Verifying RISC-V Physical Memory Protection

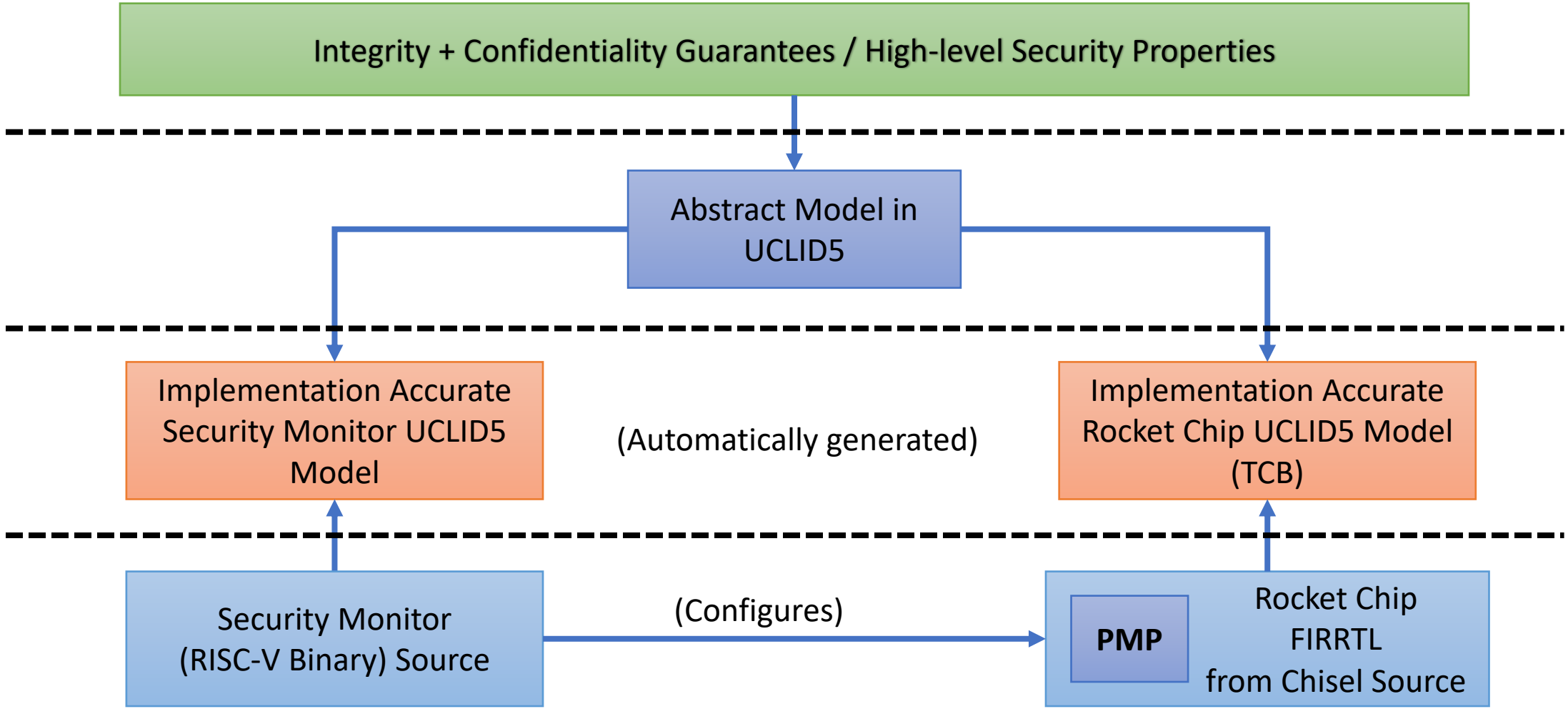
By Kevin Cheang, Cameron Rasmussen, Dayeol Lee, David W. Kohlbrenner, Krste Asanovic, Sanjit A. Seshia

First International Workshop on Secure RISC-V Architecture Design Exploration (SECRISC-V'20)

