国立大学法人
電気通信大学
The University of Electro-Communications

ICIT2023
The 2nd International Conference on Intelligence of Things

The 3rd Symposium on Computer Science & Engineering (SCSE 2023)

RISC-V *T*rusted *P*latform *M*odule (*TPM*) and *T*rusted *E*xecution *E*nvironment (*TEE*)

Trong-Thuc Hoang and Cong-Kha Pham

University of Electro-Communications (UEC), Tokyo, Japan

2023/10/27

Outline

1. Introduction
2. TPM and TEE
3. Why RISC-V?
4. Proposed System
5. Peripherals
6. Result
7. Conclusion

Outline

1. Introduction
2. TPM and TEE
3. Why RISC-V?
4. Proposed System
5. Peripherals
6. Result
7. Conclusion

The University of Electro-Communications

国立大学法人 電気通信大学

1. Introduction (2/8) University

History of UEC

1918 Established as "The Technical Institute for Wireless- Communications"

1949 Promoted to the National University status as "The **U**niversity of **E**lectro-**C**ommunications"

2004 Reformed as a National University Corporation

2013 Authorized as "The Enhancement of Research Universities"

2018 Observes its Centennial

Location of UEC Campus

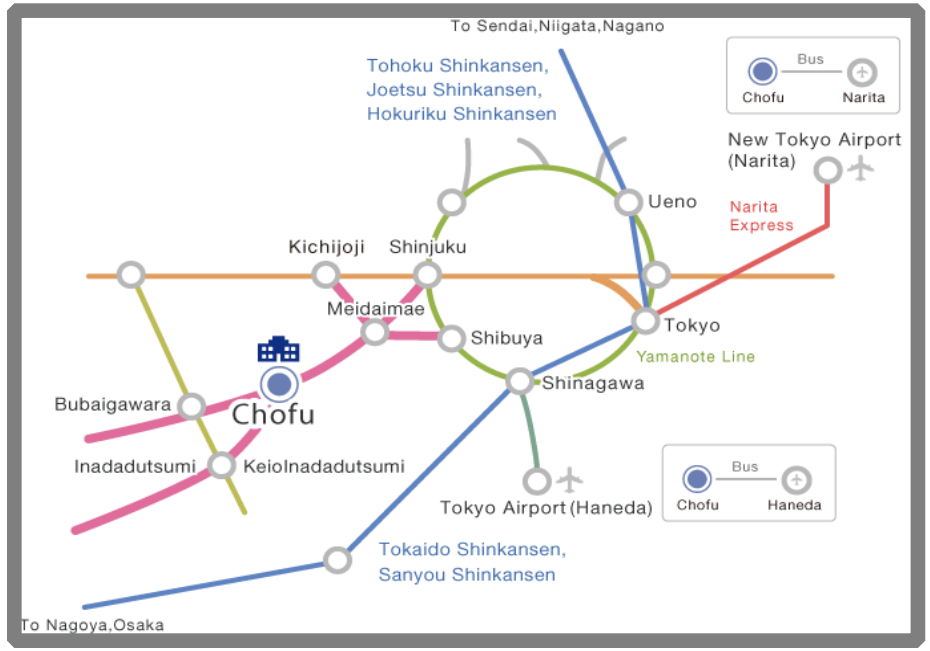
TOKYO

A convenient location

- **15 minutes to Shinjuku**, a major business center
- **1 hour from Tokyo Airport** by Airport Shuttle bus.

With beautiful suburban and historical surroundings

Access



1. Introduction (4/8) University

UEC Statistics

(as of May 1, 2023)

1. **One Undergraduate and One Graduate Schools**
   - Undergraduate School of Informatics and Engineering
   - Graduate School of Informatics and Engineering

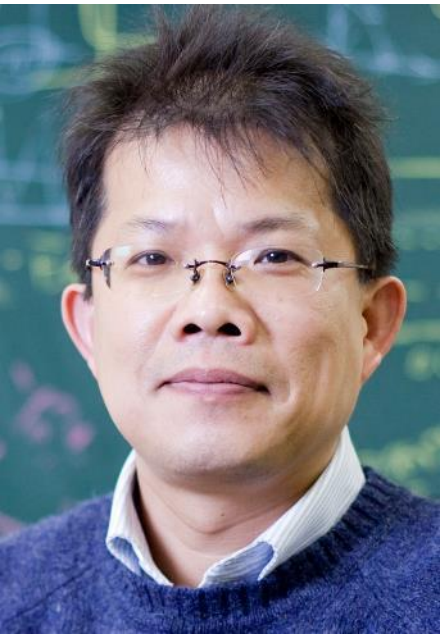
2. **Number of Students : 4,801 (305 international students)**
   - Undergraduate: 3,371
   - Graduate <Master>: 1,159
     <Doctor>: 271

3. **Number of Faculty Members : 348**
   - Professors: 135
   - Associate Professors: 123
   - Lecturers: 4
   - Assistant Professors: 42
   - Special Faculty Members : 44

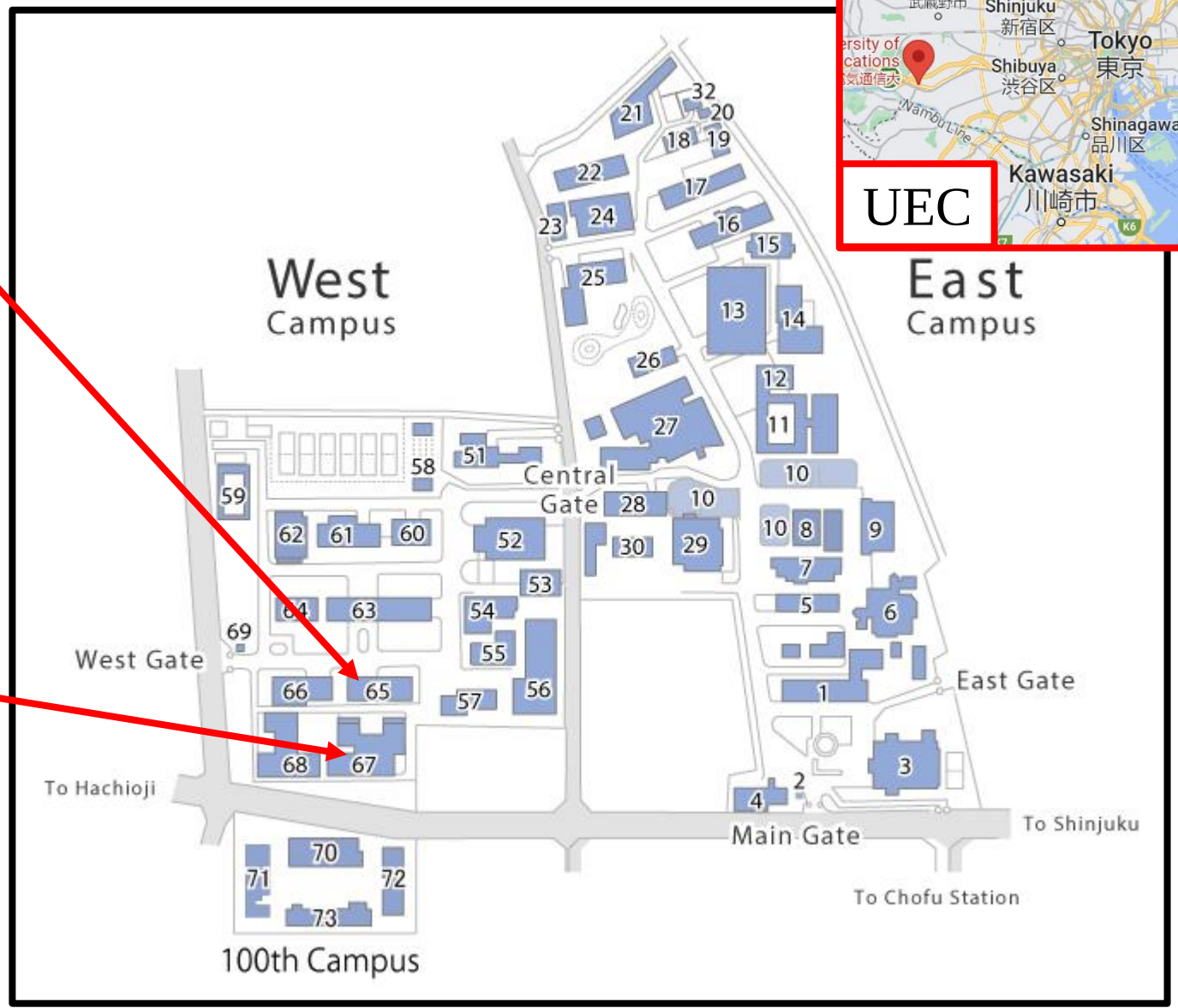
4. **Number of Administration and Technical Staffs: 199**

1. Introduction (5/8) VLSI Lab

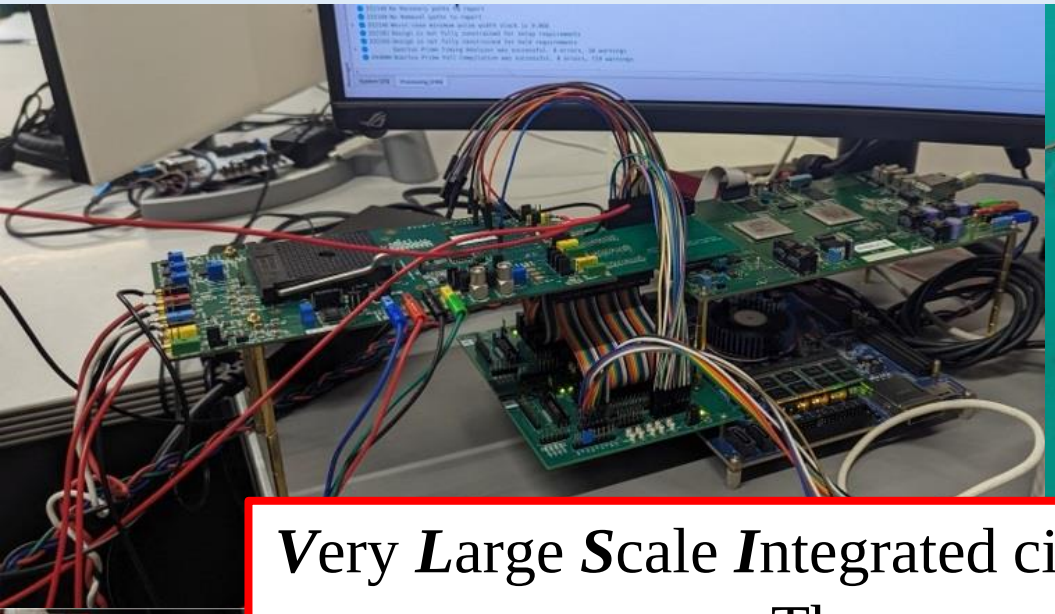
Trong-Thuc Hoang
Assistant Professor
hoangtt@uec.ac.jp
W1-507

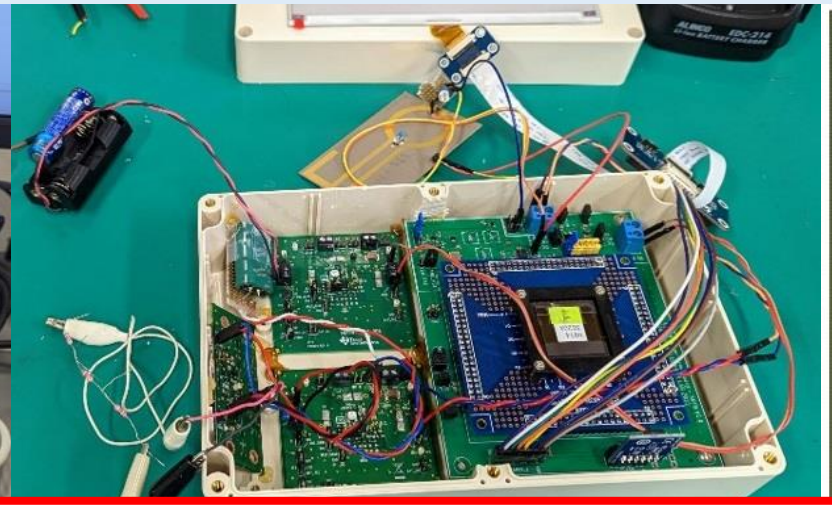


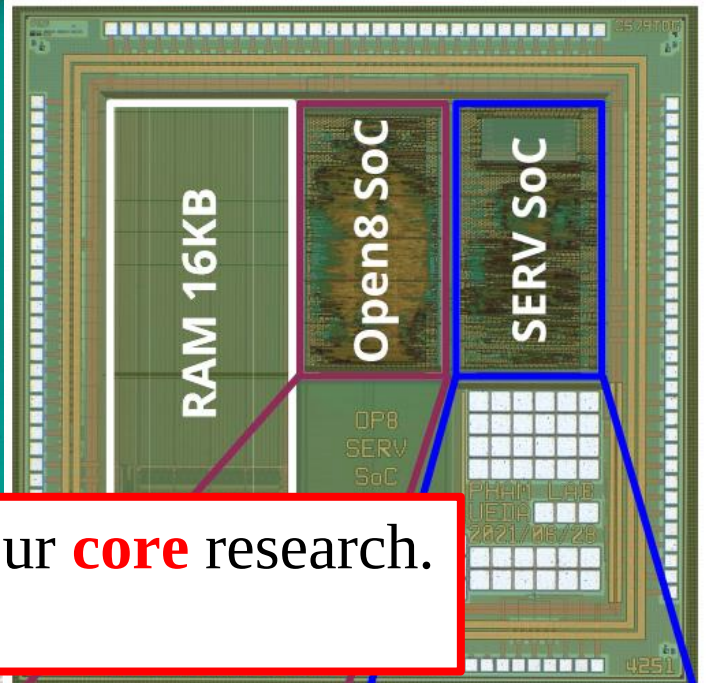
Cong-Kha Pham
Professor
phamck@uec.ac.jp
W8-214



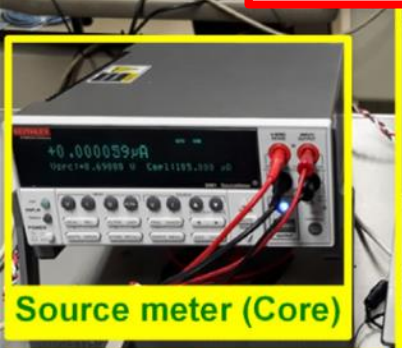
1. Introduction (6/8) VLSI Lab

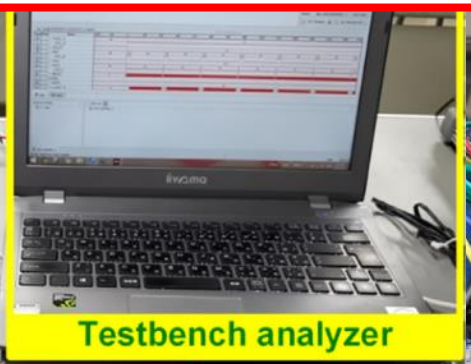


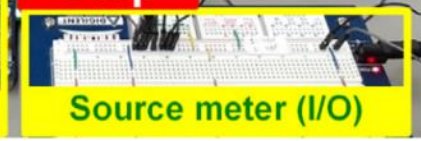


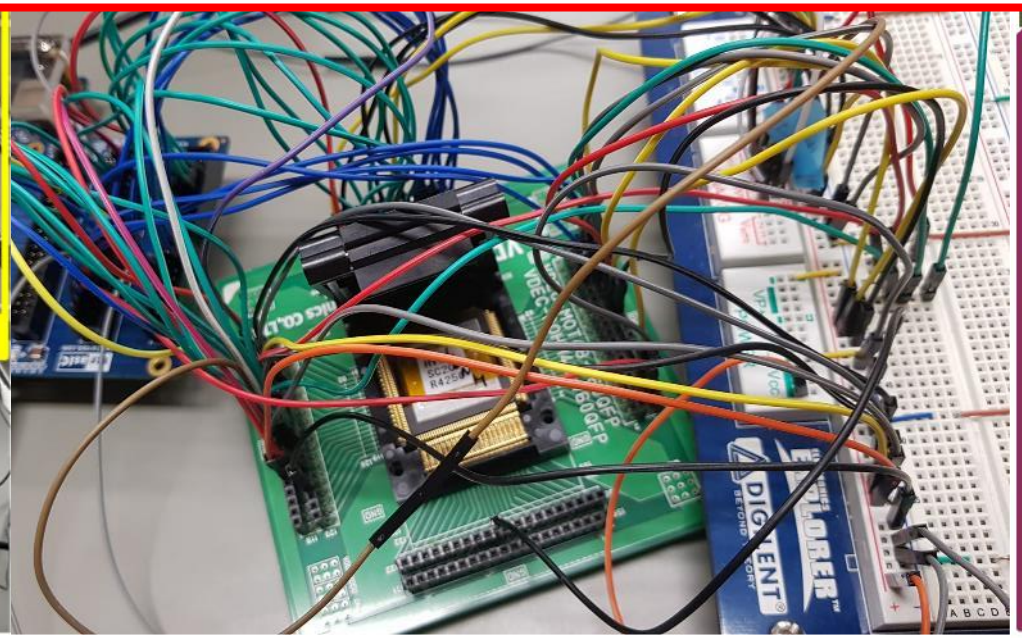


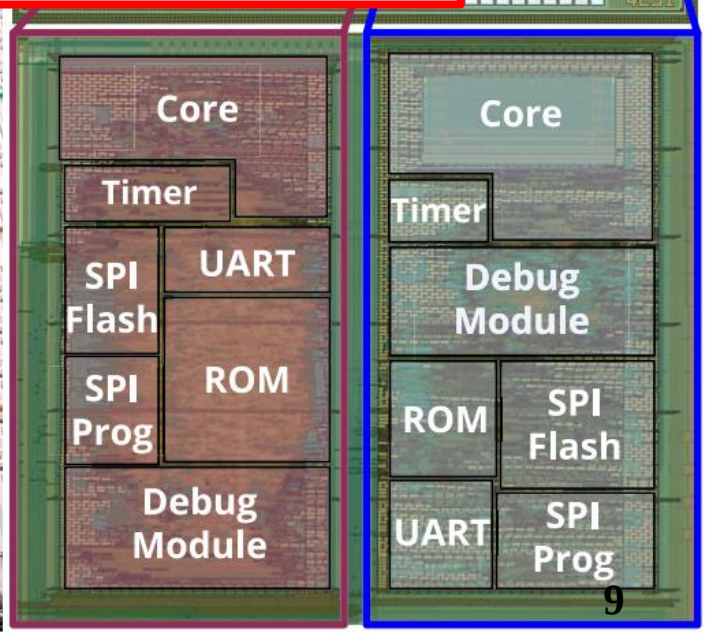
***V**ery **L**arge **S**cale **I**ntegrated circuit (**VLSI**) and **RISC-V** are our **core** research.
They serve as the <u>foundation</u> for others.*











1. Introduction (7/8) VLSI Lab

Member

(as in Oct. 2023)

- Ph.D. students: 6
- Master students: 4
- Bachelor students: 2
- Researcher: 1

Total: 13

From

- Vietnam: 7
- Japan: 5
- China: 1

1. Introduction (8/8) VLSI Lab

Core research | ***V***ery-***L***arge ***S***cale ***I***ntegrated circuit | RISC-V

Cybersecurity

Cryptosystem

- ***T***rusted ***P***latform ***M***odule
- ***T***rusted ***E***xecution ***E***nvironment

*S*ide-*C*hannel *A*ttack

- ***P***ower ***A***nalysis ***A***ttack
- Spectre attack

New Family

- ***P***ost-***Q***uantum ***C***ryptography
- ***L***ight-***W***eight ***C***ryptography

Cryptography

Security IP

- ***T***rue ***R***andom ***N***umber ***G***enerator
- ***P***hysical ***U***nclonable ***F***unction
- ***O***ne-***T***ime ***P***rogrammable

New Standard

- ***Cipher***: ChaCha20, Poly1305, AEAD, HMAC
- ***Hash***: SHA3, SHAKE
- ***Pair-key***: ECDSA, EdDSA

Internet of Things

Ultra-***L***ow ***P***ower Micro-controller

Others

Big Data

- ***F***requent ***I***tem ***C***ounter
- ***T***ext ***S***earch ***P***rocessor

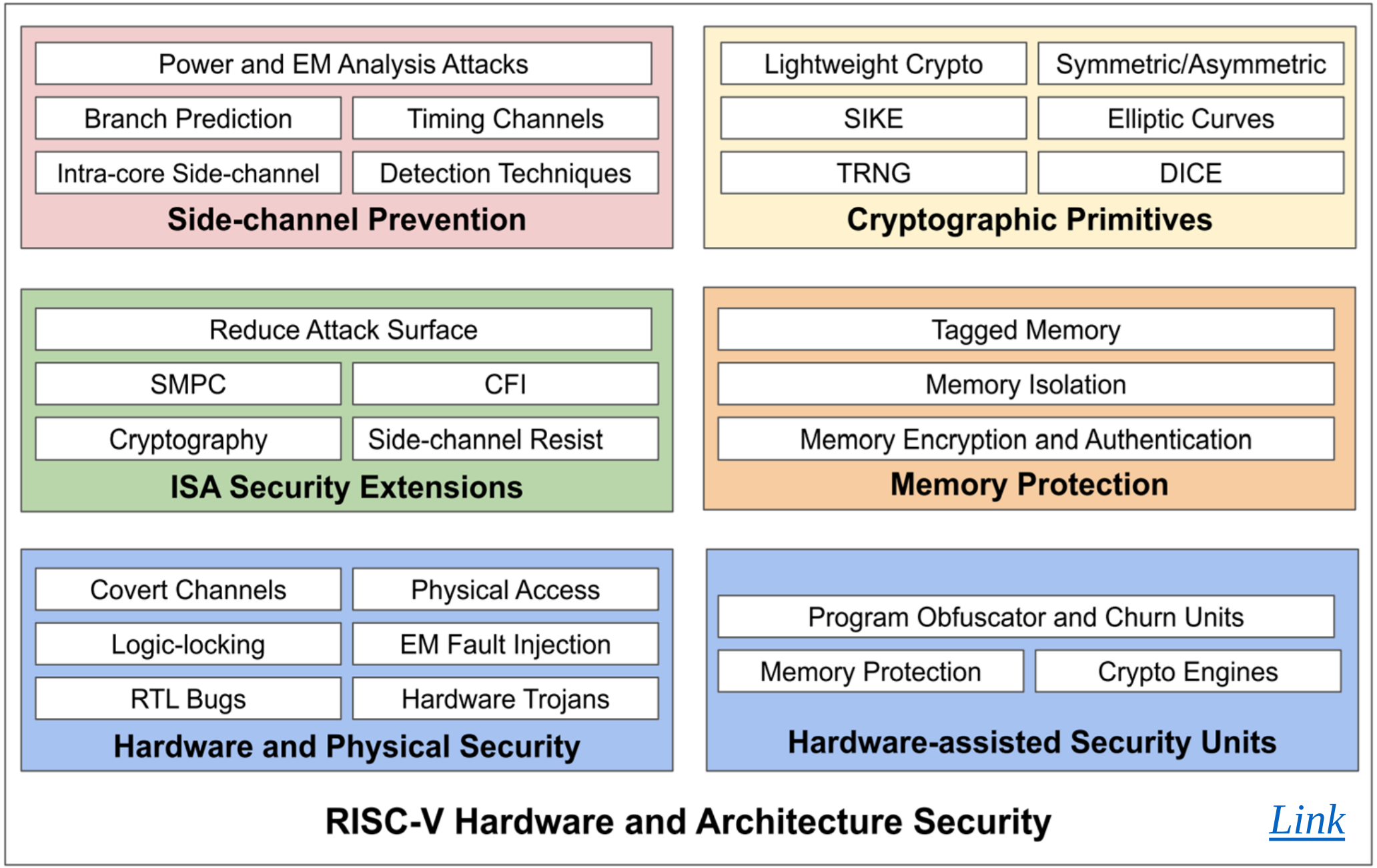
*D*igital *S*ignal *P*rocessing

- ***CO***ordinate ***R***otation ***DI***gital ***C***omputer
- ***D***iscrete ***C***osine ***T***ransform
- ***F***ast ***F***ourier ***T***ransform

Outline

1. Introduction
2. TPM and TEE
3. Why RISC-V?
4. Proposed System
5. Peripherals
6. Result
7. Conclusion

2. TPM & TEE (1/8) Cybersecurity overview



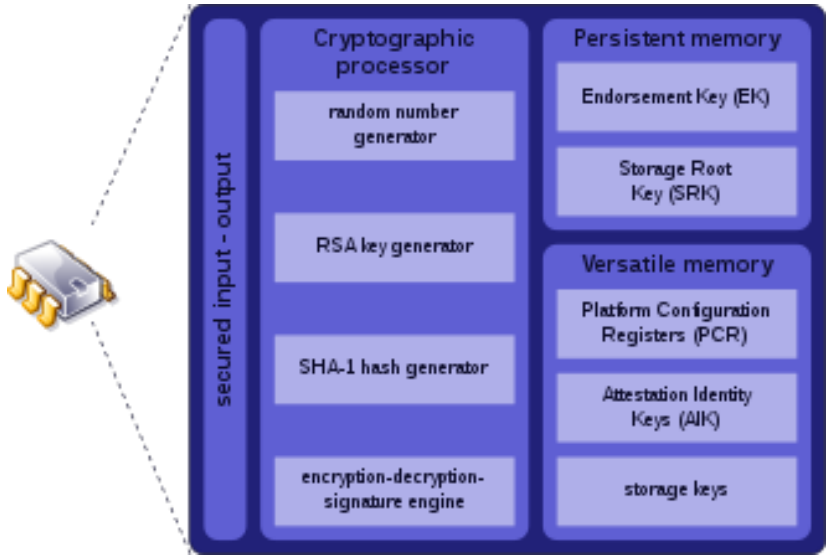
Cybersecurity is a huge field of research.

