Int. Conf. on IC Design & Technology (ICICDT 2022)

Hanoi, 21-23 Sep., 2022

RISC-V Computer System Designed for Cyber-Security

Trong-Thuc HOANG,[1] Ba-Anh DAO,[2] Anh-Tien LE,[1,2]
Van-Phuc HOANG,[3] and Cong-Kha PHAM[1]

[1] University of Electro-Communications (UEC), Tokyo, Japan
[2] Academy of Cryptography Techniques (ACT-VN), Hanoi, Vietnam
[3] Le Quy Don Technical University (LQDTU), Hanoi, Vietnam

OUTLINE

1. Introduction
2. What is RISC-V, and How Does It Affect Cyber-Security?
3. Secure Boot for Trusted Execution Environment (TEE)
4. Cache Side-channel Attack (Spectre) on Out-of-Order (OoO) Processors
5. Prevent Correlation Power Analysis (CPA) with Random Dynamic Frequency Scaling (RDFS)
6. Conclusion

OUTLINE

1. Introduction
2. What is RISC-V, and How Does It Affect Cyber-Security?
3. Secure Boot for Trusted Execution Environment (TEE)
4. Cache Side-channel Attack (Spectre) on Out-of-Order (OoO) Processors
5. Prevent Correlation Power Analysis (CPA) with Random Dynamic Frequency Scaling (RDFS)
6. Conclusion

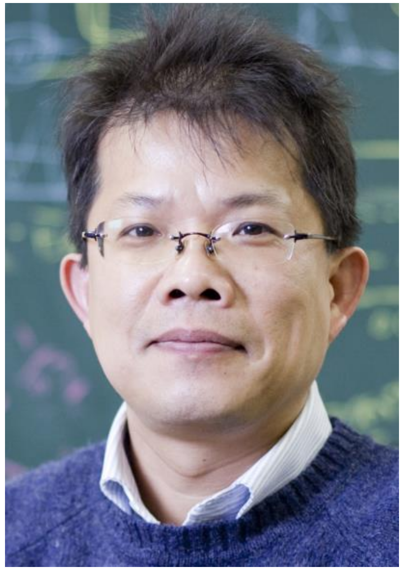
1. Introduction (1/5) Authors

Trong-Thuc HOANG,
Assistant Professor (UEC)

Ba-Anh DAO,
Dr. (ACT-VN)

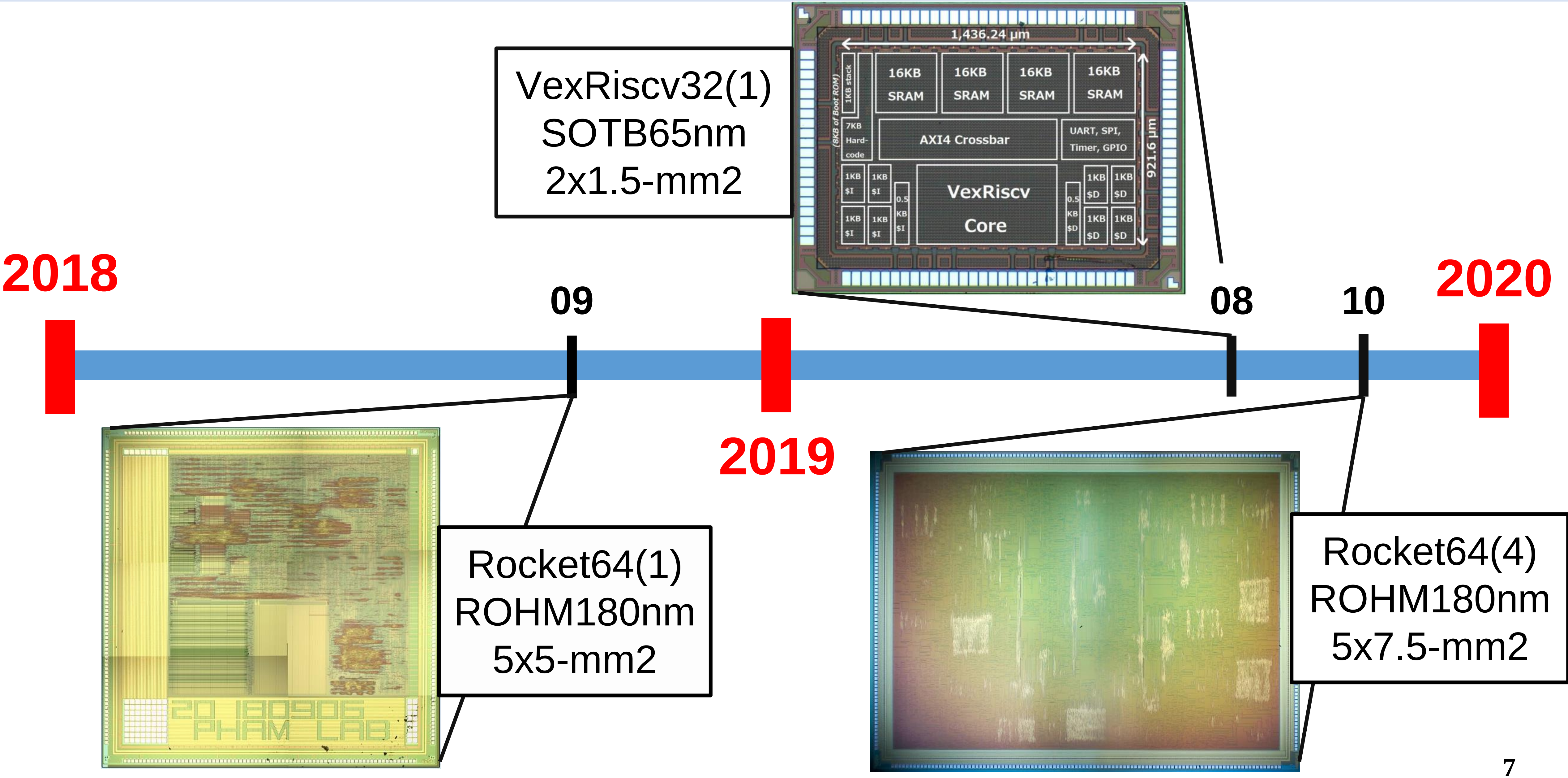
Anh-Tien LE
(UEC, ACT-VN)

Van-Phuc HOANG,
Associate Professor
(LQDTU)



Cong-Kha PHAM,
Professor (UEC)

1. Introduction (2/5) RISC-V project timeline



2018

09

Rocket64(1)
ROHM180nm
5x5-mm2

2019

08

VexRiscv32(1)
SOTB65nm
2x1.5-mm2

10

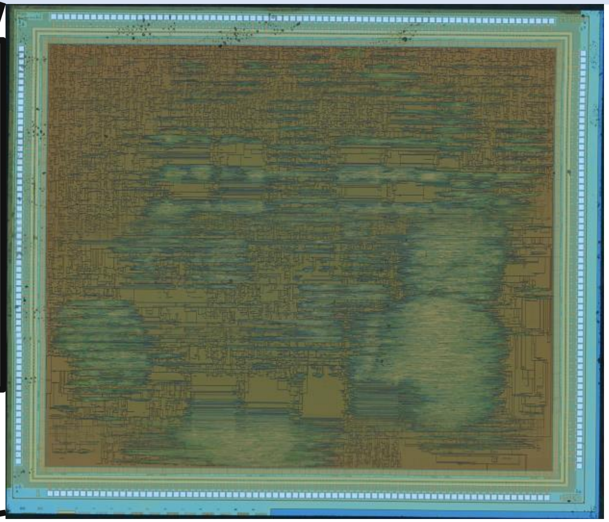
Rocket64(4)
ROHM180nm
5x7.5-mm2

2020

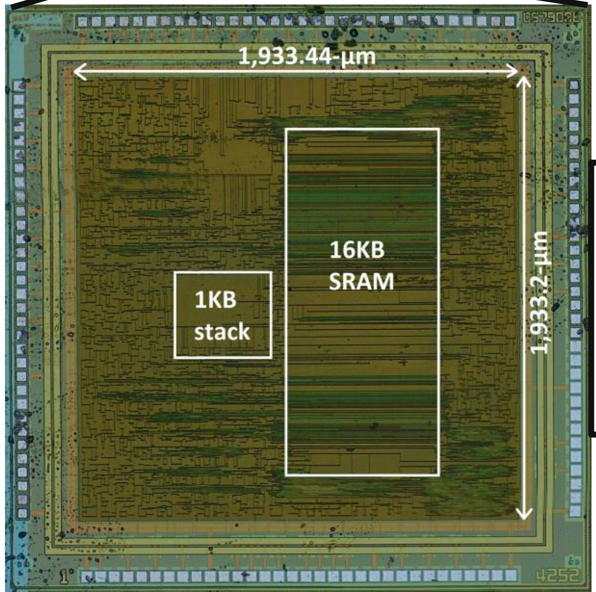
1. Introduction (3/5) RISC-V project timeline

2020

Rocket64(2)
+ Crypto-cores
ROHM180nm
5x5-mm2



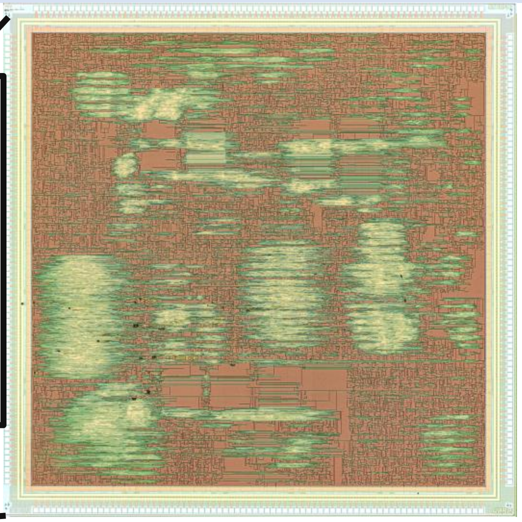
01

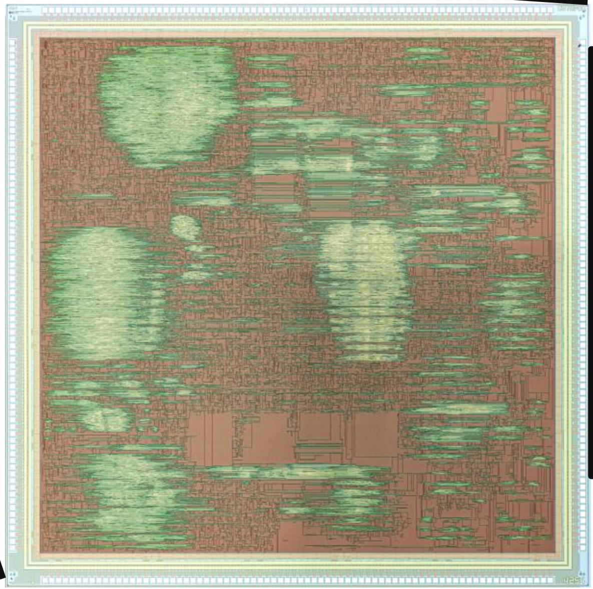


VexRiscv32(1)
ROHM180nm
2.5x2.5-mm2

06

Boom64(1) +
Crypto-cores
ROHM180nm
5x5-mm2





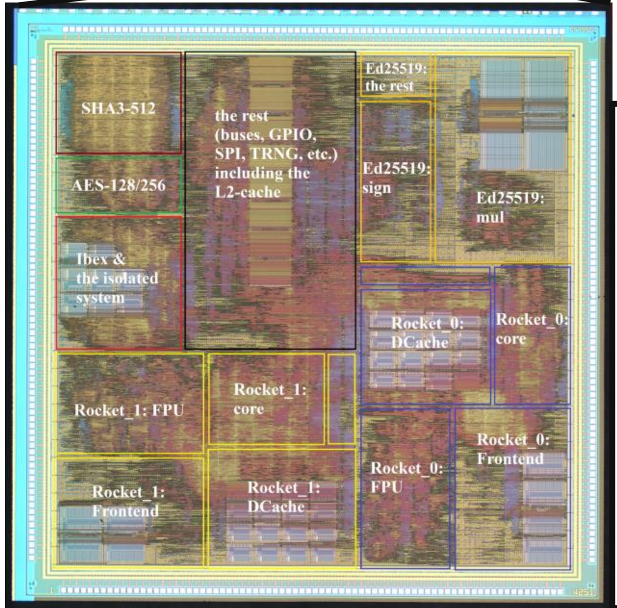
Rocket32(1)
+ Boom32(1)
+ Crypto-cores
ROHM180nm
5x5-mm2

2021

1. Introduction (4/5) RISC-V project timeline

2021

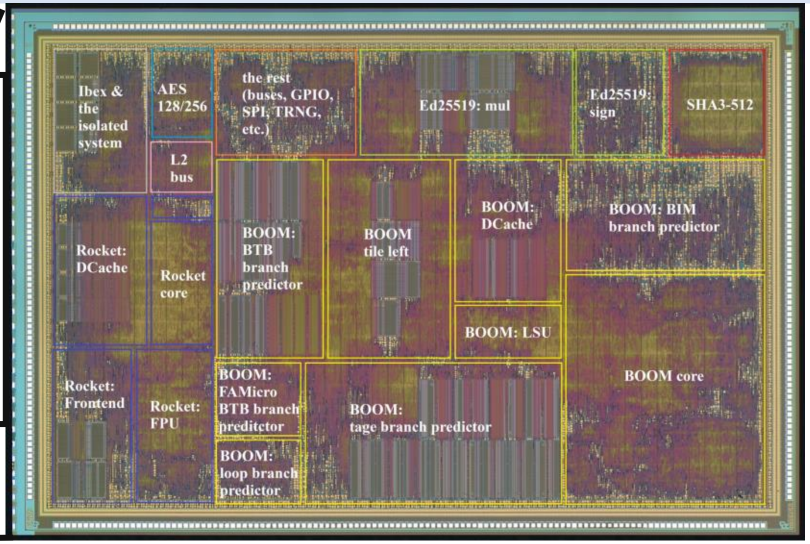
02

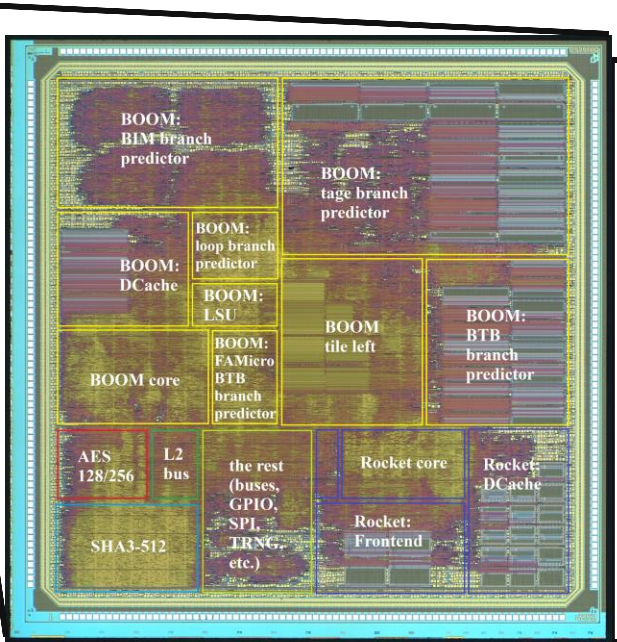


Rocket64(2)
+ Crypto-cores
+ TRNG
+ Secure boot
ROHM180nm
5x5-mm2

06

Rocket64(1) + Boom64(1)
+ Crypto-cores & TRNG
+ Secure boot
ROHM180nm : 5x7.5-mm2

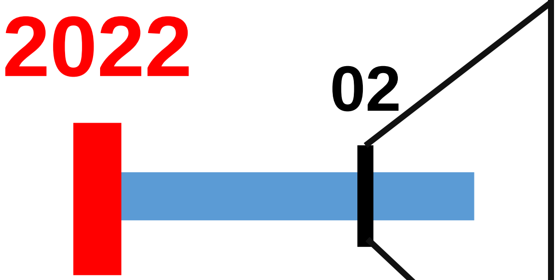

Ibex & the isolated system
AES 128/256
the rest (buses, GPIO, SPI, TRNG, etc.)
Ed25519: mul
Ed25519: sign
SHA3-512
L2 bus
Rocket: DCache
Rocket core
BOOM: BTB branch predictor
BOOM tile left
BOOM: DCache
BOOM: BIM branch predictor
BOOM: LSU
BOOM core
Rocket: Frontend
Rocket: FPU
BOOM: FAMicro BTB branch preditctor
BOOM: tage branch predictor
BOOM: loop branch predictor


BOOM: BIM branch predictor
BOOM: tage branch predictor
BOOM: loop branch predictor
BOOM: DCache
BOOM: LSU
BOOM tile left
BOOM: BTB branch predictor
BOOM core
BOOM: FAMicro BTB branch predictor
AES 128/256
L2 bus
the rest (buses, GPIO, SPI, TRNG, etc.)
Rocket core
Rocket: DCache
Rocket: Frontend
SHA3-512

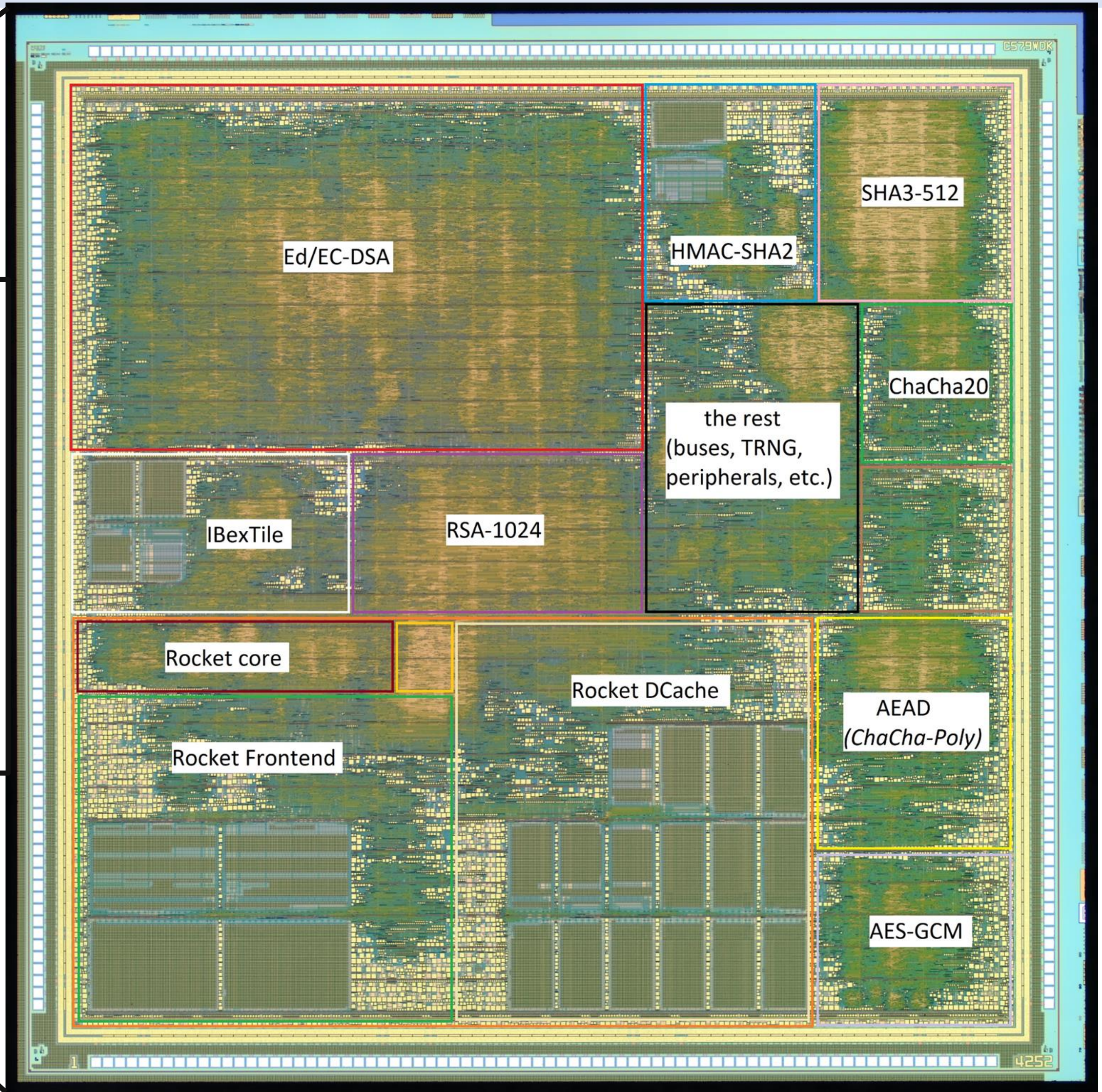
Rocket32(1)
+ Boom32(1)
+ Crypto-cores
+ TRNG
+ Secure boot
ROHM180nm
5x5-mm2

2022

1. **Introduction** (5/5) RISC-V project timeline



Rocket32(1)
+ TLS-1.3
Crypto-cores
+ TRNG
+ Secure boot
ROHM180nm
5x5-mm2



OUTLINE

1. Introduction
2. What is RISC-V, and How Does It Affect Cyber-Security?
3. Secure Boot for Trusted Execution Environment (TEE)
4. Cache Side-channel Attack (Spectre) on Out-of-Order (OoO) Processors
5. Prevent Correlation Power Analysis (CPA) with Random Dynamic Frequency Scaling (RDFS)
6. Conclusion

2. RISC-V vs. Cyber-security (1/8) What is ISA?

ISA means **I**nstruction **S**et **A**rchitecture

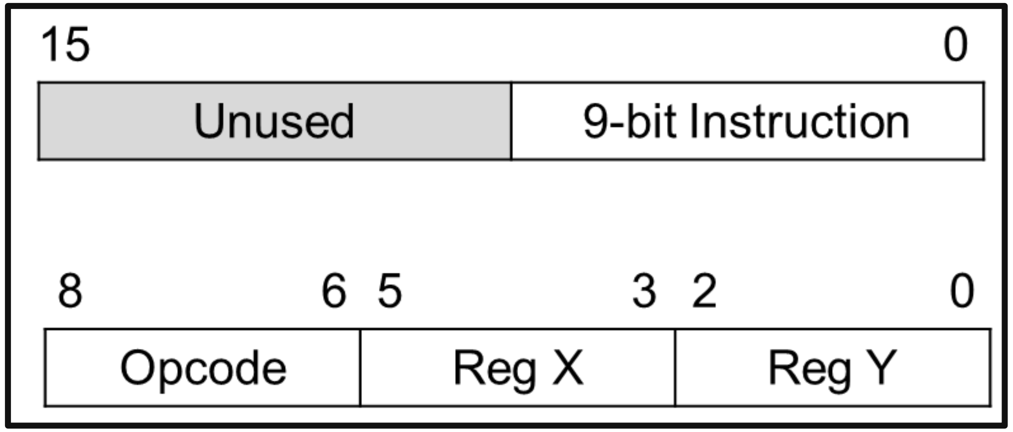
Software tools: assembler, compilers, debugger, linker, etc.

ISA: the interface between software & hardware architects

Processor: ALU, FPU, registers, CSRs, branch predictor, caches, etc.

ISA has to define all these kinds of stuffs:

1) How many instructions, and which is which?
2) In an instruction, what field means what?
3) Addressing & data-path *(8/16/32/64/128-bit)*?
4) What is supported and what is not?
5) *etc.*



2. RISC-V vs. Cyber-security (2/8) CISC vs. RISC

CISC

(**C**omplex **I**nstruction **S**et **C**omputer)

1) Emphasis on hardware
2) Includes multi-clock complex instructions
3) Memory-to-memory mindset
4) Small code sizes, high cycles/s
5) Transistors used for storing complex instructions

RISC

(**R**educed **I**nstruction **S**et **C**omputer)

1) Emphasis on software
2) Single-clock reduced instructions only

1) Register-to-register mindset
2) Large code sizes, low cycles/s
3) Spends more transistors, and most of them are used for storing data

$$\textbf{Performance} = \frac{\text{time}}{\text{program}} = \overbrace{\frac{\text{time}}{\text{cycle}} \times \frac{\text{cycle}}{\text{instruction}}}^{\text{RISC win}} \times \overbrace{\frac{\text{instruction}}{\text{program}}}^{\text{CISC win}}$$

Nowadays, almost all processors in the market are RISCs.

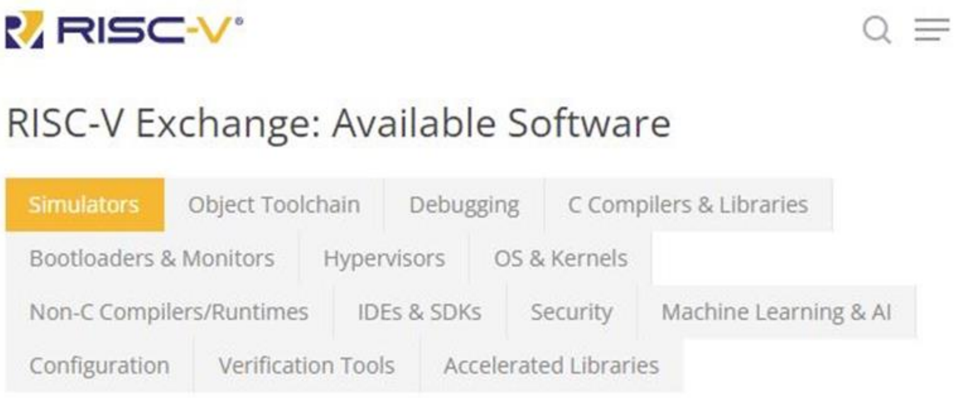
RISC-V simply means *RISC* architecture version *five*

2. RISC-V vs. Cyber-security (3/8) RISC-V ISA

Open-source **RISC-V** means open-source **ISA**, no more, no less.

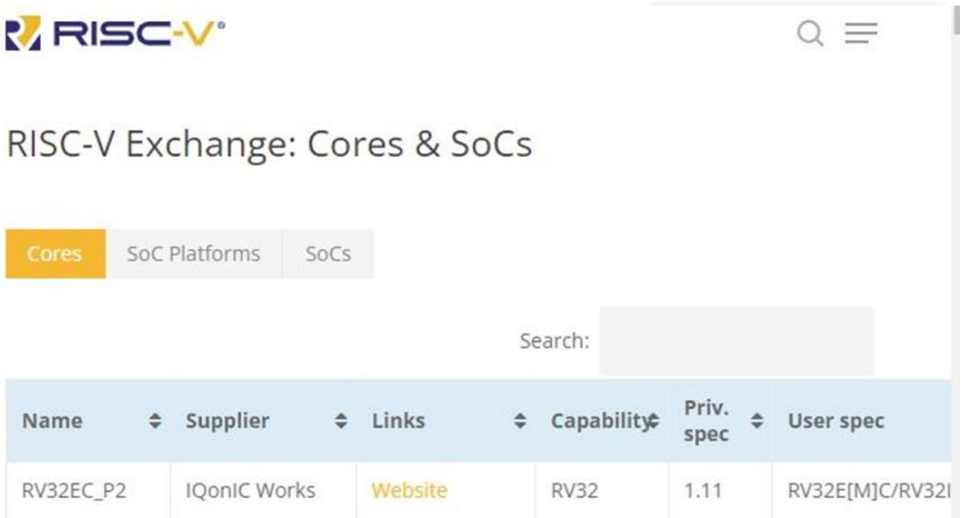
(some other common ISAs: i386, amd64, ARM 32/64, AVR, MIPS, NiosII, etc.)

