# Enabling Design Space Exploration for RISC-V Secure Compute Environments

Ayaz Akram
yazakram@ucdavis.edu
Dept. of CS, UC Davis

Venkatesh Akella
akella@ucdavis.edu
Dept. of ECE, UC Davis

Sean Peisert
sppeisert@lbl.gov
Dept. of CS, UC Davis & Lawrence Berkeley National Lab

Jason Lowe-Power
jlowepower@ucdavis.edu
Dept. of CS, UC Davis

## ABSTRACT

Cycle-level architectural simulation of Trusted Execution Environments (TEEs) can enable extensive design space exploration of these secure architectures. Existing architectural simulators which support TEEs are either based on hardware-level implementations or abstract analytic models. In this paper, we describe the implementation of the gem5 models necessary to run and evaluate the RISC-V-based open source TEE, Keystone, and we discuss how this simulation environment opens new avenues for designing and studying these trusted environments. We show that the Keystone simulations on gem5 exhibit similar performance as the previous hardware evaluations of Keystone. We also describe three simple example use cases (understanding the reason of trusted execution slowdown, performance of memory encryption, and micro-architecture impact on trusted execution performance) to demonstrate how the ability to simulate TEEs can provide useful information about their behavior in the existing form and also with enhanced designs.

## 1 INTRODUCTION AND BACKGROUND

Software simulators serve as the first level of the "agile hardware design stack" [7]. These architectural simulations are useful to iterate on high-level architectural tradeoffs and hardware/software co-design before focusing on the hardware implementation (e.g., RTL). Considering the importance of security in computing architectures, it is imperative to enable architectural simulation of hardware for accelerating security such as Trusted Execution Environments (TEEs) [16]. Cycle-level modeling of these TEE-based architectures can provide useful insights and help in early design space exploration.

As the community has recently seen, the microarchitectural implementation affects both performance and security [5, 9, 11]. To co-design the hardware and software security platforms, we need a microarchitectural simulation infrastructure which can be used to evaluate *both* performance and security while providing flexibility to modify all levels of the stack in an agile way. While functional-level modeling tools like QEMU [2] can be very fast, they do not provide detailed timing information. On the other extreme, cycle-exact models (RTL models) can provide very detailed and accurate timing information, but they are difficult to modify and very slow to simulate. These RTL models are accelerated using FPGAs (e.g., FireSim [8]), but they are still inflexible.

In this work we focus on RISC-V based TEEs (Keystone [10] for example) and a widely-used architectural simulator gem5 [12]. Currently, the RISC-V ecosystem provides support to perform functional or RTL level simulation of RISC-V TEEs using QEMU [2] or FireSim [8]. However, there is no tool or simulator available to do high-level architectural and microarchitectural studies of RISC-V TEEs at a cycle-level for early design space exploration. Until now, researchers have had to rely on analytical modeling or RTL implementations for their studies involving Keystone [18].

Keystone is proposed as a "customizable" TEE. To fully avail the benefits of this customizability, simulation support is needed so that the customized designs can be fully evaluated. A quote from the original Keystone paper [10]:

> "We advocate that the hardware should provide security primitives instead of point-wise solutions... This motivates the need for Customizable TEEs – an abstraction that allows entities that create the hardware, operate it, and develop applications to configure and deploy various TEE designs from the same base..."

In this paper, we describe the models and extensions to gem5 necessary to simulate Keystone on RISC-V to enable quantifying the impact of these *customizable* TEE designs and evaluate hardware/software co-design. With this simulation support, it is easy to pick a design and analyze how it applies to different architectures. We extend gem5's recent full system RISC-V support to also implement the physical memory protection (PMP) hardware that Keystone uses to enforce hardware isolation. We also build and test both the Berkeley Boot Loader (BBL) and OpenSBI as well as an unmodified version of Keystone's runtime, Eyrie, on gem5. We show that the Keystone simulations on gem5 exhibit similar performance trends as in the work of Lee et al. [10]. Moreover, we present three example use-cases of this work: understanding the trusted execution slowdowns, understanding the performance of hardware support for memory encryption, and understanding the impact of different microarchitectures on performance of trusted and untrusted execution on a given set of benchmarks.

The rest of this paper is organized as follows: we provide a background on Keystone, and RISC-V support in gem5 in Section 2. Section 3 discusses Keystone's support in gem5, and Section 4 evaluates this support. Section 5 presents different use cases of the


*CARRV '21, June 17, 2021, Virtual*

ability to simulate TEEs in gem5 and Section 6 discusses different possibilities this work has opened up. We conclude in Section 7.