Today we focus on *cryptosystem*.

2. TPM & TEE (2/8) TPM & TEE

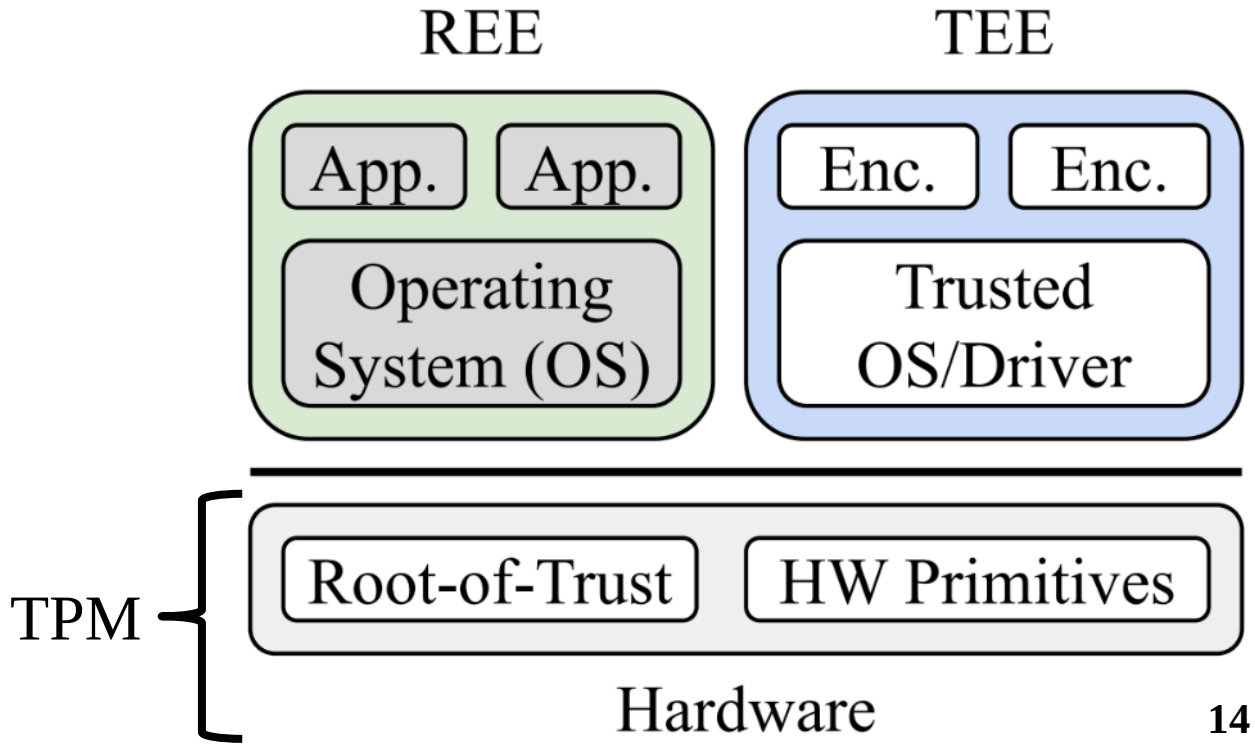
TPM = Trusted Platform Module

- TPM is for the authentication problem in a computer system.
- The main feature is remote attestation: a verifier can trust that the platform is "*clean*" (i.e., its vital data is safe and its critical software are not tampered).
- Based on TPM, other applications of *confidentiality*, *integrity*, *availability*, etc., can be developed.



TEE = Trusted Execution Environment

- TEE is the next step after TPM.
- TPM is for a trusted *hardware*; TEE is for a trusted ***O****perating* ***S****ystem* (***OS***)
- TEE needs TPM for the ***R****oot-****o****f-****T****rust* (***RoT***). Based on the RoT, the ***C****hain-****o****f-****T****rust* (***CoT***) is developed, thus creating TEE.



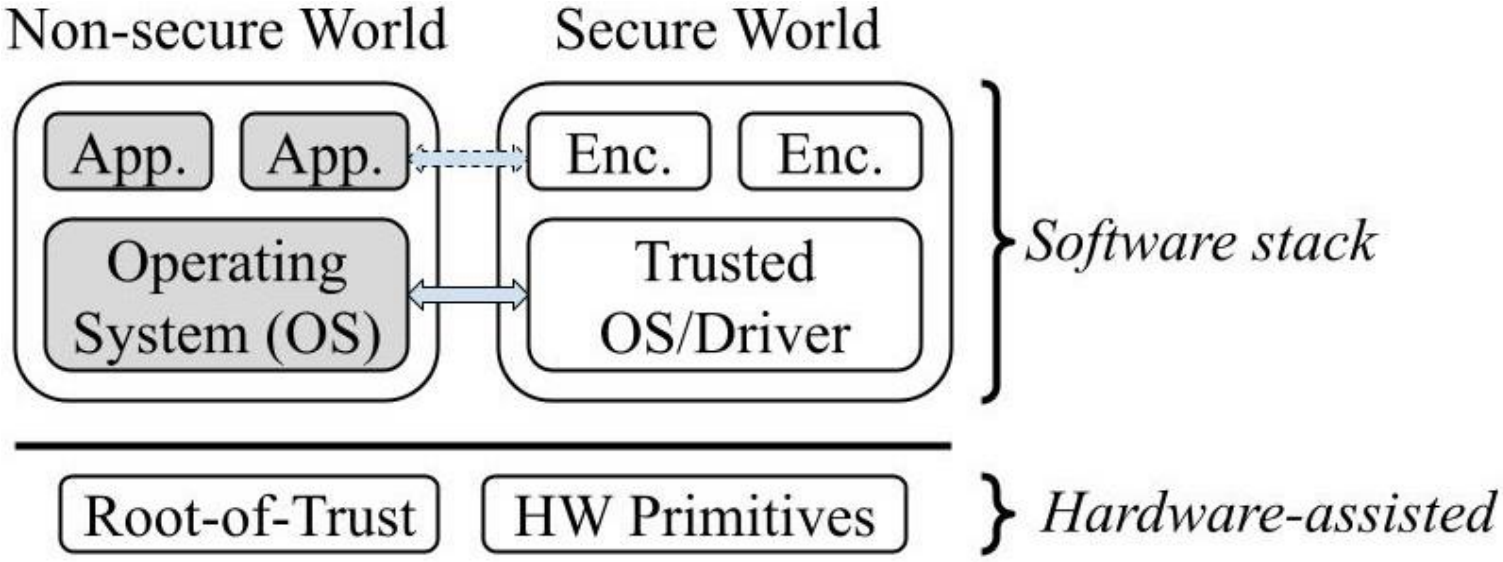
2. TPM & TEE (3/8) How TEE works?

Trusted Execution Environment (TEE) 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.

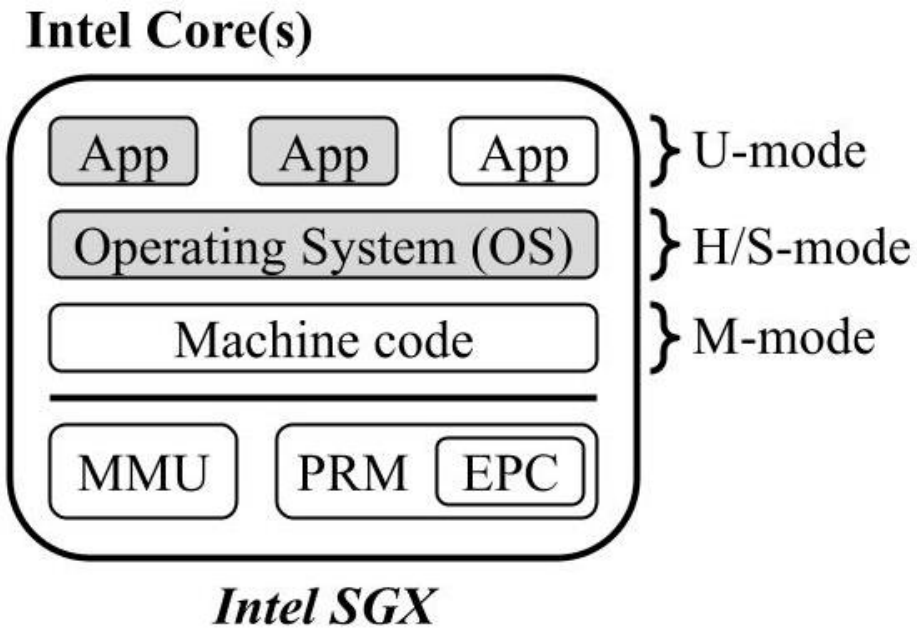
A typical TEE setup:

- 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.

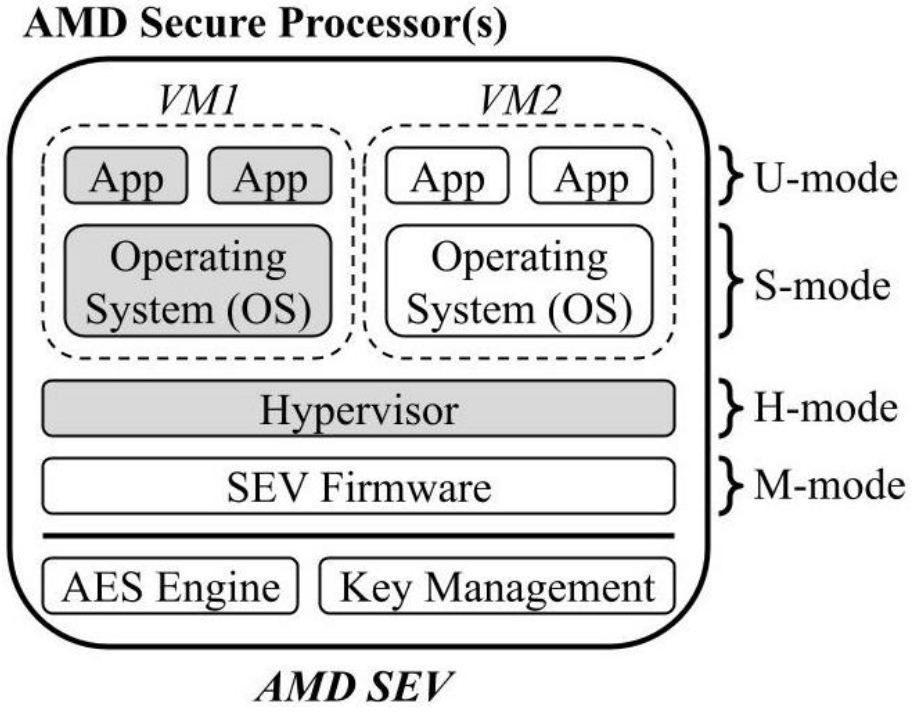


- HW primitives *(examples)*: cache flushing, cache partitioning, memory isolation, memory encryption, keys management, bus access controller, enclave encryption, and so on.

2. TPM & TEE (4/8) Several TEE examples

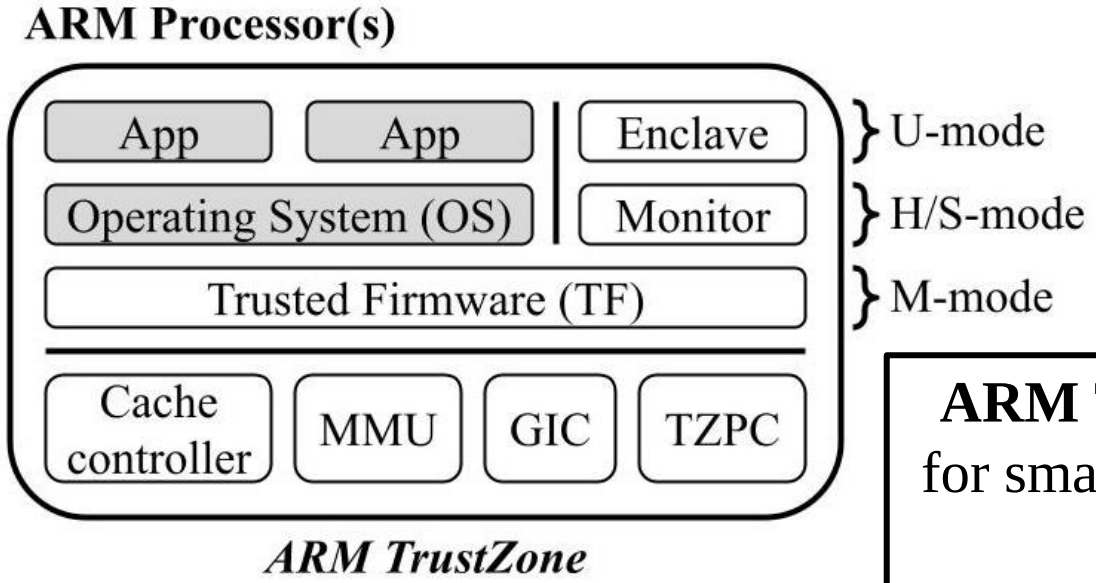


- Many TEE models were proposed: different set goals, different resources, and different developing mindsets.



Intel SGX: aiming for conventional PCs

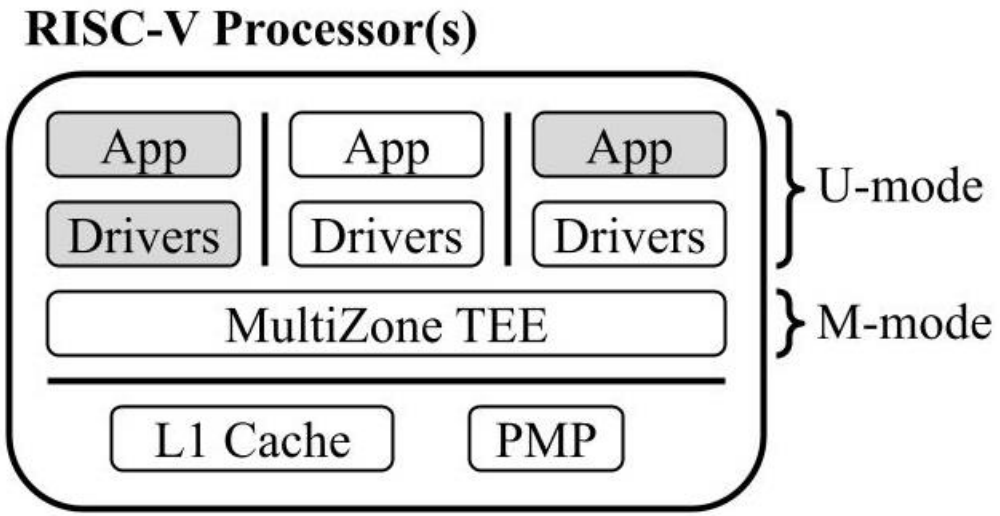
AMD SEV: aiming for server's cloud computing