RISC-V Foundation: https://riscv.org/



RISC-V Exchange: Available Software

Simulators | Object Toolchain | Debugging | C Compilers & Libraries | Bootloaders & Monitors | Hypervisors | OS & Kernels | Non-C Compilers/Runtimes | IDEs & SDKs | Security | Machine Learning & AI | Configuration | Verification Tools | Accelerated Libraries



RISC-V Exchange: Cores & SoCs

Cores | SoC Platforms | SoCs

Search:

Name	Supplier	Links	Capability	Priv. spec	User spec
RV32EC_P2	IQonIC Works	Website	RV32	1.11	RV32E[M]C/RV32I

- Official released ISA specification
- Many cores, SoCs, & software are available
- Developers can reuse each other designs & tools → significantly reducing R&D time and effort

Licensed free:
- RISC-V ISA
- RISC-V toolchain

License depended on authors/developers:
- RISC-V processors
- RISC-V software applications
- RISC-V-related products

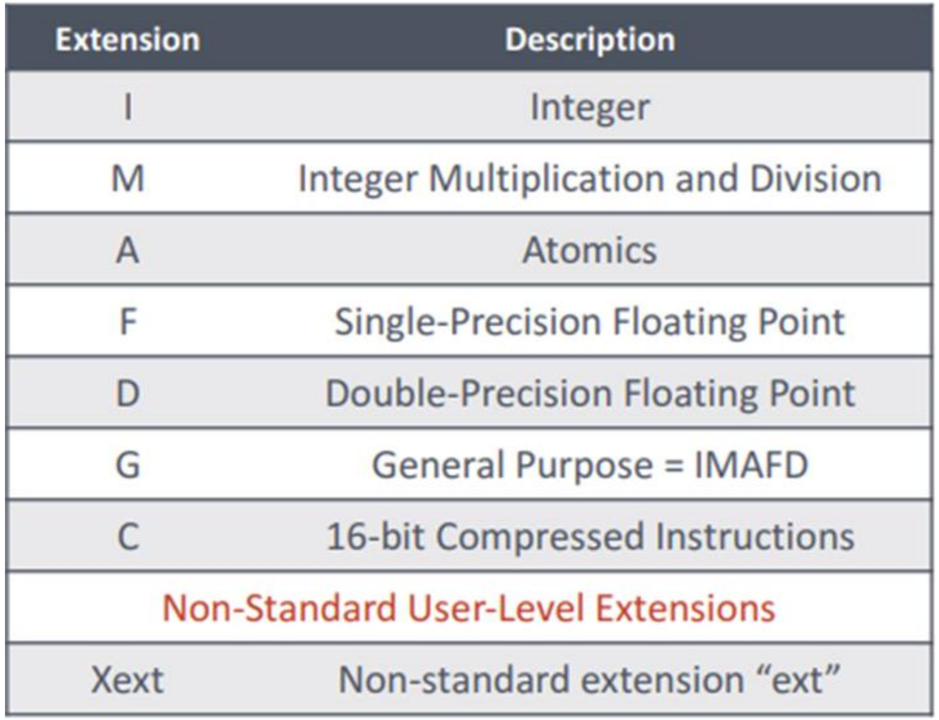
2. RISC-V vs. Cyber-security (4/8) RISC-V ISA

What makes **RISC-V** different: <u>its modular mindset</u>

(modular architecture helps fine-tune the performance based on the developer's needs [1])

Base instruction set: **I**nteger

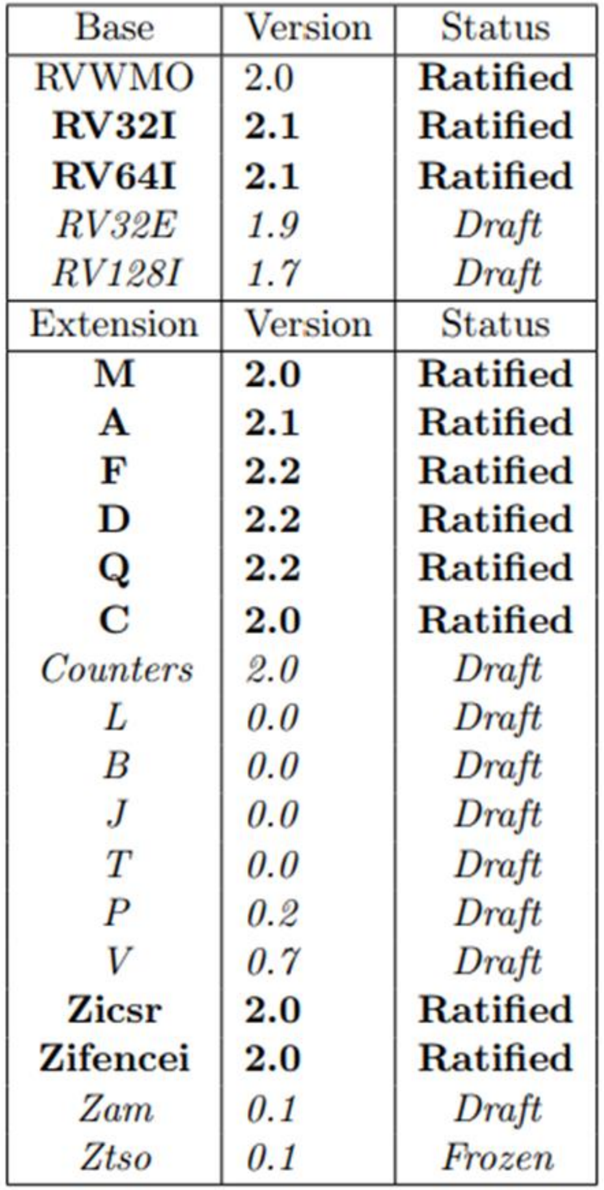
Extended instruction set: *the rest*



Extension	Description
I	Integer
M	Integer Multiplication and Division
A	Atomics
F	Single-Precision Floating Point
D	Double-Precision Floating Point
G	General Purpose = IMAFD
C	16-bit Compressed Instructions
Non-Standard User-Level Extensions	
Xext	Non-standard extension "ext"

The most common extensions: **IMAFDC** *(also known as* ***GC****)*

There are also <u>a lot more</u> than just **IMAFDC :**

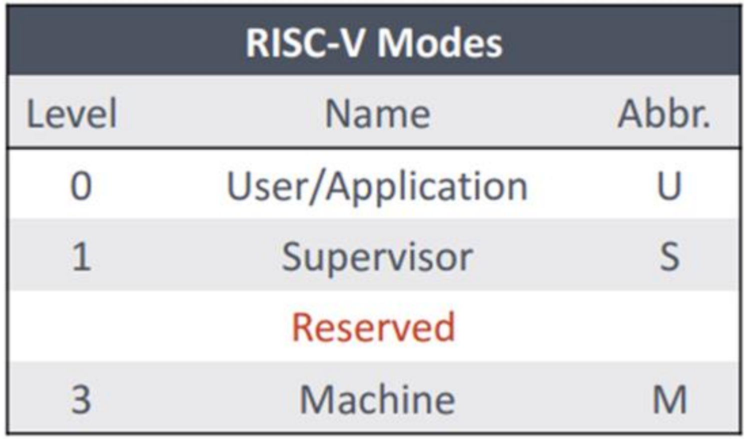


Base	Version	Status
RVWMO	2.0	Ratified
RV32I	**2.1**	**Ratified**
RV64I	**2.1**	**Ratified**
RV32E	*1.9*	*Draft*
RV128I	*1.7*	*Draft*
Extension	Version	Status
M	**2.0**	**Ratified**
A	**2.1**	**Ratified**
F	**2.2**	**Ratified**
D	**2.2**	**Ratified**
Q	**2.2**	**Ratified**
C	**2.0**	**Ratified**
Counters	*2.0*	*Draft*
L	*0.0*	*Draft*
B	*0.0*	*Draft*
J	*0.0*	*Draft*
T	*0.0*	*Draft*
P	*0.2*	*Draft*
V	*0.7*	*Draft*
Zicsr	**2.0**	**Ratified**
Zifencei	**2.0**	**Ratified**
Zam	*0.1*	*Draft*
Ztso	*0.1*	*Frozen*

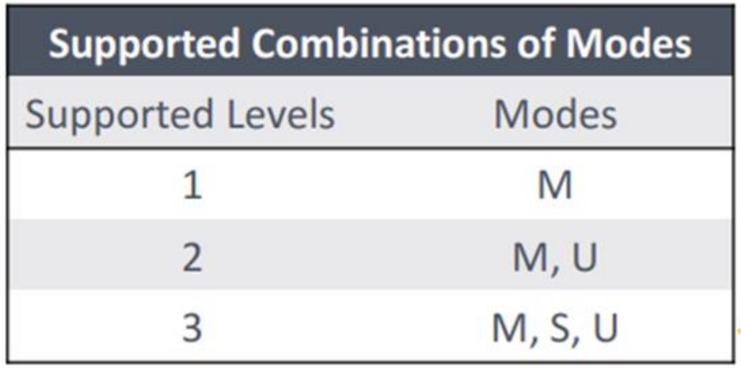
2. RISC-V vs. Cyber-security (5/8) RISC-V ISA

To support an Operating System (OS), the ISA has to support the OS stack *or the **M-/S-/U-mode***.

RISC-V privileged architecture:

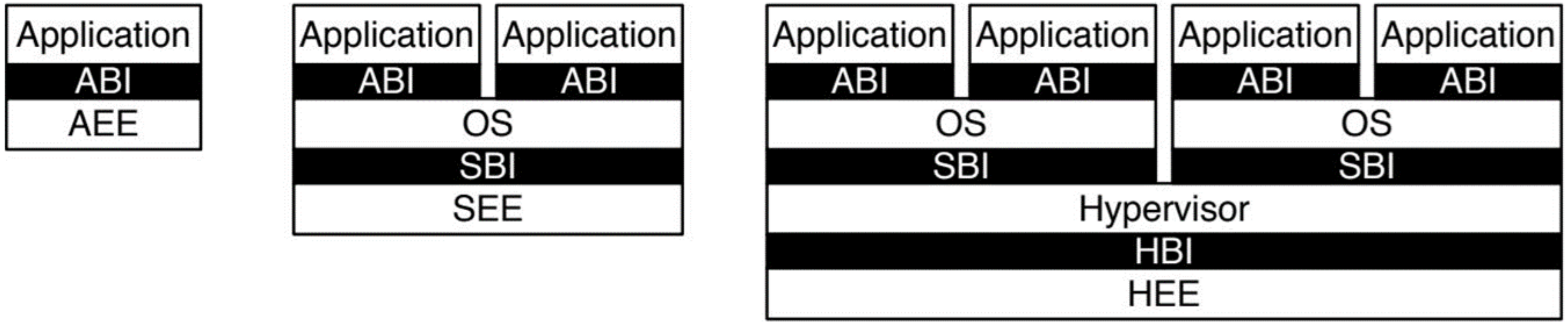


RISC-V Modes		
Level	Name	Abbr.
0	User/Application	U
1	Supervisor	S
	Reserved	
3	Machine	M



Supported Combinations of Modes	
Supported Levels	Modes
1	M
2	M, U
3	M, S, U

Different scenarios of utilizing the OS stack:

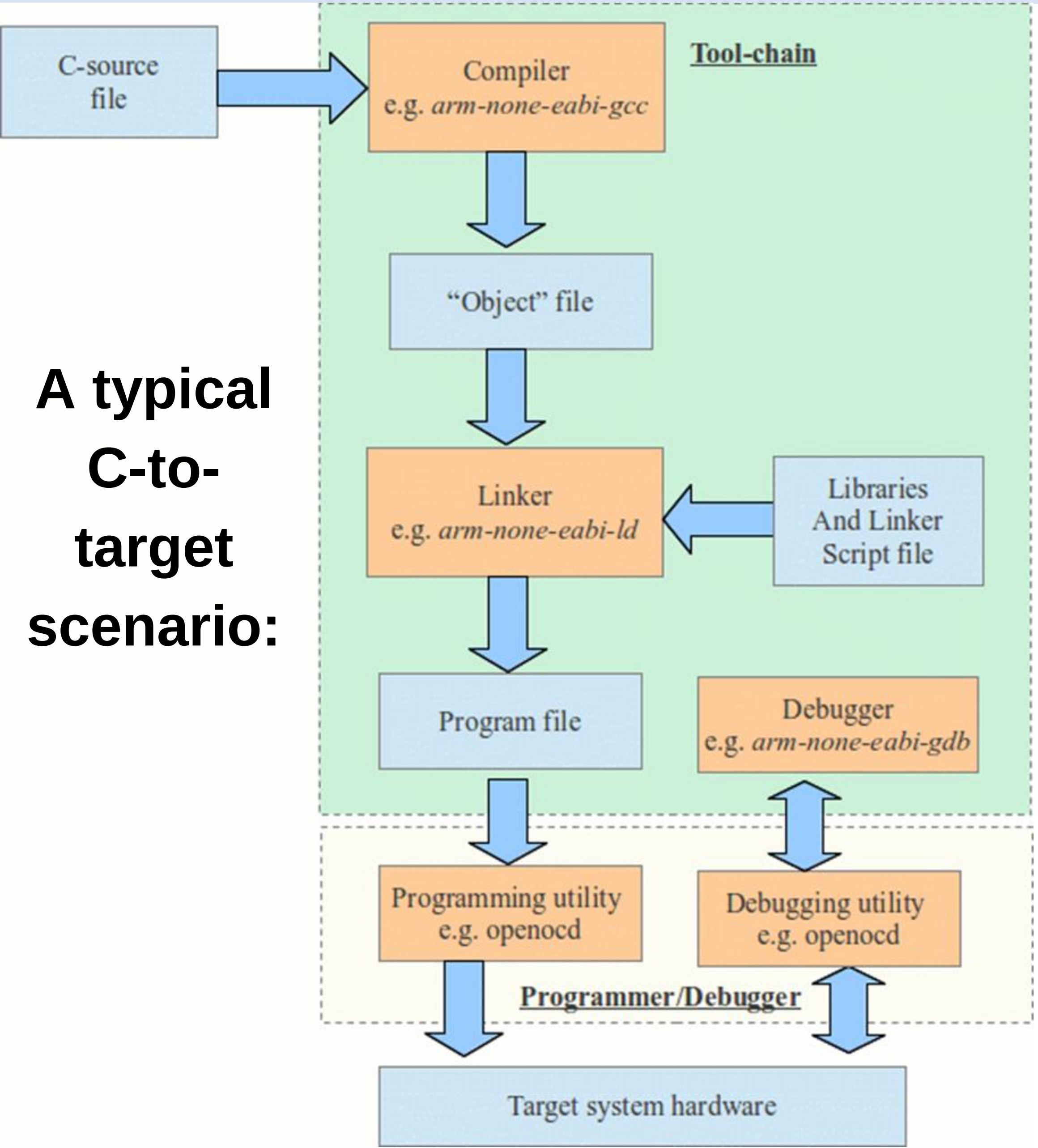


RISC-V ISA not only supports the OS stack, but also provides a **privileged architecture** [2].

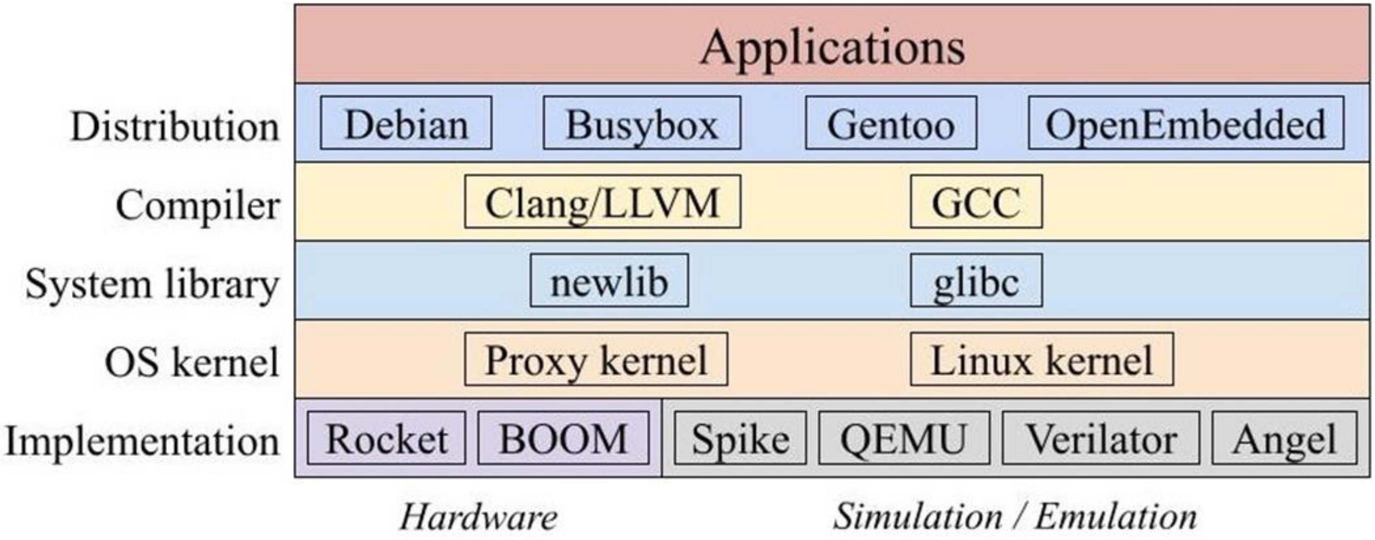
→ Better security scheme by having the hardware recognize different codes executed at different modes.

2. RISC-V vs. Cyber-security (6/8) RISC-V toolchain

A typical C-to-target scenario:



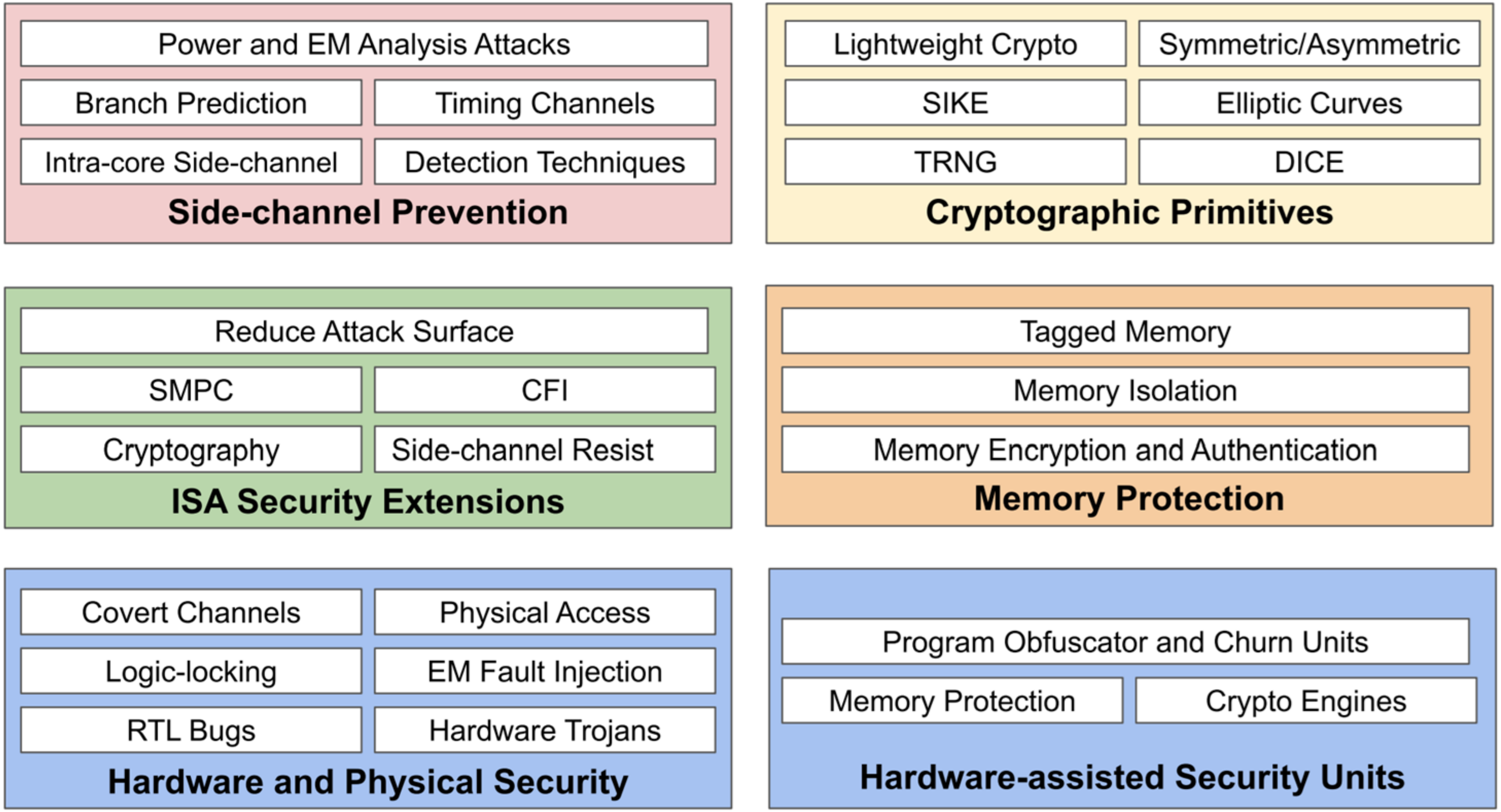
RISC-V toolchain and its ecosystem [3]:



Three most important tools:

- **GCC:** *(cross C compiler)* make a C code into assembly code
- **LD:** *(linker)* links standard libraries into the build; also links between multiple C files
- **GDB:** *(debugger)* debug the hardware/simulator/emulator

2. RISC-V vs. Cyber-security (7/8) RISC-V security



Side-channel Prevention

- Power and EM Analysis Attacks
- Branch Prediction
- Timing Channels
- Intra-core Side-channel
- Detection Techniques

Cryptographic Primitives

- Lightweight Crypto
- Symmetric/Asymmetric
- SIKE
- Elliptic Curves
- TRNG
- DICE

ISA Security Extensions

- Reduce Attack Surface
- SMPC
- CFI
- Cryptography
- Side-channel Resist

Memory Protection

- Tagged Memory
- Memory Isolation
- Memory Encryption and Authentication

Hardware and Physical Security

- Covert Channels
- Physical Access
- Logic-locking
- EM Fault Injection
- RTL Bugs
- Hardware Trojans

Hardware-assisted Security Units

- Program Obfuscator and Churn Units
- Memory Protection
- Crypto Engines

RISC-V Hardware and Architecture Security [4]

Security topics that attract attention in the RISC-V community.

2. RISC-V vs. Cyber-security (8/8) RISC-V security

The top reasons for choosing RISC-V for Cyber-security

- **For an opportunity to secure the Internet-of-Things (IoT):** cybersecurity software is ultimately ineffective. To truly address the problem, we need to address the issue at its core: directly inside the SoCs.
- **For price-sensitive applications:** specific and limited (also usually repeated) tasks can be solved cost-efficiently with a base core and a few specialized components.
- **For an open approach to cyber-security:** RISC-V's ecosystem has seen significant growth over the past few years. A Rich and strong open-source community promises equally rich and strong public libraries and open-source designs.
- **For frequently discussed and addressed security issues:** cyber-security hot topics are discussed frequently at least once a month. They are hosted by the two work groups of Cryptographic Extensions and Trusted Execution Environment (TEE) [5].

OUTLINE

1. Introduction
2. What is RISC-V, and How Does It Affect Cyber-Security?
3. Secure Boot for Trusted Execution Environment (TEE)
4. Cache Side-channel Attack (Spectre) on Out-of-Order (OoO) Processors
5. Prevent Correlation Power Analysis (CPA) with Random Dynamic Frequency Scaling (RDFS)
6. Conclusion

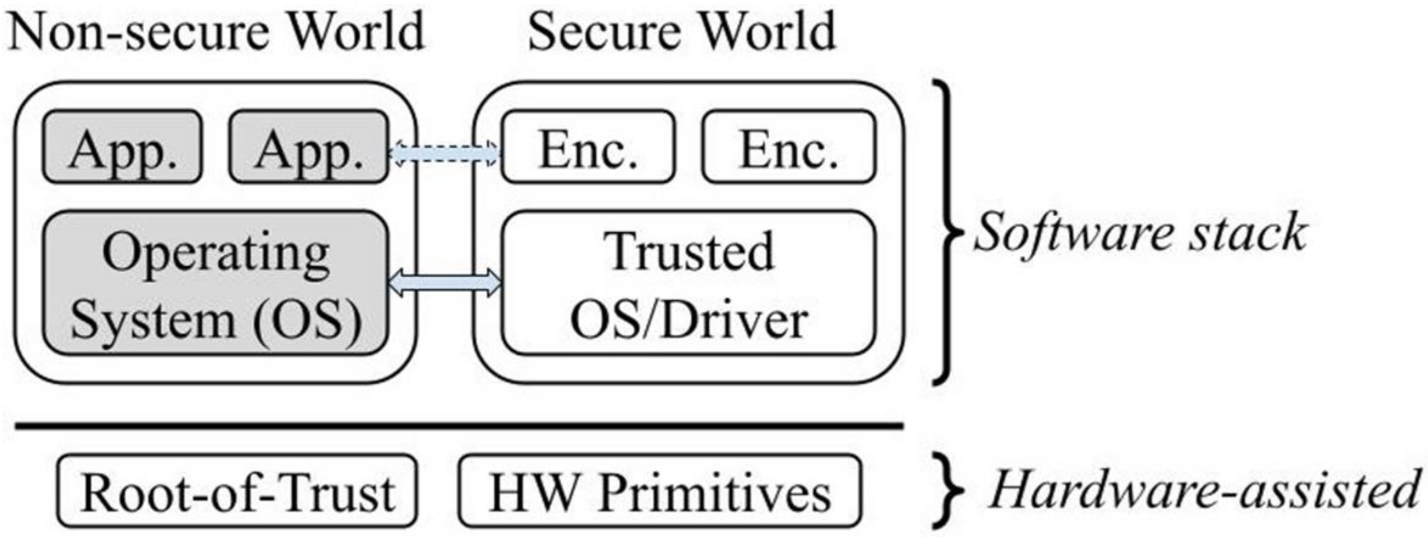
3. Secure Boot for TEE (1/23) What is TEE?

Trusted Execution Environment (TEE) [6] provides:

1. *Integrity:* the code and data cannot be tampered.
2. *Confidentiality:* the application’s content cannot be read.
3. *Attestation:* proof to a remote party that the system is safe.

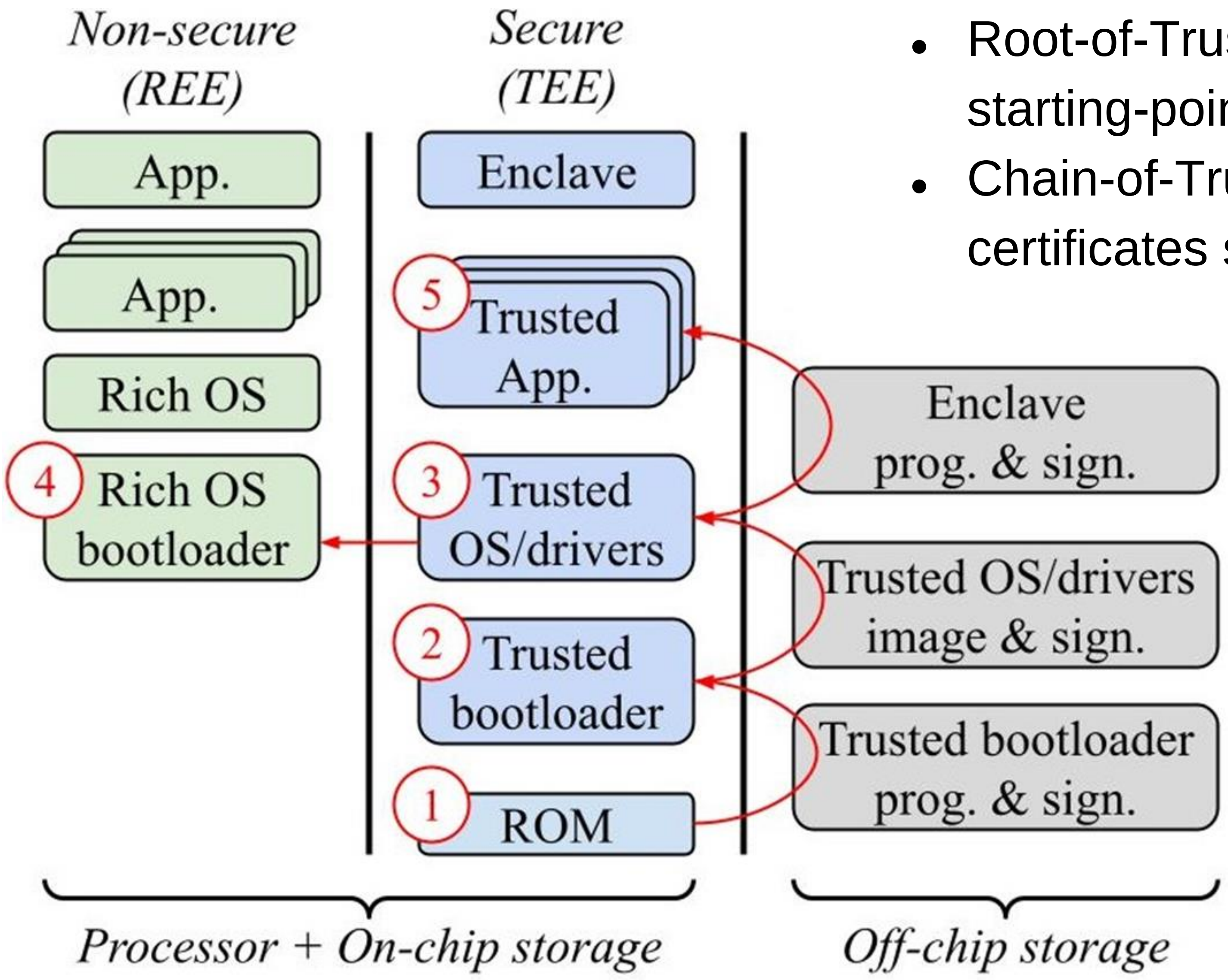
A typical TEE setup:

- Secure (trusted) vs. non-secure (untrusted) worlds.
- Barrier enforcer by: software *AND* hardware.
- All TEEs need some sort of hardware-assisted modules: Root-of-Trust (RoT) and primitives.



