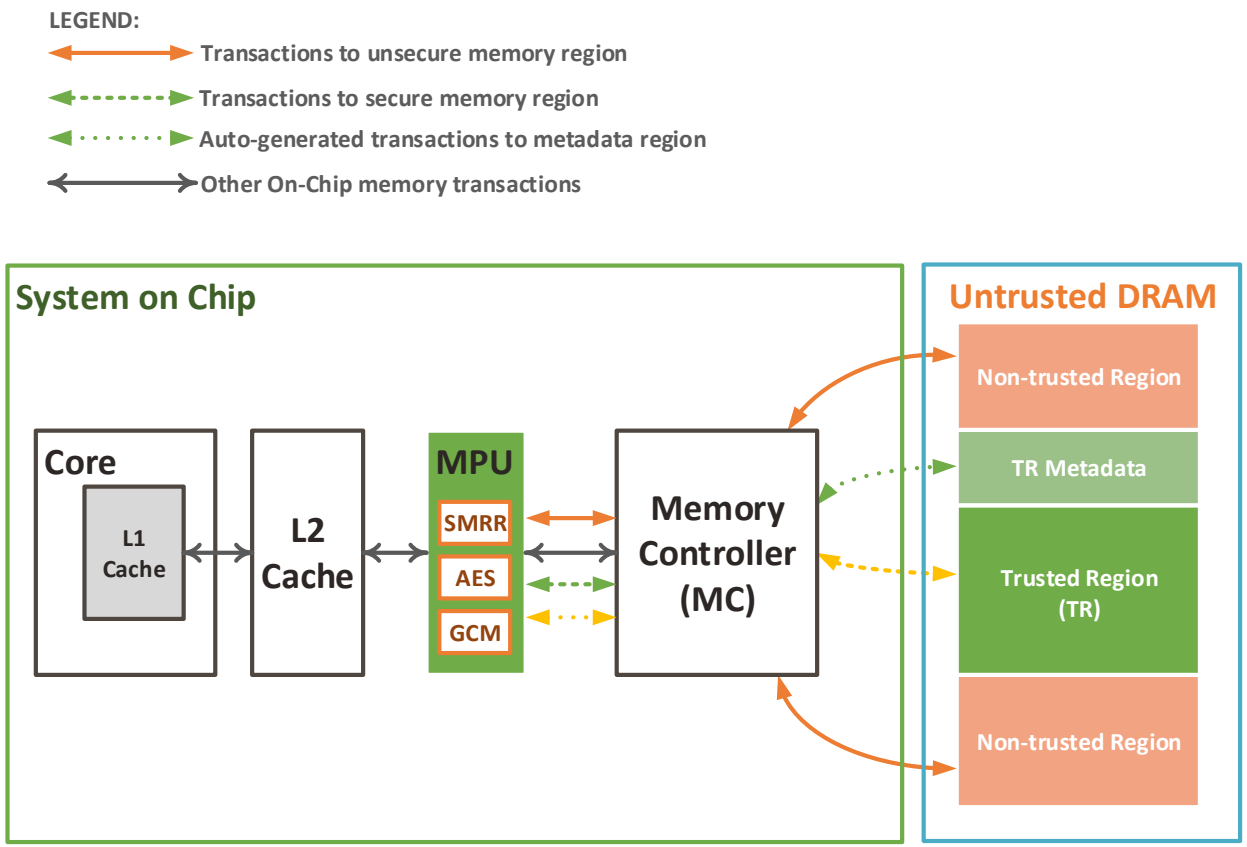
[Encryption] operates as a standard counter mode where sequence of pads is generated from a ***nonce*** and ***XORed*** with ***plaintext*** to produce the ***ciphertext***.

[Decryption] is **identical** to encryption, except that the plaintext and ciphertext are swapped.

AES GCM used at MPU (Cont.)

- Benefit of AES-GCM for RISC-V SoC Implementations:
  - GCM authentication portion has ***been proven to be as secure as*** the deployed cipher, which is the ***AES***.
  - GCM authentication can be ***largely overlapped with memory latency***. With that, program performance is not severely affected.
  - Both the encryption and authentication portions shares the same AES hardware and this offer an ***optimal cost-effective solution***

Reconfigurable Trusted Memory Region



Dynamic and reconfigurable DRAM memory region allocation.