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

ARM TrustZone: aiming for smartphones/embedded-systems

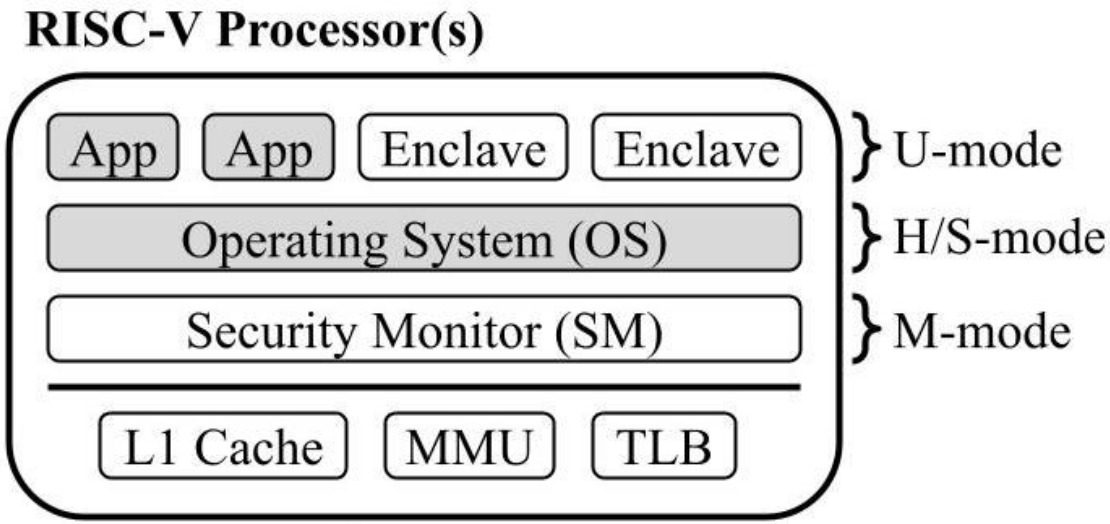
2. TPM & TEE (5/8) Several TEE examples



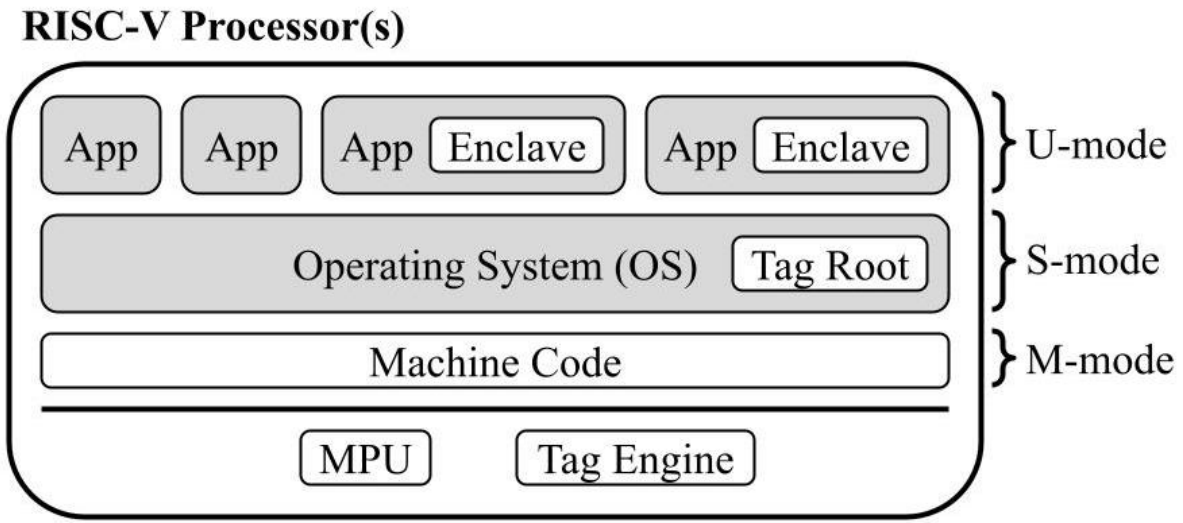
Hex-Five MultiZone

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

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



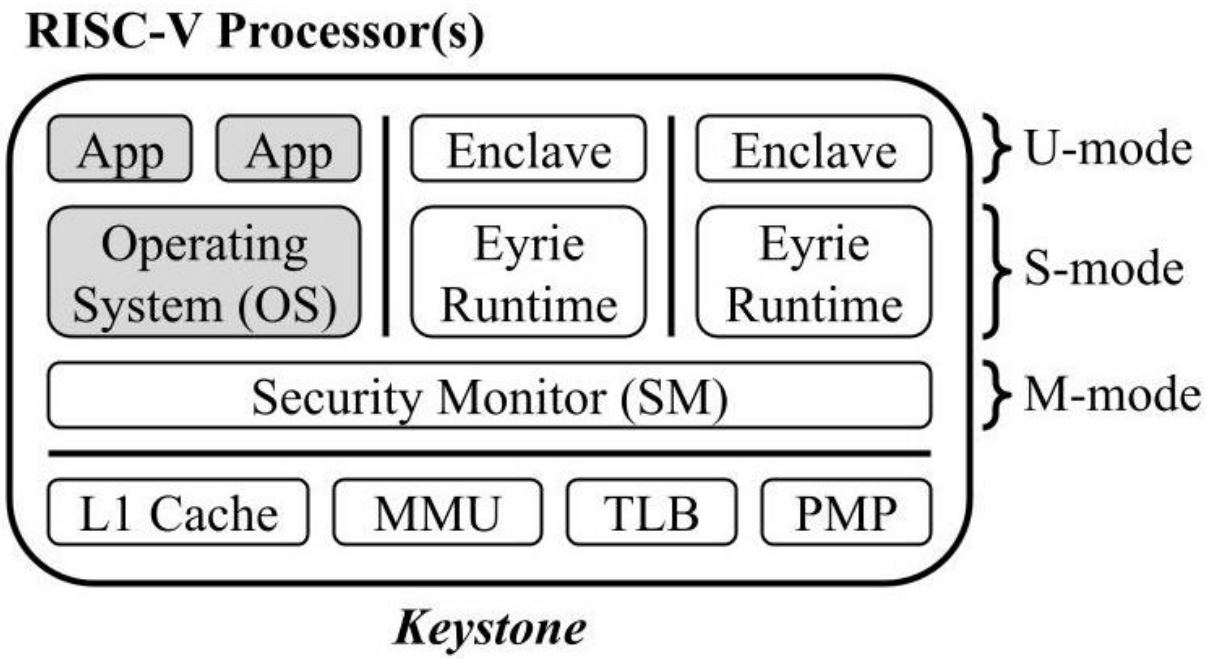
Sanctum



TIMBER-V

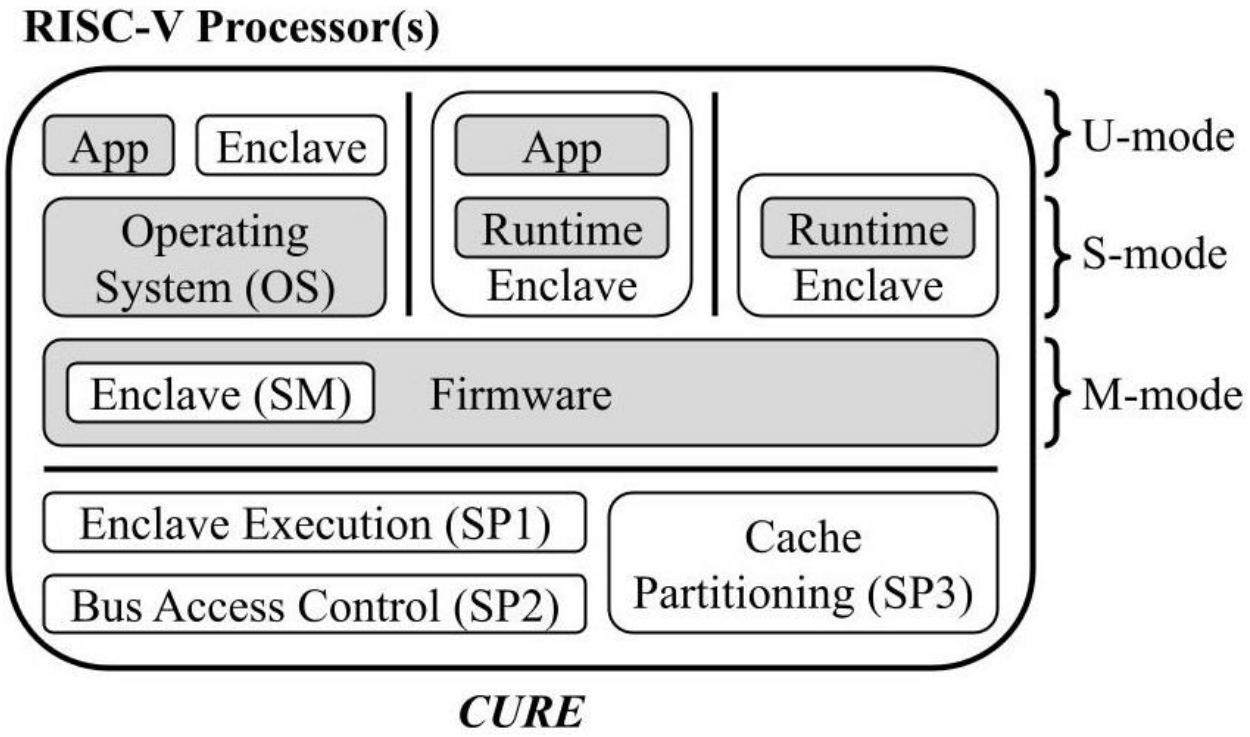
TIMBER-V: similar approach with Intel SGX, but uses strong hardware enforcers based on "Tag"-ID across the entire system

2. TPM & TEE (6/8) Several TEE examples



Keystone

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



CURE

CURE: 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)*

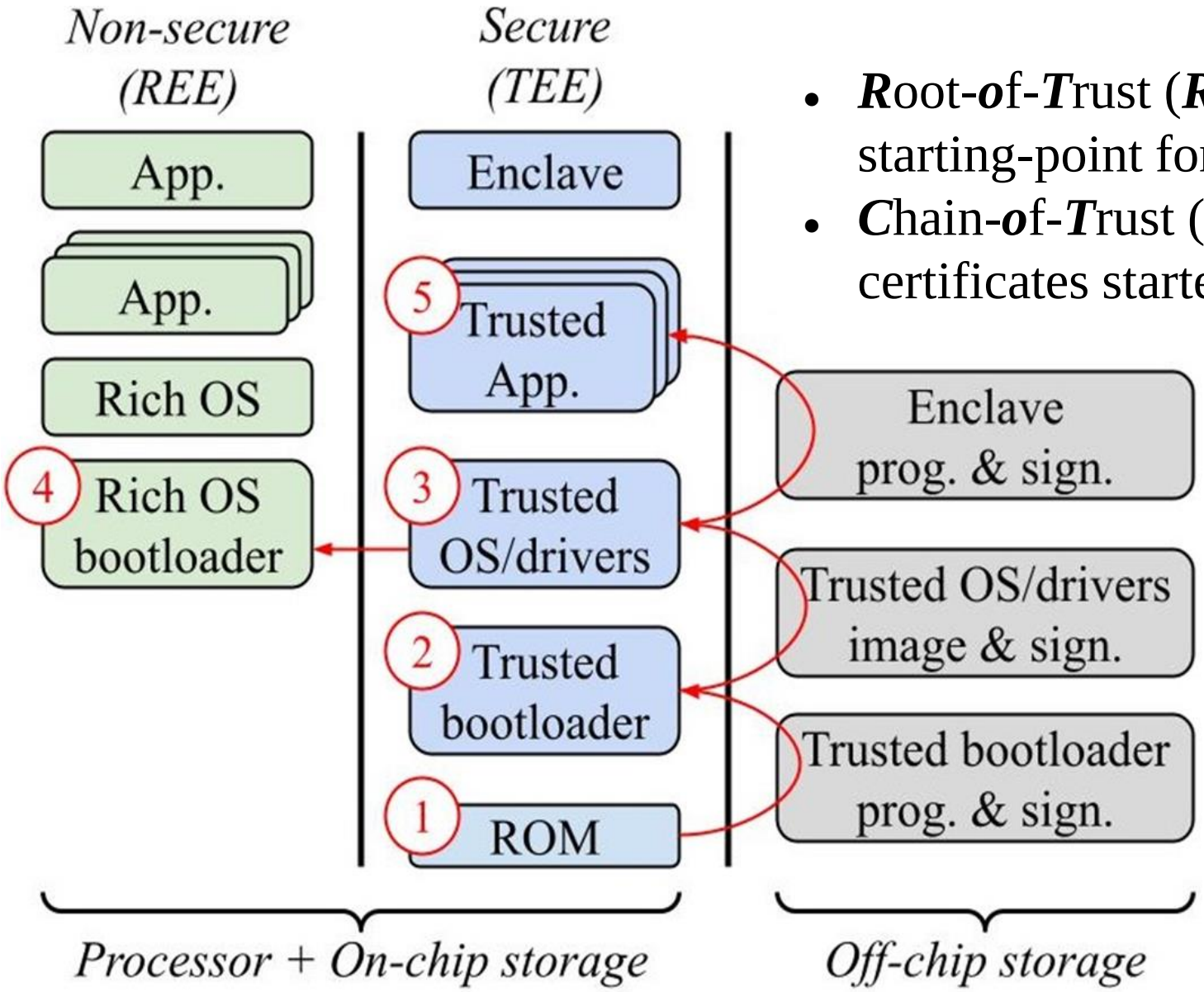
2. TPM & TEE (7/8) TEE comparison

TABLE 2.1: 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 [1]	Haven [22]	Graphene [23]	Scone [24]	TrustZone [2]	Komodo [26]	OP-TEE [27]	Sanctuary [28]	SEV [3]	SEV-ES [29]	SEV-SNP [30]	MultiZone [4]	Sanctum [5]	TIMBER-V [6]	Keystone [7]	CURE [8]
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		○	●	●	◐	○	●	●	●	●	●	●	◐	●	◐	●	○

- Various implementations for various purposes and applications:
  - ❑ **RISC-V:** with the advantage of open-source → fast to adapt and can be fine-tuned to any requirements.
  - ❑ **ARM:** aiming for SCA resilience, mostly for portable hand-held devices.
  - ❑ **Intel & AMD:** typical solution for generic PC and data center; aiming for heavy workload in those systems.

2. TPM & TEE (8/8) Secure boot in TEE



Secure boot in **TEE**:

- ***R***oot-***o***f-***T***rust (***RoT***): the first verification at reset, the starting-point for ***CoT***. This should be provided by **TPM**.
- ***C***hain-***o***f-***T***rust (***CoT***): a series of signatures & certificates started from the ***RoT*** up to the Rich OS.

Secure boot should guarantee:

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

Outline

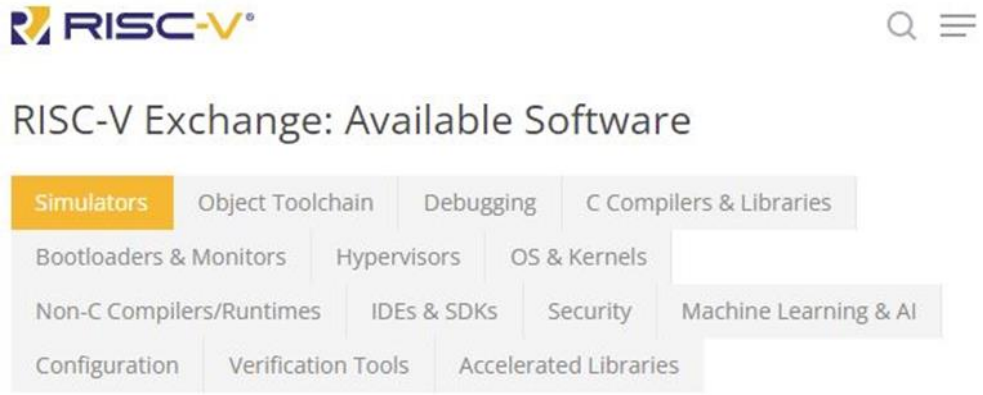
1. Introduction
2. TPM and TEE
3. Why RISC-V?
4. Proposed System
5. Peripherals
6. Result
7. Conclusion

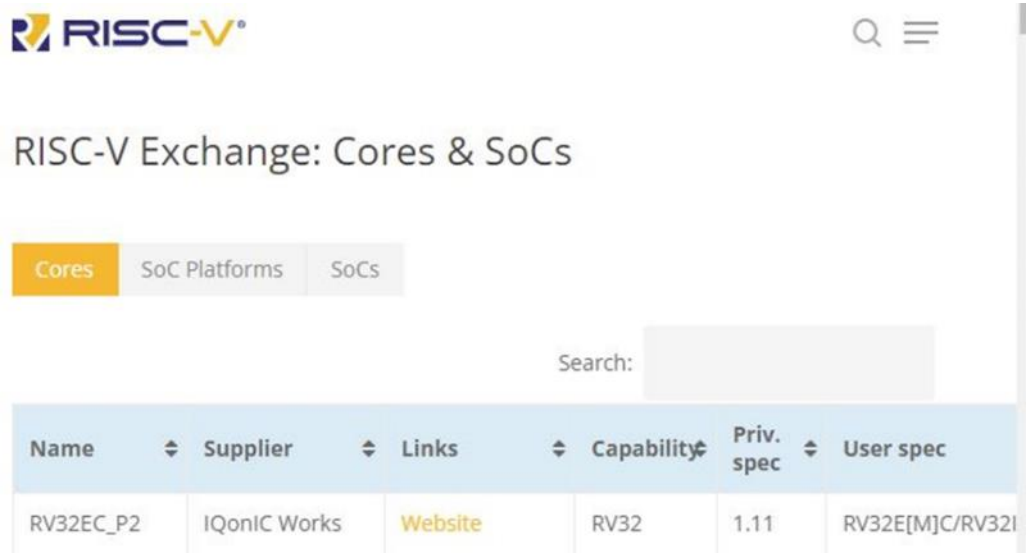
3. Why RISC-V? (1/12) RISC-V ISA

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/





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

License free:

- RISC-V ISA
- RISC-V toolchain

License depends on authors/developers:

- RISC-V processors
- RISC-V software applications
- RISC-V-related products

3. Why RISC-V? (2/12) What is ISA?

ISA means **I**nstruction **S**et **A**rchitecture.
It is the layer between software and hardware developers.

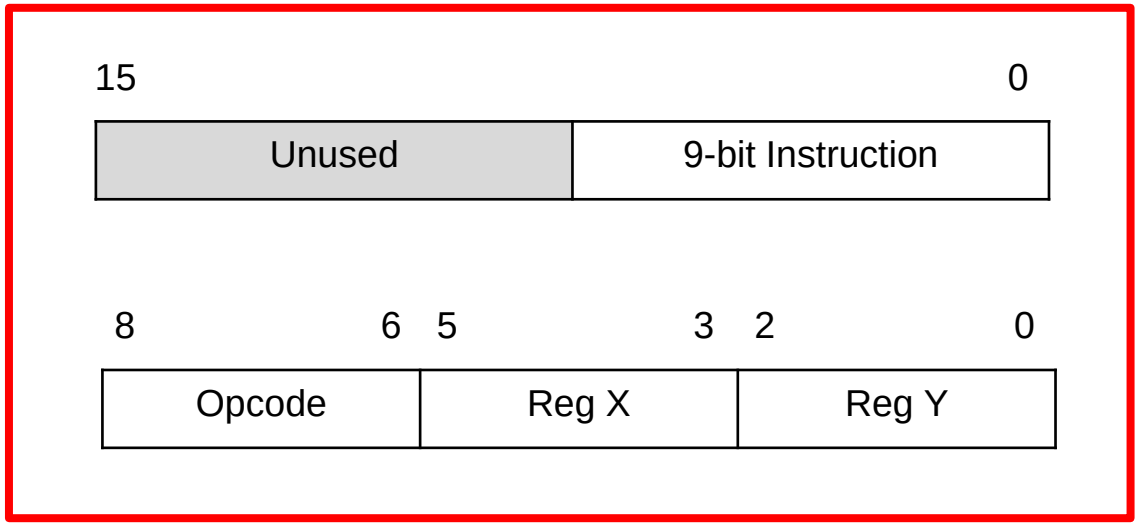
Software tools: assembler, compilers, debugger, linker, etc.

ISA: the interface between software & hardware architects

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

ISA has to define all these kinds of stuff:

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.*



3. Why RISC-V? (3/12) CISC vs. RISC

CISC

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

1) Emphasis on *hardware*
2) *Multi-clock* complex instructions
3) *Memory-to-memory* mindset
4) *Small* code size, *many* cycles per instruction
5) *Low Fmax* due to complex design
6) Most transistors are used for storing *instructions*
7) *Less memory* for storing data & program

RISC

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

1) Emphasis on *software*
2) *Single-clock* simple instructions
3) *Register-to-register* mindset
4) *Large* code size, *few* cycles per instruction
5) *High Fmax* due to simple design
6) Most transistors are used for storing *data*
7) *More memory* for storing data & program

RISC win CISC win

$$\textbf{Performance} = \frac{\text{time}}{\text{program}} = \frac{\text{time}}{\text{cycle}} \times \frac{\text{cycle}}{\text{instruction}} \times \frac{\text{instruction}}{\text{program}}$$

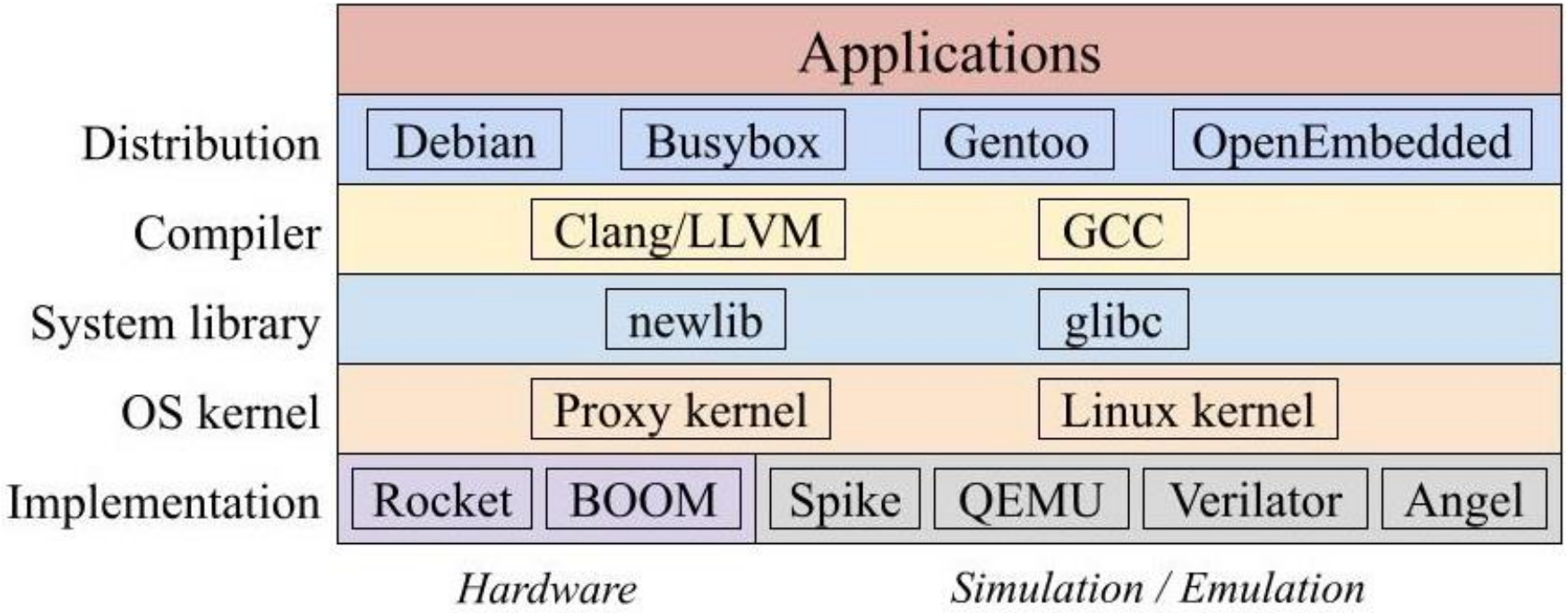
Nowadays, almost all processors in the market are RISCs.

Economic reason: memory price is way down ↓ ↓ ↓

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

3. Why RISC-V? (4/12) RISC-V toolchain

RISC-V toolchain and its ecosystem



(User's) software on the top

OS file system

Compiler

Standard libraries

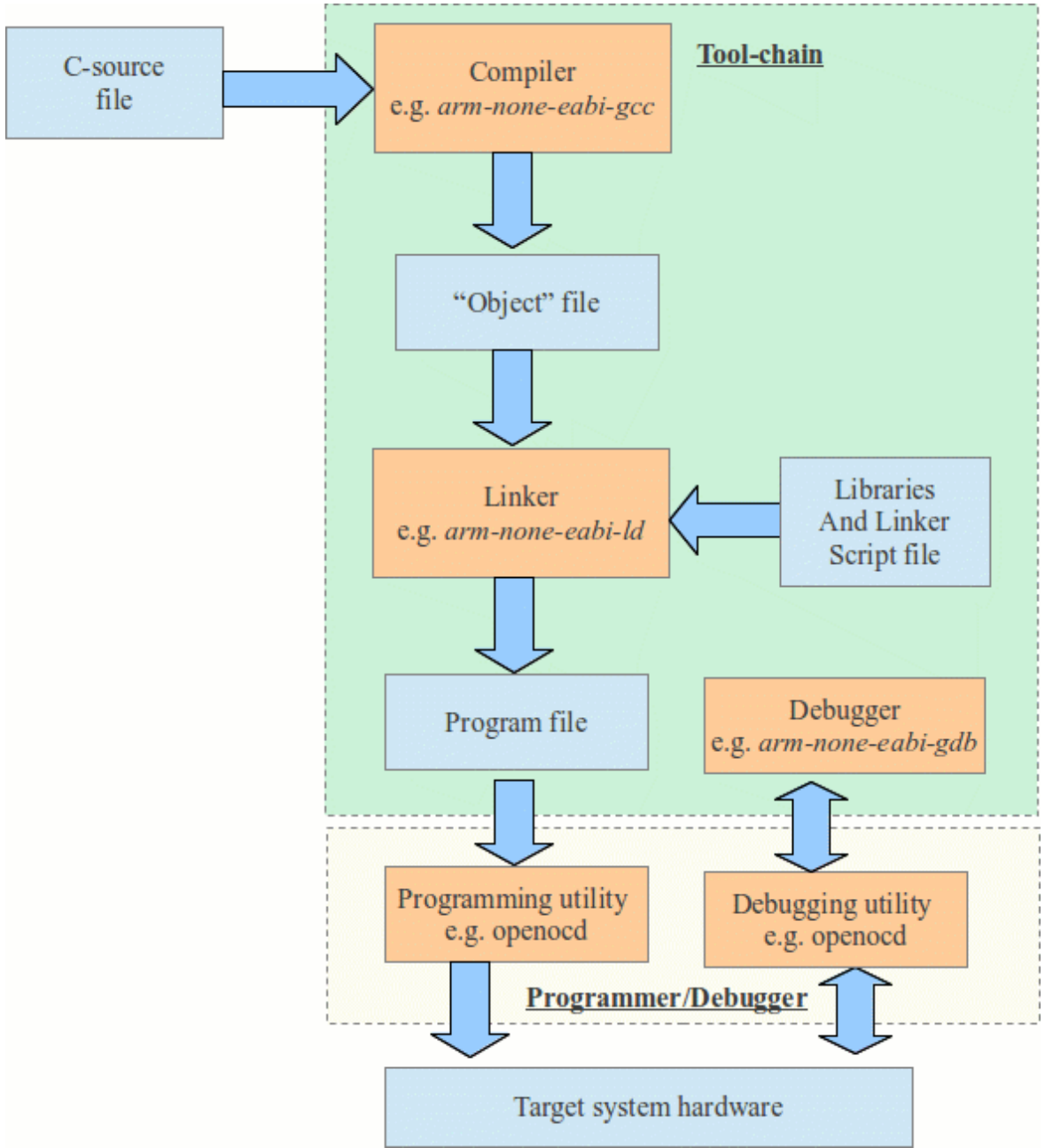
OS kernel

Hardware at the bottom

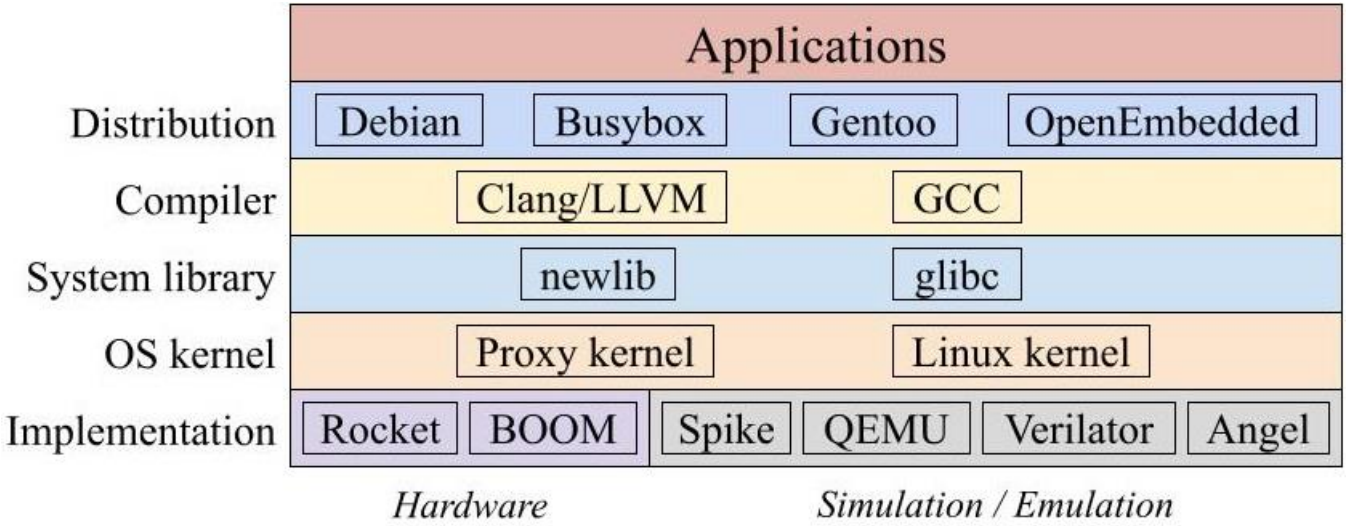
Top-down explanation:

User's applications on the top are operated in an OS file system, which then compiled by a compiler based on multiple standard libraries. After compiled, the execution file is run on the OS kernel that manages the hardware at the bottom.