- HW primitives *(examples)*: cache flushing, cache partitioning, memory isolation, memory encryption, keys management, bus access controller, enclave encryption, and so on.

3. Secure Boot for TEE (2/23) What is TEE?



Root-of-Trust (RoT) in theTEE:

- Root-of-Trust (RoT): the 1st verification at reset, the starting-point for CoT.
- Chain-of-Trust (CoT): a series of signatures & certificates started from the RoT up to the Rich OS.

Secure boot guarantee:

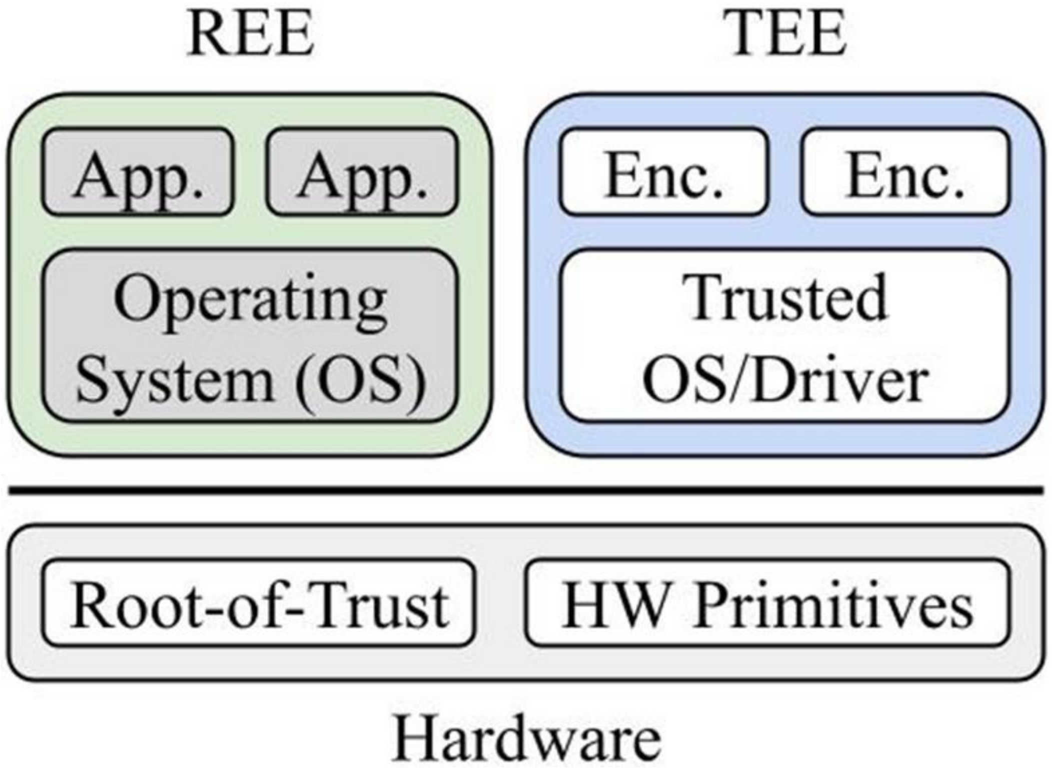
- All TEE-related assets *(code, trusted OS/drivers, hardware primitives)* are installed and at the initial states *(as expected by designers)*.
- Means: EVERYTHING is signature checked, and EVERY sensitive data are immutable or held in isolation.

3. Secure Boot for TEE (3/23) TEE vs. secure boot

TEE is just an isolated environment. It isn't, *and shouldn't be*, the Root of Trust (RoT).

Most TEE models have to assume:

- The hardware is trusted and securely booted.
- The bootloader is "bug-free," and the RoT has not been tampered.

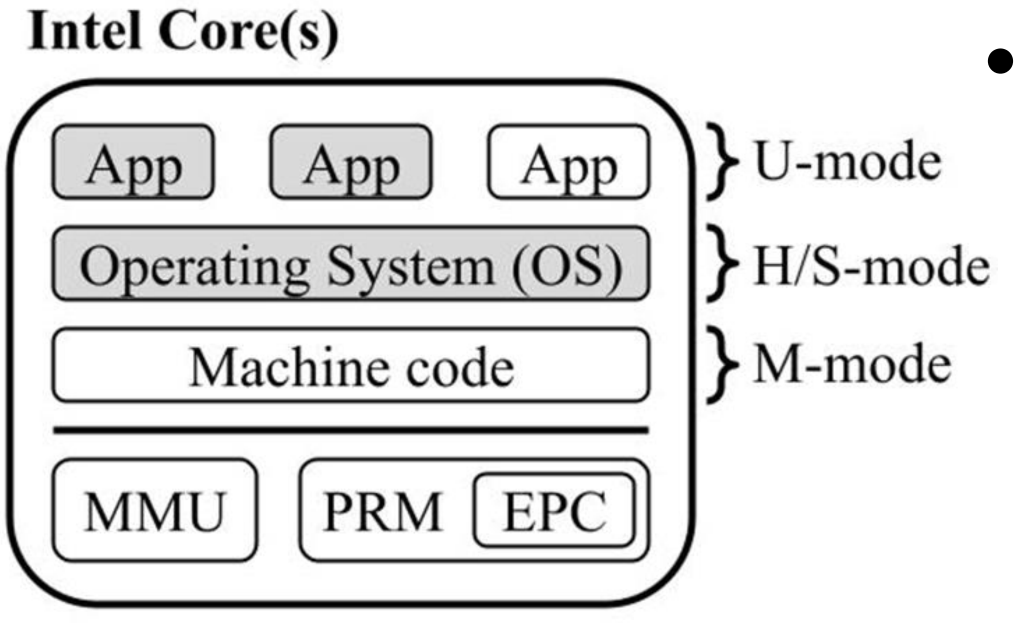


To achieve this, we can:

1. Use TEE processors for the secure boot.
2. Use extra hardware or third-party IPs.
3. Other approaches such as dynamic RoT without reset.

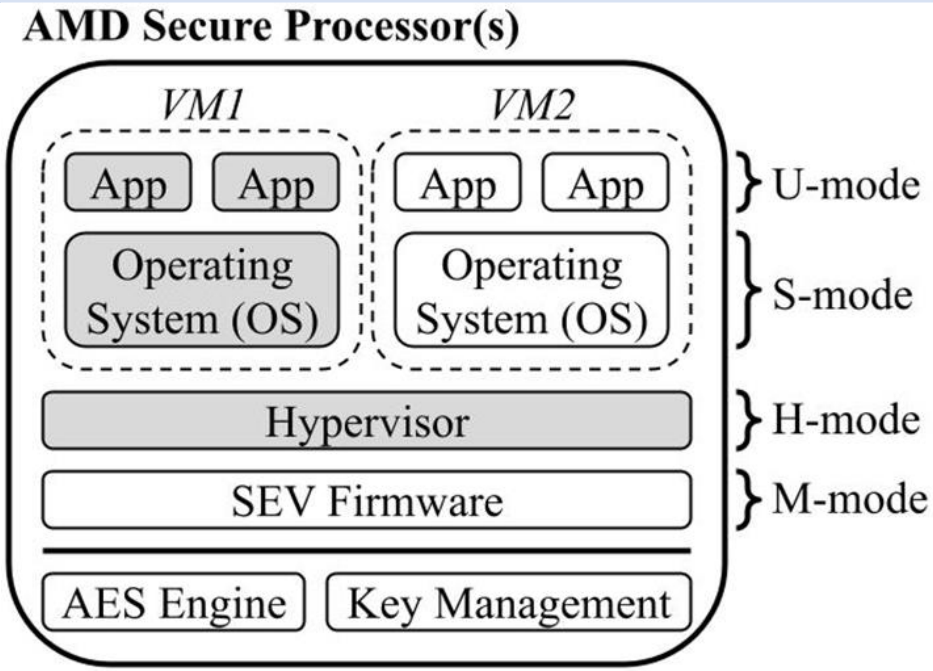
Bury root keys deep under layers of obscurity just increase the cost for attackers. The attack surface still exists as long as the secure boot process and RoT are still in the TEE system.

3. Secure Boot for TEE (4/23) TEE implementations



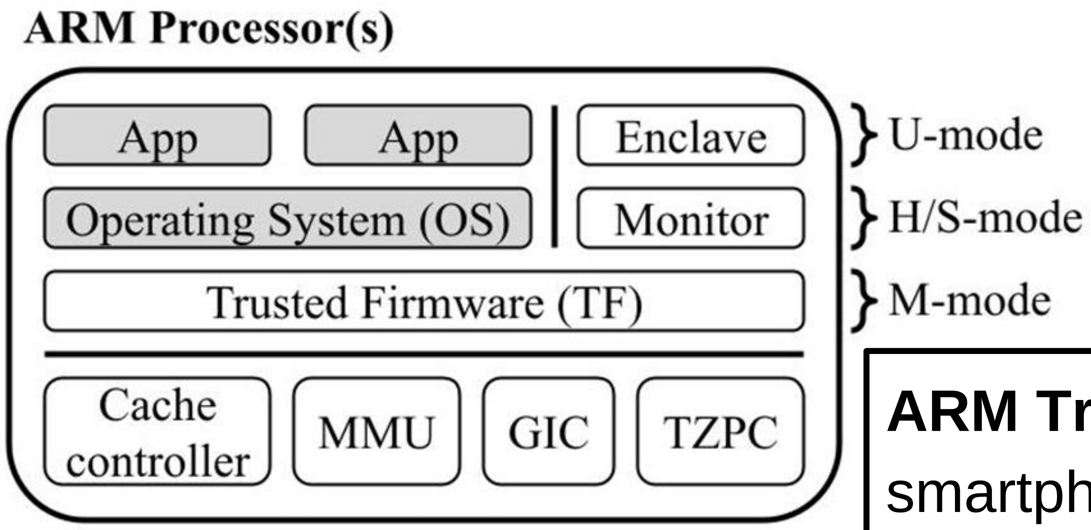
Intel SGX

- Many TEE models were proposed: different set goals, different resources, and different developing mindsets.



AMD SEV

Intel SGX [7]: aiming for conventional PCs



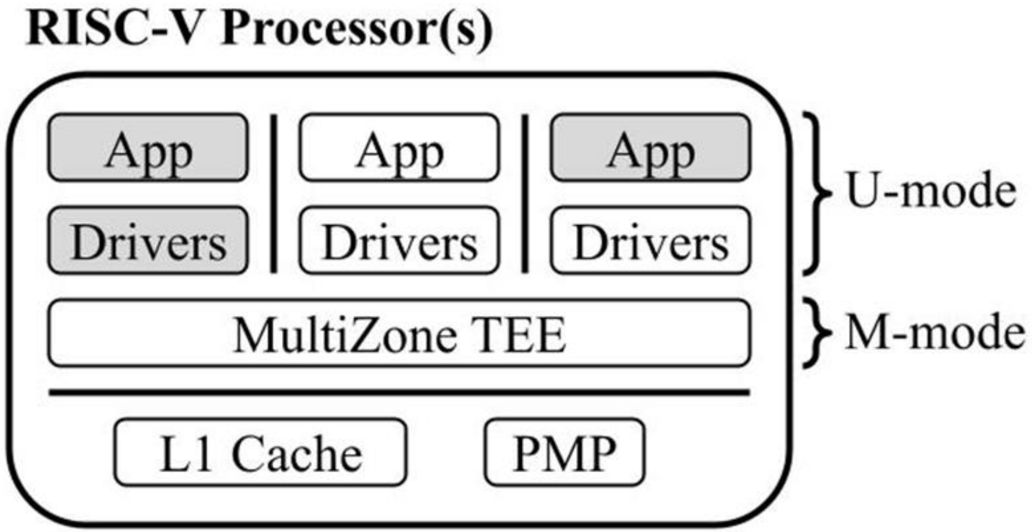
ARM TrustZone

AMD SEV [9]: aiming for server's cloud computing

ARM TrustZone [8]: aiming for smartphones/embedded-systems

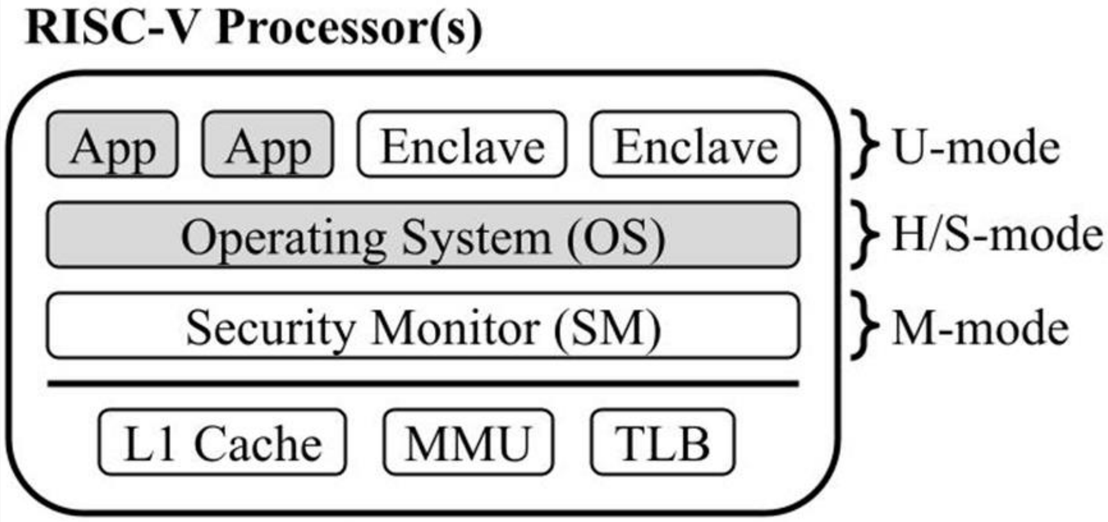
- Most closed-source TEEs are fine-tuned for their specific processors.

3. Secure Boot for TEE (5/23) TEE implementations



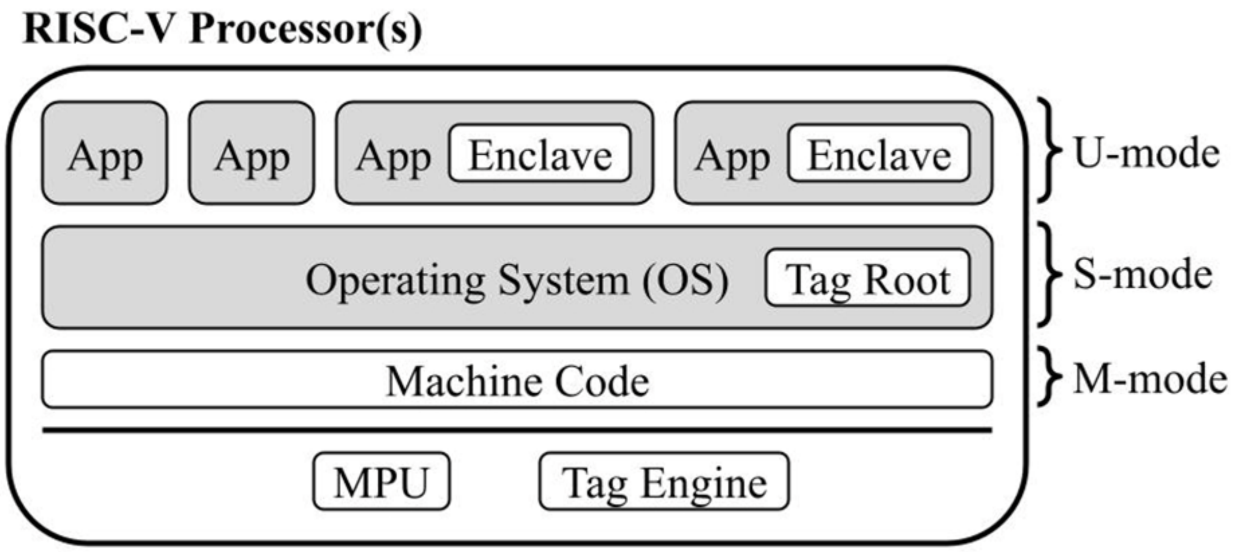
Hex-Five MultiZone

Sanctum [11]: similar approach with Intel SGX, but for RISC-V processors



Sanctum

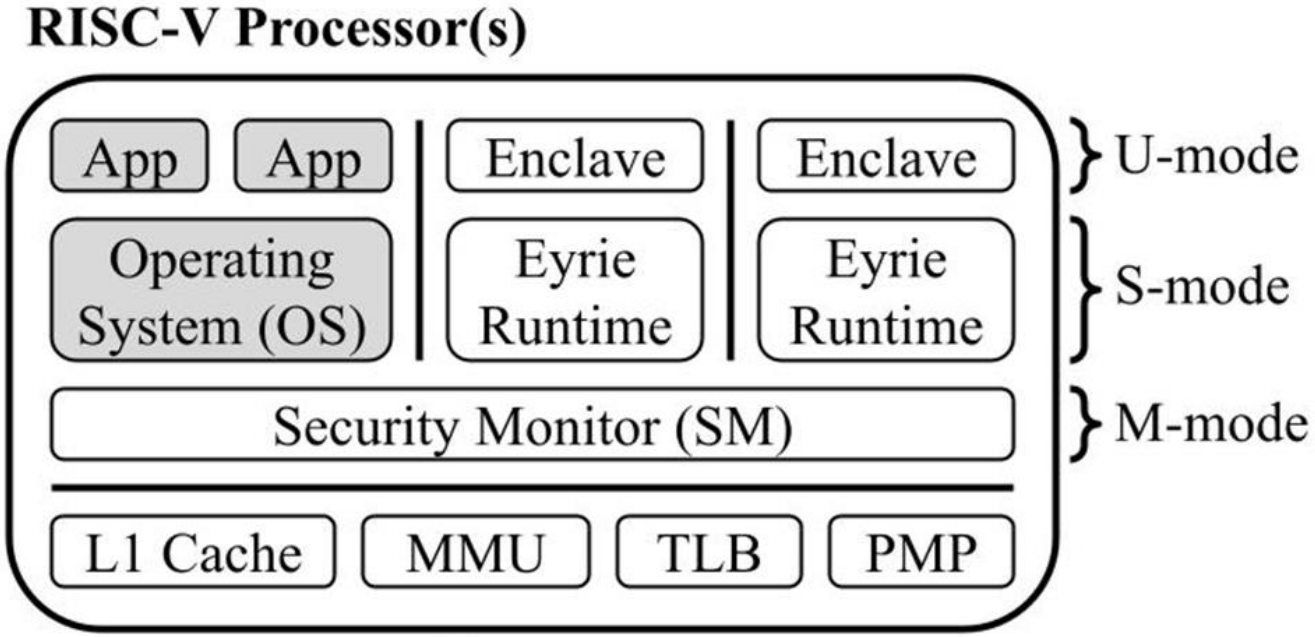
MultiZone [10]: lightweight TEE, multi-purposes, aiming for embedded/IoT applications



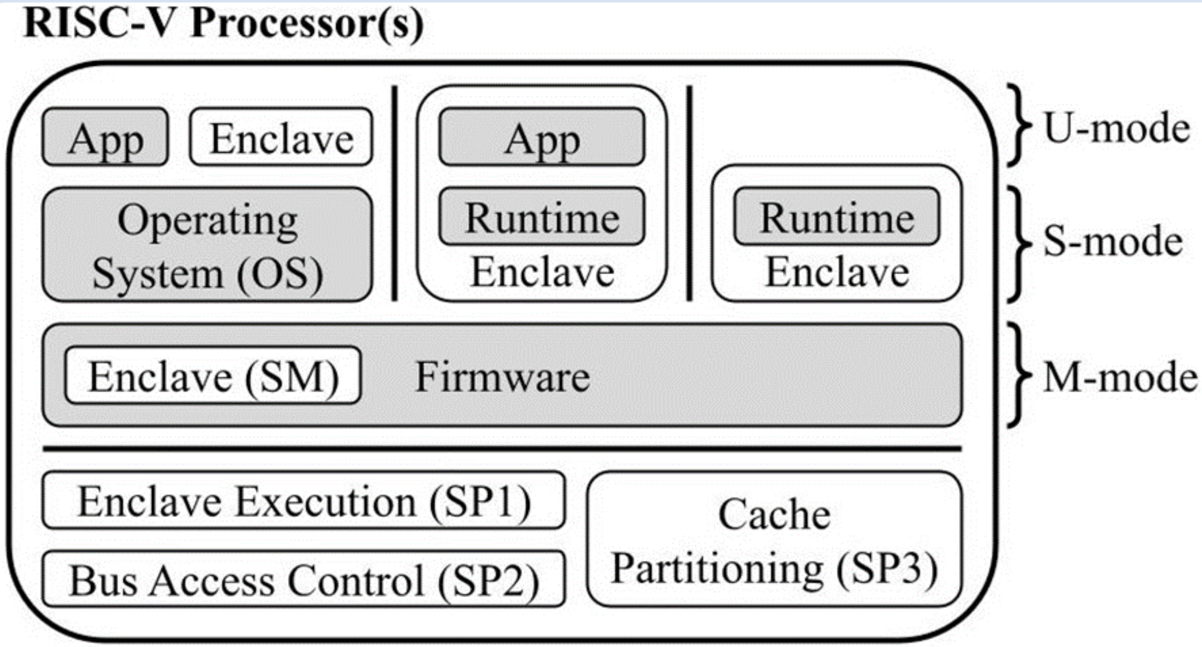
TIMBER-V

TIMBER-V [12]: similar approach with Intel SGX, but uses strong hardware enforcers based on “Tag”-ID across the entire system

3. Secure Boot for TEE (6/23) TEE implementations



Keystone

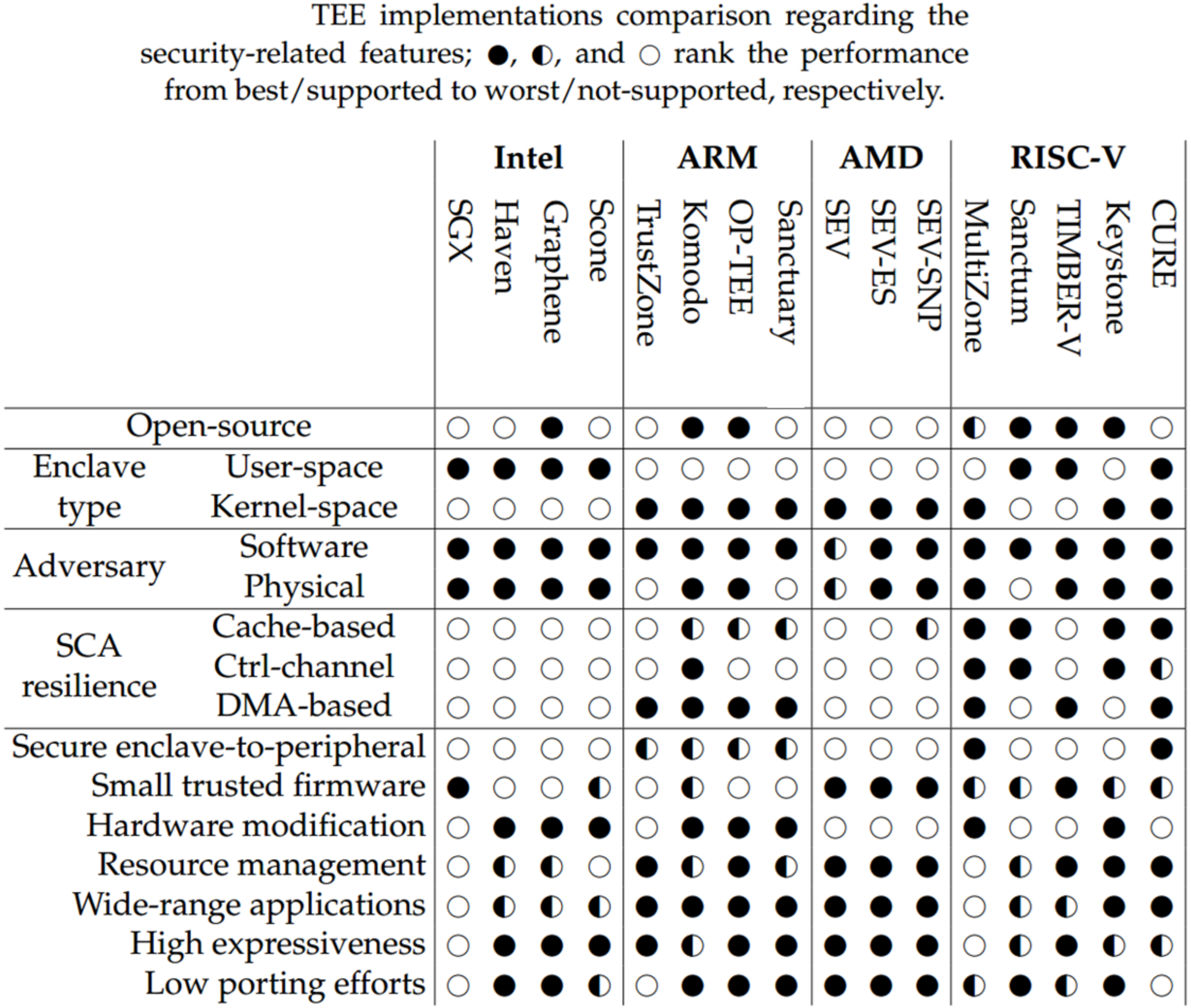


CURE

Keystone [13]: is not a specific type of TEE, but a modular TEE framework *(try its best to be hardware-agnostic)*

CURE [14]: a complete opposite with Keystone, this TEE model requires a total hardware modification across every architectural level *(but provides strong isolation with multiple types of enclaves)*

3. Secure Boot for TEE (7/23) TEEs comparison



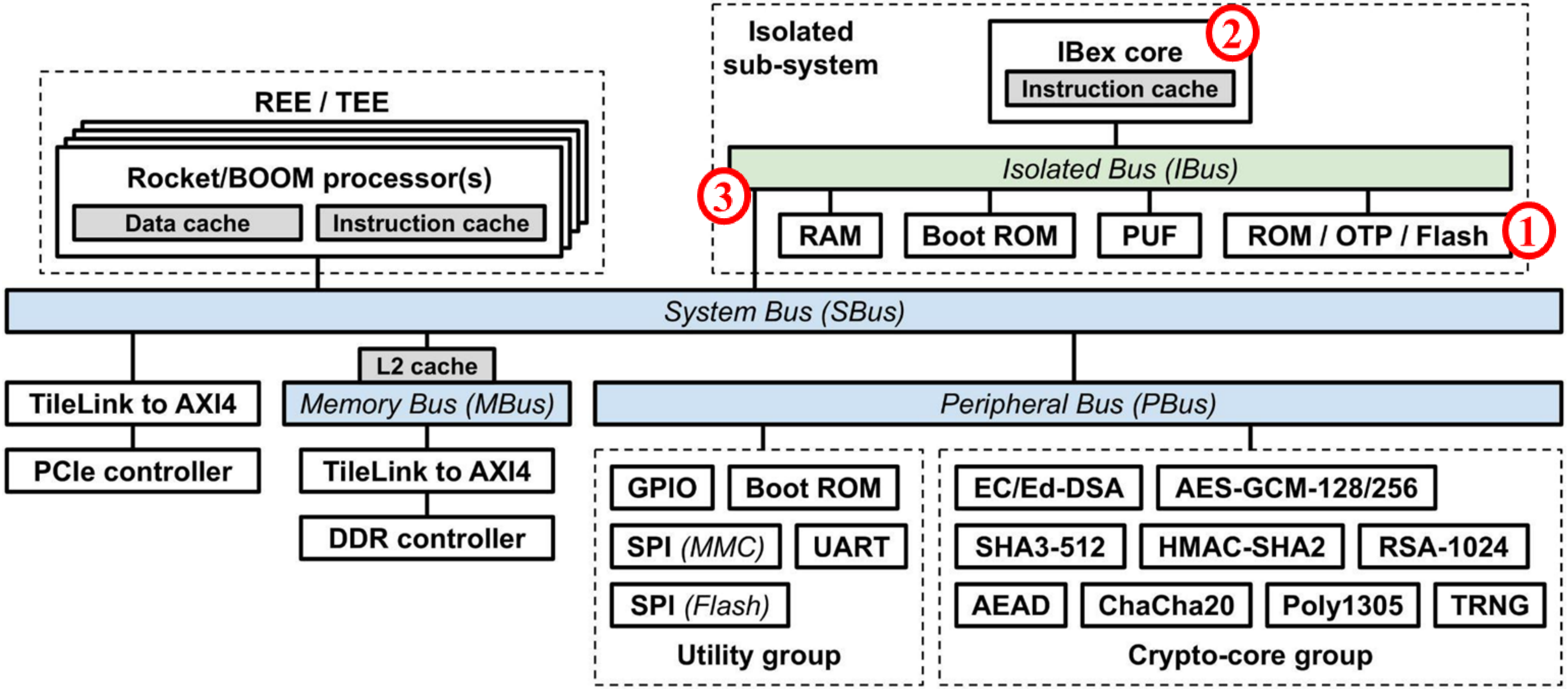
TEE implementations comparison regarding the security-related features; ●, ◐, and ○ rank the performance from best/supported to worst/not-supported, respectively.