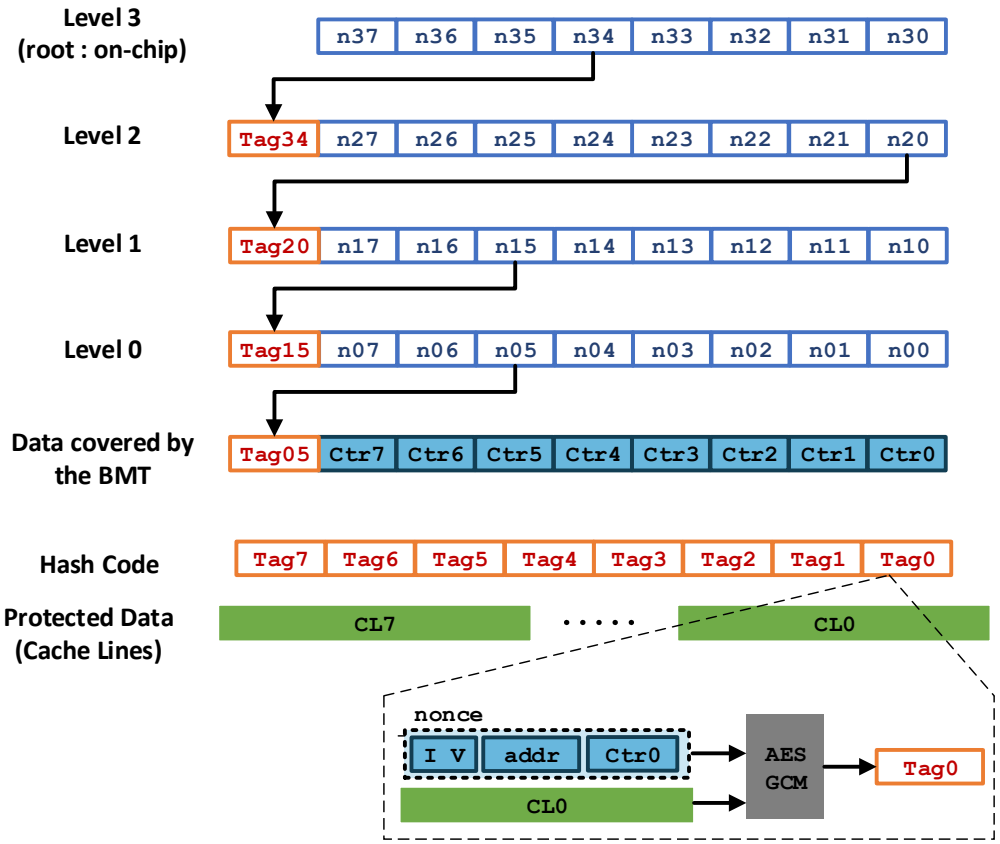
Partial Memory Encryption (PME) feature:

DRAM allocation is freed and repartitioned into finer granularity regions: ***trusted region***, ***non-trusted region*** or ***medata region***

These regions allocation is protected from interception and modification by software.

Data memory will be ***mapped*** to the allocated region according to their ***memory address***.

MPU Framework for Lightweight SoC
Integrity Tree for Memory Authentication



Branch of the Bonsai Merkle Tree (BMT)

Efficient implementation Bonsai Merkle Tree (BMT):

MAC over every *memory cache line* (CL) with a *nonce*.

Counter (Ctr) concatenated with the data *address* (addr) and *Initialization Vector* (IV) as extra input of the MAC function:

MAC = MACK (CL, {IV || addr || Ctr })

[Advantage]

BMT is applied to build the ***integrity tree over the Counters*** instead over the CL data

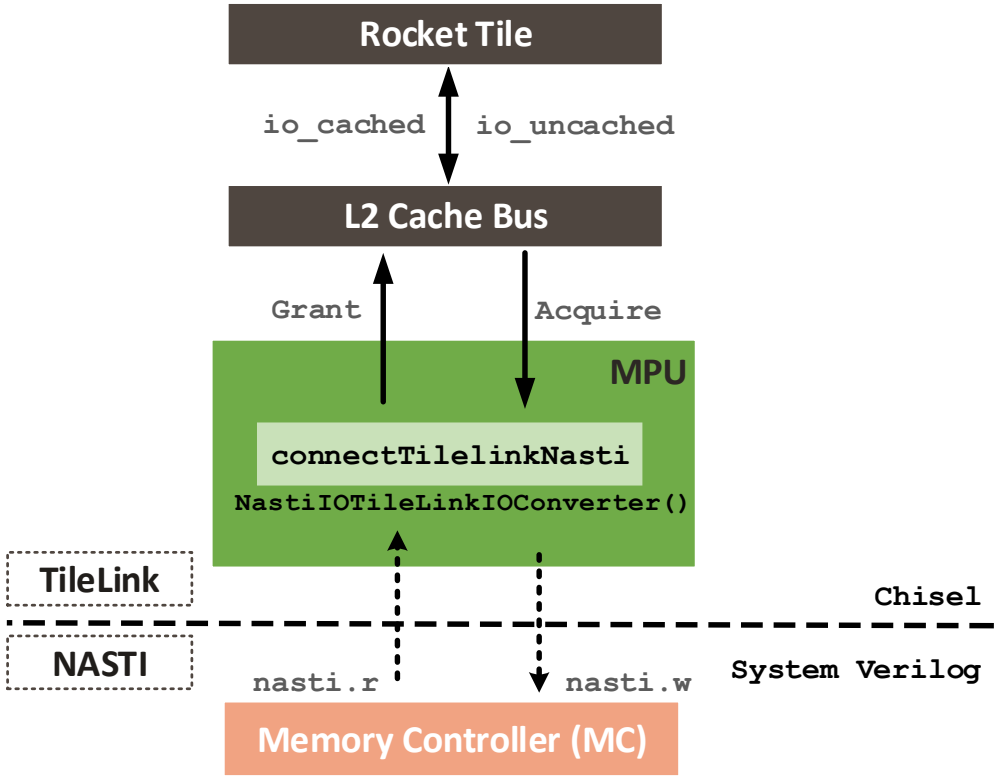
MPU Framework for Lightweight SoC
Integrity Tree for Memory Authentication