3. Why RISC-V? (5/12) RISC-V toolchain



RISC-V toolchain and its ecosystem



Three most important tools

- **GCC:** *(cross C compiler)* makes 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

3. Why RISC-V? (6/12) RISC-V extension

What makes **RISC-V** different: its modular mindset

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

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 a lot more 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*

3. Why RISC-V? (7/12) OS stack

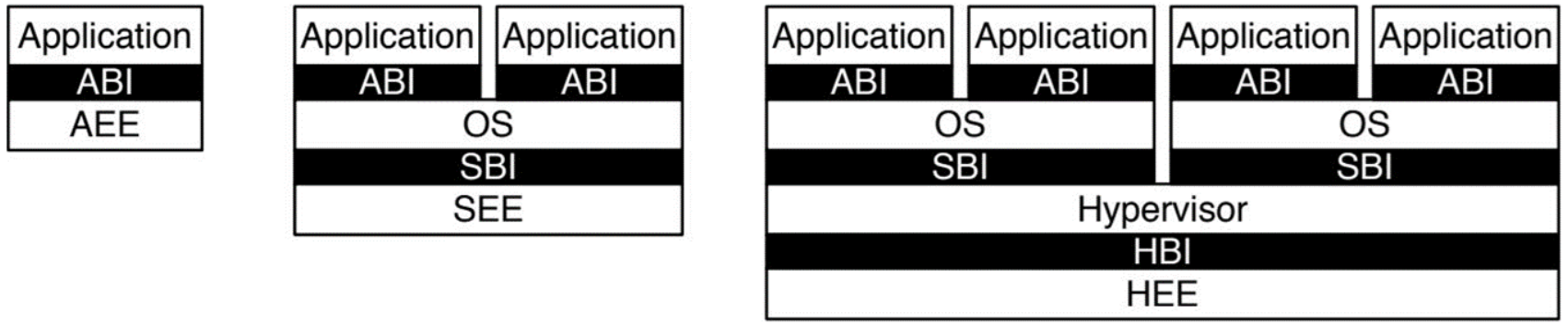
To support an Operating System (OS), the ISA has to support the OS stack *or the **M-/S-/U-mode***.

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

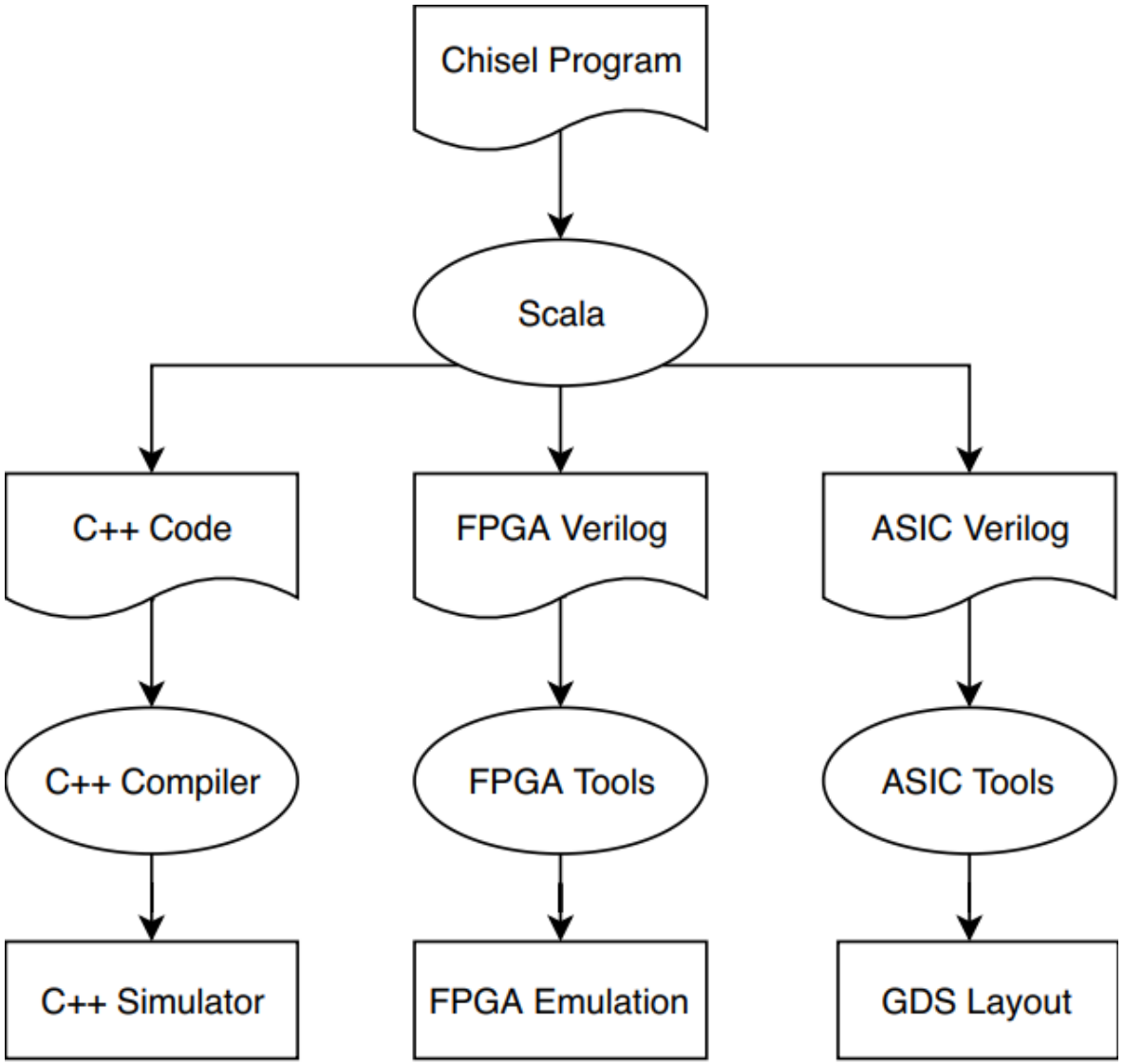
Different scenarios of utilizing the OS stack:



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

→ Better security scheme by having the hardware recognize each code's mode level. (read more at: *Link*)

3. Why RISC-V? (8/12) CHISEL



Chisel is a library.
Scala is a language.

- **Scala** itself is a *high-level object-oriented programming language*
  → It is not designed for "hardware coding."

- **Chisel** is a library attached to Scala to define a set of coding rules.
  → It is designed for "hardware coding."

- From **Scala** to **Verilog**:
  Scala → Java → FIRRTL → Verilog
  1st arrow: Scala compiler named SBT
  2nd arrow: executing Java
  3rd arrow: FIRRTL compiler

3. Why RISC-V? (9/12) Summary

RISC-V revolutionizes *computer system* design

1. **Modular at heart:**
   customizable ISA and customizable hardware
   → fine-tune the system to your specific needs.

2. **Open-source community:**
   license-free ISA, open cores and SoCs, open-source libraries, open-source software, etc. → reuse other developers' designs → save time and effort for R&D

3. **CHISEL** ***(Constructing Hardware In Scala Embedded Language)*****:**
   a new way to "coding" hardware circuits. When compiled, it will generate a true RTL Verilog code.
   → a "meta-programming" language for hardware developers with parameters and sub-designs that can be overridden or extended.
   → easy to develop "object-oriented" hardware library for reuse purpose.

3. Why RISC-V? (10/12) Common libraries

The common open RISC-V libraries that you can use

Chipyard *(contains many common and frequently used open IPs, including RISC-V processors and other peripherals such as uart, spi, sd-card, etc.)*: *https://github.com/ucb-bar/chipyard*

README.md

CHIPYARD

Chipyard Framework

chipyard-ci-process passing

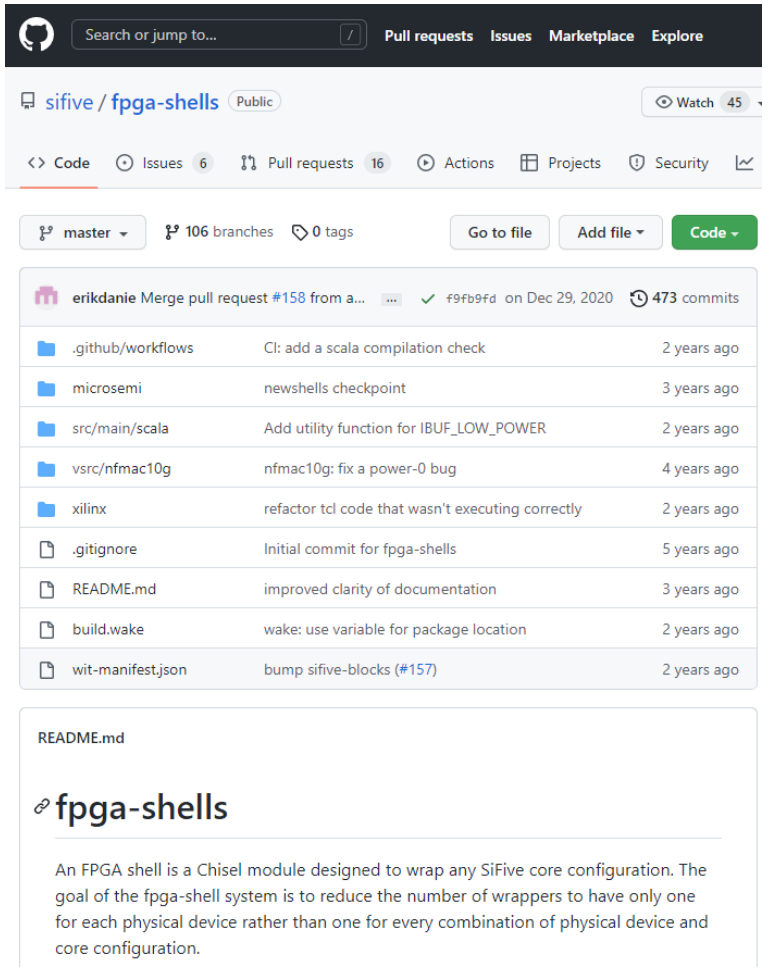
Quick Links

- **Stable Documentation:** https://chipyard.readthedocs.io/
- **User Question Forum:** https://groups.google.com/forum/#!forum/chipyard
- **Bugs and Feature Requests:** https://github.com/ucb-bar/chipyard/issues

Using Chipyard

To get started using Chipyard, see the stable documentation on the Chipyard documentation site: https://chipyard.readthedocs.io/

What is Chipyard



fpga-shells *(contains many common FPGA configurations)*: *https://github.com/sifive/fpga-shells*

3. Why RISC-V? (11/12) Common processors

Some famous RISC-V processors

Rocket is the most popular among RISC-V processors: https://github.com/chipsalliance/rocket-chip
(it is an in-of-order processor)

README.md

Rocket Chip Generator

Continuous Integration passing

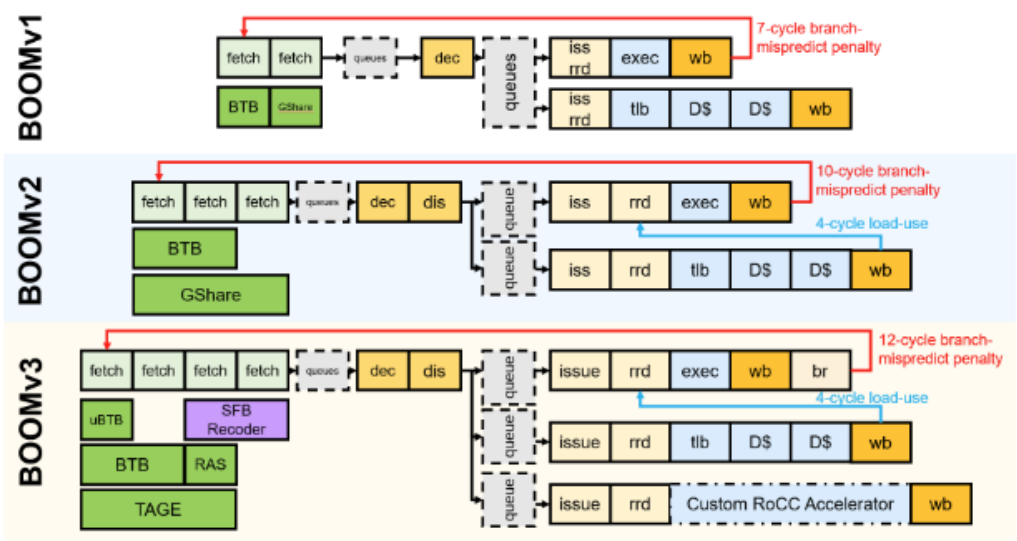
This repository contains the Rocket chip generator necessary to instantiate the RISC-V Rocket Core. For more information on Rocket Chip, please consult our technical report.

Table of Contents

- Quick instructions for those who want to dive directly into the details without knowing exactly what's in the repository.
- What's in the Rocket chip generator repository?
- How should I use the Rocket chip generator?
  - Using the cycle-accurate Verilator simulation
  - Mapping a Rocket core down to an FPGA
  - Pushing a Rocket core through the VLSI tools
- How can I parameterize my Rocket chip?
- Debugging with GDB
- Building Rocket Chip with an IDE
- Contributors

BOOM is an out-of-order processor that can rival ARM: https://github.com/riscv-boom/riscv-boom

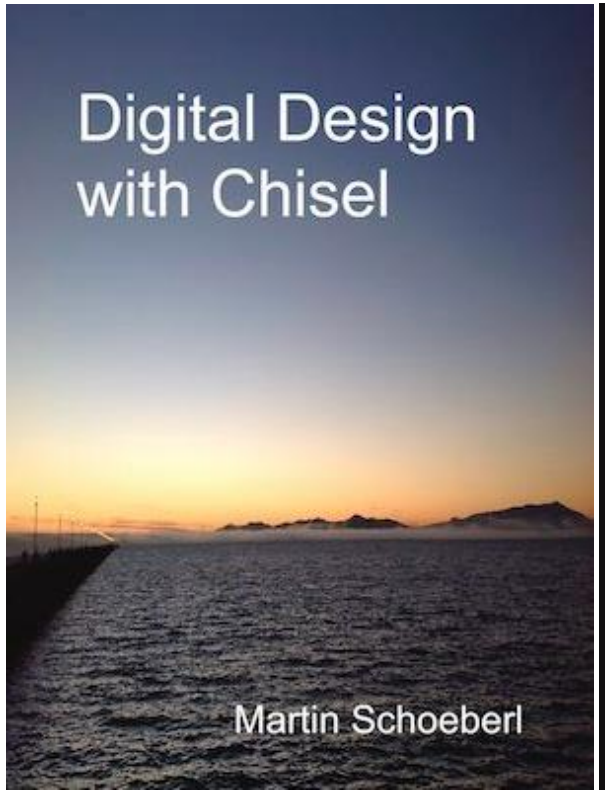
README.md

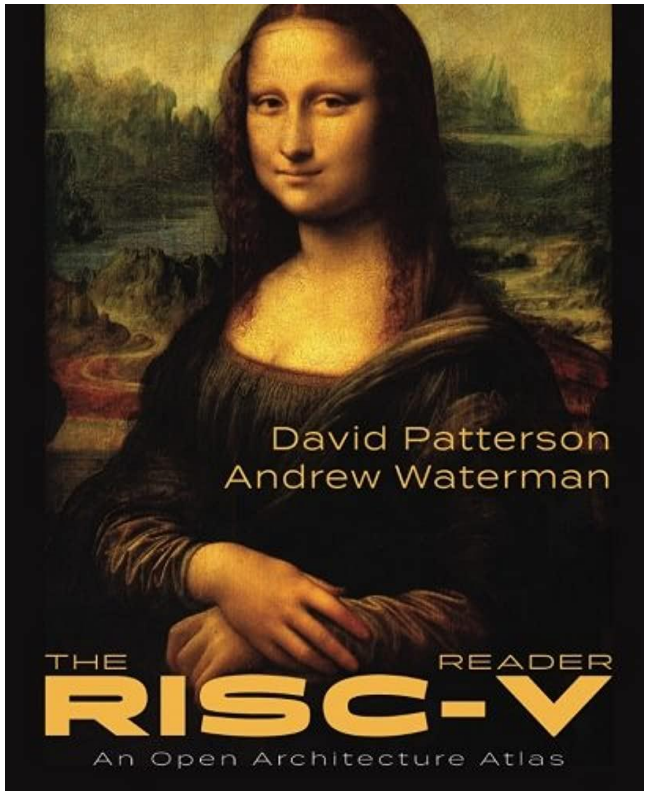


The Berkeley Out-of-Order RISC-V Processor

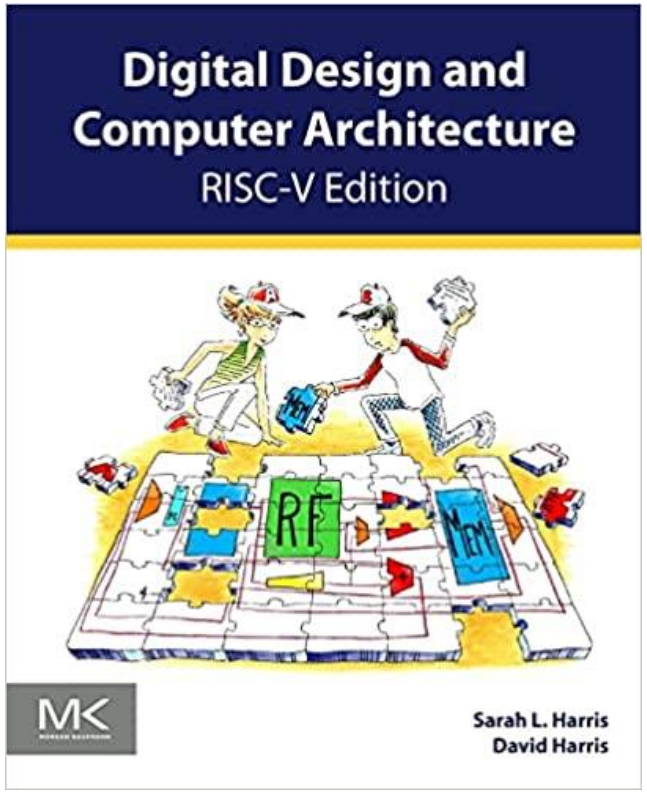
3. Why RISC-V? (12/12) Common books

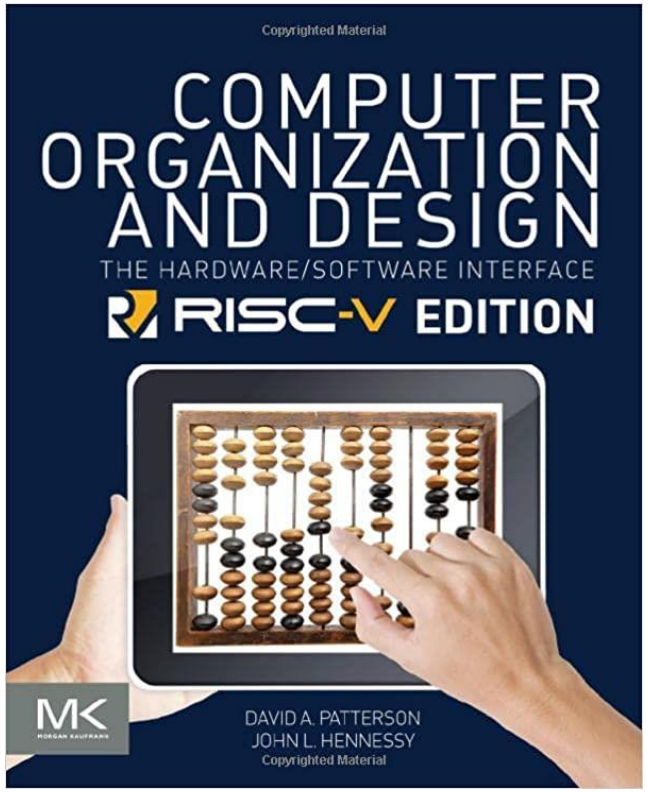
Two “must-have” books for RISC-V developers, from beginners to experts