		Intel				ARM				AMD			RISC-V				
		SGX	Haven	Graphene	Scone	TrustZone	Komodo	OP-TEE	Sanctuary	SEV	SEV-ES	SEV-SNP	MultiZone	Sanctum	TIMBER-V	Keystone	CURE
Open-source		○	○	●	○	○	●	●	○	○	○	○	◐	●	●	●	○
Enclave type	User-space	●	●	●	●	○	○	○	○	○	○	○	○	●	●	○	●
	Kernel-space	○	○	○	○	●	●	●	●	●	●	●	●	○	○	●	●
Adversary	Software	●	●	●	●	●	●	●	●	◐	●	●	●	●	●	●	●
	Physical	●	●	●	●	○	●	●	○	◐	●	●	●	○	●	●	●
SCA resilience	Cache-based	○	○	○	○	○	◐	◐	◐	○	○	◐	●	●	○	●	●
	Ctrl-channel	○	○	○	○	○	●	○	○	○	○	○	●	●	○	●	◐
	DMA-based	○	○	○	○	●	●	●	●	○	○	○	●	○	●	○	●
Secure enclave-to-peripheral		○	○	○	○	◐	◐	◐	◐	○	○	○	●	○	○	○	●
Small trusted firmware		●	○	○	◐	○	◐	○	○	●	●	●	◐	◐	●	◐	◐
Hardware modification		○	●	●	●	○	●	●	●	○	○	○	●	○	○	●	○
Resource management		○	◐	◐	○	●	◐	●	◐	●	●	●	○	◐	●	●	●
Wide-range applications		○	◐	◐	◐	●	●	●	●	●	●	●	○	◐	◐	●	●
High expressiveness		○	●	●	●	●	◐	●	●	●	●	●	○	◐	●	◐	◐
Low porting efforts		○	●	●	◐	○	●	●	●	●	●	●	◐	●	◐	●	○

- Choose Keystone for the software implementation:
  - **Open-source:** modular TEE framework, versatile usage.
  - **Kernel-space enclave:** better isolation in general.
  - **Hardware-agnostic:** does not require any special custom-built features to function.

3. Secure Boot for TEE (8/23) Propose architecture

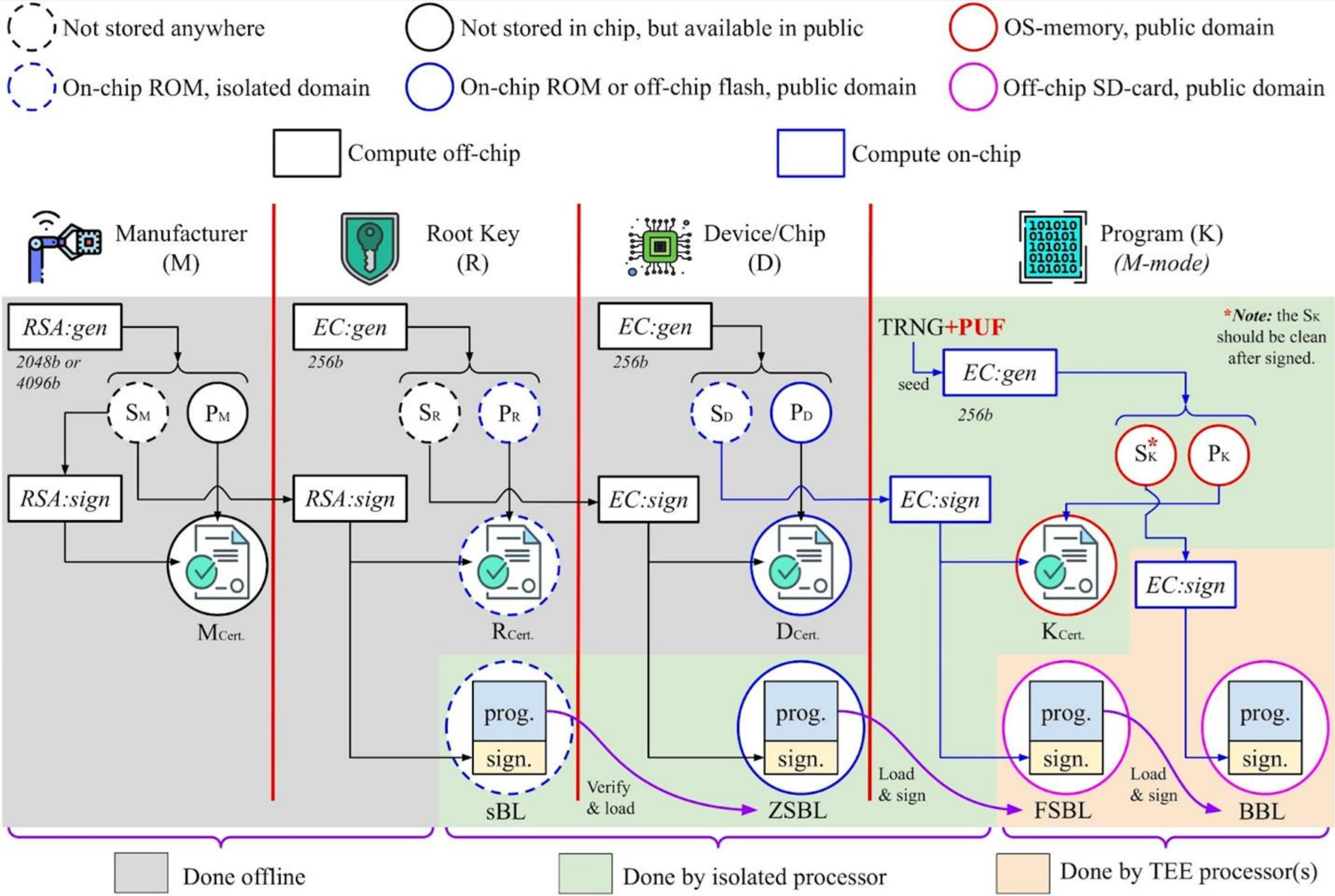
Propose: A secure boot process with RoT for TEE



Main points of the proposed architecture:

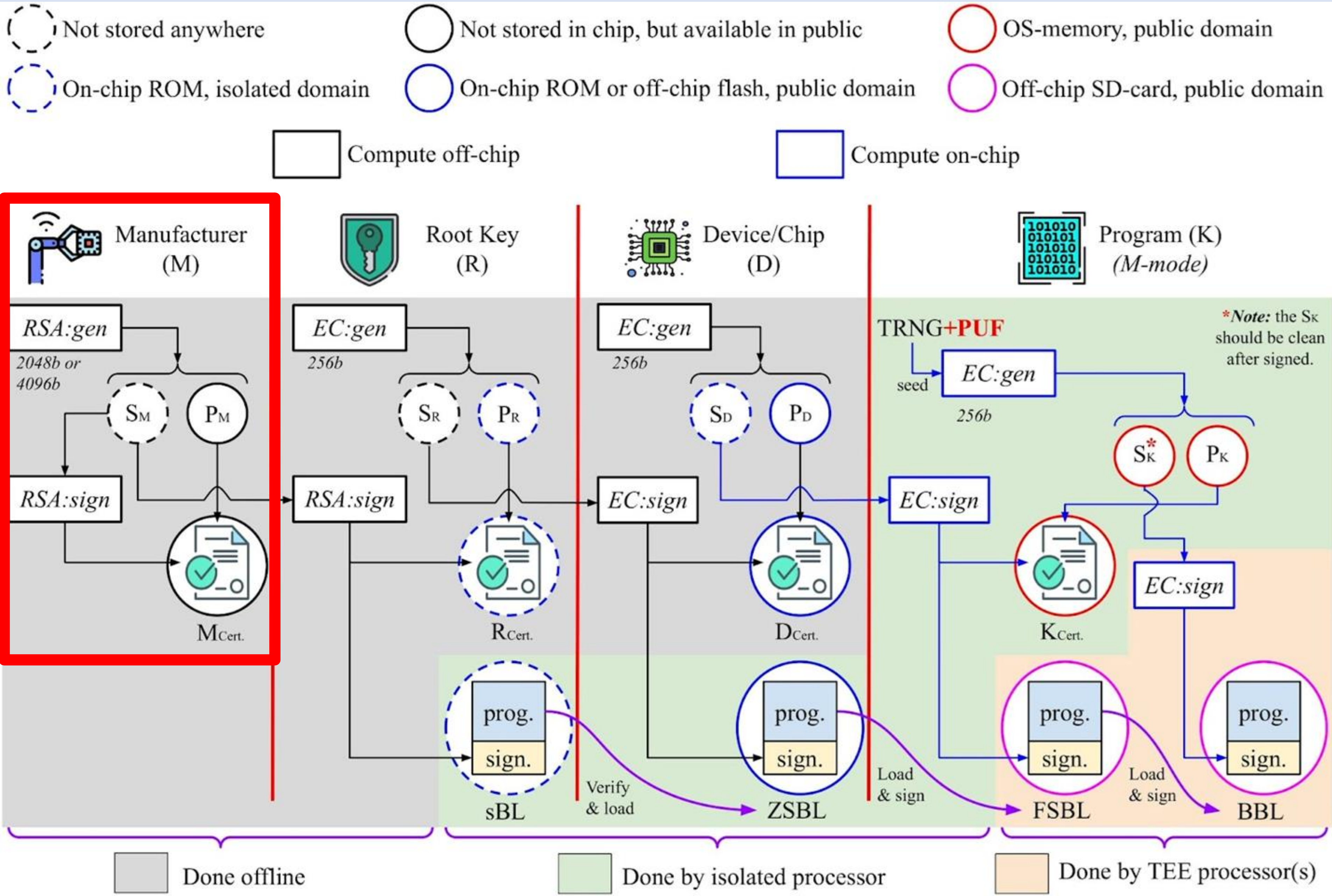
1. Root key installed at the time manufactured.
2. Hidden MCU for the flexible boot program
3. Hierarchy-bus: TEE processors cannot access RAM/ROMs in the isolated domain *(BUT the isolated core can access ALL)*

3. Secure Boot for TEE (9/23) Secure boot process



The proposed keys scheduling scheme.

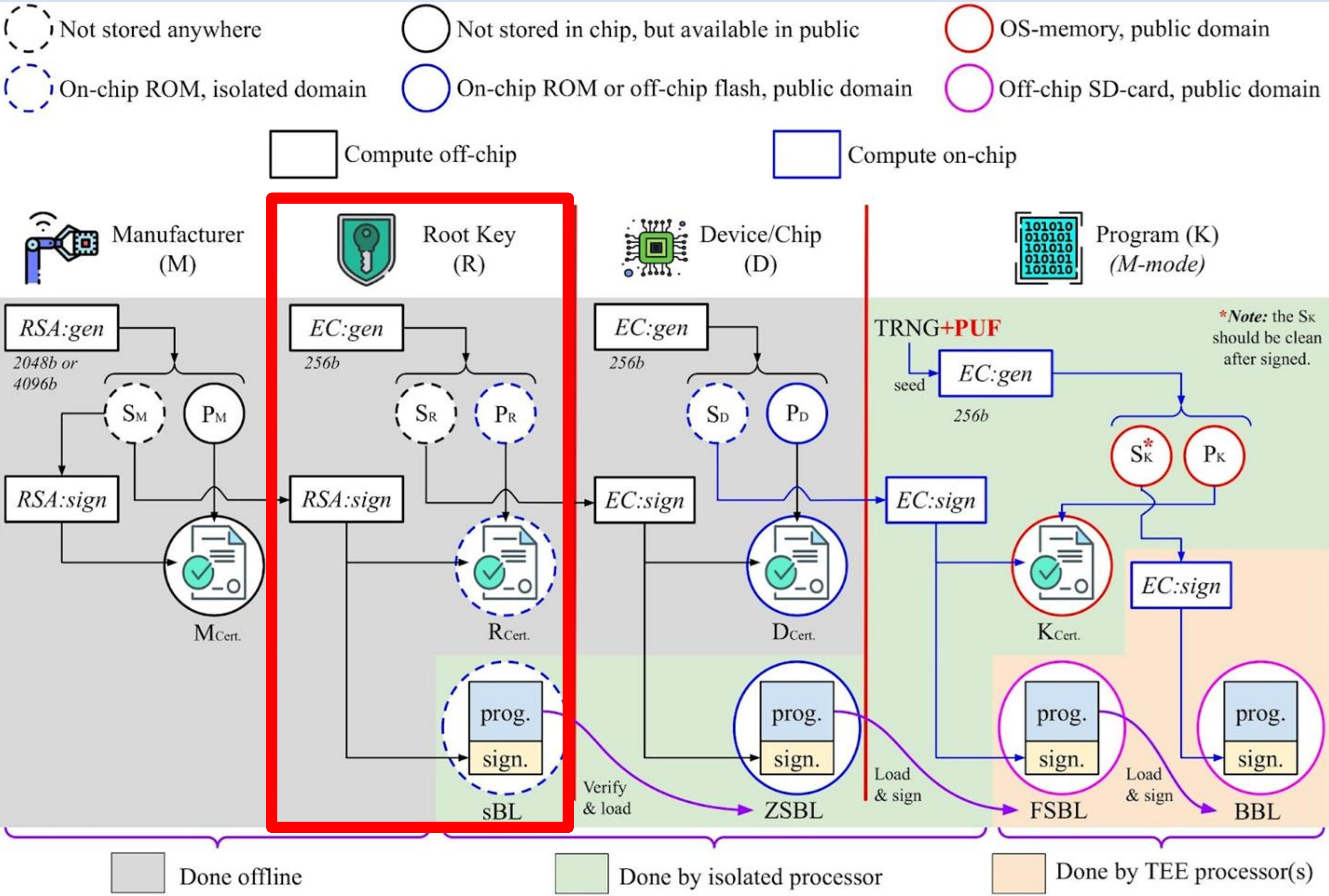
3. Secure Boot for TEE (10/23) Secure boot process



Step-by-step

- **Step 1:** The manufacturer plays the role of root CA *(public key is well-known & certificate is self-signed)*

3. Secure Boot for TEE (11/23) Secure boot process

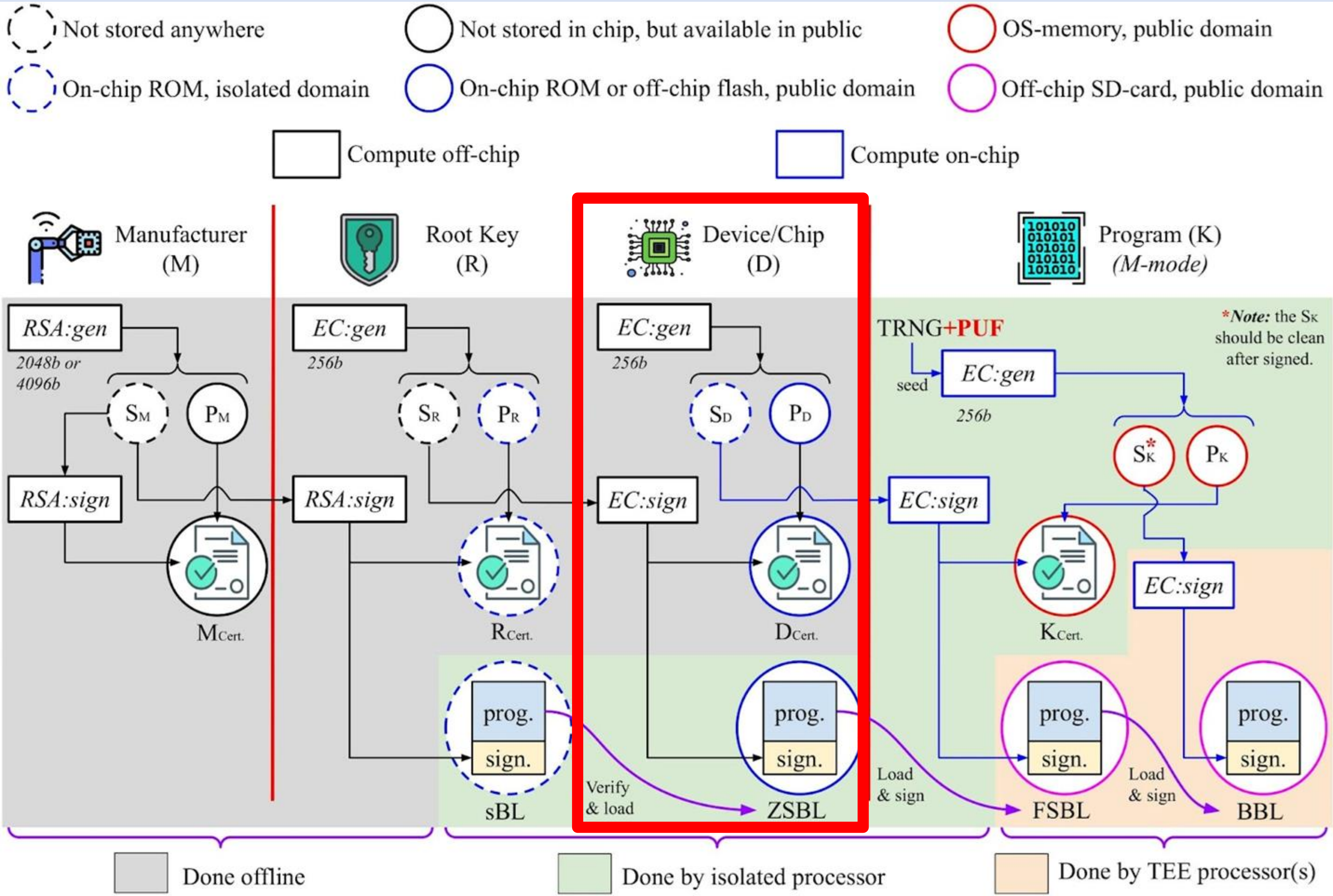


Step-by-step

- **Step 2:** manufacturer generate root SR & PR also offline, and then uses SM to sign on the PR and secure BootLoader (sBL)

sBL is stored in the same place with PR, the isolated ROM.

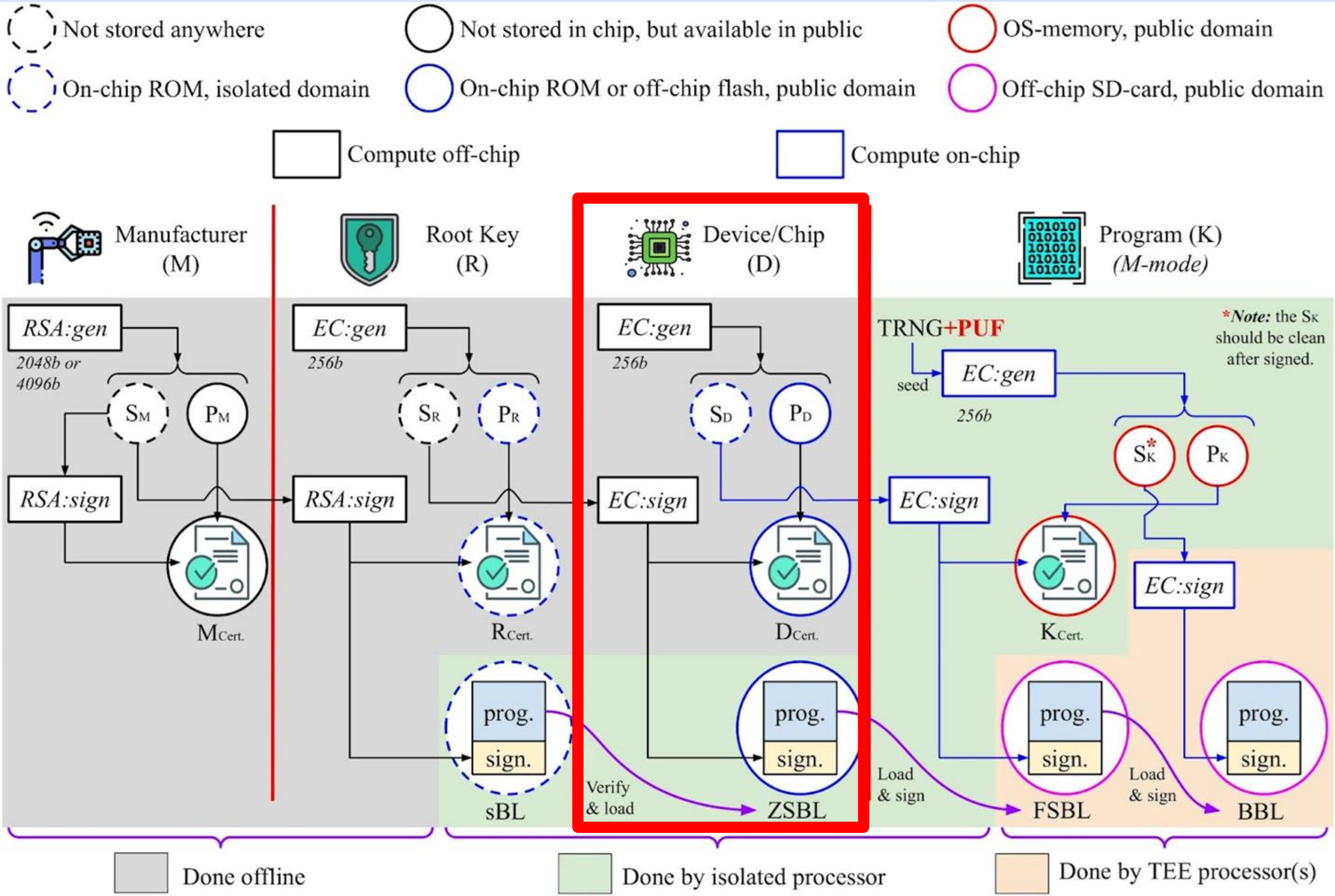
3. Secure Boot for TEE (12/23) Secure boot process



Step-by-step

- **Step 3:** *(still offline)* the manufacturer *(or the provider)* generates the pair S_D & P_D.
  Then have the root secret key generates the $D_{Cert.}$ and sign the ZSBL.

3. Secure Boot for TEE (13/23) Secure boot process

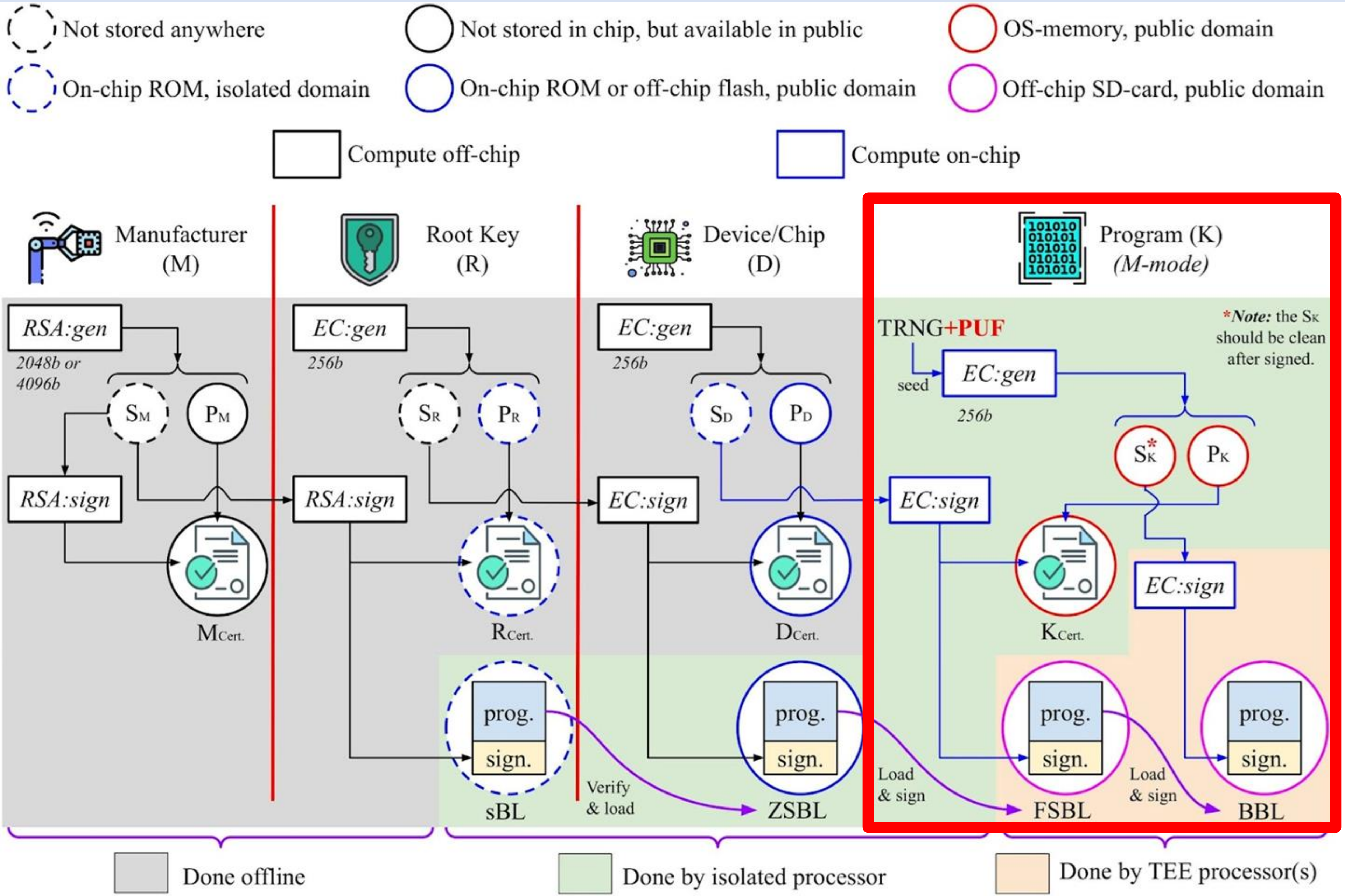


Here is the RoT

- S_D is stored in the isolated ROM.
- ZSBL & P_D could be in a flash outside.
- The very first task of the isolated processor is:
  - Verify the ZSBL signature by using the P_R

→ this allows future updates on the ZSBL.