- The ***specific instantiation*** of the MPU data structure is defined by setting
  - Cache Line: 512 bits
  - Counter: 56 bits
  - Tag (hash code): 64 bits.

- We built an ***8-ary*** tree over the cache lines' Counters;
- ***Each cache line***, ***eight Counters*** and ***one Tag*** are stored.
- The infrastructure supports ***128MB of secure memory region***.

- A tree of 6 levels, ***the root (top level)*** will be securely stored in the ***on-chip memory*** and
- the ***other levels*** stored inside the ***main memory*** at the metadata section.

MPU Integration in RISC-V Rocket
Bus Interface Conversion



Bus Interface for MPU in RISC-V Rocket.

Rocket core uses ***TileLink internal buses*** to connect tiles, caches components.

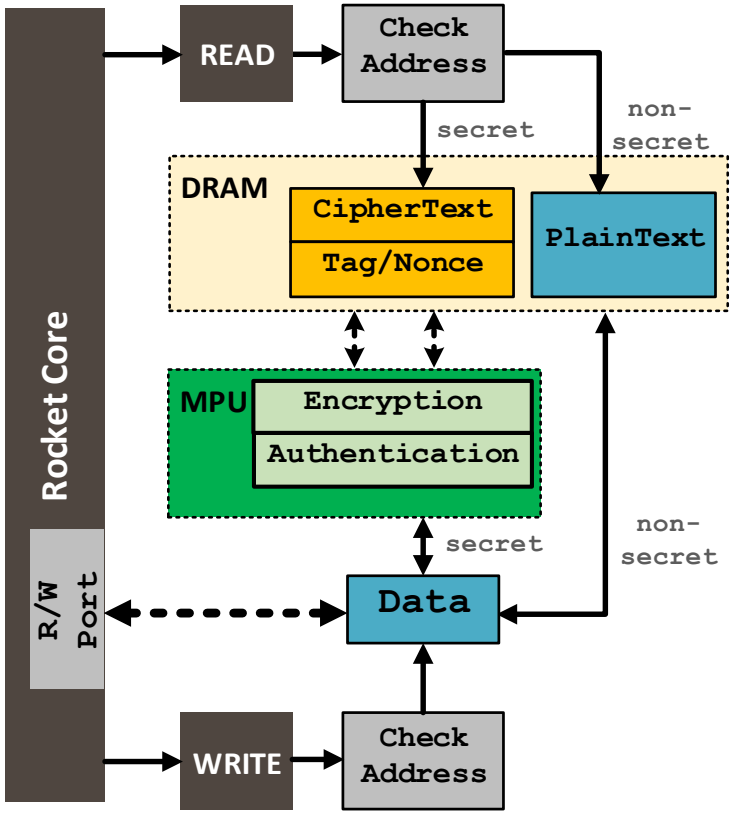
A custom bus interface, ***NASTI/NASTI-Lite*** is deployed (UC Berkeley implementation of AXI-Lite) for the ***external buses*** and as Rocket interface for all the IO devices.

Therefore, the interconnection between the cache, MC and DRAM involves ***interface convertor*** connectTilelinkNasti that requires NastiIOTileLinkIOConverter().

MPU Integration in RISC-V Rocket
Bus Interface Conversion

- For effective alignment and efficient performance, the *cryptographic primitives* are place in ***between or within*** the *connector*.
- **[Unencrypted side],** translation of the memory request, decoding write request and encoding read request are performed.
- **[Encrypted side],** support for memory read/write operations that is integrated with authenticated encryption schemes is provided.

MPU Integration in RISC-V Rocket
Memory Request Translation



Write/Read memory request flow.

Every **memory request** from *TileLink* io.tl.acquire.valid will invoke AE computation and metadata generation before storing/fetching to DRAM via NASTI buses.

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).

READ request:
the incoming *NASTI* responses (io.nasti.ar.bits) go through AE and aggregated into TileLink Grants.

Characteristic Analysis and Overview

- **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 for real world benchmarks (SPEC2006).
- **Storage overheads** are observed in terms of ***the additional memory allocated*** for the ***metadata*** in

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

Characteristic	Our Work	XOM	AEGIS	Split Counter	SecureMe	AISE
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

Conclusion

New memory protection unit (MPU) implemented into 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.
- 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