# 2 BACKGROUND

## 2.1 RISC-V Isolation Mechanisms

***RISC-V Privileged Modes:*** RISC-V provides three privilege levels to maintain vertical isolation. The least privileged mode of execution is user-mode (U-mode), where normal user applications operate. The next level is supervisor-mode (S-mode), at which operating system operates. The most privileged mode is machine-mode (M-mode) at which software like boot-loader operate. M-mode can also be considered parallel to "system management mode" in x86 architecture.

***RISC-V PMP (Physical Memory Protection):*** RISC-V's PMP feature controls access of U (user) and S (supervisor) mode to certain memory regions. The allowed access (*r-w-x*) permissions and the memory region can be configured using a set of PMP address (*pmpaddr*) and configuration registers (*pmpcfg*). These registers together constitute a PMP entry. PMP supports three addressing modes which interpret the *pmpaddr* registers differently to determine the size of a PMP region. PMP entries can configure PMP regions of arbitrary sizes (from 4 bytes to entire DRAM). Lower numbered PMP entries are given priority over higher number PMP entries. PMP entries define an *allow list* and every U/S mode access needs to fall in some PMP range, otherwise an access fault is raised.

***Virtual Memory Management:*** RISC-V provides different schemes for virtual memory management namely Sv39 (3 level page tables), Sv48 (4 level page tables) for 64-bit systems. Virtual memory is managed by the S-mode software (e.g., the OS). The ISA provides a CSR, *satp* (supervisor address and translation register), to control address translation, which contains three different fields: *MODE* (mode of address translation), *ASID* (address space identifier), and *PPN* (physical page number of page table's root page).

## 2.2 RISC-V based TEE – Keystone

Most of the existing TEEs (e.g., SGX and SEV) are closed source and it will be impossible to simulate all of their implementation details correctly. Therefore, we focus on Keystone [10] in this work, as it is an open source RISC-V based Trusted Execution Environment (TEE). Keystone is proposed as a customizable and modular TEE, which allows fine-grained TCB (trusted compute based) configuration. Keystone relies on PMP (physical memory protection) to provide memory protection.

***Keystone Security Monitor (SM):*** Keystone decouples resource management and security checks by relying on a security monitor (SM), which is M-mode (the most privileged execution mode in RISC-V) software, to enforce all security guarantees. The security monitor also manages the isolation boundary between the enclave and the untrusted OS. By relying on this M-mode software, Keystone enables programmability and agile patching in case bugs/vulnerabilities need to be fixed. When the SM boots, the first PMP entry is configured to be used by SM. Since the first PMP entry has the highest priority, it allows SM to stay inaccessible by other components (operating in U/S mode). The SM also configures the last PMP entry to cover all memory and configures the permissions of this entry to allow OS complete access to it.

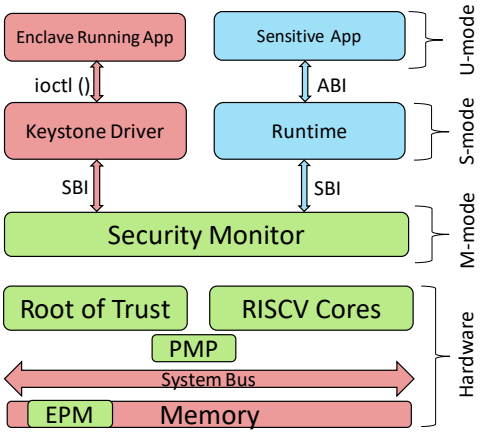


**Figure 1: High level overview of a Keystone based secure compute environment**

***Enclave:*** An enclave, in the context of Keystone, contains a sensitive user-mode application (eapp), and a supervisor mode runtime (e.g. Eyrie, seL4). Keystone uses this runtime to manage enclave application's resources like virtual memory, system call/trap handling.

There are three steps involved in an enclave's life-cycle (Figure 1 can help to understand the components that are involved in managing this life cycle):

***Enclave Creation:*** When an untrusted host (through a test runner application) calls the SM to create an enclave for a user application (we are assuming a use case where an entire unmodified application is going to execute in an enclave), the SM first allocates a region of memory (referred as enclave private memory, EPM), which is protected and isolated with the help of a PMP entry. This region of memory is also initialized with enclave's page table, the S-mode runtime that the application will be using and the application itself. At the time of creation of an enclave, the PMP status is also propagated to other cores so that EPM is kept inaccessible by other cores in the system. At the time of creation of an enclave, the SM may also validate the initial state of an enclave.

***Enclave Execution:*** When the host calls the SM to execute an enclave on a core, the SM will flip the permissions of a PMP entry to make EPM accessible by that core while the core is executing the enclave. SM makes sure that the PMP permissions are set correctly whenever a core exits/enters the enclave.