3. Secure Boot for TEE (14/23) Secure boot process

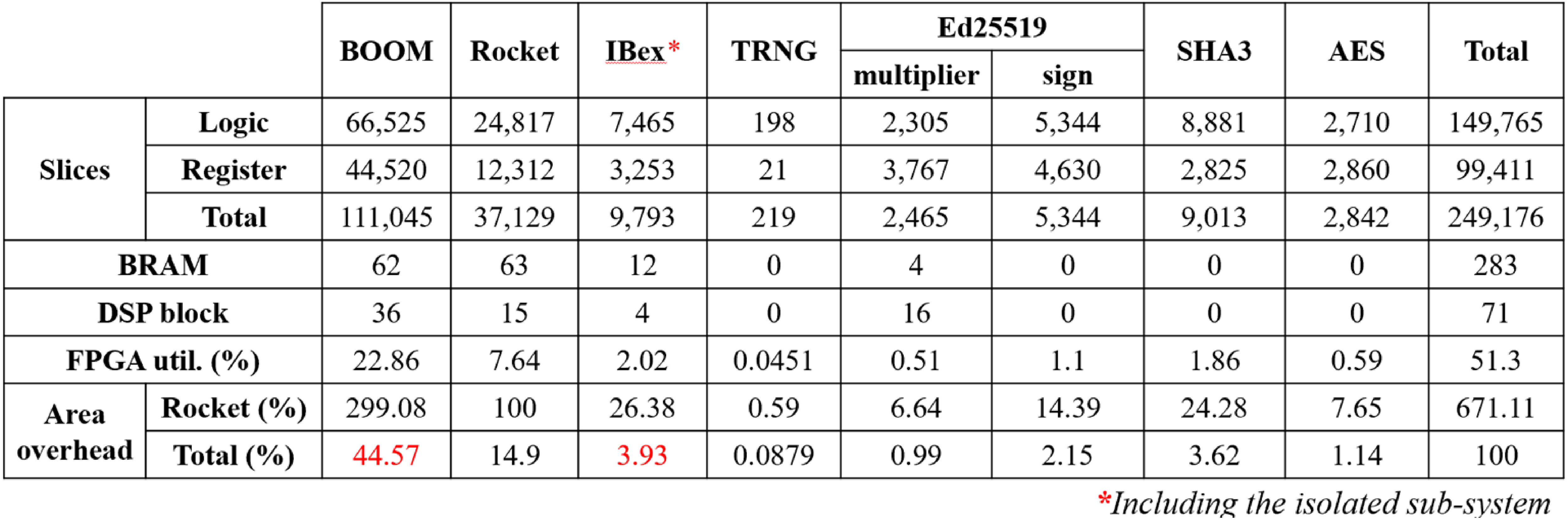


Step-by-step

- **Step 4:** *(now on-chip)* the isolated processor executes the ZSBL and:
    - Use TRNG to seed EC-genkey & create the pair of S_K & P_K
    - Load the FSBL *(hash & sign)* to the public RAM.
    - Wakes up the TEE processors

3. Secure Boot for TEE (15/23) FPGA result

Build Reports of the Proposed TEE SoC in Virtex-7 FPGA (XC7VX485T)



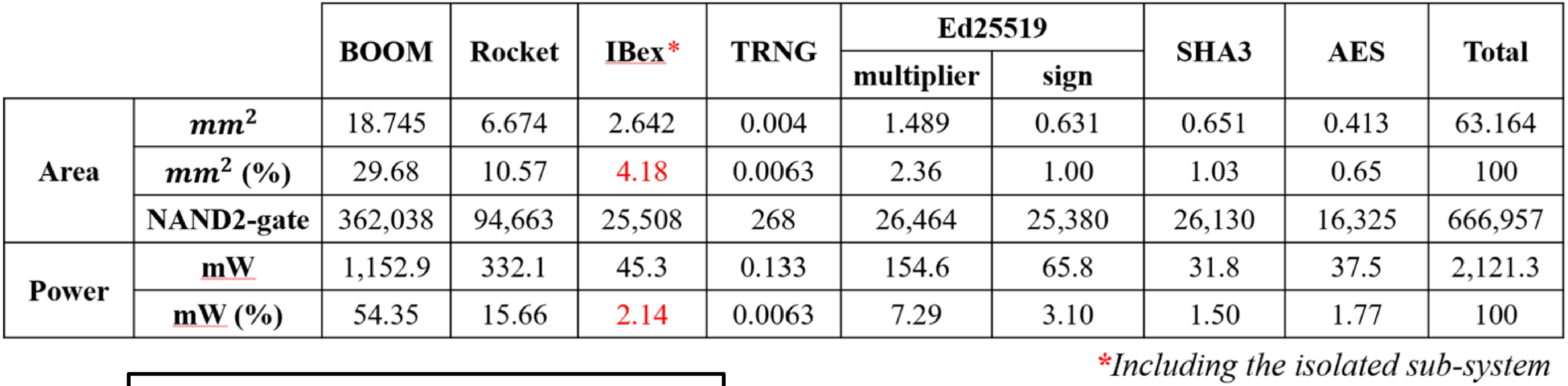
		BOOM	Rocket	IBex*	TRNG	Ed25519		SHA3	AES	Total
						multiplier	sign			
Slices	Logic	66,525	24,817	7,465	198	2,305	5,344	8,881	2,710	149,765
	Register	44,520	12,312	3,253	21	3,767	4,630	2,825	2,860	99,411
	Total	111,045	37,129	9,793	219	2,465	5,344	9,013	2,842	249,176
BRAM		62	63	12	0	4	0	0	0	283
DSP block		36	15	4	0	16	0	0	0	71
FPGA util. (%)		22.86	7.64	2.02	0.0451	0.51	1.1	1.86	0.59	51.3
Area overhead	Rocket (%)	299.08	100	26.38	0.59	6.64	14.39	24.28	7.65	671.11
	Total (%)	44.57	14.9	3.93	0.0879	0.99	2.15	3.62	1.14	100

**Including the isolated sub-system*

- Build with default configuration:
  - ISA: RV64IMAFDC.
  - Cache: 16-KB for inst. & 16-KB for data.
  - L2 cache: 512-KB.
  - Isolated sub-system: included.
  - PCIe: excluded.

3. Secure Boot for TEE (16/23) VLSI result

Synthesis results of the Proposed TEE SoC in ROHM-180nm.



		BOOM	Rocket	IBex*	TRNG	Ed25519		SHA3	AES	Total
						multiplier	sign			
Area	mm^2	18.745	6.674	2.642	0.004	1.489	0.631	0.651	0.413	63.164
	mm^2 (%)	29.68	10.57	4.18	0.0063	2.36	1.00	1.03	0.65	100
	NAND2-gate	362,038	94,663	25,508	268	26,464	25,380	26,130	16,325	666,957
Power	mW	1,152.9	332.1	45.3	0.133	154.6	65.8	31.8	37.5	2,121.3
	mW (%)	54.35	15.66	2.14	0.0063	7.29	3.10	1.50	1.77	100

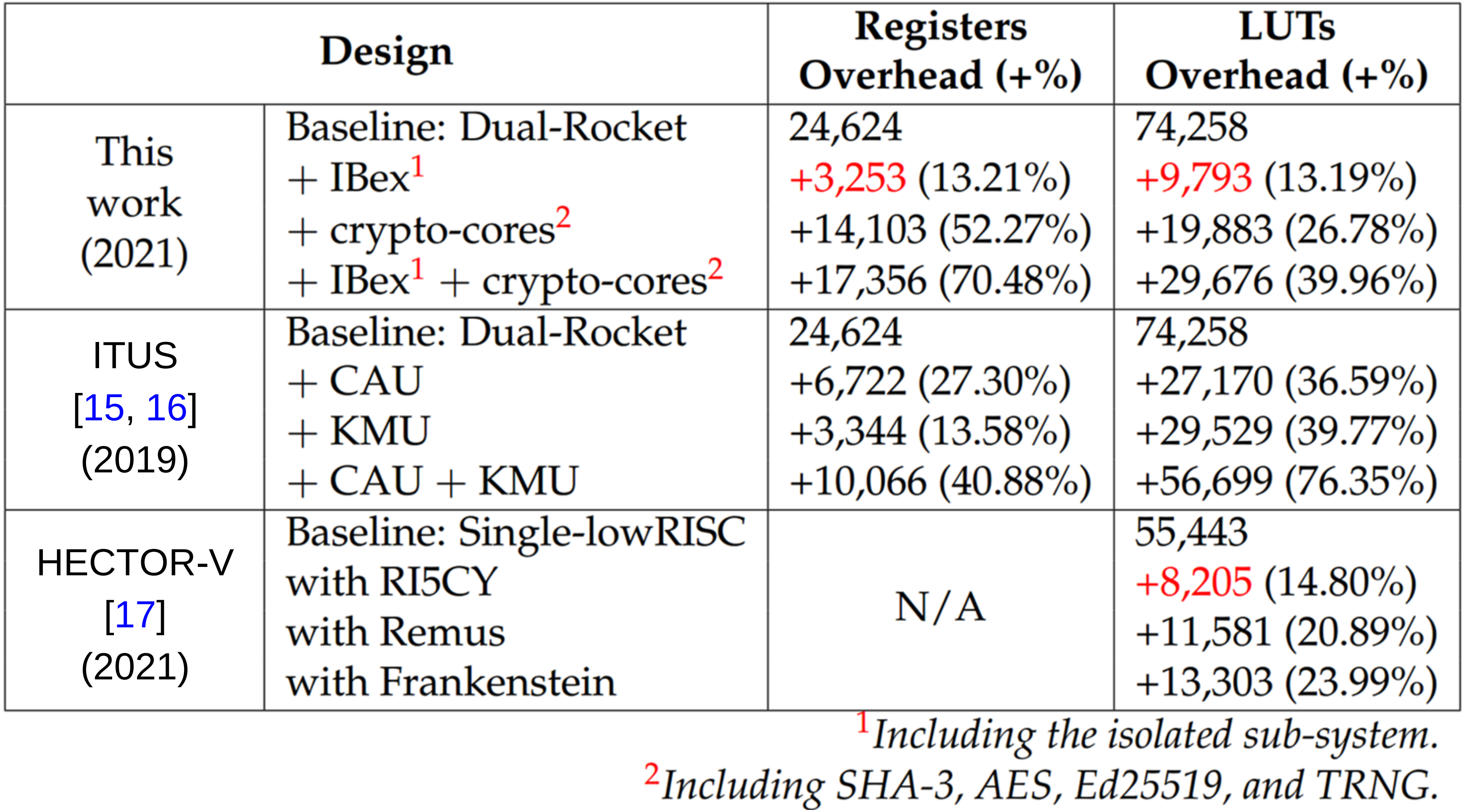
**Including the isolated sub-system*

Note:* the used tools are Cadence’.

- Build with default configuration:
  - ISA: RV64IMAFDC.
  - Cache: 16-KB for inst. & 16-KB for data.
  - L2 cache: 512-KB.
  - Isolated sub-system: included.
  - PCIe: excluded.

3. Secure Boot for TEE (17/23) Comparison

Comparison with other secure-boot RISC-V-based TEE SoCs.



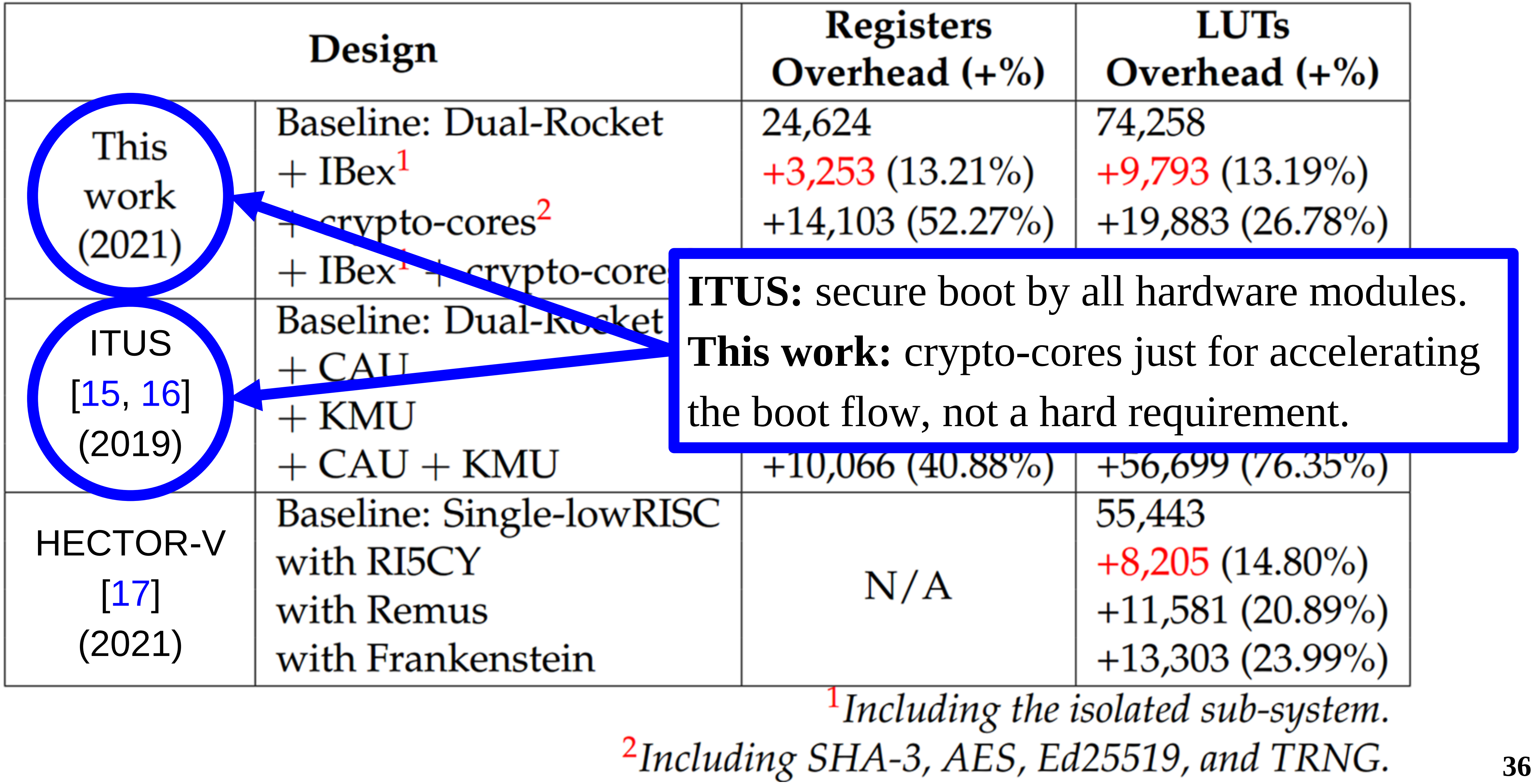
Design		Registers Overhead (+%)	LUTs Overhead (+%)
This work (2021)	Baseline: Dual-Rocket	24,624	74,258
	+ IBex[1]	+3,253 (13.21%)	+9,793 (13.19%)
	+ crypto-cores[2]	+14,103 (52.27%)	+19,883 (26.78%)
	+ IBex[1] + crypto-cores[2]	+17,356 (70.48%)	+29,676 (39.96%)
ITUS [15, 16] (2019)	Baseline: Dual-Rocket	24,624	74,258
	+ CAU	+6,722 (27.30%)	+27,170 (36.59%)
	+ KMU	+3,344 (13.58%)	+29,529 (39.77%)
	+ CAU + KMU	+10,066 (40.88%)	+56,699 (76.35%)
HECTOR-V [17] (2021)	Baseline: Single-lowRISC	N/A	55,443
	with RI5CY		+8,205 (14.80%)
	with Remus		+11,581 (20.89%)
	with Frankenstein		+13,303 (23.99%)

[1] *Including the isolated sub-system.*

[2] *Including SHA-3, AES, Ed25519, and TRNG.*

3. Secure Boot for TEE (18/23) Comparison

Comparison with other secure-boot RISC-V-based TEE SoCs.



Design		Registers Overhead (+%)	LUTs Overhead (+%)
This work (2021)	Baseline: Dual-Rocket	24,624	74,258
	+ IBex[1]	+3,253 (13.21%)	+9,793 (13.19%)
	+ crypto-cores[2]	+14,103 (52.27%)	+19,883 (26.78%)
	+ IBex[1] + crypto-core	[illegible]	[illegible]
ITUS [15, 16] (2019)	Baseline: Dual-Rocket	[illegible]	[illegible]
	+ CAU	[illegible]	[illegible]
	+ KMU	[illegible]	[illegible]
	+ CAU + KMU	+10,066 (40.88%)	+56,699 (76.35%)
HECTOR-V [17] (2021)	Baseline: Single-lowRISC	N/A	55,443
	with RI5CY		+8,205 (14.80%)
	with Remus		+11,581 (20.89%)
	with Frankenstein		+13,303 (23.99%)

[1] *Including the isolated sub-system.*

[2] *Including SHA-3, AES, Ed25519, and TRNG.*

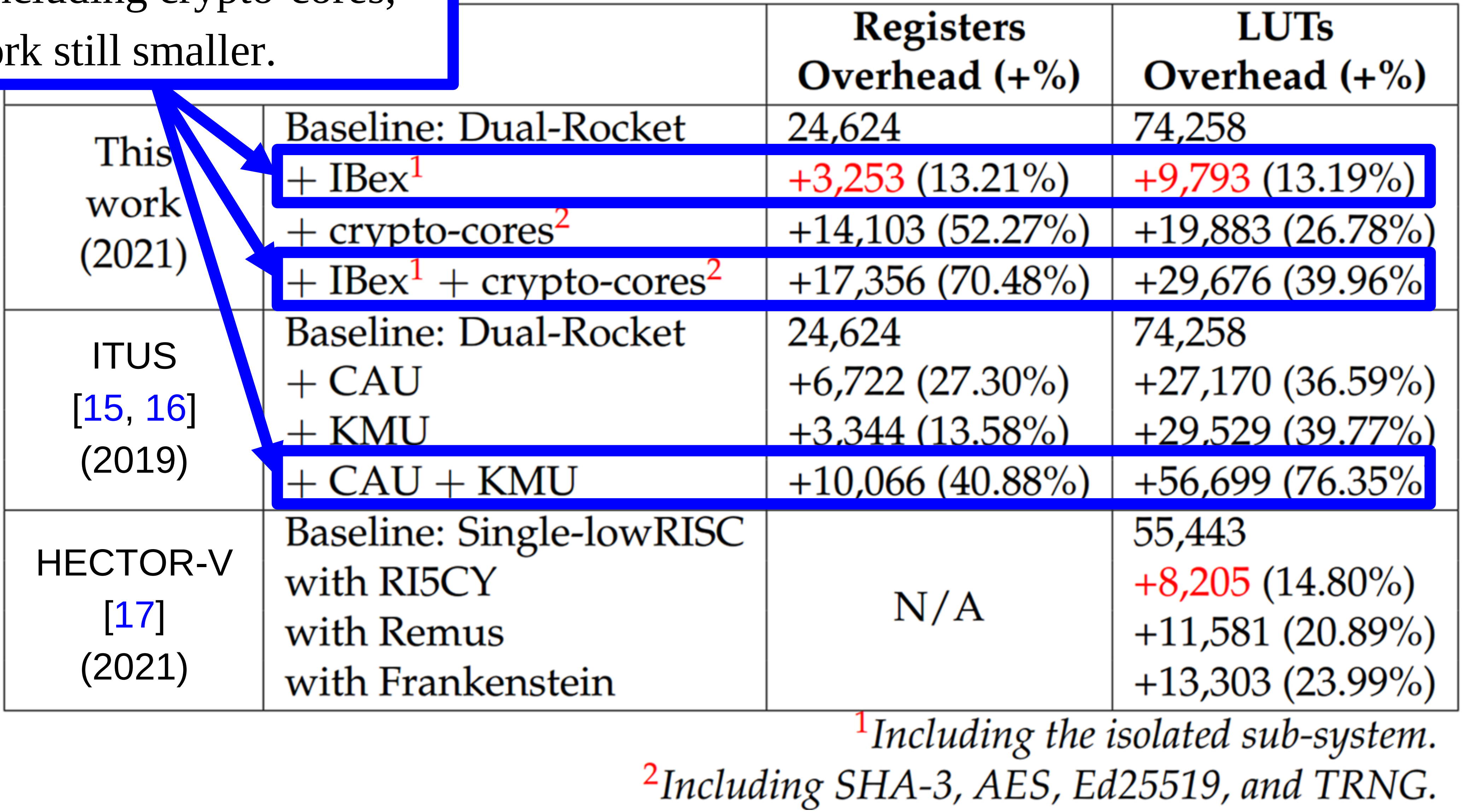
ITUS: secure boot by all hardware modules.

This work: crypto-cores just for accelerating the boot flow, not a hard requirement.

3. Secure Boot for TEE (19/23) Comparison

Even including crypto-cores,
this work still smaller.

other secure-boot RISC-V-based TEE SoCs.



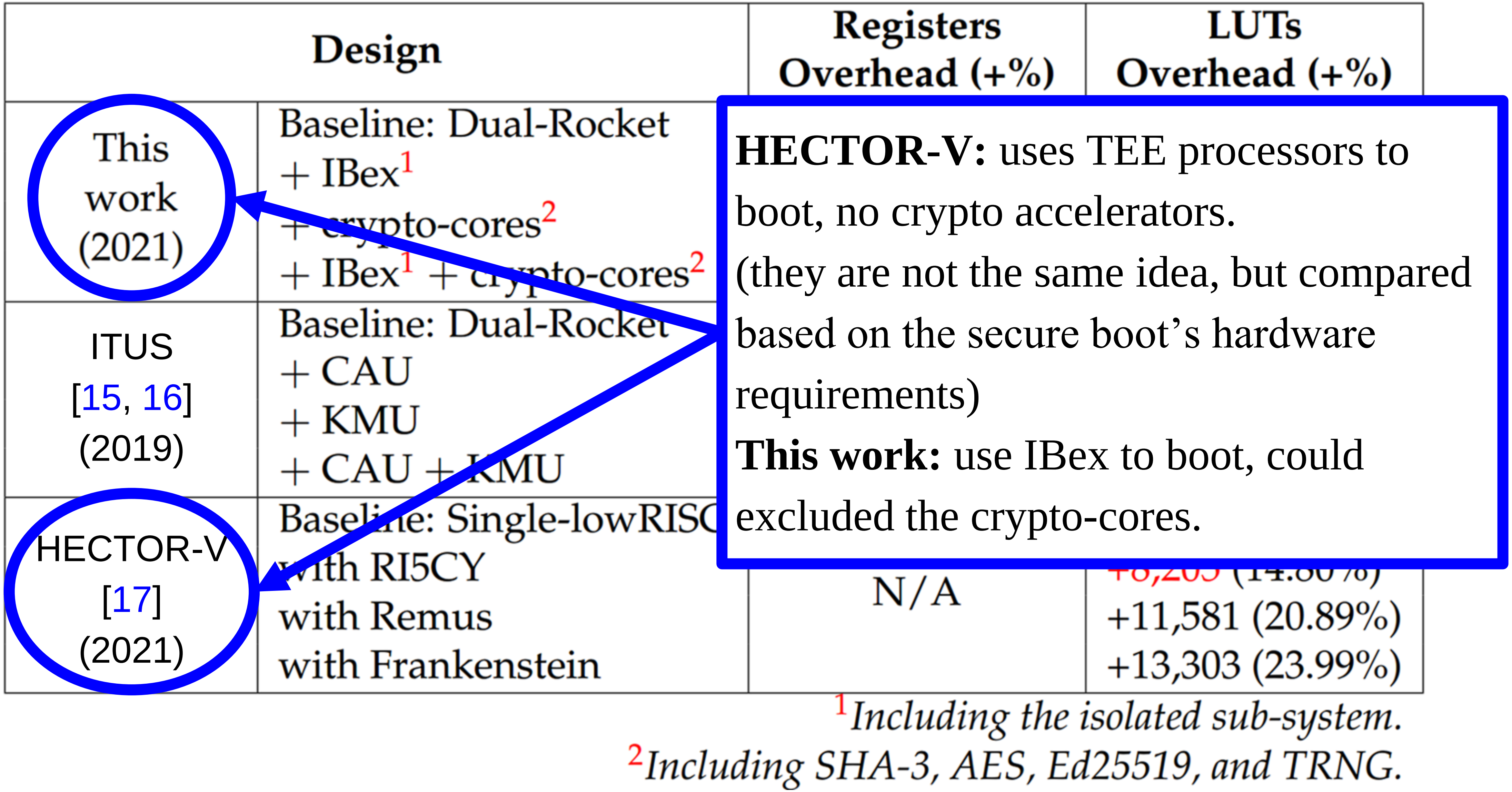
		Registers Overhead (+%)	LUTs Overhead (+%)
This work (2021)	Baseline: Dual-Rocket	24,624	74,258
	+ IBex[1]	+3,253 (13.21%)	+9,793 (13.19%)
	+ crypto-cores[2]	+14,103 (52.27%)	+19,883 (26.78%)
	+ IBex[1] + crypto-cores[2]	+17,356 (70.48%)	+29,676 (39.96%
ITUS [15, 16] (2019)	Baseline: Dual-Rocket	24,624	74,258
	+ CAU	+6,722 (27.30%)	+27,170 (36.59%)
	+ KMU	+3,344 (13.58%)	+29,529 (39.77%)
	+ CAU + KMU	+10,066 (40.88%)	+56,699 (76.35%
HECTOR-V [17] (2021)	Baseline: Single-lowRISC	N/A	55,443
	with RI5CY		+8,205 (14.80%)
	with Remus		+11,581 (20.89%)
	with Frankenstein		+13,303 (23.99%)

[1] *Including the isolated sub-system.*

[2] *Including SHA-3, AES, Ed25519, and TRNG.*

3. Secure Boot for TEE (20/23) Comparison

Comparison with other secure-boot RISC-V-based TEE SoCs.



Design		Registers Overhead (+%)	LUTs Overhead (+%)
This work (2021)	Baseline: Dual-Rocket	[illegible]	[illegible]
	+ IBex[1]	[illegible]	[illegible]
	+ crypto-cores[2]	[illegible]	[illegible]
	+ IBex[1] + crypto-cores[2]	[illegible]	[illegible]
ITUS [15, 16] (2019)	Baseline: Dual-Rocket	[illegible]	[illegible]
	+ CAU	[illegible]	[illegible]
	+ KMU	[illegible]	[illegible]
	+ CAU + KMU	[illegible]	[illegible]
HECTOR-V [17] (2021)	Baseline: Single-lowRISC	N/A	[illegible]
	with RI5CY		+8,203 (14.80%)
	with Remus		+11,581 (20.89%)
	with Frankenstein		+13,303 (23.99%)

[1] *Including the isolated sub-system.*

[2] *Including SHA-3, AES, Ed25519, and TRNG.*

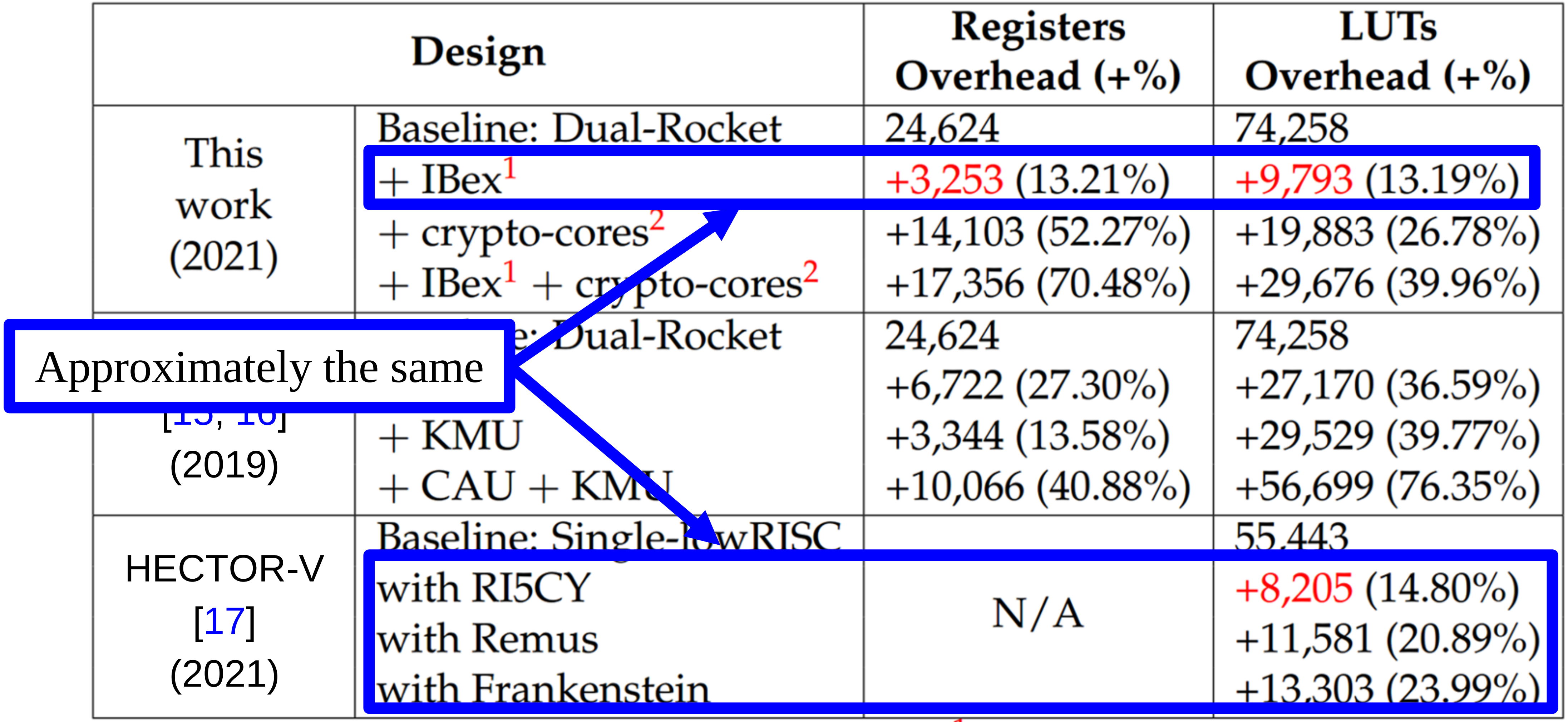
HECTOR-V: uses TEE processors to boot, no crypto accelerators.