Motivation

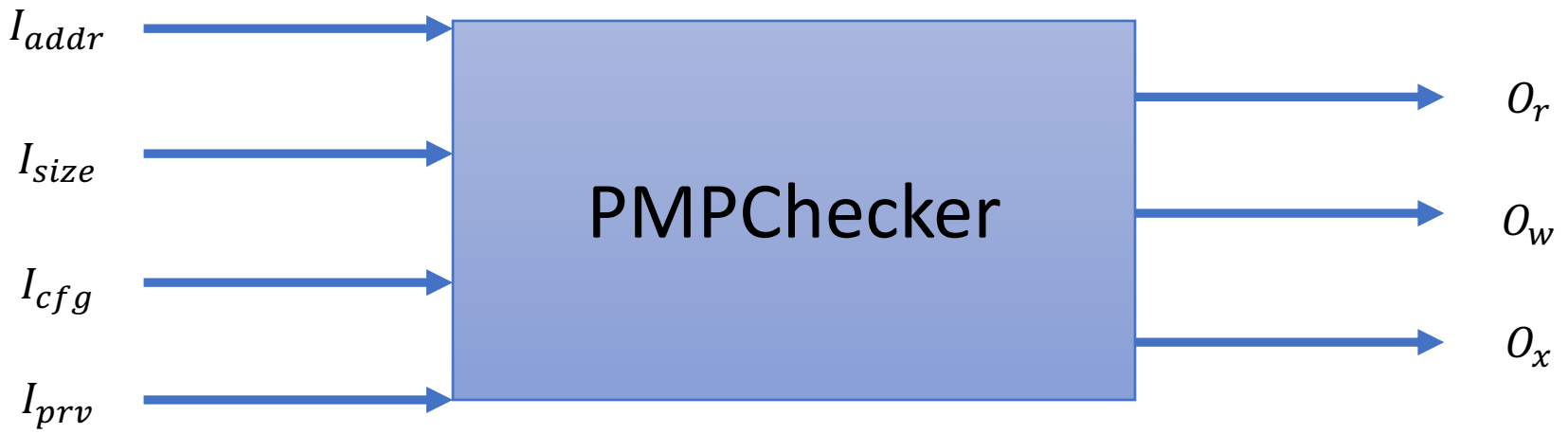
- Physical memory protection (PMP) is a standard RISC-V feature that allows firmware to specify **physical memory regions**
  - Controls memory access permissions
  - E.g. Used in Keystone's security monitor for memory isolation
- Systems that use the PMP feature depend on strong security guarantees provided by the PMPChecker mechanism
  - E.g. Memory isolation via integrity and confidentiality properties
- First step to verify RISC-V's Keystone
  - Keystone is a platform for architecting trusted execution environments
  - Composed of hardware (Rocket Chip) and software (security monitor)

Motivation


Integrity + Confidentiality Guarantees / High-level Security Properties
Abstract Model in UCLID5
Implementation Accurate Security Monitor UCLID5 Model
(Automatically generated)
Implementation Accurate Rocket Chip UCLID5 Model (TCB)
Security Monitor (RISC-V Binary) Source
(Configures)
PMP
Rocket Chip FIRRTL from Chisel Source

Problem Statement

- Given an implementation of the PMP feature, how can we verify its correctness?