***Enclave Destruction:*** When a host requests the SM to destroy an enclave, the SM clears the memory region (EPM) belonging to that enclave and also frees the PMP entry corresponding to this enclave.

## 2.3 gem5 and RISC-V

The gem5 simulation framework provides a wide variety of models of processors (single-cycle, in-order, out-of-order), cache subsystems, and memories [3, 12]. It supports different ISAs and two main modes of execution, SE-mode (*System Emulation*) and FS-mode (*Full System*). In SE mode, gem5 emulates the simulated program's system calls, thus obviating the need to simulate an operating system. gem5's FS mode allows booting an unmodified operating system

which can then run applications as they would on a normal operating system.

The initial RISC-V ISA support in gem5 by Roelke et al. provided ability to simulate RV64G in SE mode for single-core systems [15]. Ta et al. provided support for RISC-V synchronization instructions, thread related syscalls (e.g., futex, clone), and atomic operations to enable simulation of multi-core systems in gem5 [17].

Later, main parts of the privileged RISC-V ISA were implemented in gem5 to provide full system support for RISC-V [12]. The full-system RISC-V in gem5 supports Sv39 paging (39-bit virtual address space), three-level page table walks and TLB accesses. This provided support to boot a bare metal OS. Recently this support has been extended to allow (unmodified) RISC-V Linux kernel booting on gem5. This ability to simulate RISC-V Linux was made possible by the addition of CLINT (Core Local Interrupt Controller) and PLIC (Platform Level Interrupt Controller) devices along with the ability to generate device tree needed by the boot loader. Both BBL[1] (Berkeley Boot Loader) and OpenSBI[2] (Open Source Supervisor Binary Interface) boot-loaders have been tested to run on gem5.

## 3 KEYSTONE IN GEM5

In this work, we extended the privileged ISA support to add RISC-V PMP (physical memory protection) hardware in gem5 which enables running Keystone's Security Monitor (SM) on gem5. Figure 2 provides an overview of the PMP implementation in gem5. There are three components which interact with each other: the ISA subsystem, the MMU unit, and the PMP unit. Any read of the PMP registers returns the registers' value, and writing to a PMP register (eventually) triggers a call to update the PMP rules which are maintained in a PMP table (set of PMP entries). When a memory access is made and the MMU (TLB or page table walker) has generated the physical address corresponding to a program (virtual) address, a call is made to PMP unit to detect if a PMP check should be made or not (depending on the current mode of execution) [19]. If a check is desired, the PMP table is consulted to find out if there is an entry match/mismatch for both address and the permissions. If no match is found a fault is raised, otherwise control returns back to MMU with a successful check.

Keystone's SM is shipped as a part of both BBL and OpenSBI bootloaders. We have tested both bootloaders with the SM on gem5. We further set-up all Keystone components for simulation on gem5 and performed different tests to check the validity of runs. The components include:

- Bootloader (OpenSBI)
- SM (compiled as a part of OpenSBI)
- Linux kernel (compiled with OpenSBI)
- Keystone driver
- Benchmarks/tests with a test runner application
- Buildroot based disk image

Detailed instructions on how to build these components and use them with gem5 are provided in https://github.com/darchr/Keystone-experiments. This repository also contains scripts to launch gem5 based Keystone experiments using gem5art [4] (a tool to run gem5 tests in a structured and reproducible way).

[1]https://github.com/riscv/riscv-pk
[2]https://github.com/riscv/opensbi

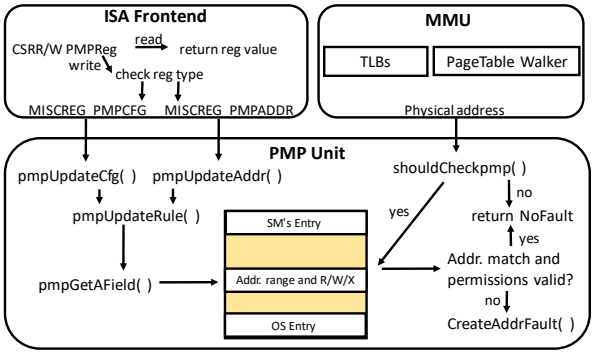


**Figure 2: PMP implementation in gem5**

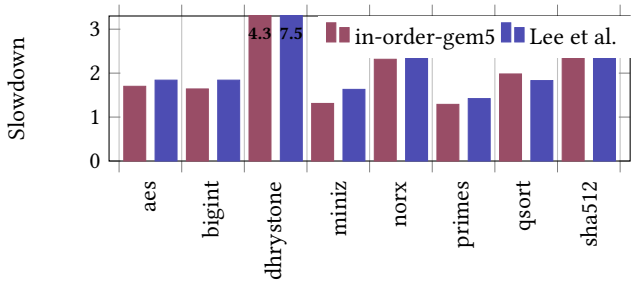


**Figure 3: Comparison of slowdowns (incurred by trusted execution using Keystone) between gem5 and Lee et al. [10]. This slowdown includes enclave creation and management time as well.**