(they are not the same idea, but compared based on the secure boot's hardware requirements)

This work: use IBex to boot, could excluded the crypto-cores.

3. Secure Boot for TEE (21/23) Comparison

Comparison with other secure-boot RISC-V-based TEE SoCs.

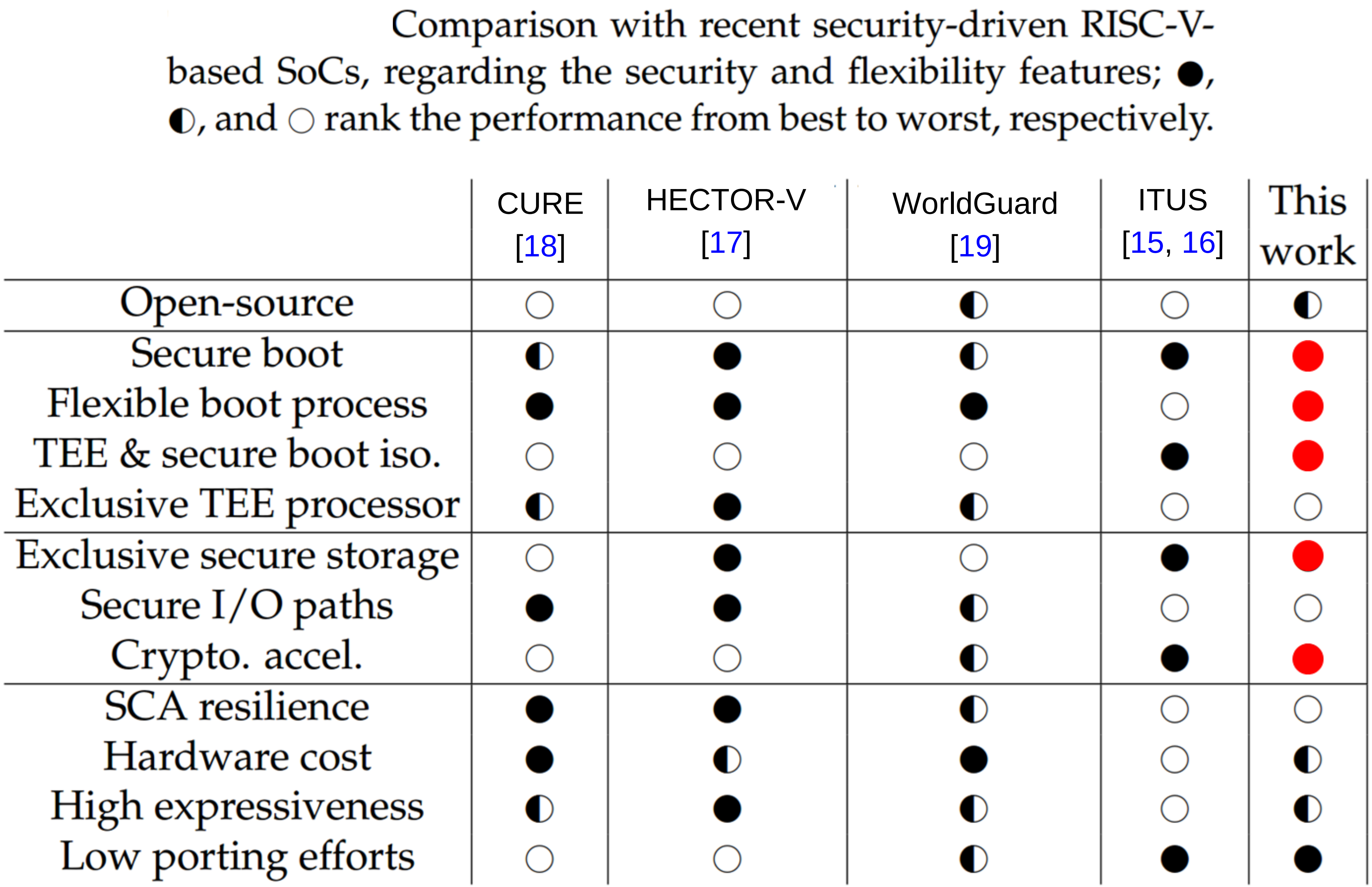


Design		Registers Overhead (+%)	LUTs Overhead (+%)
This work (2021)	Baseline: Dual-Rocket	24,624	74,258
	+ IBex[1]	+3,253 (13.21%)	+9,793 (13.19%)
	+ crypto-cores[2]	+14,103 (52.27%)	+19,883 (26.78%)
	+ IBex[1] + crypto-cores[2]	+17,356 (70.48%)	+29,676 (39.96%)
[15, 16] (2019)	Baseline: Dual-Rocket	24,624	74,258
		+6,722 (27.30%)	+27,170 (36.59%)
	+ KMU	+3,344 (13.58%)	+29,529 (39.77%)
	+ CAU + KMU	+10,066 (40.88%)	+56,699 (76.35%)
HECTOR-V [17] (2021)	Baseline: Single-lowRISC	N/A	55,443
	with RI5CY		+8,205 (14.80%)
	with Remus		+11,581 (20.89%)
	with Frankenstein		+13,303 (23.99%)

Approximately the same

[1] *Including the isolated sub-system.*
[2] *Including SHA-3, AES, Ed25519, and TRNG.*

3. Secure Boot for TEE (22/23) Comparison



Comparison with recent security-driven RISC-V-based SoCs, regarding the security and flexibility features; ●, ◐, and ○ rank the performance from best to worst, respectively.

	CURE [18]	HECTOR-V [17]	WorldGuard [19]	ITUS [15, 16]	This work
Open-source	○	○	◐	○	◐
Secure boot	◐	●	◐	●	🔴
Flexible boot process	●	●	●	○	🔴
TEE & secure boot iso.	○	○	○	●	🔴
Exclusive TEE processor	◐	●	◐	○	○
Exclusive secure storage	○	●	○	●	🔴
Secure I/O paths	●	●	◐	○	○
Crypto. accel.	○	○	◐	●	🔴
SCA resilience	●	●	◐	○	○
Hardware cost	●	◐	●	○	◐
High expressiveness	◐	●	◐	○	◐
Low porting efforts	○	○	◐	●	●

Achieved:

- Secure boot process with RoT for TEE.
- Flexible boot flow.
- Complete isolation between the boot process and the TEE domain.
- Has exclusive storage for boot program only.
- Cryptographic accelerators are available.

3. Secure Boot for TEE (23/23) Summary

Key Achievements

1. **TEE-HW with cryptographic accelerations:** custom hardware was made for accelerating the TEE boot flow.
2. **TEE-HW with isolated RoT:** the heterogeneous architecture was proposed to isolate the RoT from the TEE side.
   - The manufacturer and root keys are stored at the time manufactured.
   - The ability to make a secure direct connection from the isolated bus to outside peripherals.
   - The secure boot flow is executed by the isolated environment.
   - The bootloader program is flexible and can be updated.

OUTLINE

1. Introduction
2. What is RISC-V, and How Does It Affect Cyber-Security?
3. Secure Boot for Trusted Execution Environment (TEE)
4. Cache Side-channel Attack (Spectre) on Out-of-Order (OoO) Processors
5. Prevent Correlation Power Analysis (CPA) with Random Dynamic Frequency Scaling (RDFS)
6. Conclusion

4. Spectre attack in OoO Processors (1/12)

Spectre - Cache side-channel attack

Target: RISC-V Out-of-order BOOM

First variants:

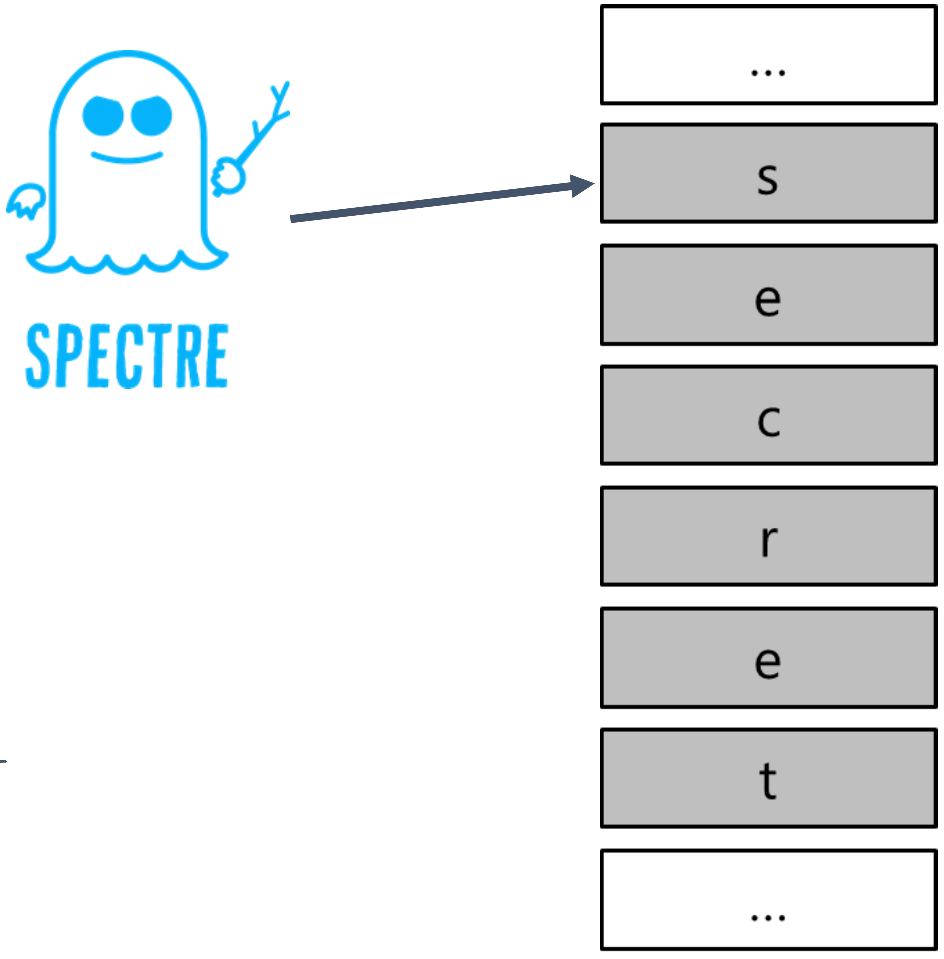
- Spectre v1: Bound Check Bypass
- Spectre v2: Branch Target Injection

BOOM

L1 Cache

- Branch Predictor Unit
- Speculative Execution
- Caching
- …

BOOM suitable for Spectre



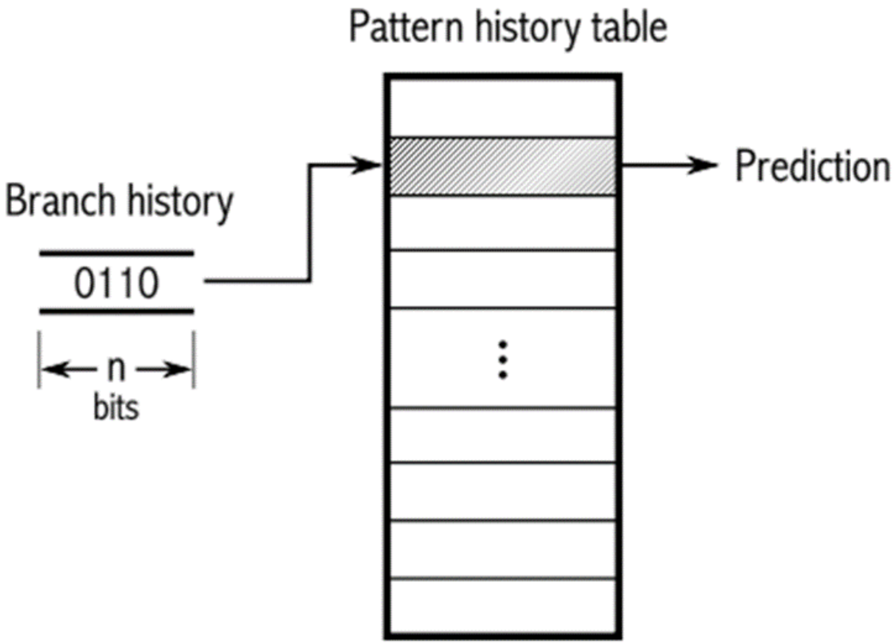
Cache memory

4. Spectre attack in OoO Processors (2/12)

Speculative execution example:

IF (a < 10)
Run B

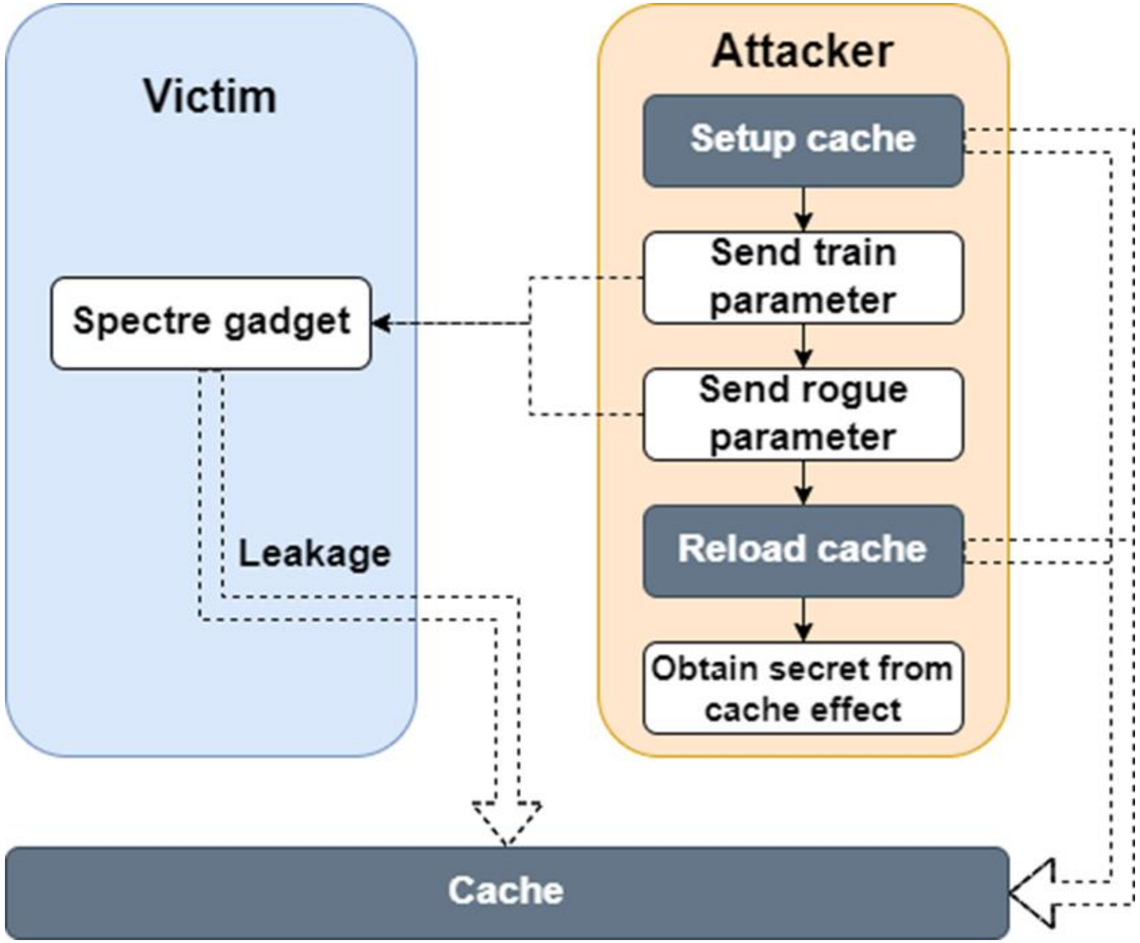
User input	**Process**
a = 1	**TRUE** => Execute **B**
a = 2	**TRUE** => Execute **B**
a = 3	**TRUE** => Execute **B**
…	**TRUE** => Execute **B**
a = X	*Guess as* **TRUE** => Execute **B**



4. Spectre attack in OoO Processors (3/12)

Typical attack strategy:

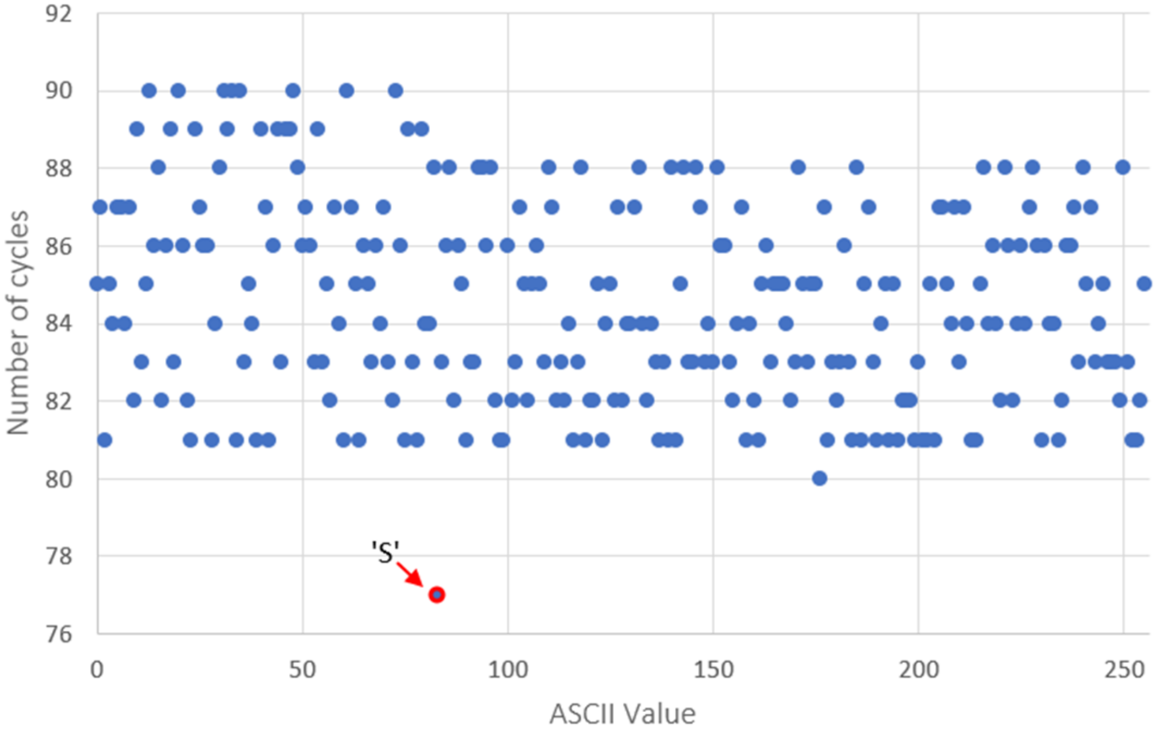
1. Setup processor cache, for example, fill or flush all the cache lines, as in **timing attacks** approaches.
2. Force **mis-speculation** in victim code to leak secret into a side-channel
3. Attacker recovers secret from **side-channel effect** in the cache (usually the **access load time).**



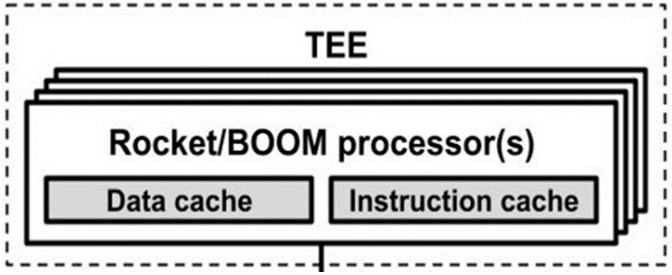
4. Spectre attack in OoO Processors (4/12)

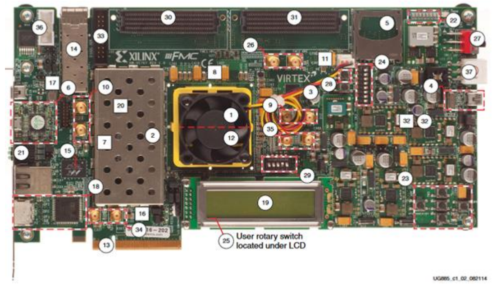
Implement RISC-V processor

- BOOM core: exploited
- Rocket core: not exploited

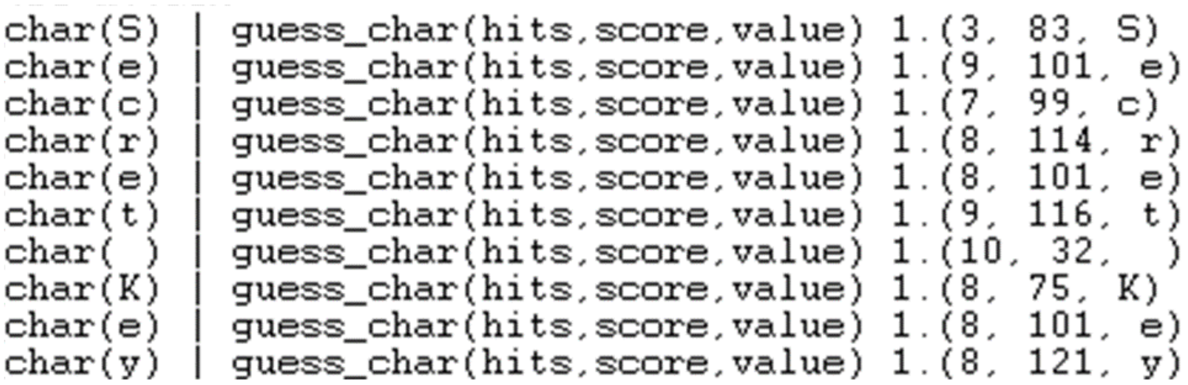


Observe cache accessing time after an attack attempt





FPGA VC707



```
char(S) | guess_char(hits,score,value) 1.(3, 83, S)
char(e) | guess_char(hits,score,value) 1.(9, 101, e)
char(c) | guess_char(hits,score,value) 1.(7, 99, c)
char(r) | guess_char(hits,score,value) 1.(8, 114, r)
char(e) | guess_char(hits,score,value) 1.(8, 101, e)
char(t) | guess_char(hits,score,value) 1.(9, 116, t)
char( ) | guess_char(hits,score,value) 1.(10, 32,  )
char(K) | guess_char(hits,score,value) 1.(8, 75, K)
char(e) | guess_char(hits,score,value) 1.(8, 101, e)
char(y) | guess_char(hits,score,value) 1.(8, 121, y)
```

Attack log (success case)

4. Spectre attack in OoO Processors (5/12)

Software mitigation method

- Fence instructions
- Speculation Load Hardening

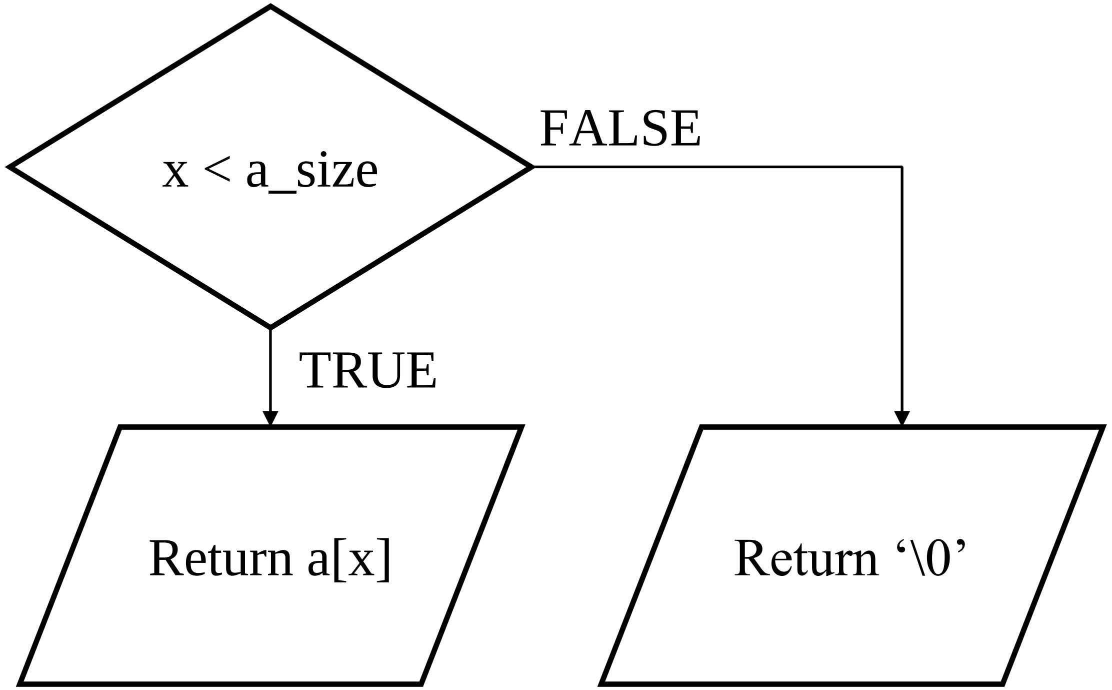
Properties:

- Modify to strengthen victim program
- Require to re-compile source code
- Affect on performance

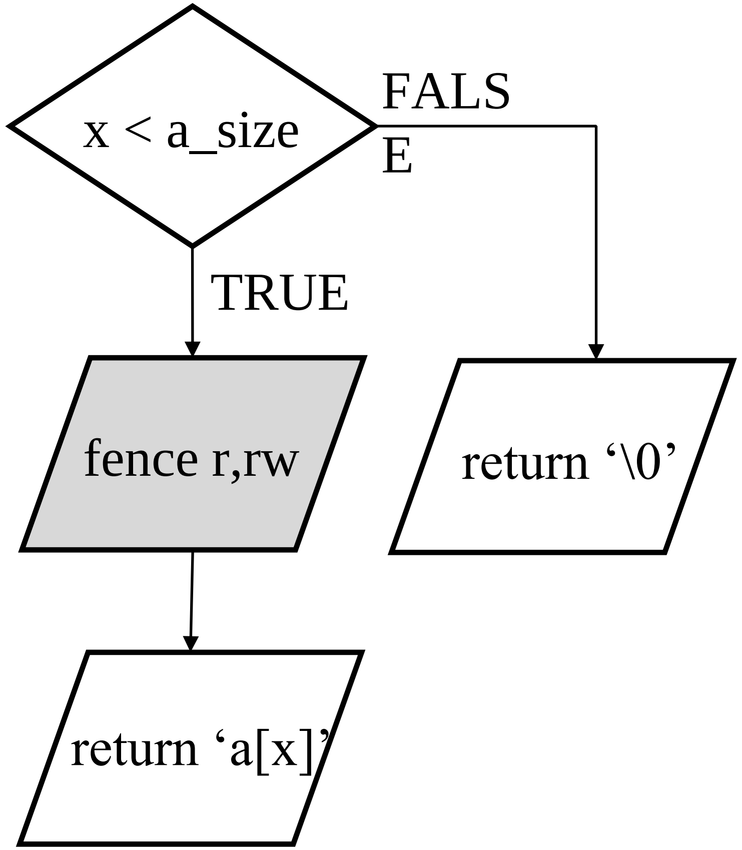
```
if (x < a_size)
     return
a[x];
else
     return ‘0’;
```