Main Contributions

1. Automated Verification Workflow for the PMPChecker

2. Formal Specifications for the PMPChecker
   - Define PMP primitives
   - PMPChecker Functional Specifications

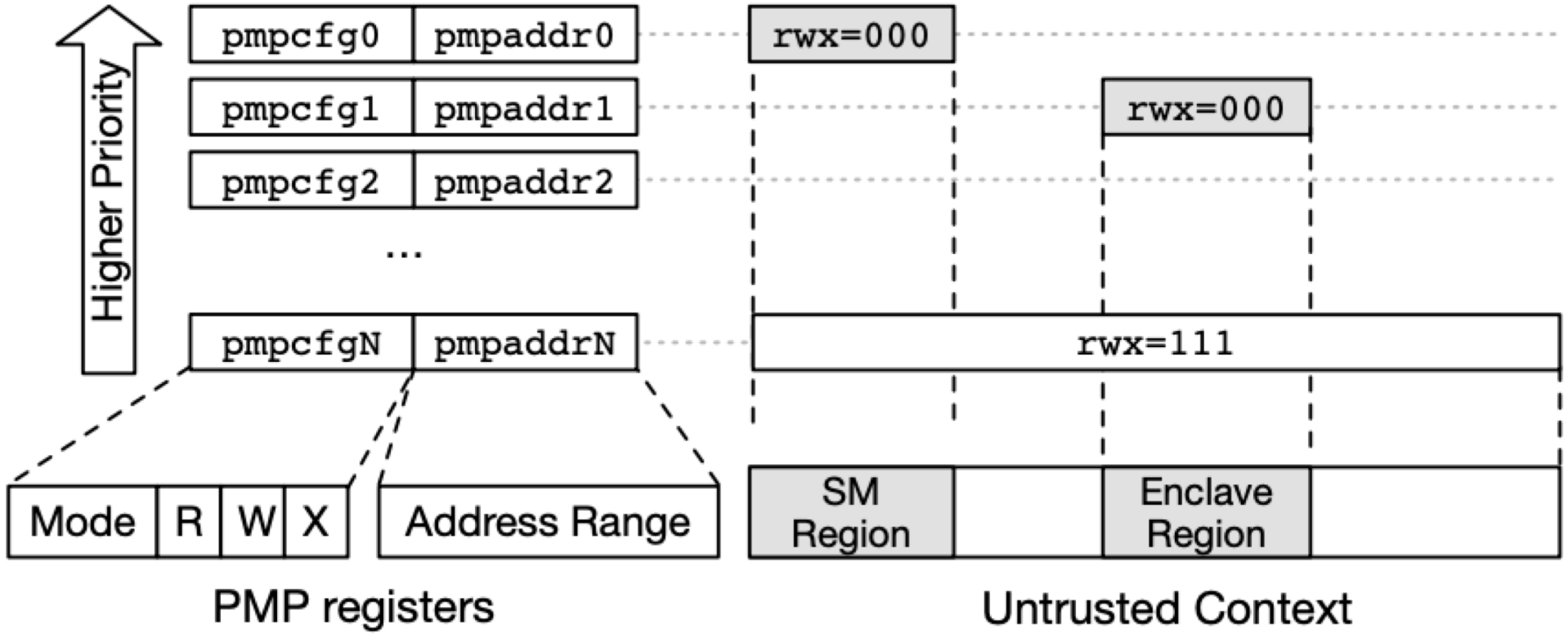
Related Work

- Existing implementations of enclaves like Intel's SGX lacks transparency
  - Formal correctness properties
  - High level security properties
- Non-commercial implementations of enclaves such as MIT's Sanctum need more assurance
  - No verification at the hardware level
- A Formal Foundation for Secure Remote Execution of Enclaves [CCS' 17]
  - Defines a trusted abstract platform
  - Does not extend to RISC-V's PMP

Background: PMP

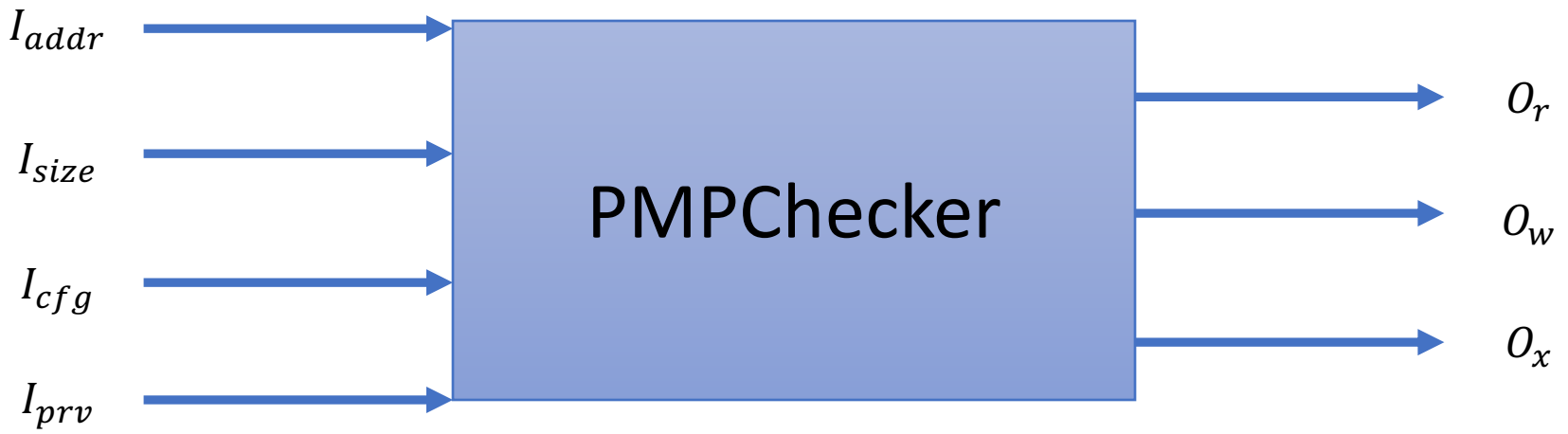
- PMP controls the access permissions to physical memory regions using a set of control status registers (CSR) in the RISC-V architecture
- Each core may have 0-16 PMP registers
  - PMP Configuration
    - Addressing mode
    - Permission Bits
  - PMP Address
    - Address range (based on an addressing mode): NAPOT, TOR, NA4
- PMP entries act as a whitelist