## 4 VALIDATION

In this section, we validate and evaluate Keystone's implementation in gem5. We relied on the following actions for the functional validation of this implementation:

- We performed physical memory access checks using Linux Busybox utility to test functionality of PMP, which passed successfully.
- We successfully ran primary Keystone tests, which, in addition to performing some basic functionality tests, check if an enclave access control is violated or not.
- Finally, we successfully tested the workloads used by Lee et al. [10] and found similar performance results.

In addition to the functional validation, we also validated the performance of gem5's Keystone implementation. To investigate this, we performed some experiments and collected performance numbers for Keystone benchmarks on gem5 and compared them with the performance numbers published in the Keystone paper [10].

Figure 3 shows a comparison of the slowdown experienced from enabling trusted execution on two different gem5 CPU models, and the slowdown numbers taken from the work of Lee et al. [10] for rv8 benchmark suite [1]. This figure shows that the Keystone simulations on gem5 exhibit similar performance numbers and trends as in the work of Lee et al. [10]. The slowdown numbers shown in Figure 3 include benchmark execution as well as the enclave creation, destruction and management time. *dhrystone* which has the smallest execution time in normal (untrusted) execution shows

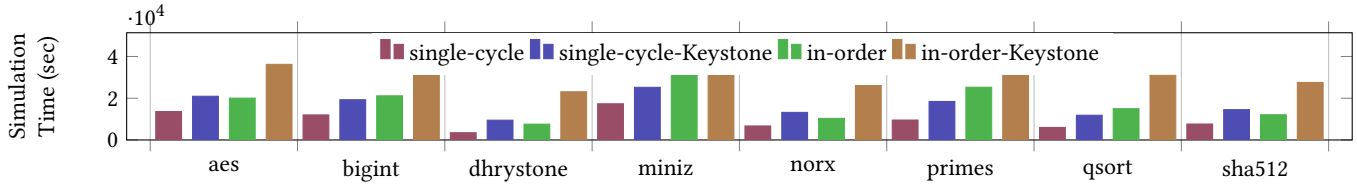


**Figure 4: Time taken by gem5 to simulate rv8 [1] benchmarks on a single cycle (TimingSimpleCPU) and an in order (MinorCPU) CPU models of gem5 with and without Keystone.**

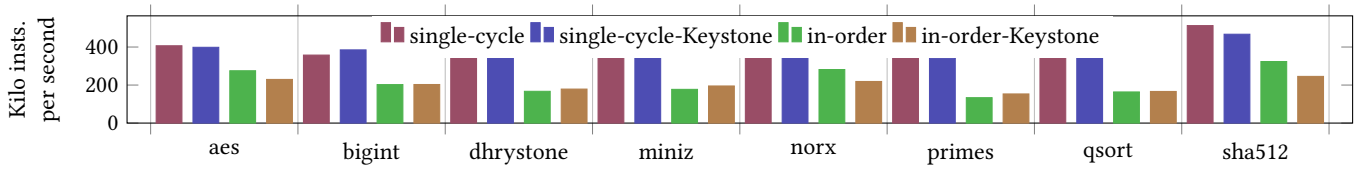


**Figure 5: Kilo simulated instructions per host second while simulating these benchmarks on a single cycle (TimingSimpleCPU) and an in order (MinorCPU) CPU models of gem5 with and without Keystone.**

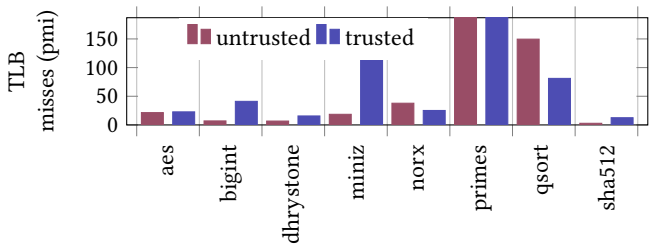


**Figure 6: Comparison for TLB misses (per million instructions) for untrusted and trusted gem5 runs.**

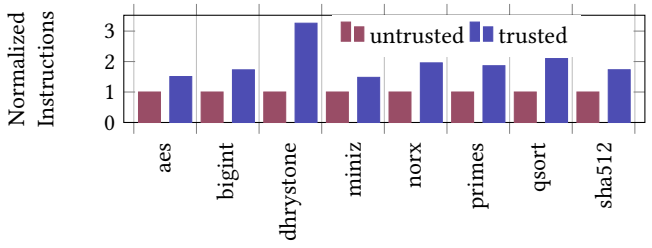


**Figure 7: Comparison of instructions for untrusted and trusted runs (normalized to untrusted count)**

the biggest overhead for trusted execution, because the cost of enclave creation, and management becomes more dominant due to its small execution time. Similar is the case for *sha*512 and *norx*, which have slightly higher execution time compared to *dhrystone*, but still relatively less in comparison to other workloads.