Original target for spectre attack

4. Spectre attack in OoO Processors (6/12)



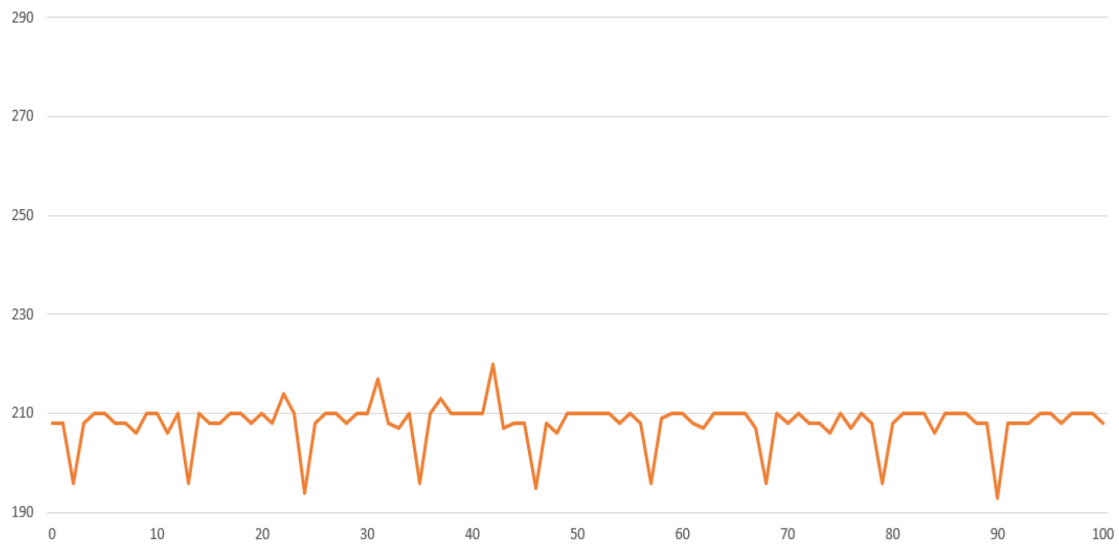
Original code



Fence instructions
=> Force in-order execution

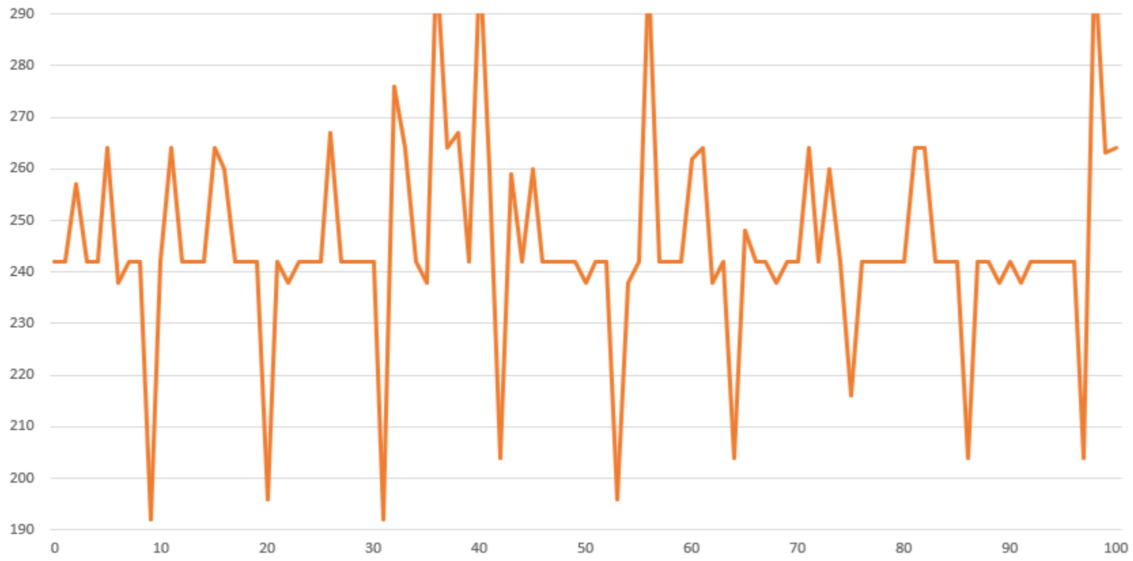
4. Spectre attack in OoO Processors (7/12)

Performance measure



No mitigation

- Normal execution cycle: 210



Mitigation using fence

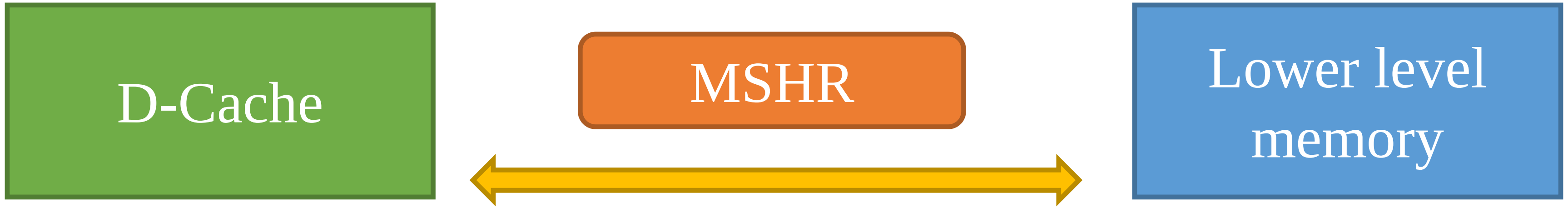
- Normal execution cycle: 242 - 290
- Performance loss: 15 – 43%

4. Spectre attack in OoO Processors (8/12)

Hardware mitigation method: modifying MSHRs

- MSHRs: miss status holding registers
- Located in The Load/Store Unit (LSU)
- Handling data forwarding when mis-speculative events

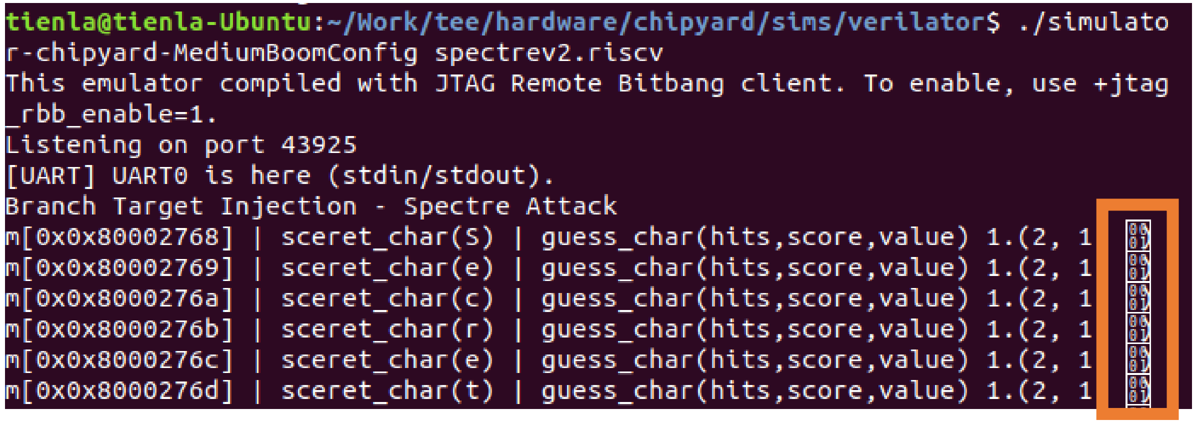
=> Delay the data forwarding when mis-speculative



4. Spectre attack in OoO Processors (9/12)

- Config: Single core RISC-V MediumBoom
- Verilator software simulation
- Secure Boom: modify MSHRs for Spectre-resistant

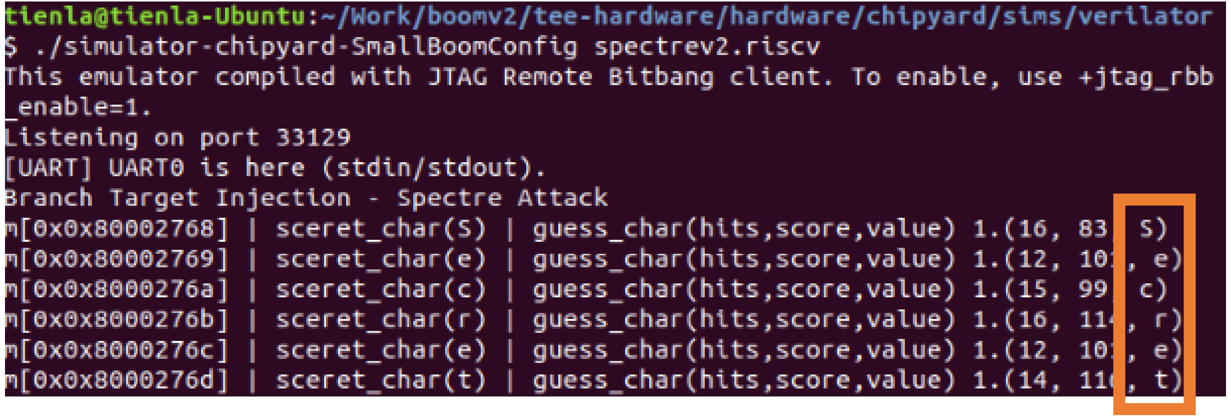
Secure Boom v2 & Spectre v2



```
tienla@tienla-Ubuntu:~/Work/tee/hardware/chipyard/sims/verilator$ ./simulato
r-chipyard-MediumBoomConfig spectrev2.riscv
This emulator compiled with JTAG Remote Bitbang client. To enable, use +jtag
_rbb_enable=1.
Listening on port 43925
[UART] UART0 is here (stdin/stdout).
Branch Target Injection - Spectre Attack
m[0x0x80002768] | sceret_char(S) | guess_char(hits,score,value) 1.(2, 1  ▯
m[0x0x80002769] | sceret_char(e) | guess_char(hits,score,value) 1.(2, 1  ▯
m[0x0x8000276a] | sceret_char(c) | guess_char(hits,score,value) 1.(2, 1  ▯
m[0x0x8000276b] | sceret_char(r) | guess_char(hits,score,value) 1.(2, 1  ▯
m[0x0x8000276c] | sceret_char(e) | guess_char(hits,score,value) 1.(2, 1  ▯
m[0x0x8000276d] | sceret_char(t) | guess_char(hits,score,value) 1.(2, 1  ▯
```

X

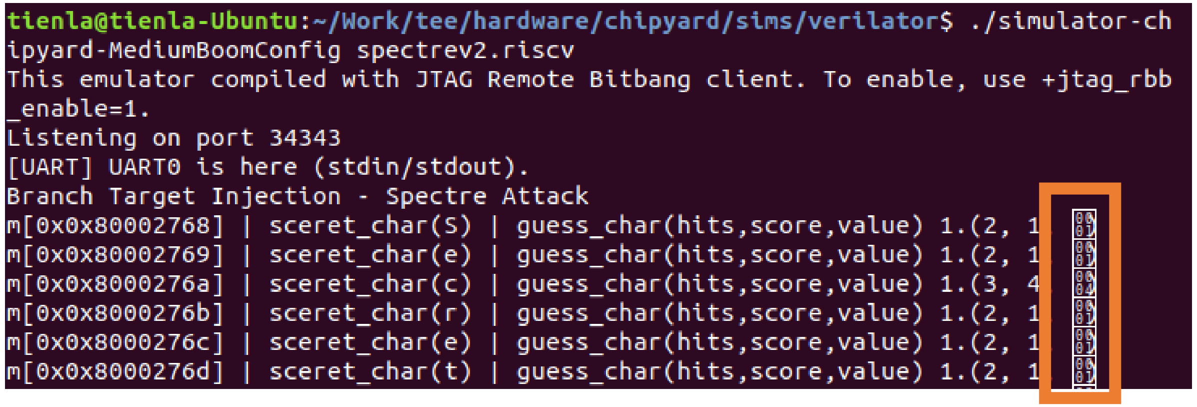
Boom v3 & Spectre v2



```
tienla@tienla-Ubuntu:~/Work/boomv2/tee-hardware/hardware/chipyard/sims/verilator
$ ./simulator-chipyard-SmallBoomConfig spectrev2.riscv
This emulator compiled with JTAG Remote Bitbang client. To enable, use +jtag_rbb
_enable=1.
Listening on port 33129
[UART] UART0 is here (stdin/stdout).
Branch Target Injection - Spectre Attack
m[0x0x80002768] | sceret_char(S) | guess_char(hits,score,value) 1.(16, 83   S)
m[0x0x80002769] | sceret_char(e) | guess_char(hits,score,value) 1.(12, 10 , e)
m[0x0x8000276a] | sceret_char(c) | guess_char(hits,score,value) 1.(15, 99   c)
m[0x0x8000276b] | sceret_char(r) | guess_char(hits,score,value) 1.(16, 11 , r)
m[0x0x8000276c] | sceret_char(e) | guess_char(hits,score,value) 1.(12, 10 , e)
m[0x0x8000276d] | sceret_char(t) | guess_char(hits,score,value) 1.(14, 11 , t)
```

O

Secure Boom v3 & Spectre v2

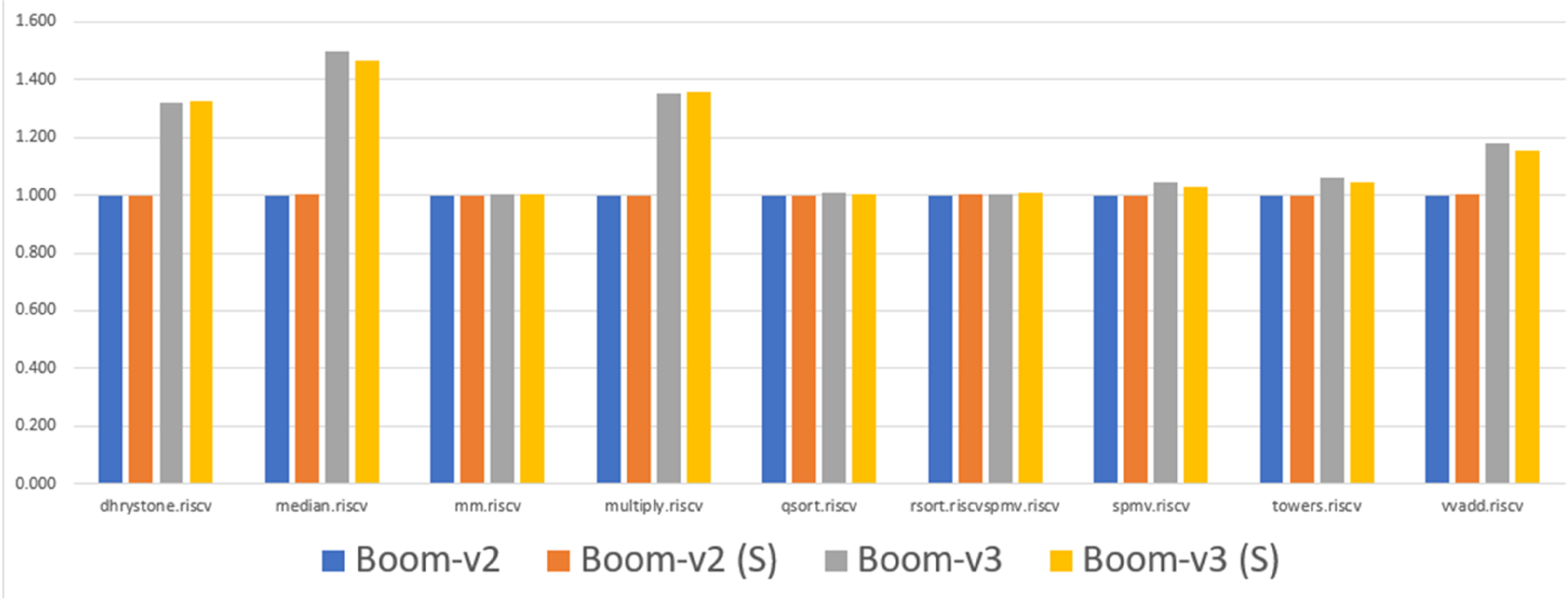


```
tienla@tienla-Ubuntu:~/Work/tee/hardware/chipyard/sims/verilator$ ./simulator-ch
ipyard-MediumBoomConfig spectrev2.riscv
This emulator compiled with JTAG Remote Bitbang client. To enable, use +jtag_rbb
_enable=1.
Listening on port 34343
[UART] UART0 is here (stdin/stdout).
Branch Target Injection - Spectre Attack
m[0x0x80002768] | sceret_char(S) | guess_char(hits,score,value) 1.(2, 1  ▯
m[0x0x80002769] | sceret_char(e) | guess_char(hits,score,value) 1.(2, 1  ▯
m[0x0x8000276a] | sceret_char(c) | guess_char(hits,score,value) 1.(3, 4  ▯
m[0x0x8000276b] | sceret_char(r) | guess_char(hits,score,value) 1.(2, 1  ▯
m[0x0x8000276c] | sceret_char(e) | guess_char(hits,score,value) 1.(2, 1  ▯
m[0x0x8000276d] | sceret_char(t) | guess_char(hits,score,value) 1.(2, 1  ▯
```

X

4. Spectre attack in OoO Processors (10/12)

- Benchmark riscv-tests
- The performance ratio of Boom-v2 is set as 1.0



4. Spectre attack in OoO Processors (11/12)

- MSHRs resources utilization

Configuration	**LUT**	**FF**
Normal MSHR	1926	1120
Secure MSHR	1980 (2.8%)	1124 (0.4%)

4. Spectre attack in OoO Processors (12/12)

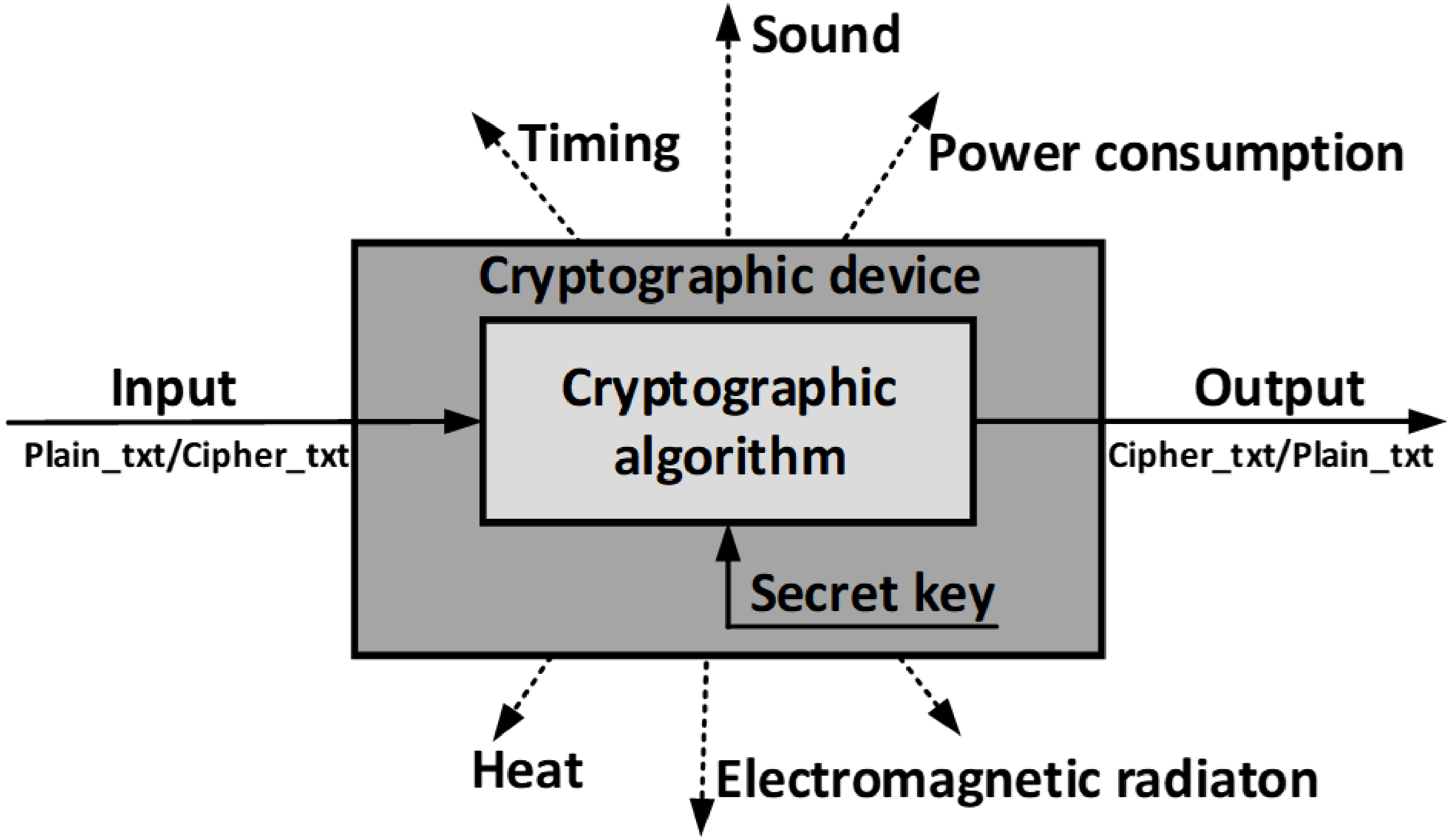
Spectre on RISC-V Boom	Detection	Prevention	
		Software approach	Hardware approach(*)
Idea or Research	● Analyse hardware performance counter ● Use machine learning	• Fence instruction • Speculation Load Hardening (Index masking)	• Modifying MSHRs
Benefits	● High accuracy and simple (>95%) ● Low performance overhead (~2%)	• Strengthen victim program • Simple to implement	• Low performance overhead
Drawbacks	● Need to find action after detection ● Need to re-create model for new threat	• Require to re-compile victim code • High performance overhead: 15-43%	• Complicated. • Time consuming to verify and develop.

(*) Currently on research stage

OUTLINE

1. Introduction
2. What is RISC-V, and How Does It Affect Cyber-Security?
3. Secure Boot for Trusted Execution Environment (TEE)
4. Cache Side-channel Attack (Spectre) on Out-of-Order (OoO) Processors
5. Prevent Correlation Power Analysis (CPA) with Random Dynamic Frequency Scaling (RDFS)
6. Conclusion

5. Prevent CPA with RDFS (1/10)



A cryptographic device leaks side-channel information

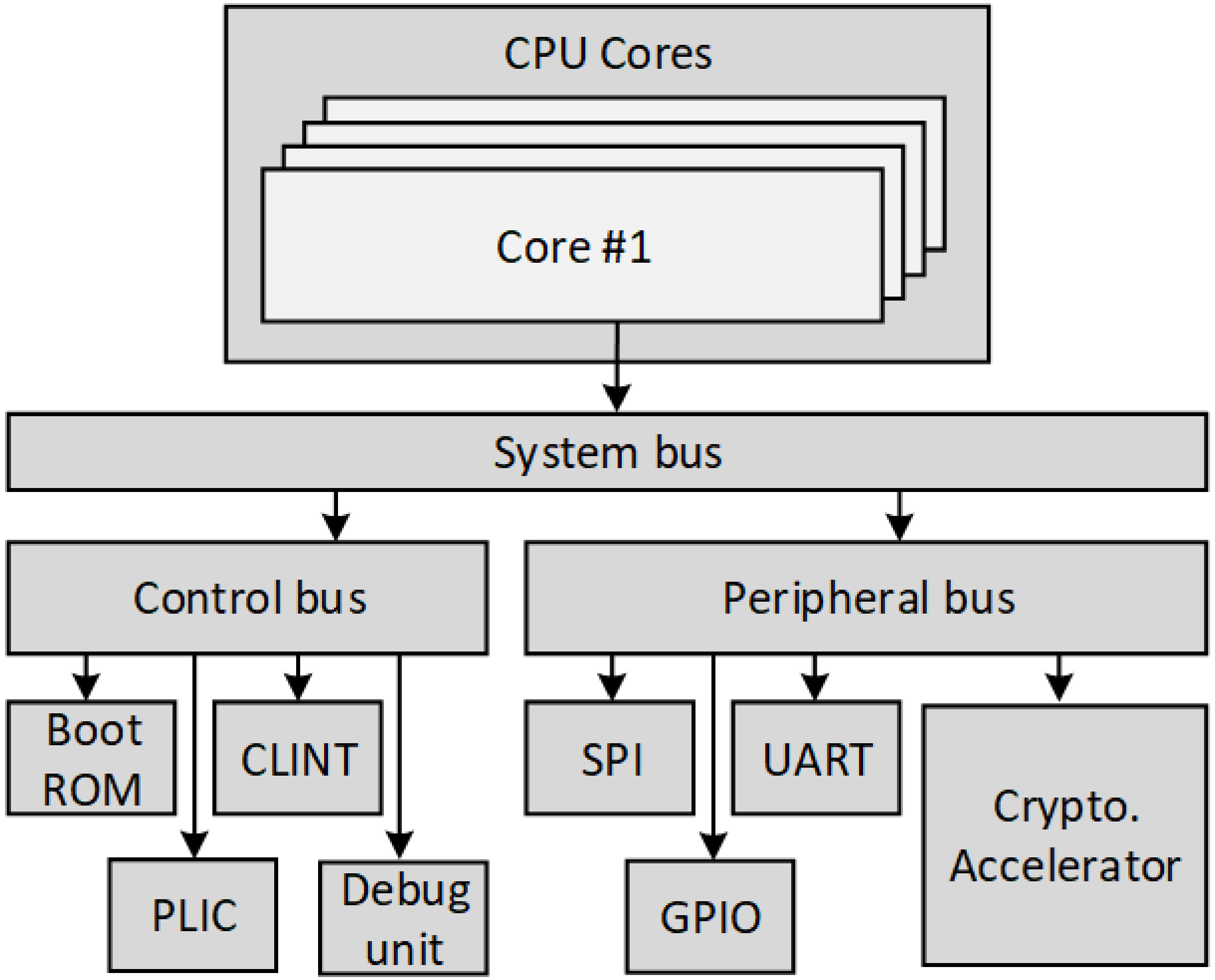
Side-channel attacks:

Exploit unavoidable side-channel information in cryptanalysis.

Power Analysis attacks:

Using Power consumption or Electromagnetic radiation.