RISC-V books that often used in universities for teaching



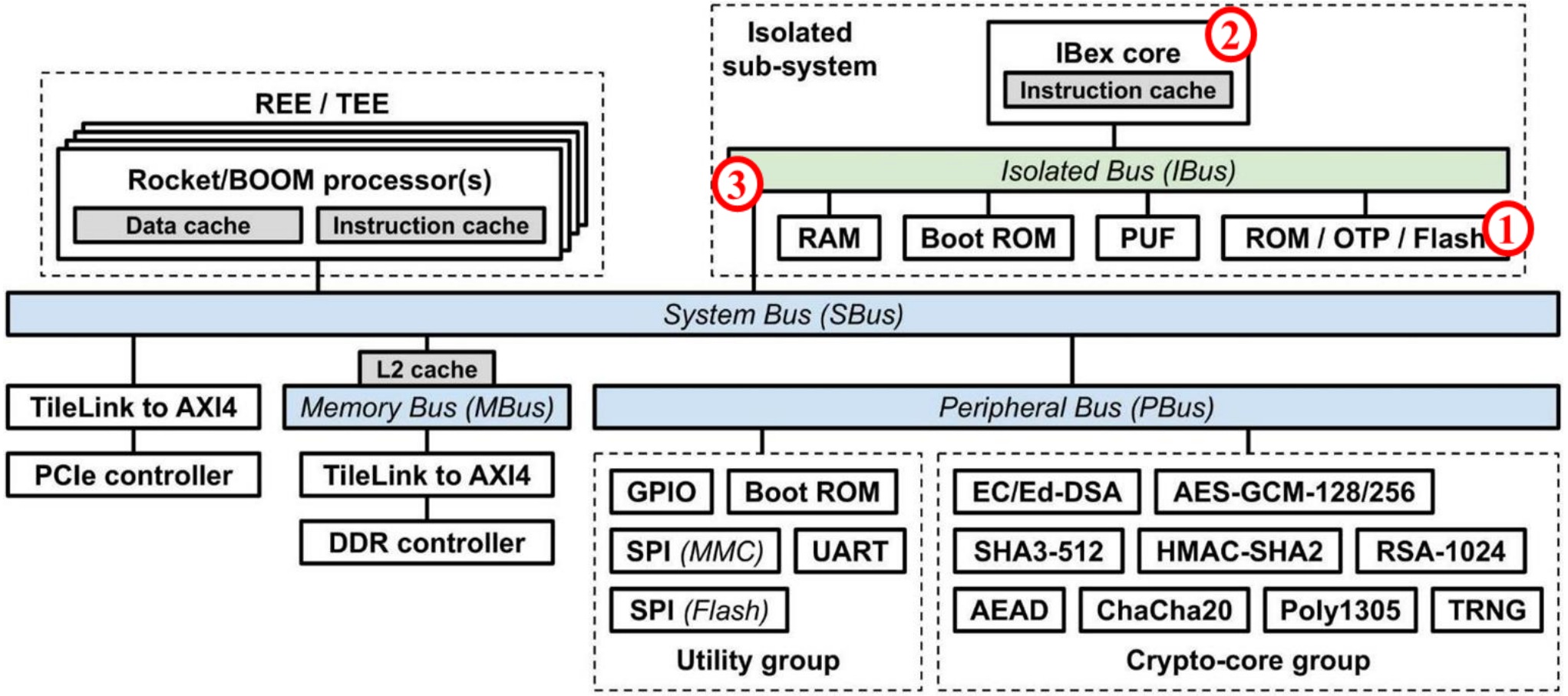


Outline

1. Introduction
2. TPM and TEE
3. Why RISC-V?
4. Proposed System
5. Peripherals
6. Result
7. Conclusion

4. Proposed System (1/8) Hardware modification

Proposed secure boot process with Root-of-Trust for **TEE**

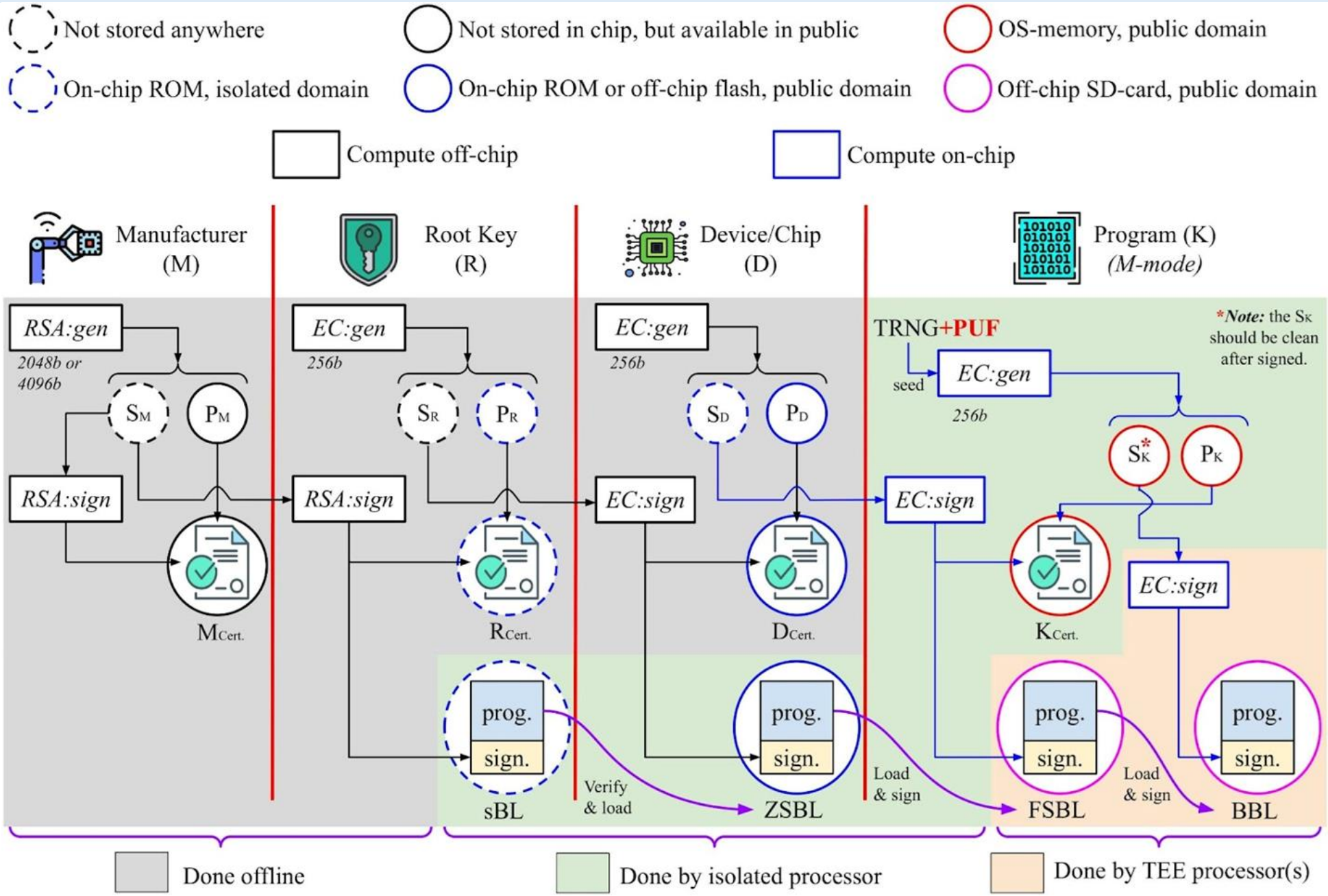


1. Root key installed at the time manufactured.
2. Hidden MCU for the flexible boot program

3. Hierarchy-bus: TEE processors cannot access RAM/ROMs in the isolated domain *(BUT the isolated core can access ALL)*

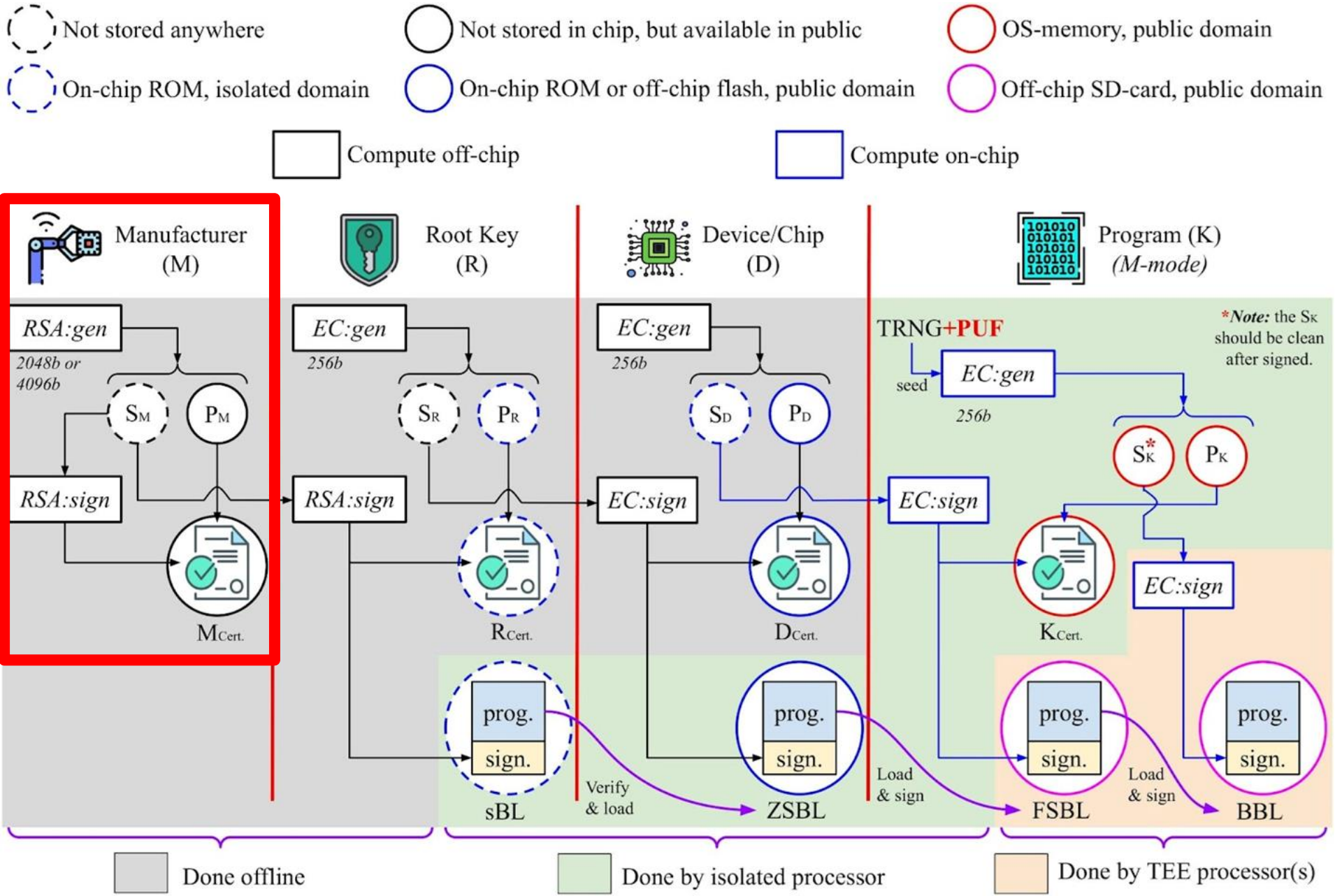
4. Proposed System (2/8) Key scheme

The proposed keys scheduling scheme



[Link]

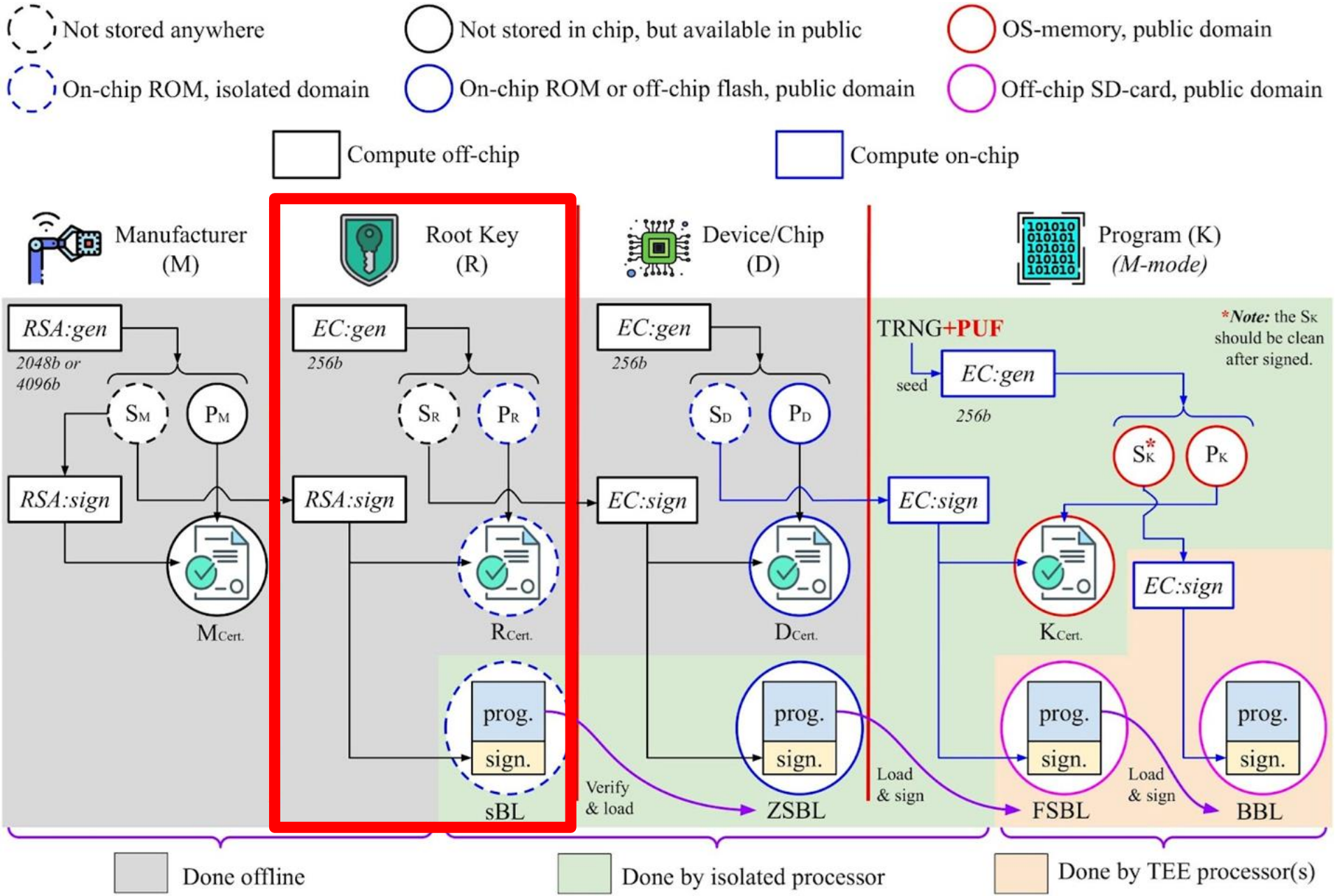
4. Proposed System (3/8) Key scheme: root CA



Step-by-step

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

4. Proposed System (4/8) Key scheme: developer cert.

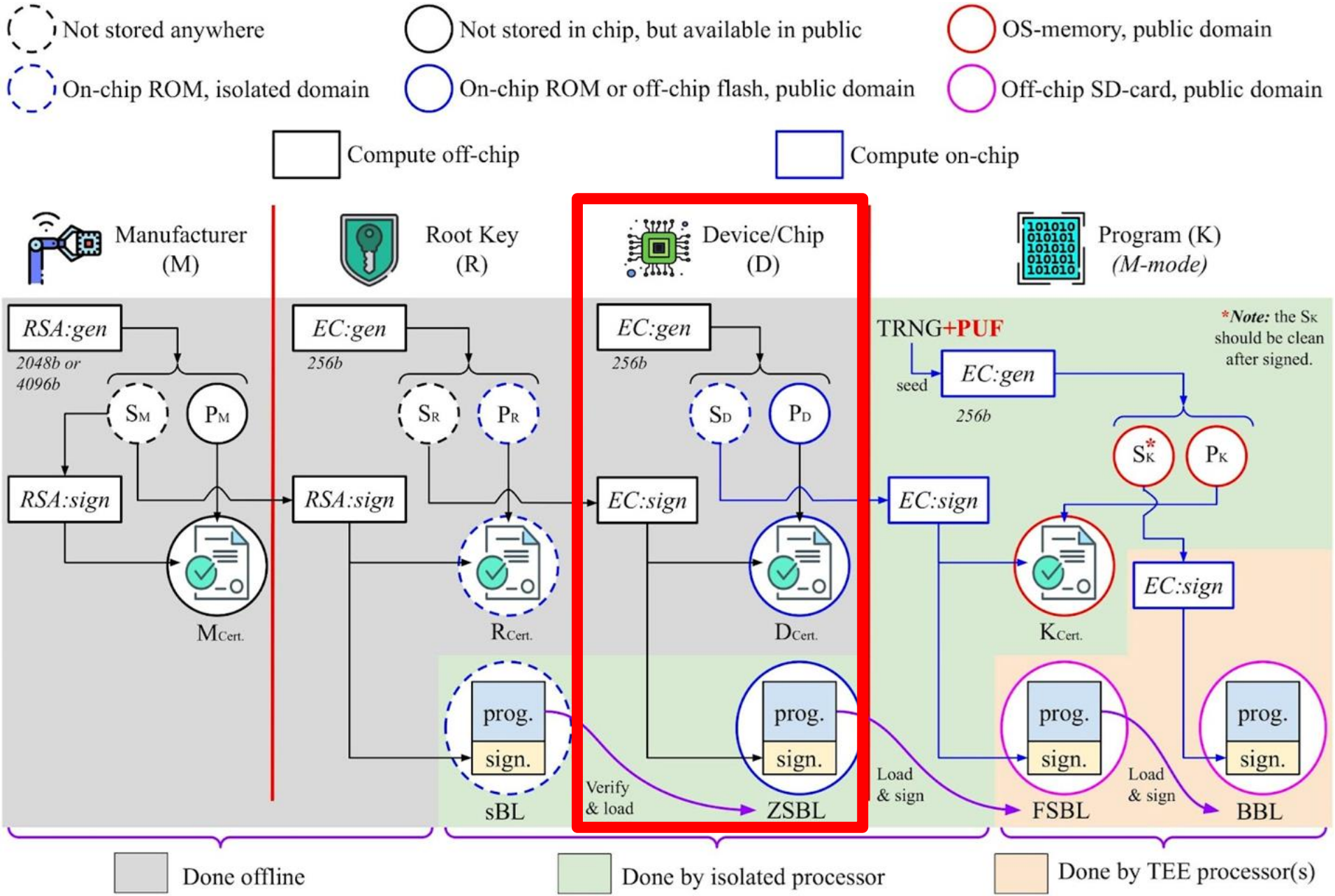


Step-by-step

- **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.

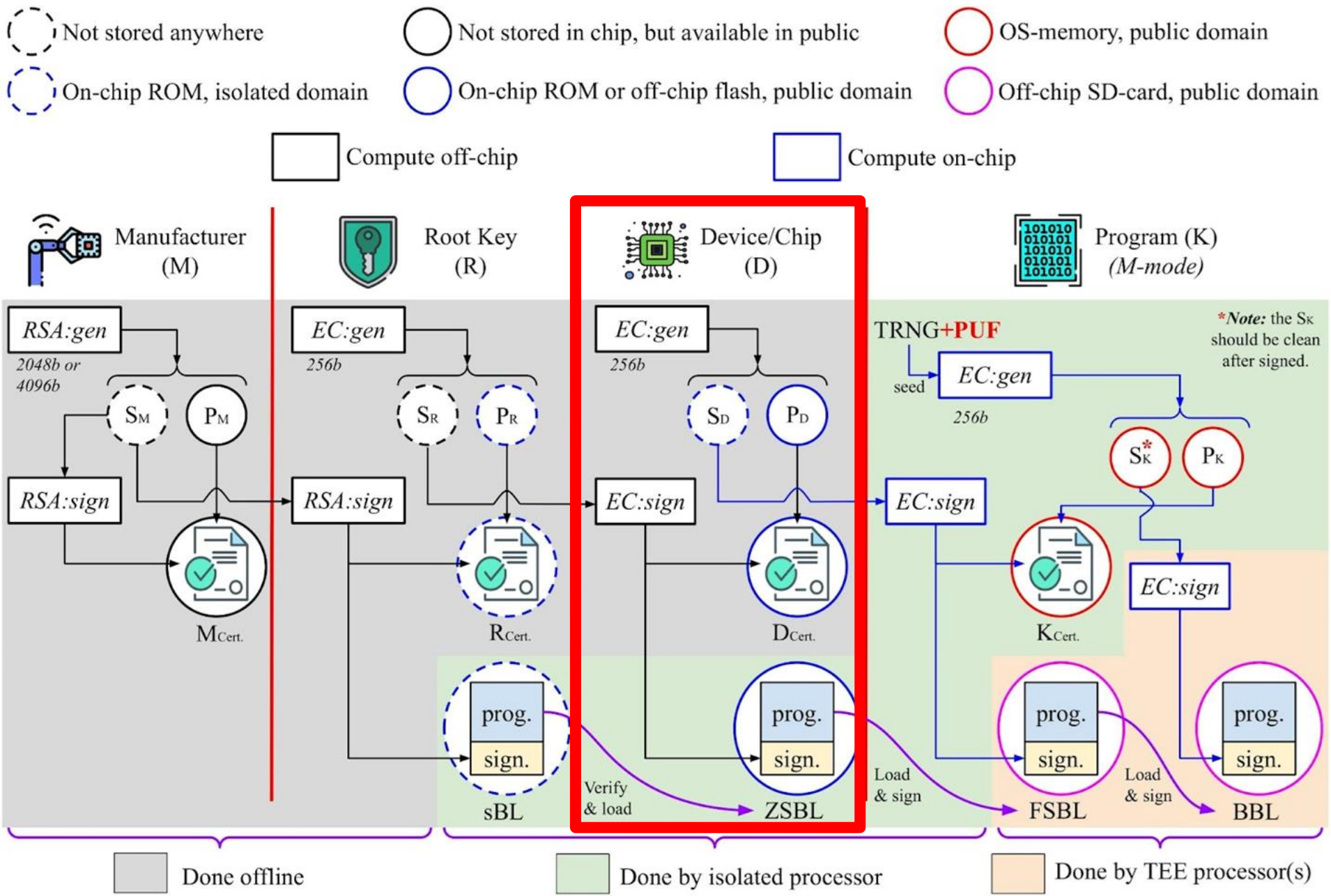
4. Proposed System (5/8) Key scheme: product cert.



Step-by-step

- **Step 3:** *(still offline)* the manufacturer *(or the provider)* generates the pair SD & PD.
  Then have the root secret key generates the DCert. and sign the ZSBL.