Figure 4 shows the performance of gem5 itself (i.e., the time taken by gem5 to perform a simulation). Simulating an in order CPU (called MinorCPU in gem5) takes more time in comparison to a single cycle CPU (called TimingSimpleCPU in gem5). It should be noted that the difference in simulation time of a trusted and untrusted execution is because of the difference in amount of instructions/work that is simulated. Figure 5 which shows thousand instructions (simulated) per second numbers, provides a more clear picture, where we can observe that gem5 simulates trusted or untrusted system with the same throughput. However, there can be a difference in throughput for different modeled CPUs.

## 5 USE CASES

In this section, we discuss some of the use cases of the ability to simulate secure compute environments.

### 5.1 Analyzing microarchitecture to understand performance

The gem5 simulator has the ability to produce detailed simulated performance statistics, which can help understanding the performance characteristics of secure architectures. Many of these statistics are hard to collect on the real hardware (e.g., number of cycles when certain pipeline stages are idle, number of squashed instructions due to mis-speculation, etc.).

In this section, we investigate the reason for slowdown of the trusted execution shown in Figure 3. One (potentially) expensive operation that Keystone performs during enclave context management is flushing the TLBs (translation look-aside buffers). So, one can expect that these flushes would lead to extra TLB misses when a workload is executing in trusted fashion. Figure 6 shows a comparison of the number of TLB misses (per million instructions) between untrusted and trusted runs. Though, there is a reasonable difference in the number of TLB misses between the two configurations, the actual number of TLB misses per million instructions seem to be low for these workloads (except *primes*). Figure 7 shows the total number of committed instructions for untrusted and trusted configurations (normalized to untrusted configuration). The difference in the number of instructions seem to correlate well with the slowdown numbers shown in Figure 3 This indicates that the main reason of the execution time difference between the trusted and

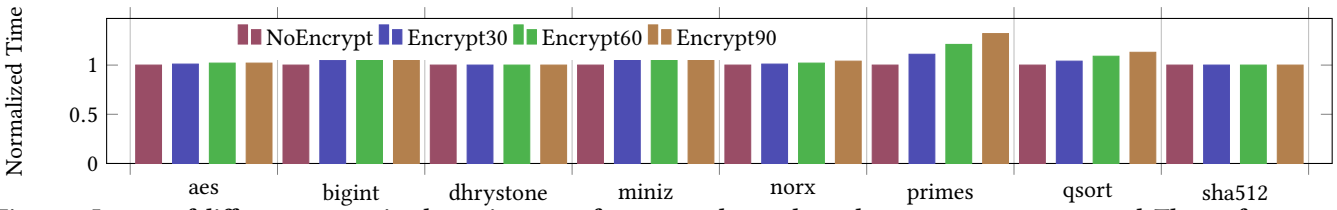


**Figure 8: Impact of different encryption latencies on performance when only enclave accesses are encrypted. The performance implications of single cycle and in order CPU are similar.**

untrusted configurations for these benchmarks is the difference in the number of executed instructions which corresponds to the extra work done to create, initialize, and manage the enclave. This ability to analyze a large set of architectural or microarchitectural events can open doors for security properties' analysis as well.

## 5.2 Performance of memory encryption

In this subsection, we present a use-case of adding new hardware structures for security and show the ability to study the potential impact of such structures.

We model a memory encryption engine, with different encryption/decryption latencies, which will encrypt cache blocks belonging to an enclave whenever they leave the last level cache. The blocks are decrypted on their way back to caches.

We assume direct mode encryption, which can expose the encryption latency to read accesses and can affect overall performance. One could imagine a part of this latency consumed for providing some kind of integrity protection as well. For example, a MAC can be computed for each block and can used to later authenticate the validity of that block. However, we do not model any integrity protection scheme in this work.

Figure 8 shows execution time for four different configurations: trusted with encryption disabled (NoEncrypt), trusted with encryption enabled and encryption latency of 30 cycles (Encrypt30), 60 cycles (Encrypt60), and 90 cycles (Encrypt90). As shown in Figure 8, the performance impact of encryption for rv8 benchmarks is mostly small. These experiments are performed on default gem5 configuration for single cycle (TimingSimpleCPU) and in order (MinorCPU) CPUs. The largest slowdown is 32% for *primes* with 90 cycles encryption latency. This is an expected behavior as most of these workloads are not memory intensive.