5. Prevent CPA with RDFS (2/10)



Example of Cryptographic SoC

Countermeasures for Cryptographic SoC:

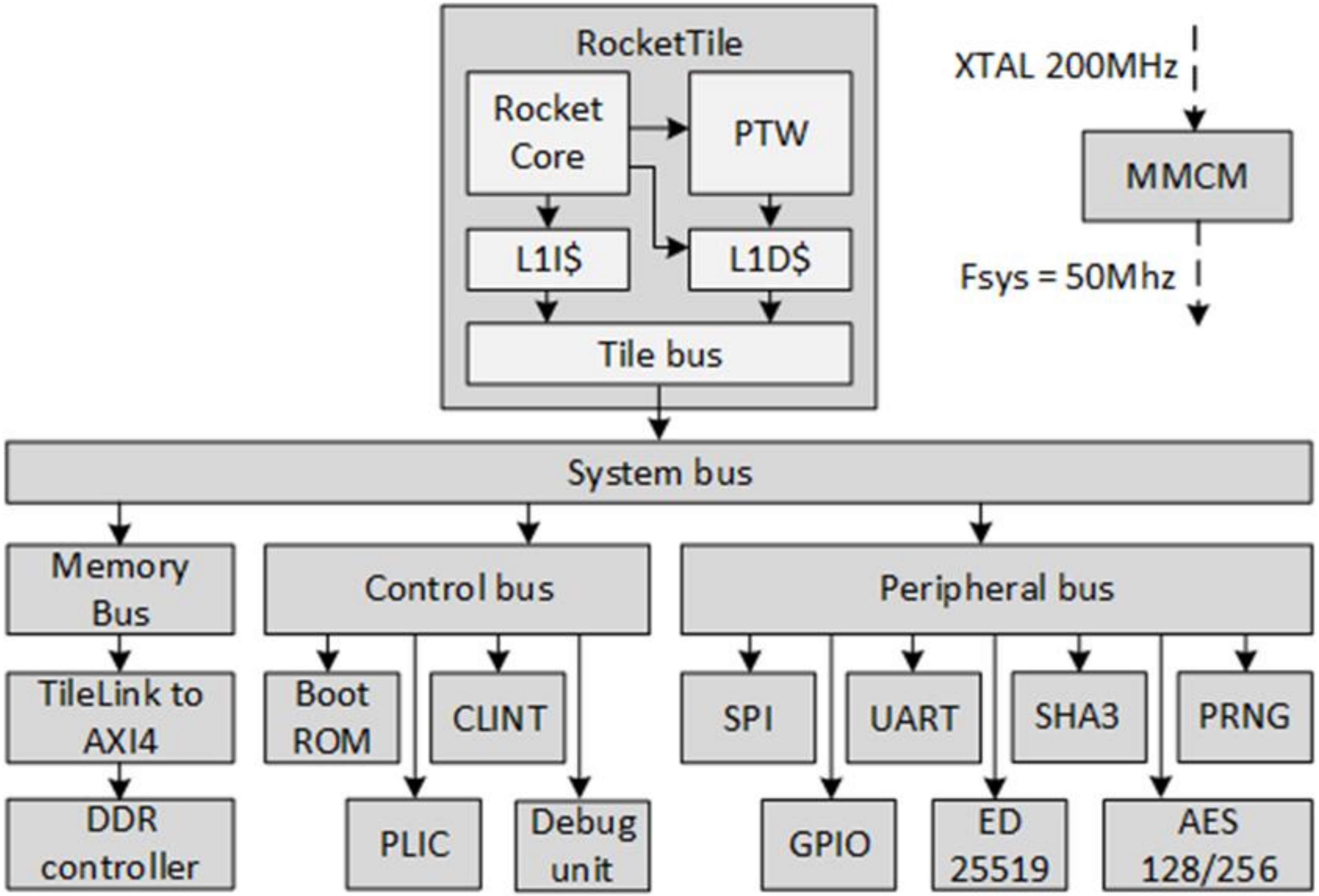
Existing techniques are not suitable:

- Masking: Reduce performance, Increase power, area.
- Hiding: Huge hardware overheads.

⇒ Proposed Ideas:

- Randomly scale the clock freq. of Crypto.Acc. after each encryption/decryption.
- Only applied to the Crypto.Acc.
- Create as many Clock frequencies as possible.

5. Prevent CPA with RDFS (3/10)

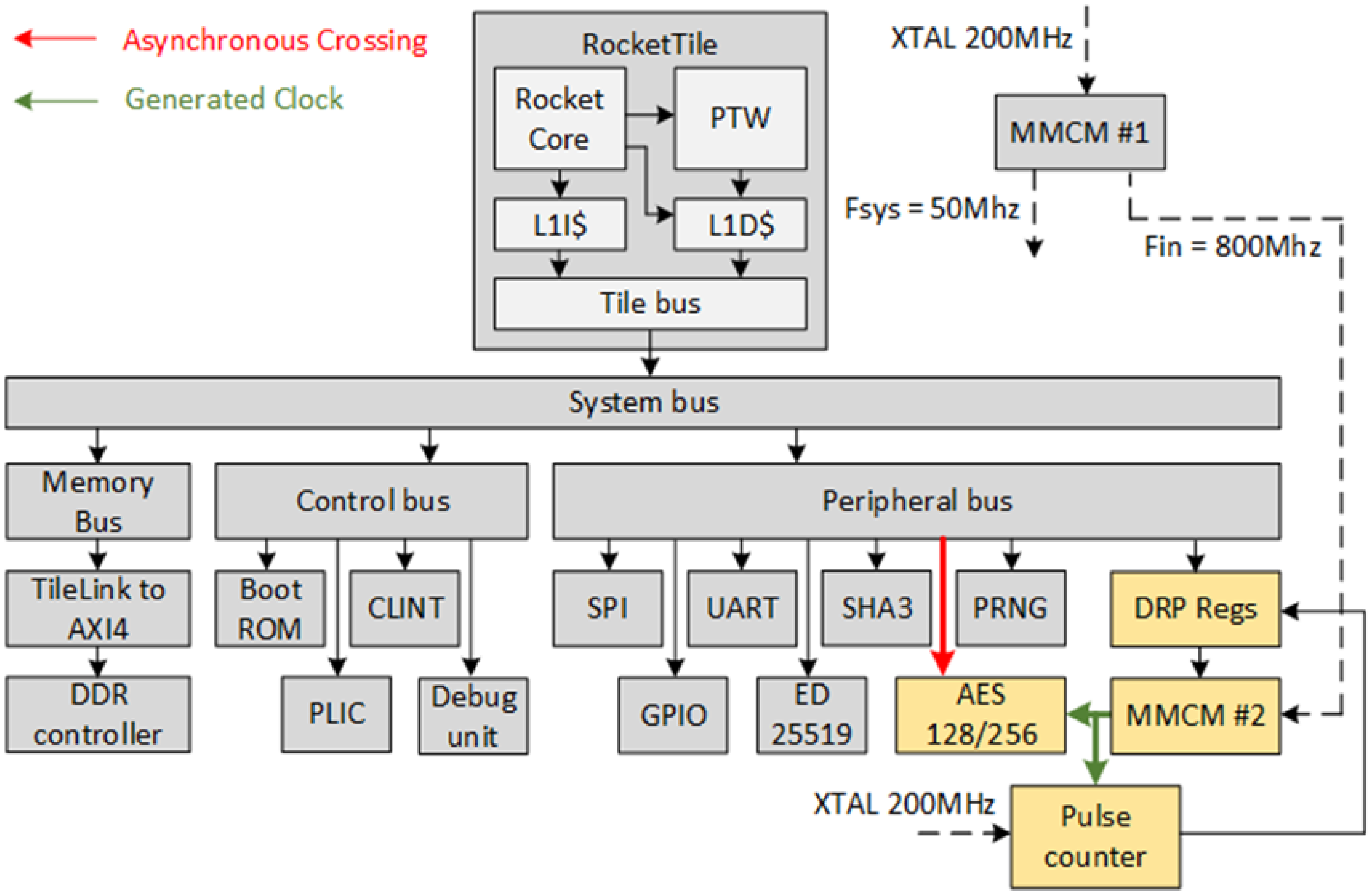


Unprotected Cryptographic SoC (TEE-Hardware)

Unprotected SoC (TEE-hardware):

- 32-bit RISC-V SoC
- DDR Controller ⇒ support Linux OS
- Crypto. Accelerator:
  - AES-128/256
  - SHA3
  - ED25519
  - PRNG
- Fixed system Clock: Fsys = 50MHz

5. Prevent CPA with RDFS (4/10)

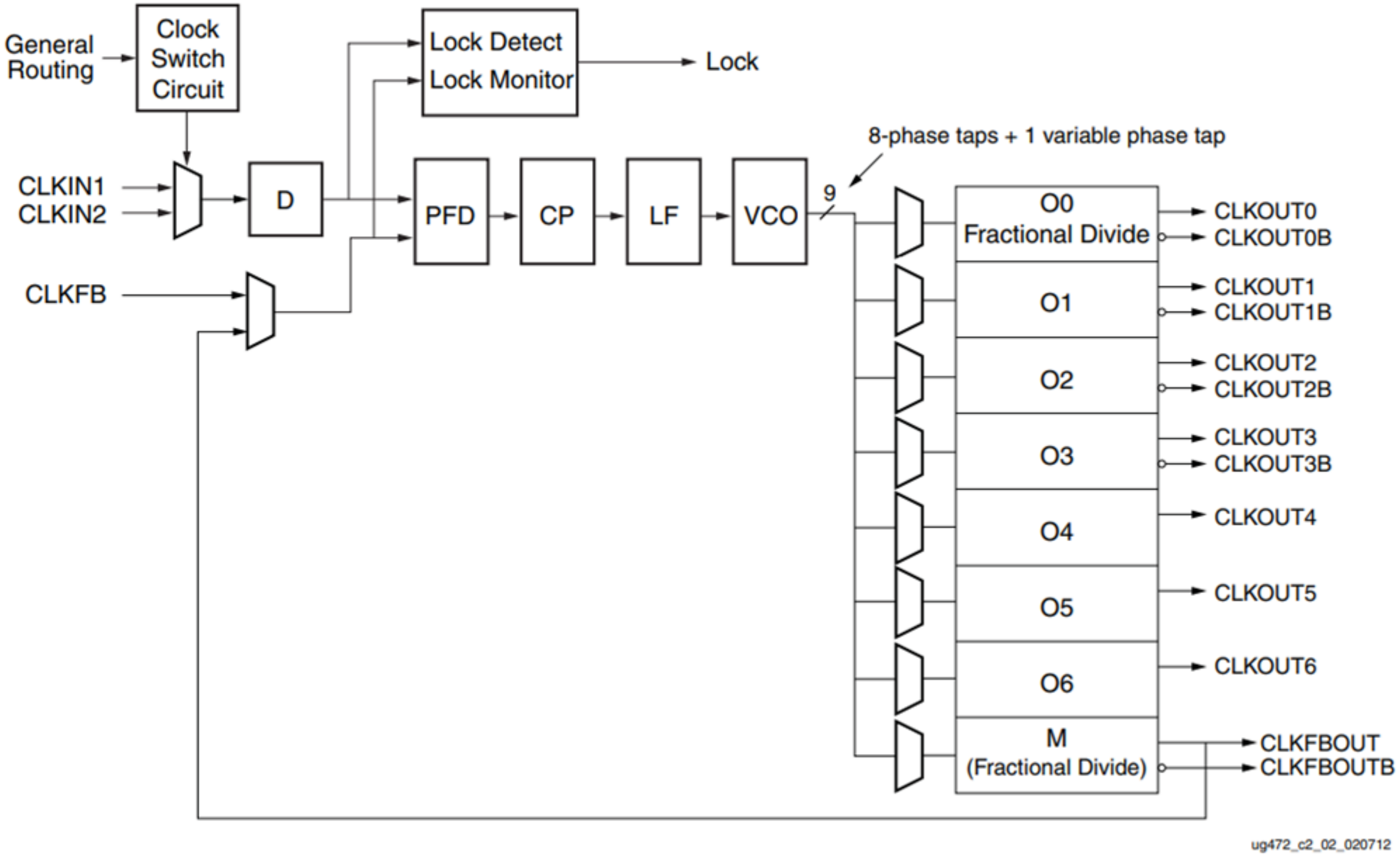


Protected Cryptographic SoC with RDFS [21]

Protected SoC with RDFS:

- Add Clock Generation peripherals (use Xilinx's Clock Manager IP)
- Create > 219.000 frequencies (in range from 50MHz to 100MHz)
- Verify accuracy by Pulse counter
- Only applied to AES-128 module
- Scale AES's CLK after each encryption

5. Prevent CPA with RDFS (5/10)



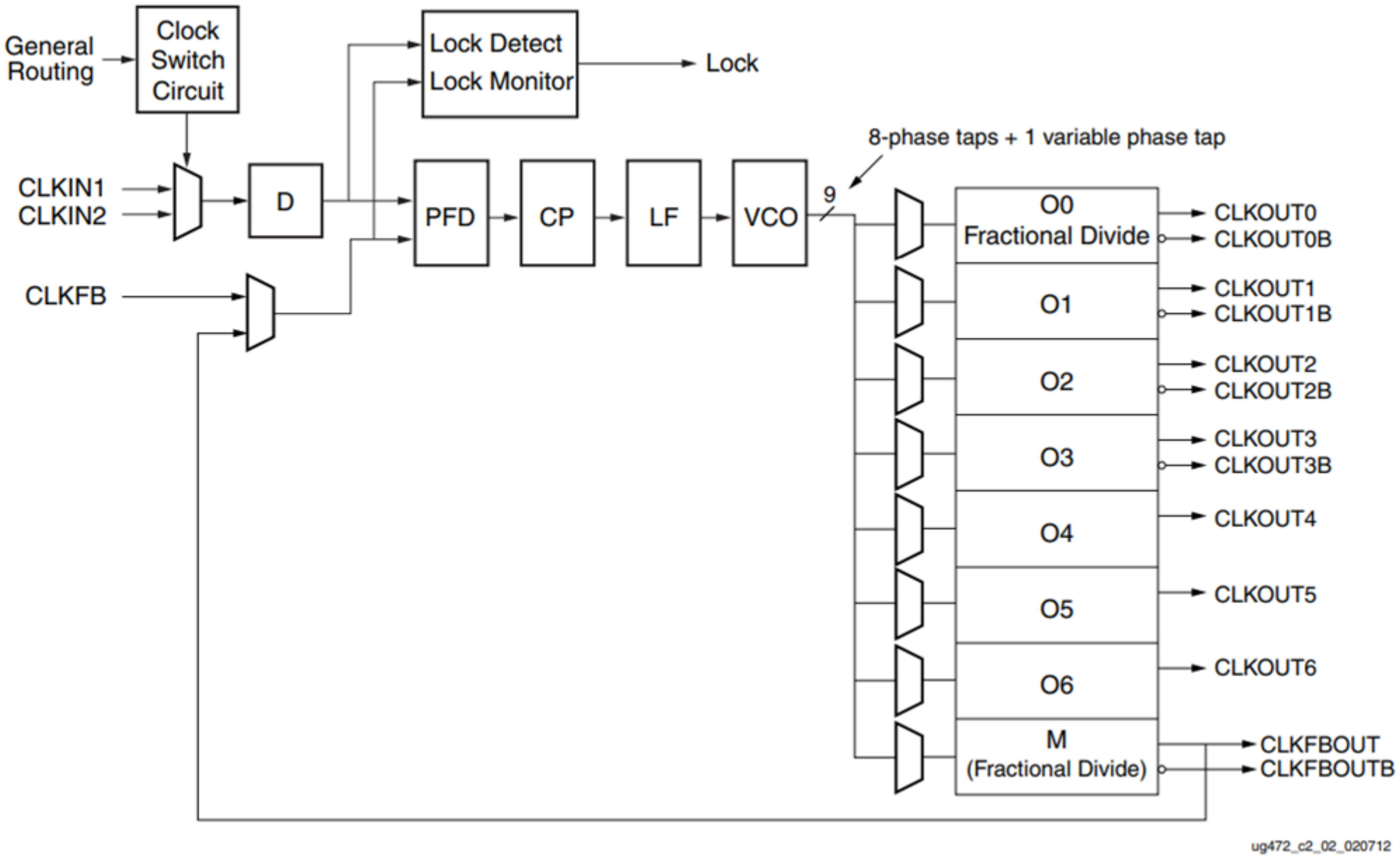
Xilinx 7-Series' Mixed Mode Clock Manager [22]

$$F_{CLKOUT_i} = \frac{F_{in} \times M}{D \times O_i}; i \in (0,6)$$

Constraints for Kintex-7 devices:

- $10\text{MHz} \leq F_{in} \leq 800\text{MHz}$
- $10\text{MHz} \leq (F_{in} / D) \leq 450\text{MHz}$
- $600\text{MHz} \leq F_{VCO} \leq 1200\text{MHz}$
- $1 \leq D \leq 106$ (integer)
- $2 \leq M \leq 64$ (fractional with 0.125 increment)
- $2 \leq O_0 \leq 128$ (fractional with 0.125 increment)
- $1 \leq O_{1\text{-}6} \leq 128$ (integer)

5. Prevent CPA with RDFS (6/10)



Xilinx 7-Series' Mixed Mode Clock Manager [22]

How to use:

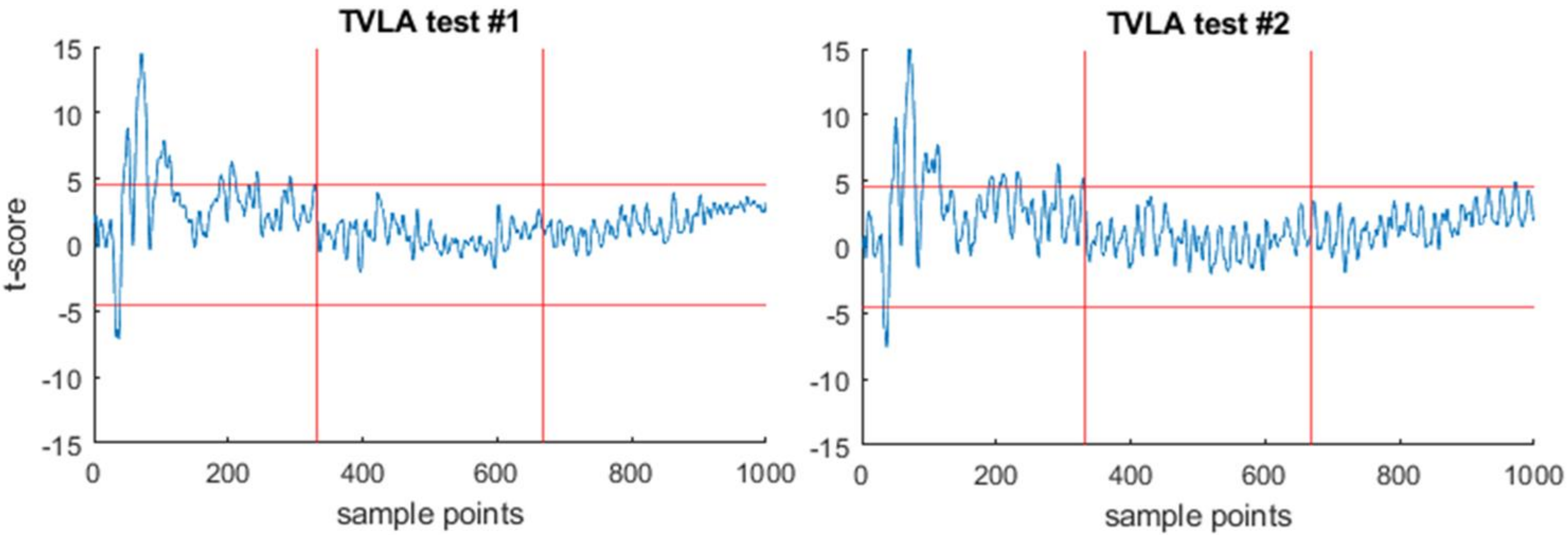
- Generate all possible settings for D,M,O_0
- Store setting values as C header in your code
- Randomly select, apply a new setting after each encryption or decryption

5. Prevent CPA with RDFS (7/10)

Implementation results on Kintex-7 FPGA

	Available	Original Unprotected SoC		AES accelerator		Protected SoC		Hardware Overhead (%)
		Utilization	(%)	Utilization	(%)	Utilization	(%)	
LUT	101400	48989	48.31	3169	3.13	51047	50.34	4.20
FF	202800	39298	19.38	3307	1.63	39516	19.49	0.55
BRAM	325	30	9.23	0	0	30	9.23	0
MMCM	8	2	25	0	0	3	37.5	50

5. Prevent CPA with RDFS (8/10)



Test Vector Leakage Assessment (TVLA) results:

- RDFS with 219,412 clk freq. (50MHz - 100MHz)
- Does not detect any leakage in 5 million power traces

5. Prevent CPA with RDFS (9/10)

CPA attacks		#1	#2	#3	#4
Para-meters	Target device	Unprotected SoC	Unprotected SoC	Unprotected SoC	Protected SoC
	Operating mode	Bare-metal	Bare-metal	Linux OS	Bare-metal
	Measuring method	Single acquisition	Averaging-64	Averaging-64	Single acquisition
	Power model	Hamming Weight model			
	Number of attacking traces	70,000	18,000	20,000	5 million
Attack Results	Number of byte revealed	12/16	13/16	13/16	0/16
	Min traces required	1,642 to 58,685	465 to 7,613	1,650 to 19,591	N/A
	Average traces required	28,683	1,928	10,175	N/A

Correlation Power Analysis (CPA) results:

- Cannot extract any byte of secret key with 5 million power traces

5. Prevent CPA with RDFS (10/10)

DLSCA attack #		#1	#2	#3	#4	#5
Para-meters	**Target device**	Unprotected SoC	Unprotected SoC	Unprotected SoC	Protected SoC	Protected SoC
	Operating mode	Bare-metal	Bare-metal	Linux OS	Bare-metal	Bare-metal
	Measuring method	Single acquisition	Averaging-64	Averaging-64	Single acquisition	Single acquisition
	Number of profiling traces	60,000	15,000	17,000	1,000,000	60,000
	Number of attacking traces	12,000	3,000	3,000	100,000	12,000
Attack Results	**Number of byte revealed**	16/16	16/16	9/16	13/16	0/16
	Min traces required	4,231	805	2,022	45,924	N/A

Deep Learning based Side Channel Attacks (DLSCA) results:

- Extremely powerful, can break RDFS countermeasure
- Require 16.67 times number of profiling traces
- Require 8.33 times number of attacking traces

OUTLINE

1. Introduction
2. What is RISC-V, and How Does It Affect Cyber-Security?
3. Secure Boot for Trusted Execution Environment (TEE)
4. Cache Side-channel Attack (Spectre) on Out-of-Order (OoO) Processors
5. Prevent Correlation Power Analysis (CPA) with Random Dynamic Frequency Scaling (RDFS)
6. Conclusion

Conclusion (1/1)

Keys takeaway

1. RISC-V is an opportunity to secure the IoT and is friendly with price-sensitive applications. It is an open approach to cyber-security with a solid and rich open-source community.
2. A Trusted Execution Environment (TEE) is the formal way to do the trusted vs. untrusted execution domains. However, TEE should not handle the Root-of-Trust (RoT) due to security concerns. Therefore, a platform that can provide a secure boot process with RoT utterly inaccessible from the TEE processors after boot is necessary.
3. RISC-V Out-of-order Processor has been proved vulnerable against cache side-channel attack (Spectre). Fortunately, detection and mitigation methods have been studied and implemented. Our approach for a secure MSHR (miss status holding register) has demonstrated a low-performance loss and small resource utilization solution.
4. Power Analysis attacks are powerful tools to break the security of cryptographic devices. Using RISC-V architecture, designers can easily apply suitable countermeasures to improve the system's resistance against these kinds of attacks.

THANK YOU

References (1/4)

1. A. Waterman and K. Asanovíc, “The RISC-V Instruction Set Manual Volume I: Unprivileged ISA,” SiFive Inc. and EECS Dep., Univ. of California, Berkeley, Dec. 2019. [Online] https://riscv.org/wp-content/uploads/2019/12/riscv-spec-20191213.pdf
2. A. Waterman, K. Asanović, and John Hauser, “The RISC-V Instruction Set Manual Volume II: Privileged Architecture,” SiFive Inc. and EECS Dep., Univ. of California, Berkeley, Dec. 2021. [Online] https://github.com/riscv/riscv-isa-manual/releases/download/Priv-v1.12/riscv-privileged-20211203.pdf
3. RISC-V GNU Compiler Toolchain. [Online] https://github.com/riscv-collab/riscv-gnu-toolchain
4. Tao Lu, “A Survey on RISC-V Security: Hardware and Architecture,” arXiv:2107.04175v1 [cs.CR], Jul. 2021. [Online] https://arxiv.org/pdf/2107.04175.pdf
5. Helena Handschuh, “RISC-V: An Open Approach to System Security,” Mar. 16, 2020. [Online] https://riscv.org/blog/2020/03/risc-v-an-open-approach-to-system-security/
6. M. Bailleu, D. Dragoti, P. Bhatotia, and C. Fetzer, “TEE-Perf: A Profiler for Trusted Execution Environments,” in *Proc. of Annual IEEE/IFIP Int. Conf. on Dependable Syst. and Networks (DSN)*, Jun. 2019, Portland, OR, USA, pp. 414-421.

References (2/4)

7. Intel Corp., “Intel Software Guard Extensions (Intel SGX) Developer Guide.” [Online] https://download.01.org/intel-sgx/linux-1.7/docs/Intel_SGX_Developer_Guide.pdf
8. S. Pinto and N. Santos, “Demystifying Arm TrustZone: A Comprehensive Survey,” in *ACM Comput. Surv.*, vol. 51, no. 6, pp. 1-36, Nov. 2019.
9. R. Buhren, C. Werling, and J.-P. Seifert, “Insecure Until Proven Updated: Analyzing AMD SEV's Remote Attestation,” in *Proc. of ACM SIGSAC Conf. on Computer and Comm. Secu. (CCS)*, Nov. 2019, London, UK, pp. 1087–1099.
10. Hex Five Security, Inc., “MultiZone Hex-Five Security.” [Online] https://hex-five.com/
11. V. Costan, I. Lebedev, and S. Devadas, “Sanctum: Minimal Hardware Extensions for Strong Software Isolation,” in *Proc. of Secu. Symp. (USENIX)*, Aug. 2016, Austin, TX, USA, pp. 857-874.
12. S. Weiser, M. Werner, F. Brasser, M. Malenko, S. Mangard, and A.-R. Sadeghi, “TIMBER-V: Tag-Isolated Memory Bringing Fine-grained Enclaves to RISC-V,” in *Proc. of Network and Distributed Syst. Secu. Symp. (NDSS)*, Feb. 2019, San Diego, CA, USA, pp. 1-15.
13. D. Lee, D. Kohlbrenner, S. Shinde, K. Asanovic, and D. Song, “Keystone: An Open Framework for Architecting Trusted Execution Environments,” in *Proc. of European Conf. on Computer Syst. (EUROSYS)*, Apr. 2020, Heraklion, Greece, pp. 1-16.

References (3/4)

14. R. Bahmani, F. Brasser, G. Dessouky, P. Jauernig, M. Klimmek, A.-R. Sadeghi, and E. Stapf, “CURE: A Security Architecture with CUstomizable and Resilient Enclaves,” in *Proc. of USENIX Secu. Symp. (USENIX Security)*, Aug. 2021, Virtual Event, pp. 1073-1090.
15. J. H.-Yahya, M. M. Wong, V. Pudi, S. Bhasin, and A. Chattopadhyay, “Lightweight Secure-Boot Architecture for RISC-V System-on-Chip,” in *Proc. of Int. Symp. on Quality Electronic Design (ISQED)*, Mar. 2019, Santa Clara, CA, USA, pp. 216-223.
16. V. B. Y. Kumar, A. Chattopadhyay, J. H.-Yahya, and A. Mendelson, “ITUS: A Secure RISC-V System-on-Chip,” in *Proc. of IEEE Int. System-on-Chip Conf. (SOCC)*, Sep. 2019, Singapore, Singapore, pp. 418-423.
17. P. Nasahl, R. Schilling, M. Werner, and S. Mangard, “HECTOR-V: A Heterogeneous CPU Architecture for a Secure RISC-V Execution Environment,” in *Proc. of ACM Asia Conf. on Computer and Comm. Secu. (ASIA CCS)*, Jun. 2021, Virtual Event, Hong Kong, China, pp. 187-199.
18. R. Bahmani, F. Brasser, G. Dessouky, P. Jauernig, M. Klimmek, A.-R. Sadeghi, and E. Stapf, “CURE: A Security Architecture with CUstomizable and Resilient Enclaves,” in *Proc. of USENIX Secu. Symp. (USENIX Security)*, Aug. 2021, Virtual Event, pp. 1073-1090.
19. SiFive, Inc., “Securing The RISC-V Revolution.” [Online] https://www.sifive.com/technology/shield-soc-security

References (4/4)

20. A. Le et al, "Experiment on replication of side channel attack via cache of RISC-V Berkeley out-of-order machine (BOOM) implemented on FPGA", CARRV 2020.
21. Dao, B.A., Hoang, T.T., Le, A.T., Tsukamoto, A., Suzaki, K. and Pham, C.K., 2021. Correlation Power Analysis Attack Resisted Cryptographic RISC-V SoC With Random Dynamic Frequency Scaling Countermeasure. *IEEE Access*, *9*, pp.151993-152014.
22. Guide, Xilinx User. "seriens FPGAs clocking resources,”." *UG472, Jun* 12 (7): 2015.[Online] https://docs.xilinx.com/v/u/en-US/ug472_7Series_Clocking