4. Proposed System (6/8) Key scheme: updatable ZSBL

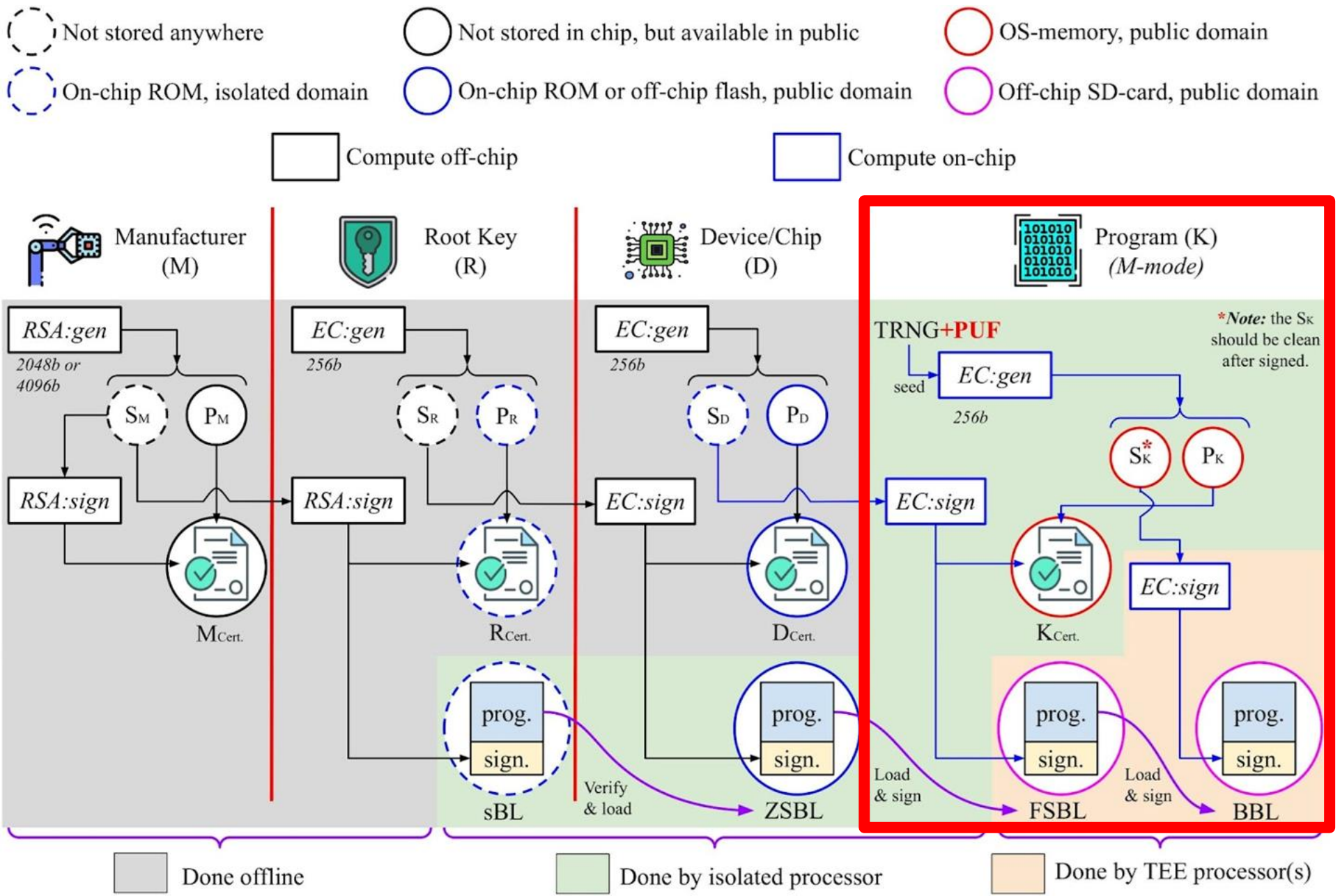


Here is the RoT

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

→ this allows future updates on the ZSBL.

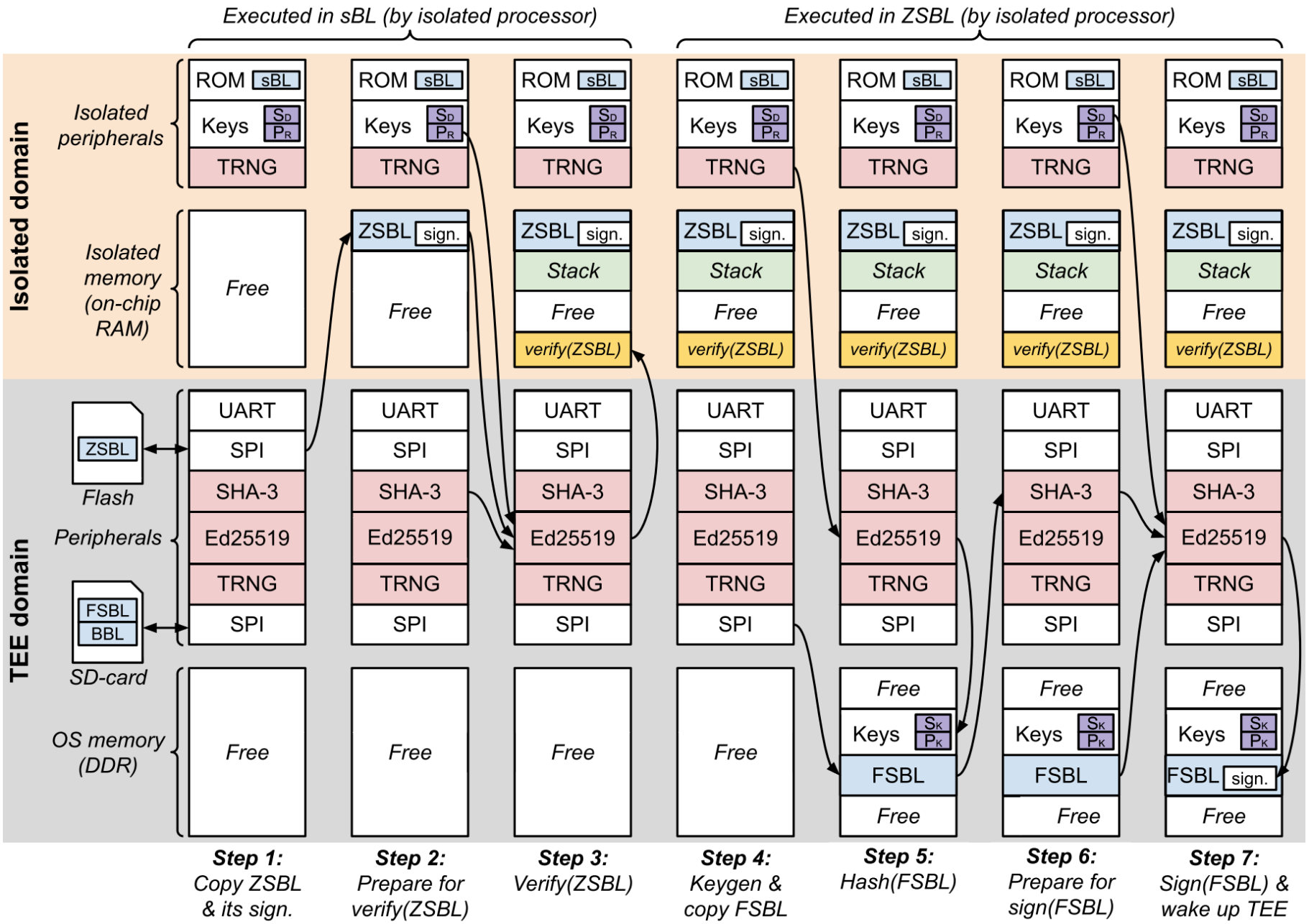
4. Proposed System (7/8) Key scheme: program cert.



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 SK & PK
  - Load the FSBL *(hash & sign)* to the public RAM.
  - Wakes up the TEE processors

4. Proposed System (8/8) Detail boot flow

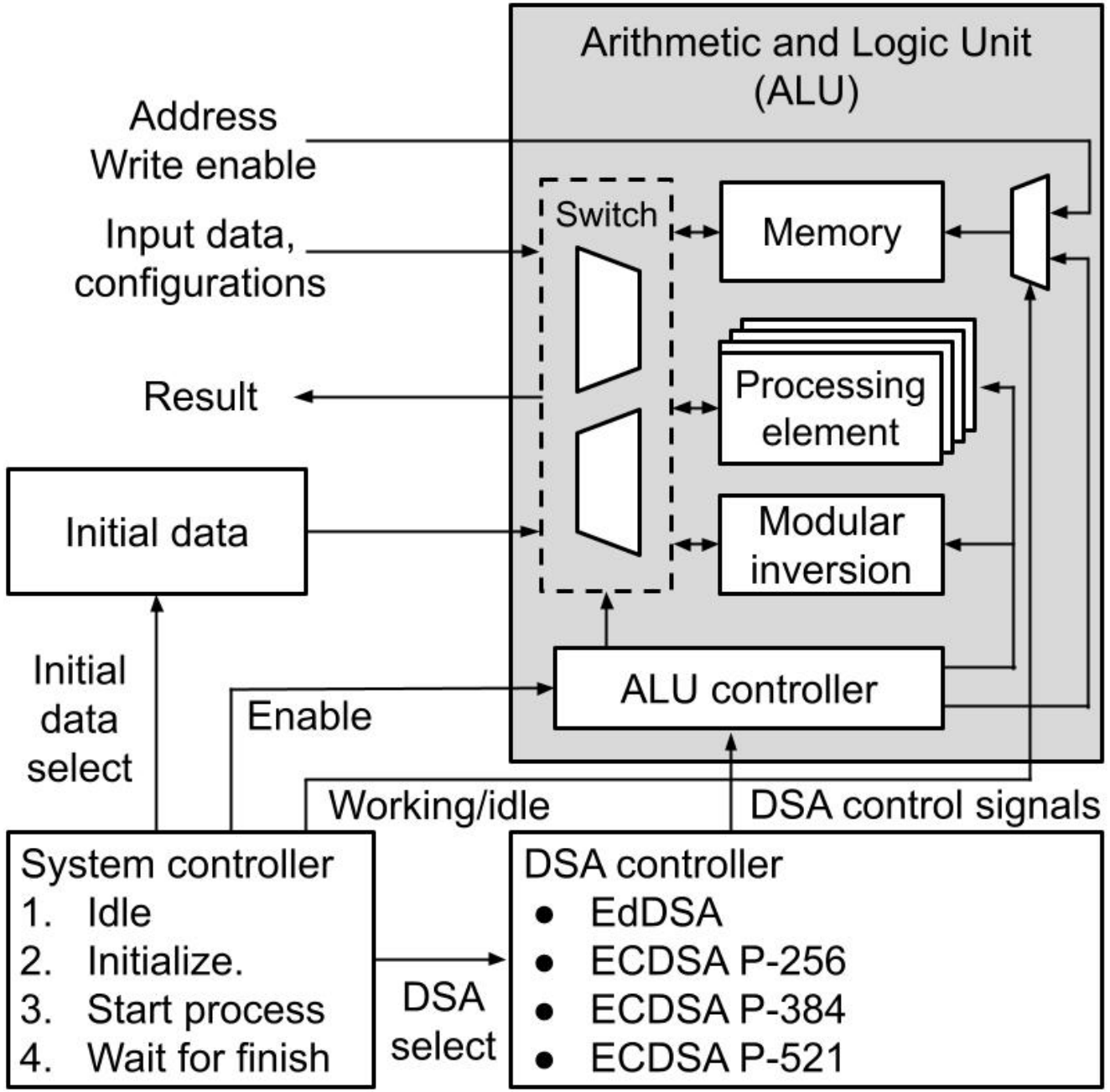


The detail boot flow based on the proposed key scheduling.

Outline

1. Introduction
2. TPM and TEE
3. Why RISC-V?
4. Proposed System
5. Peripherals
6. Result
7. Conclusion

5. Peripherals (1/14) EC/Ed-DSA

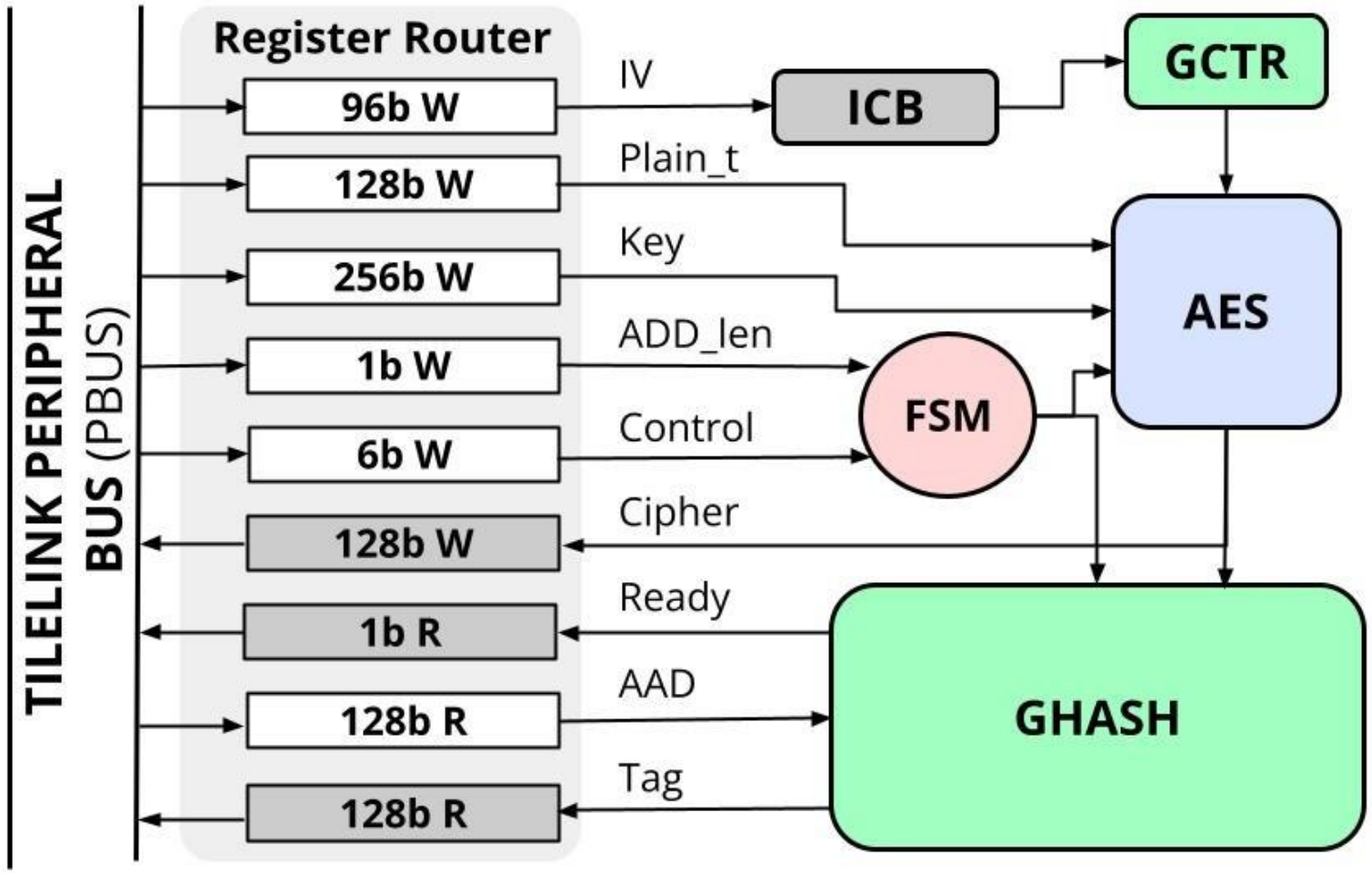


Ecliptic ***C***urve (***EC***) and
Edwards-curve (***Ed***)
Digital ***S***ignature ***A***lgorithm (***DSA***)

- Support four curves: three of ECDSA and one of EdDSA
- Support 256-bit, 384-bit, and 512-bit
- Support functions: gen-key, sign, and verify

Link

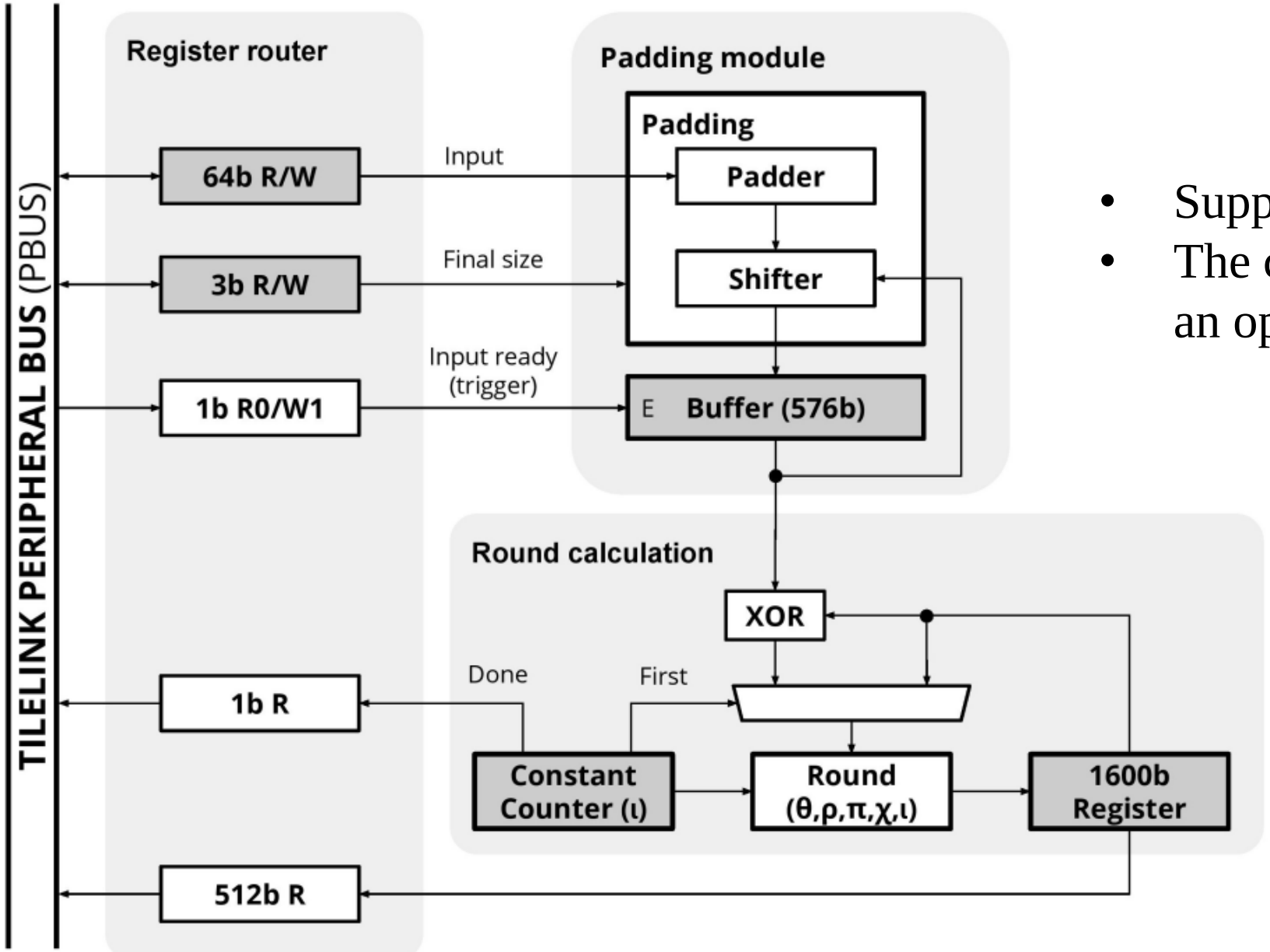
5. Peripherals (2/14) AES-GCM



Advanced ***E***ncryption ***S***tandard (***AES***) with ***G***alois/***C***ounter ***M***ode (***GCM***)

- Support encryption and decryption.
- Support 128-bit and 256-bit.

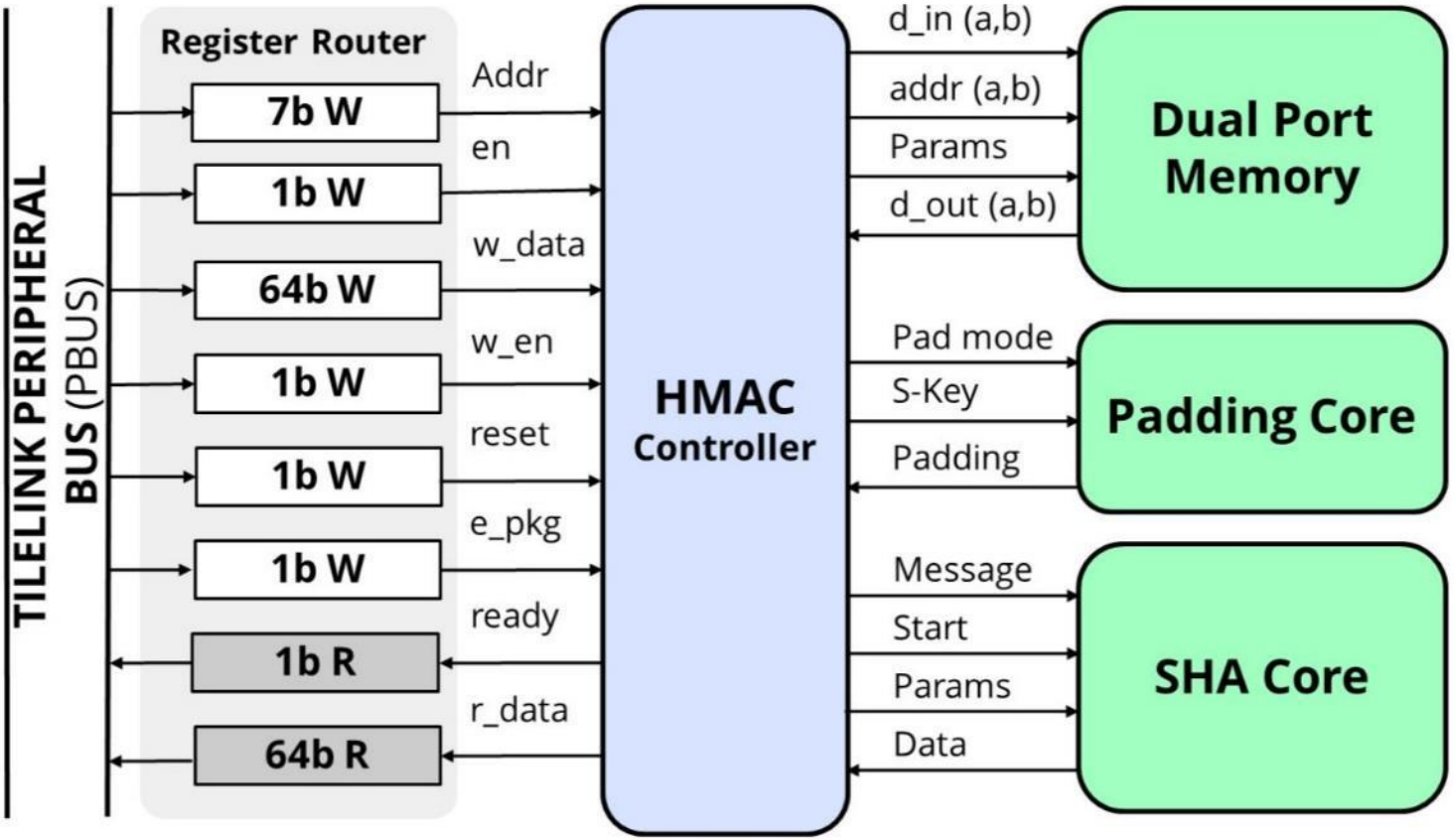
5. Peripherals (3/14) SHA3



SHA3-512

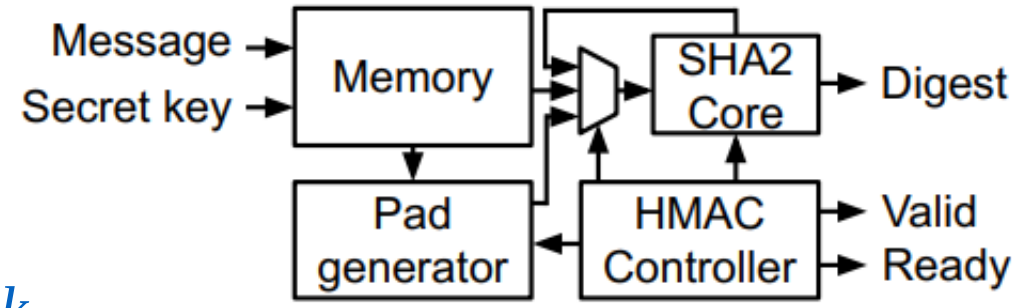
- Support 512-bit.
- The core was developed based on an open-source project (*link*).