Background: PMP



Background: PMPChecker

- The PMPChecker is the module (written in Chisel) that is used to check whether the memory access is permitted



Formal Specification

- Input and Output Definitions
  - $I_{addr} \in \{0,1\}^N$: address to the PMPChecker, where $N = XLEN$
  - $2^{I_{size}},\ I_{size} \in \{0,1\}^2$: size of the memory access
  - $I_{cfg} \in \{l, x, w, r\}$: struct of 1-bit variables, where l, x, w, r are the lock, read, write, and execute permission bits
  - $I_{prv} \in \{0,1\}^2$: privilege mode
  - $O_r \in \{0,1\}$: read permission output bit
  - $O_w \in \{0,1\}$: write permission output bit
  - $O_x \in \{0,1\}$: execute permission output bit

Formal Specification

- PMP Primitive Functions
  - A: set of addresses
  - $r'(addr, i) \mapsto \{0,1\}$: predicate that determines if address $addr$ is contained in the i-th region
  - $r'_{lo}(i) \mapsto A$: returns the low address boundary for the i-th region
  - $r'_{hi}(i) \mapsto A$: returns the high address boundary for the i-th region
  - $a'(addr, i) \mapsto \{0,1\}$: predicate that determines if $addr$ is aligned to the i-th region's addressing mode

  - $r(\mathrm{addr}, \mathrm{i})$: predicate that determines if the address is between the low and high boundaries of the region
  - $a(addr, i)$: predicate that determines if the address is within the region's range (as defined by r) then so should the last byte be

Formal Specification

- Primary PMP Functional Property
  1. If the address is not contained in any region, return the default permission bits
     1. High privilege => full permissions
     2. Low privilege => no permissions
  2. If we are operating in high privilege mode
     1. And if the region **is not locked** => full permissions
     2. If the region **is locked** => permissions according to PMP CSRs
  3. Deny accesses that exceed the regions boundaries

SEC RISC-V Workshop 2020

Formal Specification

- Primary PMP Functional Property

$$(\mathcal{I}_{prv} = low) \Rightarrow$$

SEC RISC-V Workshop 2020

Formal Specification

1. If the address is not contained in any region, return the default permission bits
    1. High privilege => full permissions
    2. Low privilege => no permissions

$$\forall addr \in \mathcal{A}, \neg(\exists i \in \mathcal{N}, r(addr, i)) \Rightarrow$$

SEC RISC-V Workshop 2020

Formal Specification

2. If we are operating in high privilege mode
    1. And if the region **is not locked** => full permissions
    2. If the region **is locked** => permissions according to PMP CSRs
3. Deny accesses that exceed the regions boundaries

$$(\mathcal{I}_{prv} \neq low) \Rightarrow$$

SEC RISC-V Workshop 2020

Evaluation

- Verified the PMP FIRRTL implementation from Rocket Chip core
  - By encoding the functional correctness of the PMPChecker
  - Verified using UCLID5 Model Checker

Evaluation

- Chisel implementation of the PMPChecker contains 48 LoC
- UCLID5 Model contains 1125 LoC
- Verifying the problem using UCLID5
  - Z3 SMT solver as backend
  - 1-step induction
  - 41.331s on average
  - 2.6 GHz Intel Core i7 machine with 16 GB RAM on OSX
- Revealed that we missed specifying an unimplemented feature (hypervisor mode)

Future Work

- Rocket core enforces PMP rules using multiple hardware components
  - PMPChecker
  - Translation look-aside buffer (TLB)
  - Page table walker (PTW)
- Higher-level properties such as memory isolation relies on software
  - E.g. Keystone's security monitor

Summary

- Provided a formal specification of the PMPChecker
- Introduced and implemented a workflow using the Chisel generator and LIME transpiler to automatically generate a model of the PMPChecker
- Verified the functional property of the PMPChecker

SEC RISC-V Workshop 2020

Questions?