We further studied the impact of adding memory encryption on two microbenchmarks which load from and store to array of different sizes (8MB, 16MB, 32MB, 64MB, 128MB). These load and store operation are independent. Figure 8 shows the slowdown of these microbenchamrks (relative to no encryption) with different array sizes, A noteworthy point is that this test is performed on a simulated system which models an in order pipeline and has a last level cache of size 32MB. Therefore, the slowdown observed due to encryption for array size 32MB and under is very small. For 64MB and 128MB, the slowdown is around 16% for both loads and stores.

We performed a second experiment with memory encryption design. We evaluate systems in which only a subset of memory is encrypted. For example, solutions like AMD'S SME (secure memory encryption) transparently encrypt all memory with a single key.

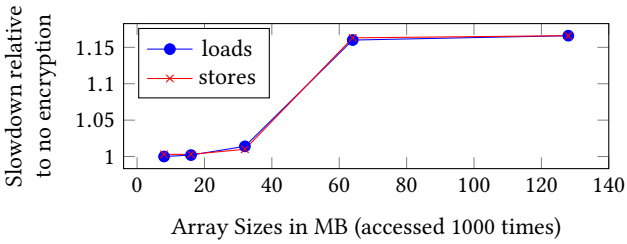


**Figure 9: Encryption slowdown for independent load and store operations for different array sizes with an encryption latency of 30 cycles.**

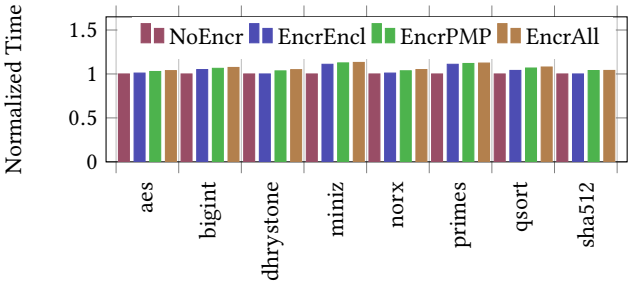


**Figure 10: Enclave vs all memory encryption**

The alternatives include encrypting accesses of only sensitive applications (e.g., enclaves) or somewhere in between. To understand how these choices might impact performance, we compared the execution time of three different configurations relative to no encryption. EncrEncl means that only the memory accesses belonging to an enclave are encrypted. In case of Keystone, this means that any physical memory access which passes a PMP check and does not fall in the first or last PMP entry. EncrPMP encrypts all memory accesses that required a PMP check and have passed the check. EncrAll encrypts all memory accesses even if they do not require a PMP check (e.g., M mode accesses). The encryption latency used for all three configurations is 30 cycles. Figure 10 shows that the slowdown increases if we move towards EncrAll configuration; however, the difference is small. We conclude that selective encryption will not provide much benefit (in terms of performance) over transparently encrypting everything for these workloads.

## 5.3 Micro-architecture impact on performance of trusted execution

The final use-case discusses how changing the microarchitecture can impact performance of the trusted execution and how would it relate to the performance of the untrusted execution on the same

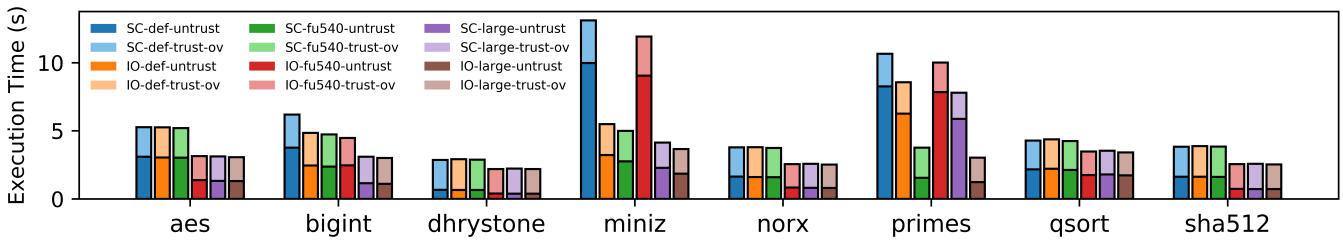


**Figure 11: Microarchitecture impact on performance of secure compute environments. In the legend entries SC: single cycle, IO: in-order, def: default configuration from Table 1, fu540: fu540-like configuration from Table 1, and large: large configuration from Table 1. 'trust-ov' stands for overhead of trusted execution.**

**Table 1: Main feature of the configurations tested on gem5**

| Feature | default | fu540-like | large |
|---|---|---|---|
| **Dcache size** | 32KB | 32KB | 512KB |
| **Dcache assoc.** | 8 | 8 | 8 |
| **L2 cache** | N/A | 2MB | 16MB |
| **L2 cache assoc.** | N/A | 16 | 32 |
| **DTLB entries** | 64 | 128 | 2048 |

platform. A secure computer architect can be interested in this kind of analysis while working on a new system. Cycle-level simulation is a quick way to perform this kind of design space exploration.