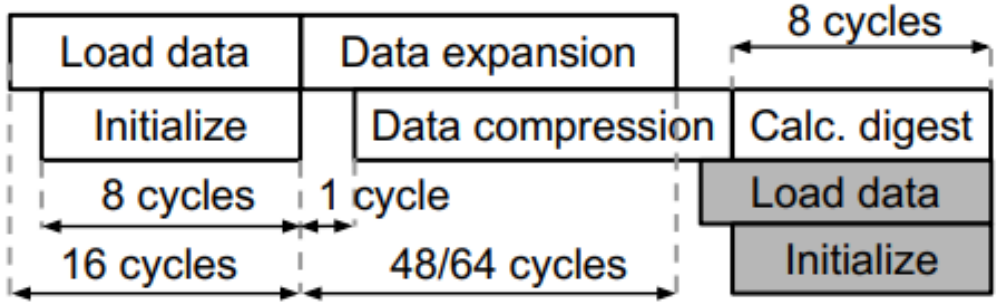
5. Peripherals (4/14) HMAC-SHA2



Hash-based ***M***essage ***A***uthentication ***C***ode (***HMAC***) with SHA2

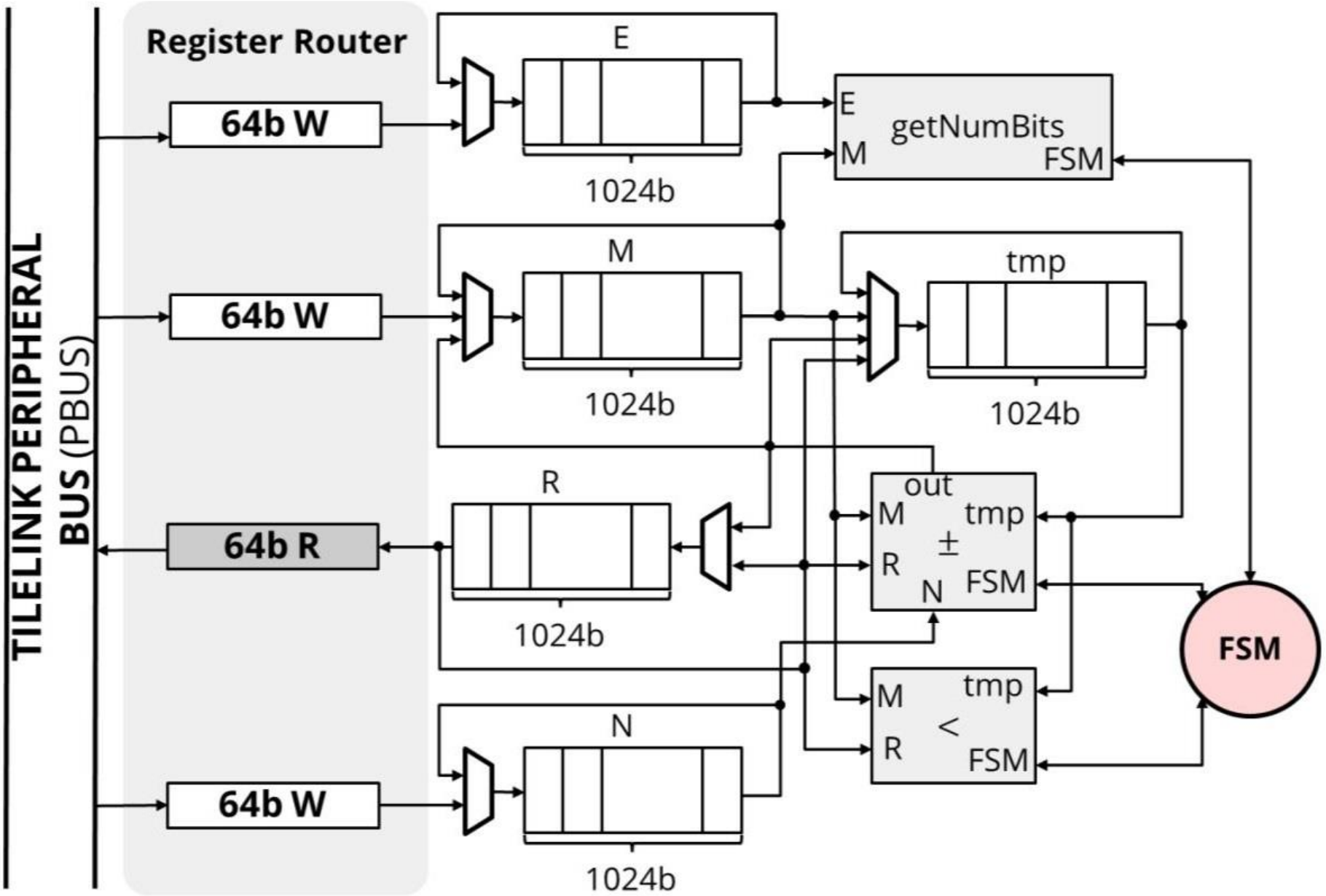
- Two modes: HMAC-SHA2 or SHA2 only.
- Support: 256-bit, 384-bit, and 512-bit.





[Link]

5. Peripherals (5/14) RSA



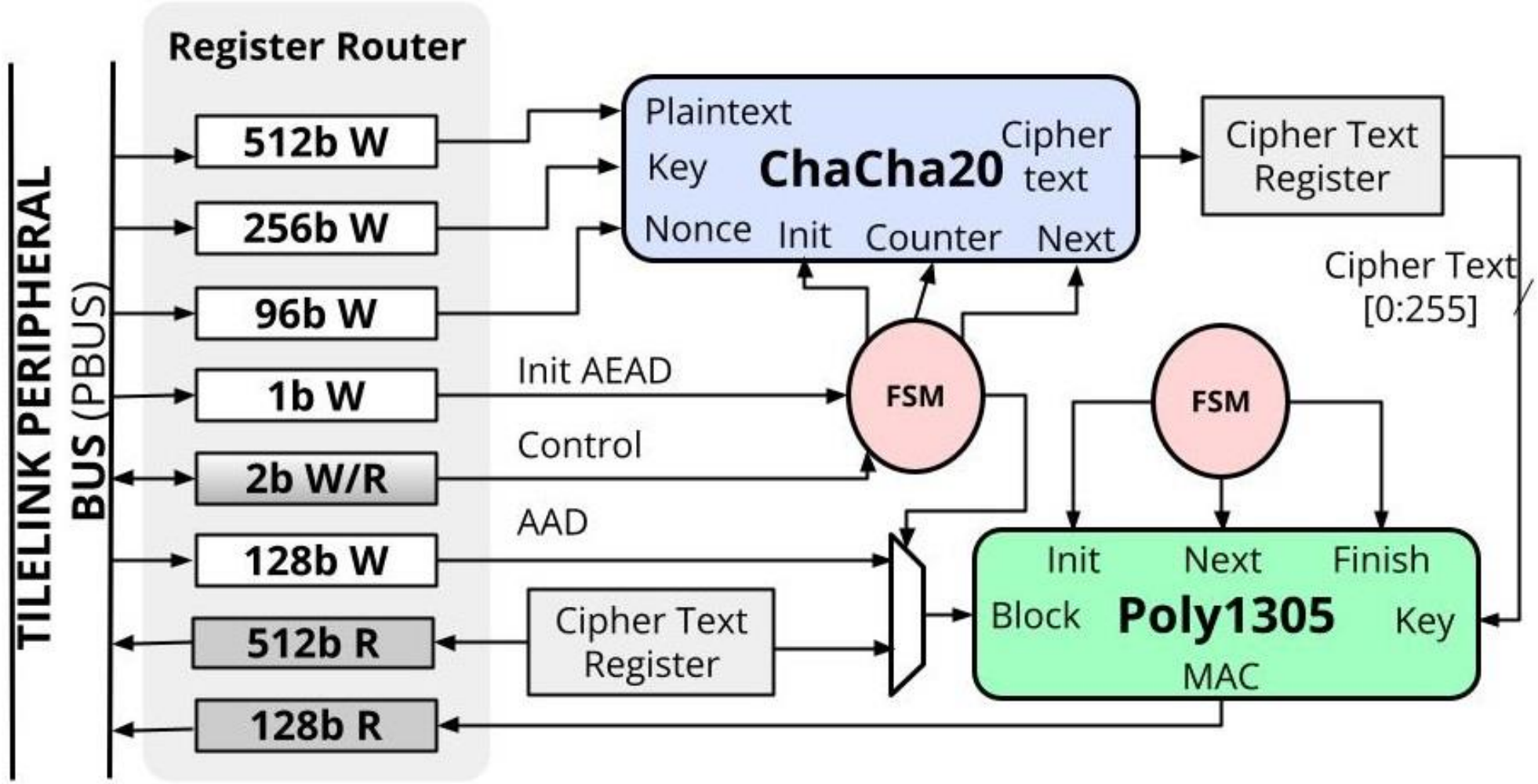
RSA-1024

- Minimize the area by using less “big”-registers as much possible
- Small tasks are done by primitives such as *getNumBits* (number of meaning LSBs), ±, and <
- Primitive functions execute 32-bit at a time

5. Peripherals (6/14) AEAD

Authenticated Encryption with Associated Data (*AEAD*) (*link*)

- Use *ChaCha20* as a stream cipher and *Poly1305* as a MAC.



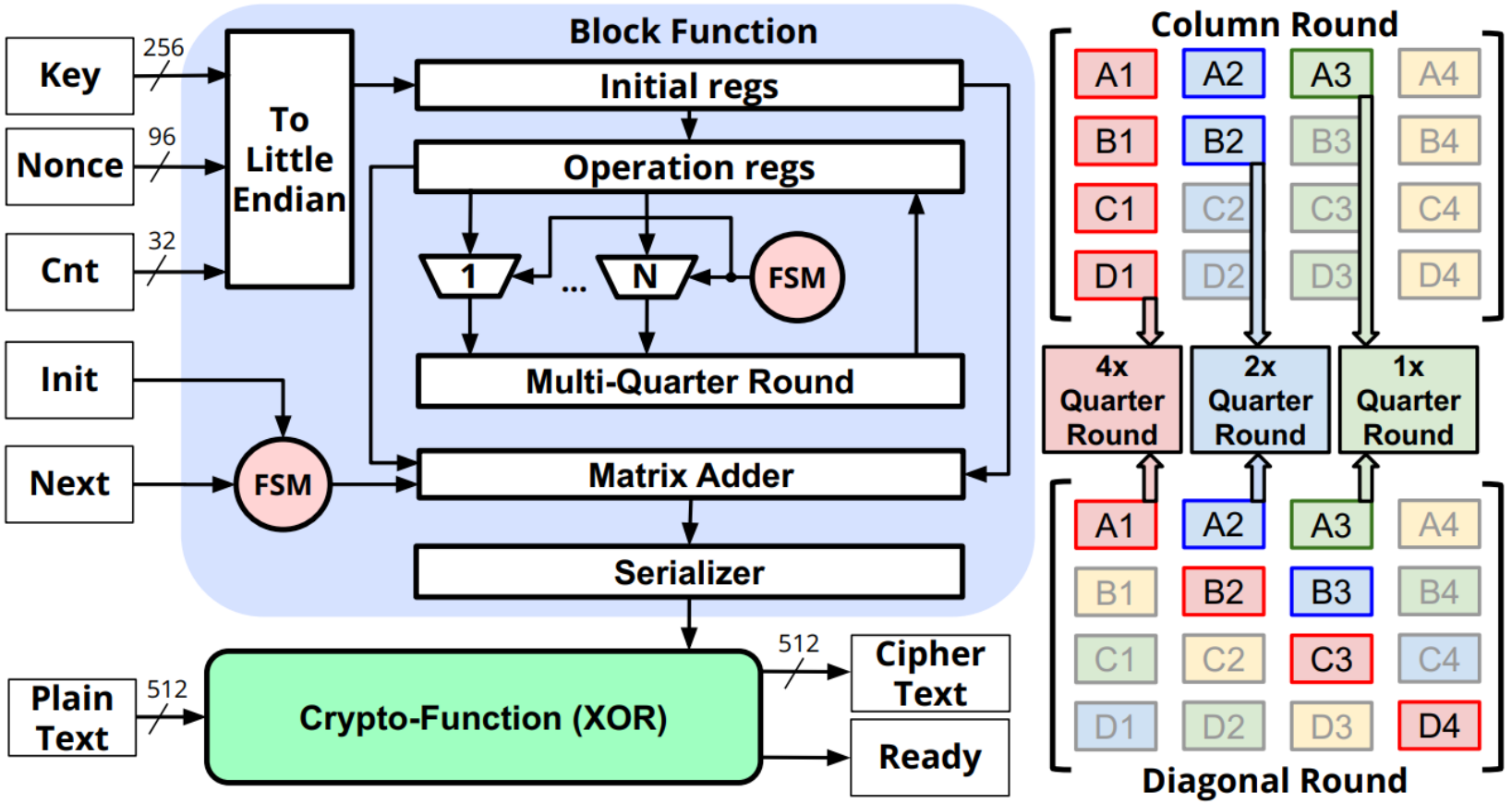
Link1

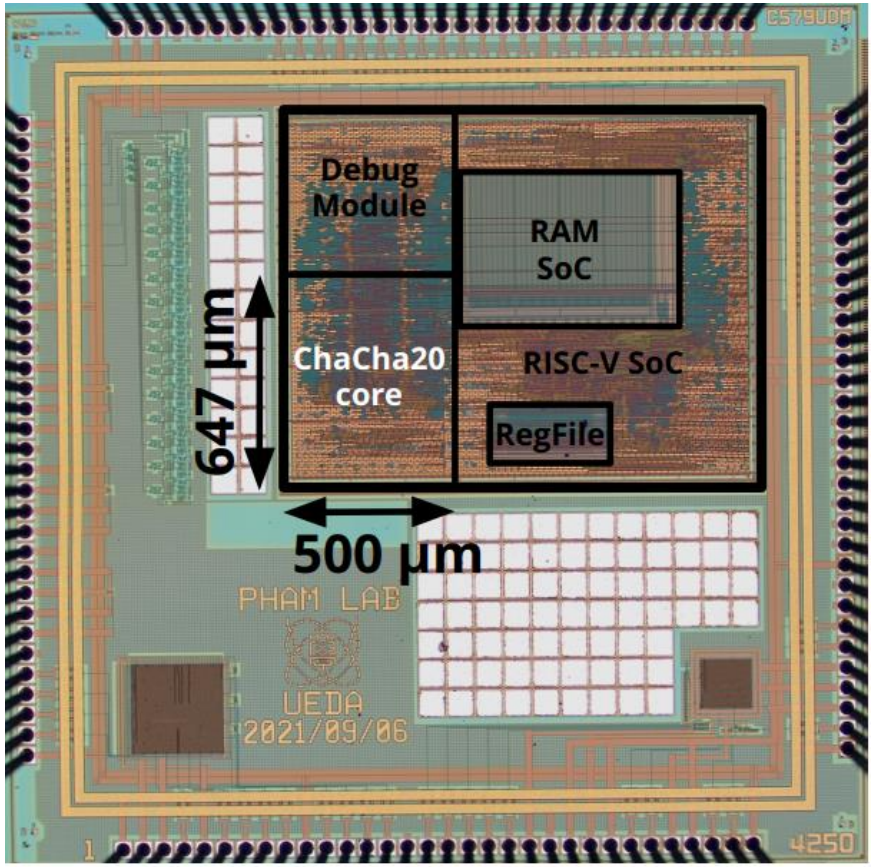
Link2

5. Peripherals (7/14) ChaCha20

ChaCha20

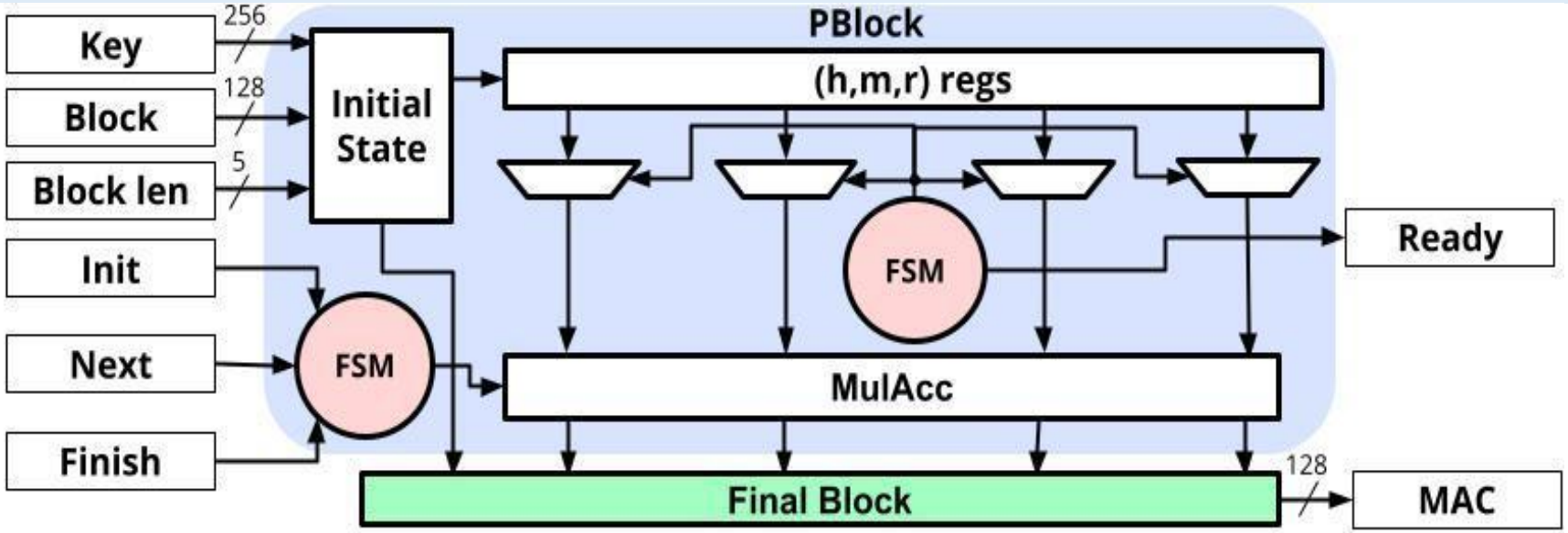
- A stream cipher that was standardized recently (*link*).
- Can work alone or team-up with *Poly1305* to perform ***A***uthenticated ***E***ncryption with ***A***dditional ***D***ata (***AEAD***).



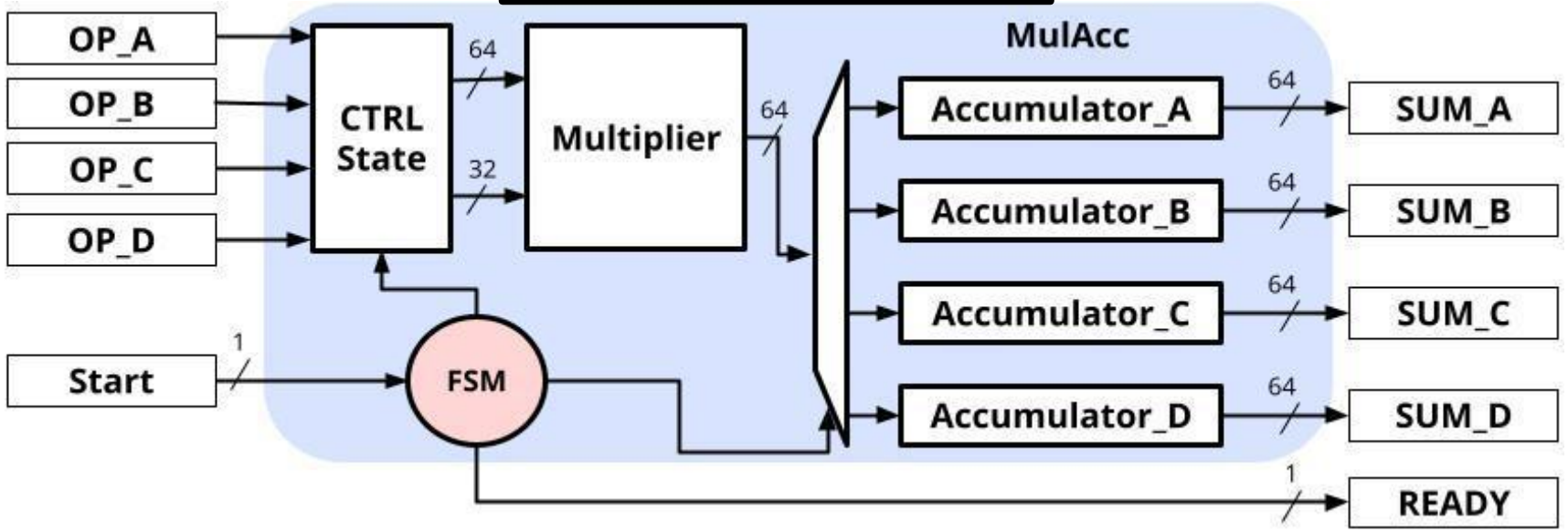


Link

5. Peripherals (8/14) Poly1305



MulAcc sub-module

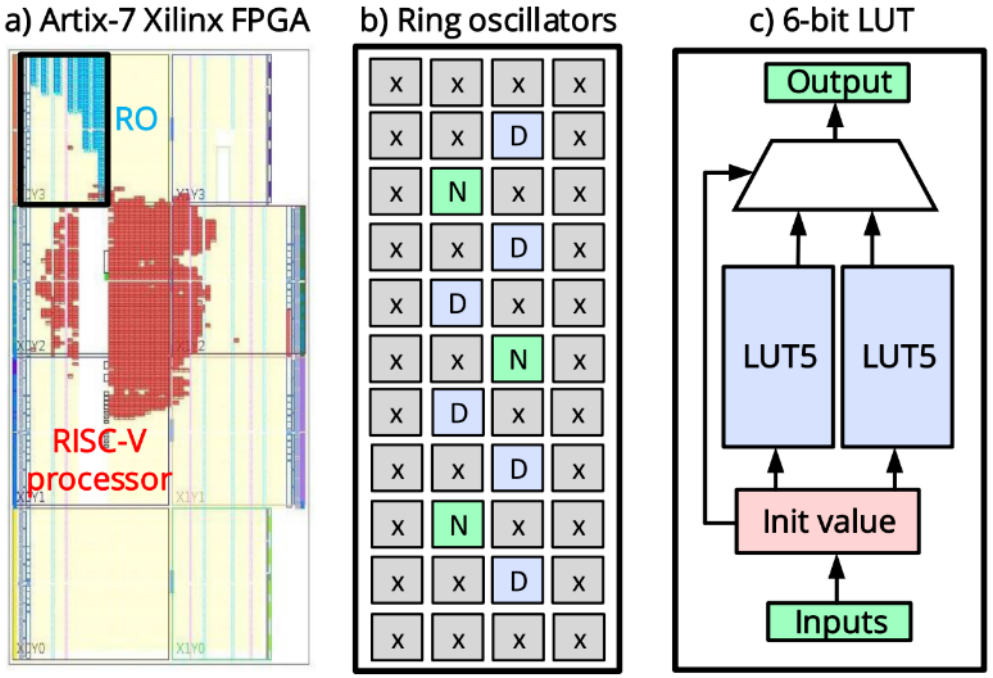


Poly1305

- A ***M***essage ***A***uthentication ***C***ode (***MAC***) that was standardized recently (*link*).
- Can work alone or team-up with *ChaCha20* to perform ***A***uthenticated ***E***ncryption with ***A***dditional ***D***ata (***AEAD***).