As an experiment, we picked single cycle (TimingSimpleCPU) and in order (MinorCPU) CPUs of gem5 and configured their memory and cache subsystems in three different ways (thus leading to six total configurations). The three memory and cache subsystems refer to def (default gem5 configuration), fu540 (fu540 like configuration), and large (a configuration with large structures and low latencies). Table 1 provides some details of these configurations.

We executed rv8 benchmarks in untrusted and trusted manner for all the six configurations, thus leading to 12 runs for a single benchmark. Figure 11 shows the execution time for all of these runs for each benchmark. We can observe that the overall execution time goes down as we move towards more aggressive configurations, however the ratio of trusted to untrusted execution time for each configuration stays similar. In other words, even on aggressive configuration, trusted execution incurs similar performance penalty (relative to untrusted execution) as it does on a simple configuration.

## 6 DISCUSSION

Importantly, the ability to simulate secure compute environments in cycle-level simulators like gem5 can enable new ways to evaluate security of computer architectures. The gem5 simulator provides an ability to inspect architectural and microarchitectural state at any point in time, which can reveal important information about the vulnerability of the system.

We envision that this work can provide an opportunity to perform a quantified vulnerability analysis for TEEs and other secure compute environments. Prior work has proposed techniques to evaluate the vulnerabilities in architectures in a systematic way. For instance, Mukherjee et al. proposed a metric, *Architecture Vulnerability Factor (AVF)*, to estimate vulnerability of hardware structures to produce errors because of faults [14], and Demme et al. proposed ways to measure the possibility of information leakage because of side channel attacks through a metric called *Side-channel Vulnerability Factor, SVF* [6]. *SVF* measures correlation between what can be observed by an attacker and the execution pattern of a victim. This helps to estimate a system's vulnerability to side channels.

Similar to these vulnerability factors, we imagine developing ways to estimate the exposure of a systems' architectural and microarchitectural state to un-trusted entities in a simulated environment. Being able to simulate secure environments at a reasonable detail provides us an opportunity to observe the state of an entire system or parts of a system (e.g., only the trusted components) at any point of execution to perform the types of analyses referred above. Moreover, researchers have previously used gem5 to evaluate speculative execution attacks (like Spectre and Meltdown) [13] or to evaluate defence mechanisms against these attacks [20, 21]. Thus, we believe that enabling architectural cycle-level simulation for an *open source* trusted execution environment will open novel research directions for the future.

## 7 CONCLUSION

In this work, we show that the cycle-level architectural simulation of secure architectures can be impactful as it opens practical ways to perform design space exploration of these architectures from both a performance and security perspective. We added support to run RISC-V based secure execution environments in gem5 and performed some experiments using Keystone and presented some use cases to make a case for the usability of simulation support for trusted environments. We look forward to the future research opportunities enabled by this new simulation infrastructure.

## ACKNOWLEDGEMENT

This work was supported by the Director, Office of Science, Office of Advanced Scientific Computing Research, of the U.S. Department of Energy under Contract No. DE-AC02-05CH11231 and the National Science Foundation under Grant No. CNS1925724. Any opinions, findings, conclusions, or recommendations expressed in this material are those of the authors and do not necessarily reflect those of the sponsors of this work. We would also like to thank the gem5 community and especially Peter Yuen for his work on RISC-V full system Linux boot support in gem5.

# REFERENCES


[1] 2021. rv8-bench. https://github.com/michaeljclark/rv8-bench. https://github.com/michaeljclark/rv8-bench [Online; accessed 5-May-2021].

[2] Fabrice Bellard. 2005. QEMU, a Fast and Portable Dynamic Translator. In *USENIX Annual Technical Conference, FREENIX Track*. Anaheim, CA, 41–46.

[3] Nathan Binkert, Bradford Beckmann, Gabriel Black, Steven K Reinhardt, Ali Saidi, Arkaprava Basu, Joel Hestness, Derek R Hower, Tushar Krishna, Somayeh Sardashti, et al. 2011. The gem5 Simulator. *ACM SIGARCH Computer Architecture News* 39, 2 (May 2011), 1–7.

[4] Bobby R Bruce, Ayaz Akram, Hoa Nguyen, Kyle Roarty, Mahyar Samani, Marjan Friborz, Trivikram Reddy, Matthew D Sinclair, and Jason Lowe-Power. 2021. Enabling Reproducible and Agile Full-System Simulation. In *2021 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. IEEE, 183–193.

[5] Claudio Canella, Jo Van Bulck, Michael Schwarz, Moritz Lipp, Benjamin Von Berg, Philipp Ortner, Frank Piessens, Dmitry Evtyushkin, and Daniel Gruss. 2019. A systematic evaluation of transient execution attacks and defenses. In *28th {USENIX} Security Symposium ({USENIX} Security 19)*. 249–266.

[6] John Demme, Robert Martin, Adam Waksman, and Simha Sethumadhavan. 2012. Side-channel vulnerability factor: A metric for measuring information leakage. In *2012 39th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 106–117.

[7] John L Hennessy and David A Patterson. 2019. A new golden age for computer architecture. *Commun. ACM* 62, 2 (2019), 48–60.

[8] Sagar Karandikar, Howard Mao, Donggyu Kim, David Biancolin, Alon Amid, Dayeol Lee, Nathan Pemberton, Emmanuel Amaro, Colin Schmidt, Aditya Chopra, et al. 2018. FireSim: FPGA-accelerated cycle-exact scale-out system simulation in the public cloud. In *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 29–42.

[9] 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. 2019. Spectre Attacks: Exploiting Speculative Execution. In *40th IEEE Symposium on Security and Privacy (S&P'19)*.

[10] Dayeol Lee, David Kohlbrenner, Shweta Shinde, Krste Asanović, and Dawn Song. 2020. Keystone: An open framework for architecting trusted execution environments. In *Proceedings of the Fifteenth European Conference on Computer Systems*. 1–16.

[11] 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. 2018. Meltdown: Reading Kernel Memory from User Space. In *27th USENIX Security Symposium (USENIX Security 18)*.

[12] Jason Lowe-Power, Abdul Mutaal Ahmad, Ayaz Akram, Mohammad Alian, Rico Amslinger, Matteo Andreozzi, Adrià Armejach, Nils Asmussen, Brad Beckmann, Srikant Bharadwaj, et al. 2020. The gem5 simulator: Version 20.0+. *arXiv preprint arXiv:2007.03152* (2020).

[13] Jason Lowe-Power, Venkatesh Akella, Matthew K Farrens, Samuel T King, and Christopher J Nitta. 2018. Position Paper: A case for exposing extra-architectural state in the ISA. In *Proceedings of the 7th International Workshop on Hardware and Architectural Support for Security and Privacy*. 1–6.

[14] Shubhendu S Mukherjee, Christopher Weaver, Joel Emer, Steven K Reinhardt, and Todd Austin. 2003. A systematic methodology to compute the architectural vulnerability factors for a high-performance microprocessor. In *Proceedings. 36th Annual IEEE/ACM International Symposium on Microarchitecture, 2003. MICRO-36.* IEEE, 29–40.

[15] Alec Roelke and Mircea R Stan. 2017. Risc5: Implementing the RISC-V ISA in gem5. In *First Workshop on Computer Architecture Research with RISC-V (CARRV)*.

[16] Mark Russinovich, Manuel Costa, Cédric Fournet, David Chisnall, Antoine Delignat-Lavaud, Sylvan Clebsch, Kapil Vaswani, and Vikas Bhatia. 2021. Toward Confidential Cloud Computing: Extending hardware-enforced cryptographic protection to data while in use. *Queue* 19, 1 (2021), 49–76.

[17] Tuan Ta, Lin Cheng, and Christopher Batten. 2018. Simulating multi-core RISC-V systems in gem5. In *Workshop on Computer Architecture Research with RISC-V*.

[18] Justin Tullos, Scott Graham, and Pranav Patel. 2021. Applied Analytical Model for Latency Evaluation of RISC-V Security Monitor. In *16th International Conference on Cyber Warfare and Security*. Academic Conferences Limited, 354.

[19] Andrew Waterman, Yunsup Lee, Rimas Avizienis, David A Patterson, and Krste Asanovic. 2015. *The risc-v instruction set manual volume 2: Privileged architecture version 1.7*. Technical Report. University of California at Berkeley Berkeley United States.

[20] Mengjia Yan, Jiho Choi, Dimitrios Skarlatos, Adam Morrison, Christopher Fletcher, and Josep Torrellas. 2018. Invisispec: Making speculative execution invisible in the cache hierarchy. In *2018 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 428–441.

[21] Jiyong Yu, Mengjia Yan, Artem Khyzha, Adam Morrison, Josep Torrellas, and Christopher W Fletcher. 2019. Speculative taint tracking (stt) a comprehensive protection for speculatively accessed data. In *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*. 954–968.