5. Peripherals (9/14) TRNG

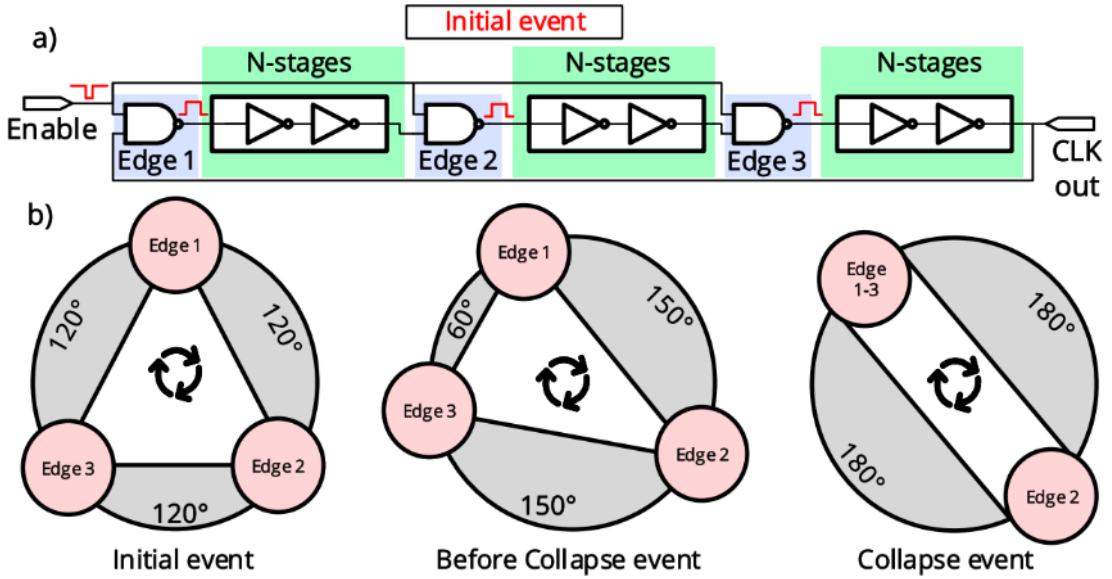
Our ***T***rue ***R***andom ***N***umber ***G***enerator (***TRNG***) is based on the frequency collapse phenomenon of ***R***ing ***O***scillators (***RO***s).



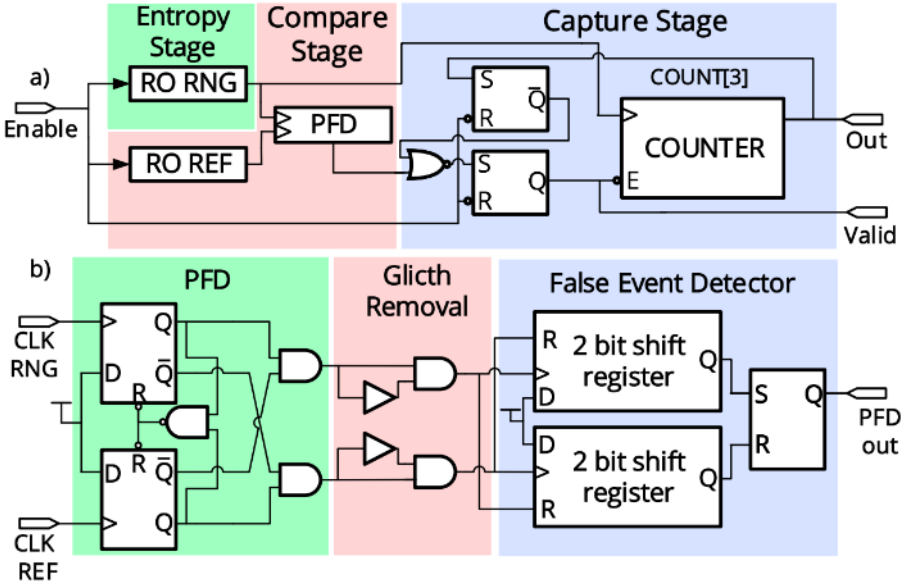
The idea can be implemented in FPGA.

[Link]

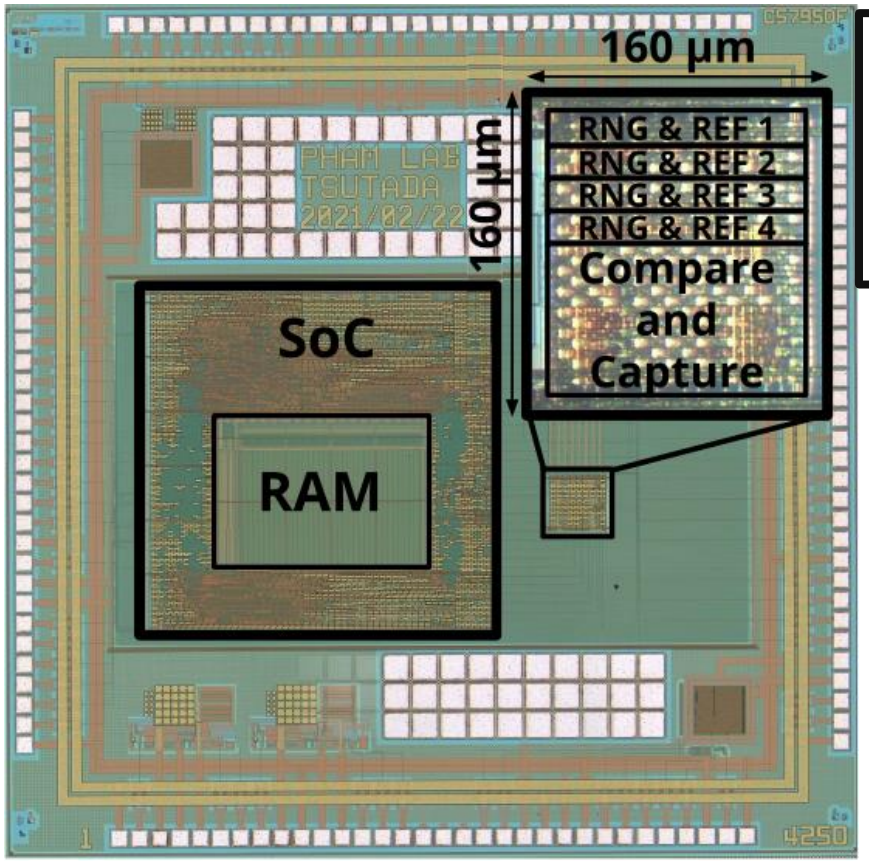
Three edge multimodal **ROs** → frequency collapse from three edges to *stable* two edges.



The proposed **TRNG** system based on multimodal **ROs**.

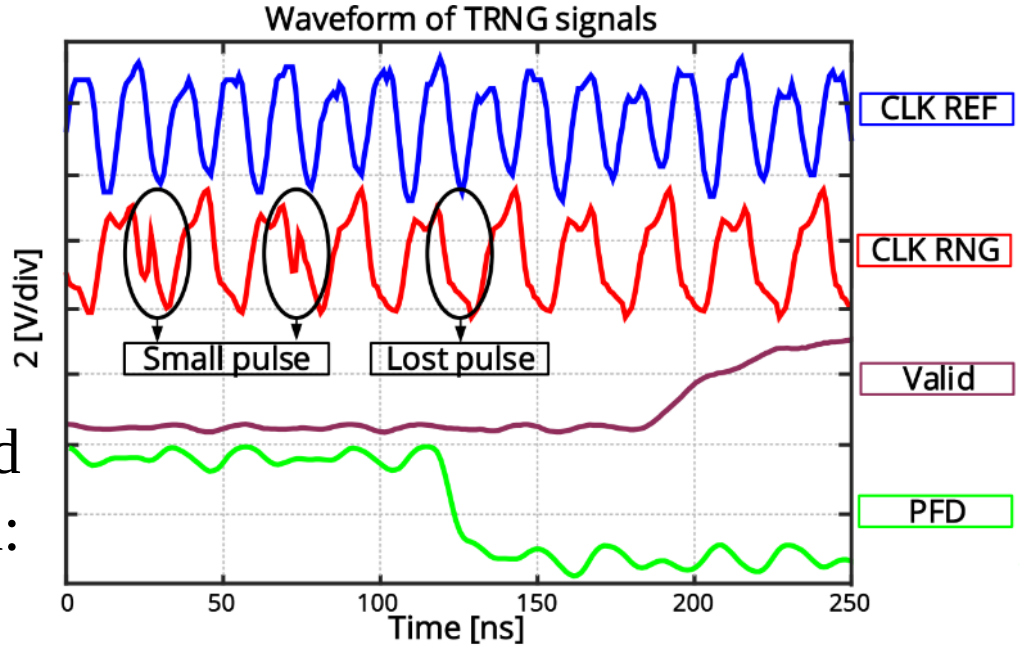


5. Peripherals (10/14) TRNG

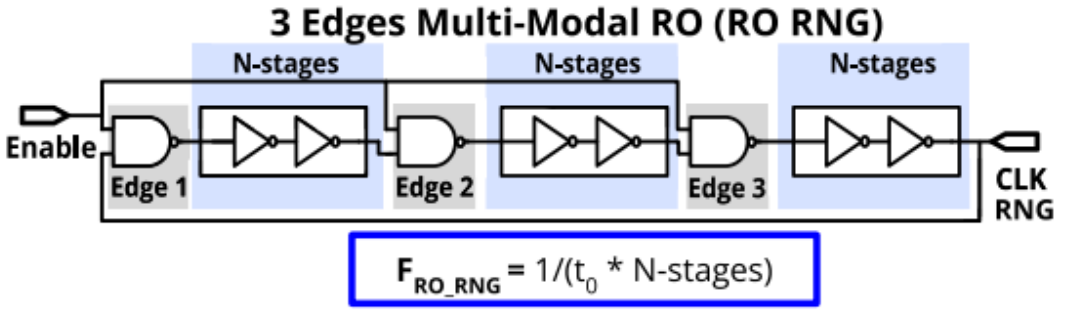


2.5×2.5-mm2
ROHM180nm
on *2021/02*

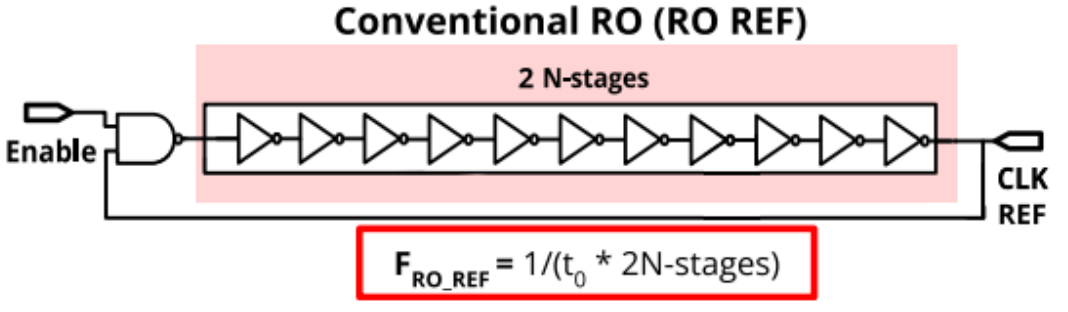
The wanted waveform:



Proposed:



Conventional:



The proposed design passed (*link*):

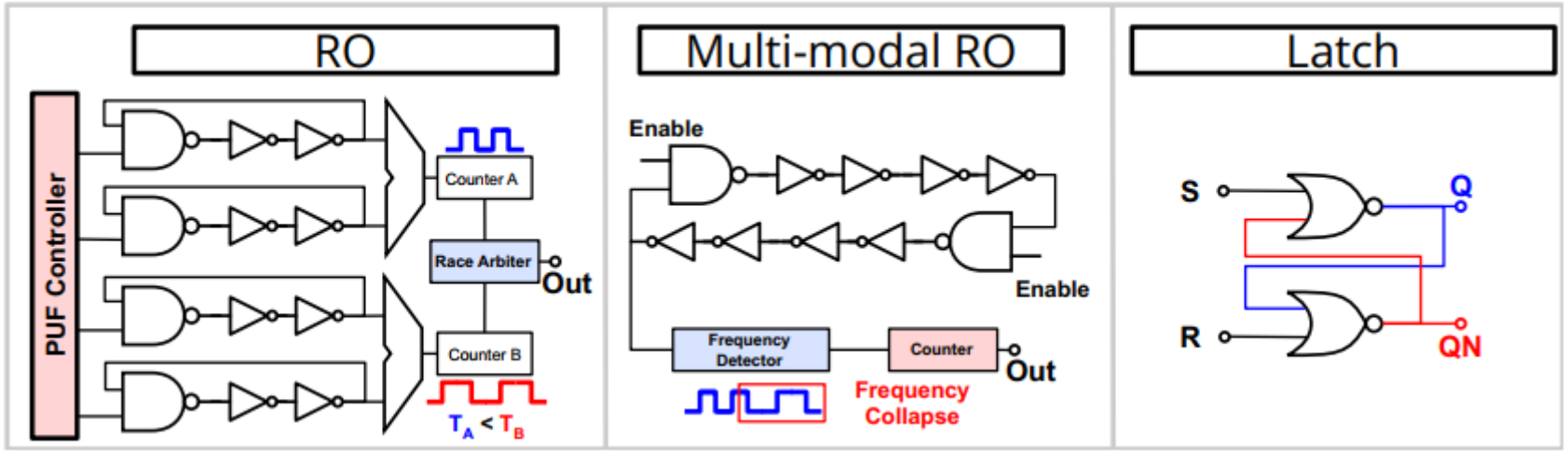
- NIST SP800-90B & SP800-22
- AIS31 & AIS20
- PVT healthy test

5. Peripherals (11/14) PUF in VLSI

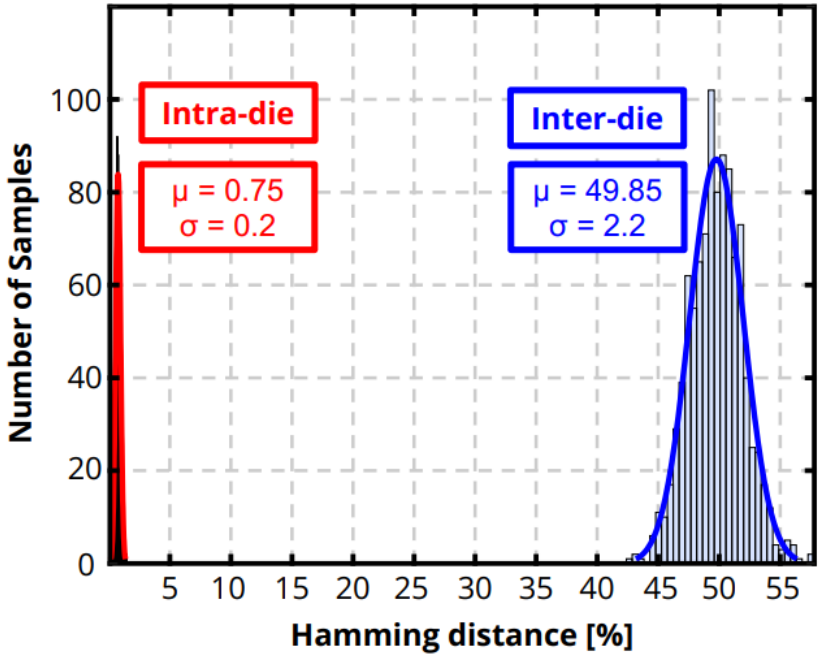
***P**hysical **U**nclonable **F**unction (**PUF**) is a physical "*object*" that serves as a unique identifier for *each* device, although the implementation is the same for *all* devices.

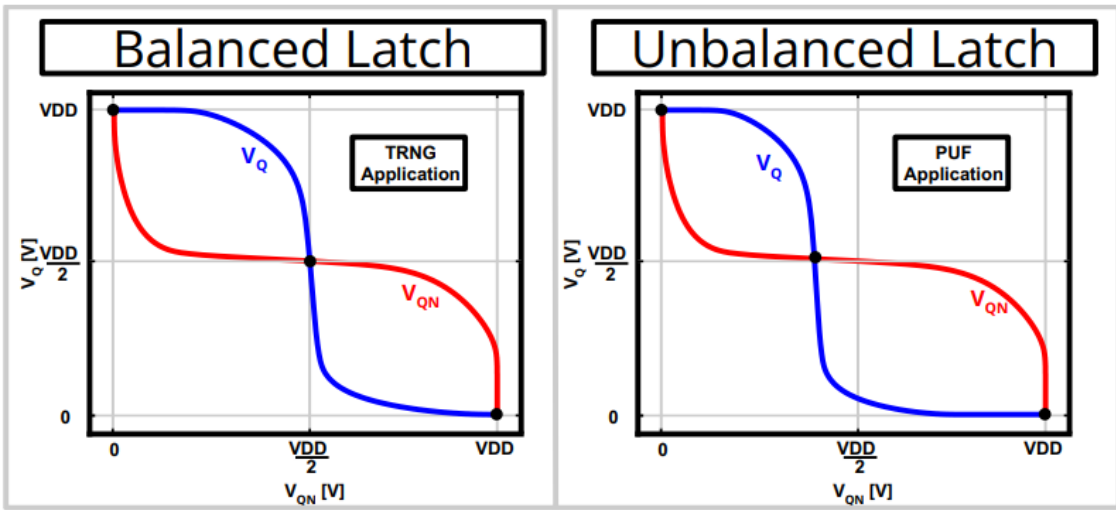
Example: fingerprints. Everybody has a fingerprint, but no two fingerprints are alike.

PUF architecture based on *RO*, *multimodal RO*, and *latches*:



Passed the intra/inter-die tests:



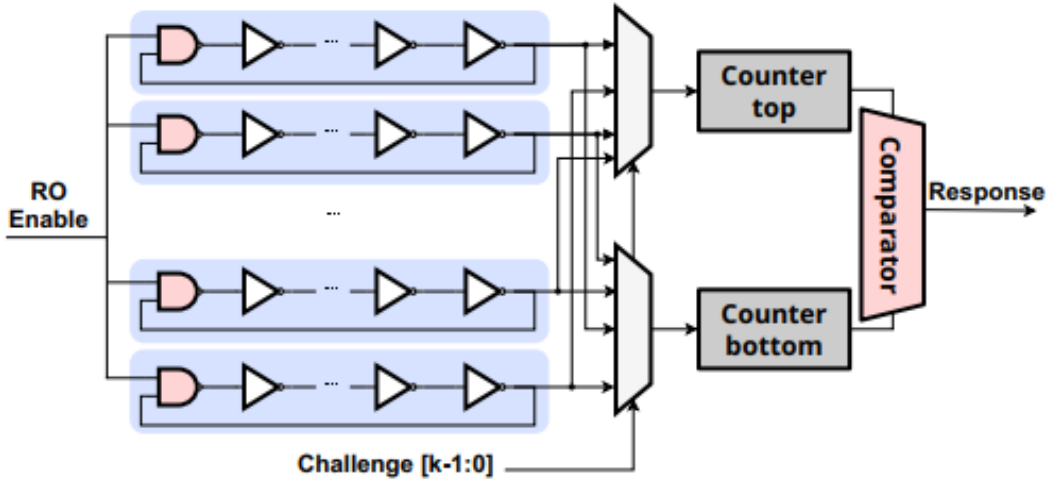


The main mechanism is based on the metastability of latches.

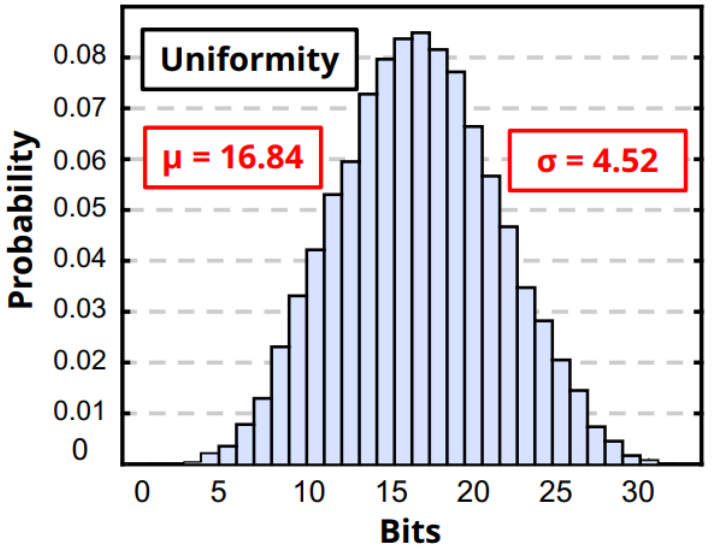
Link

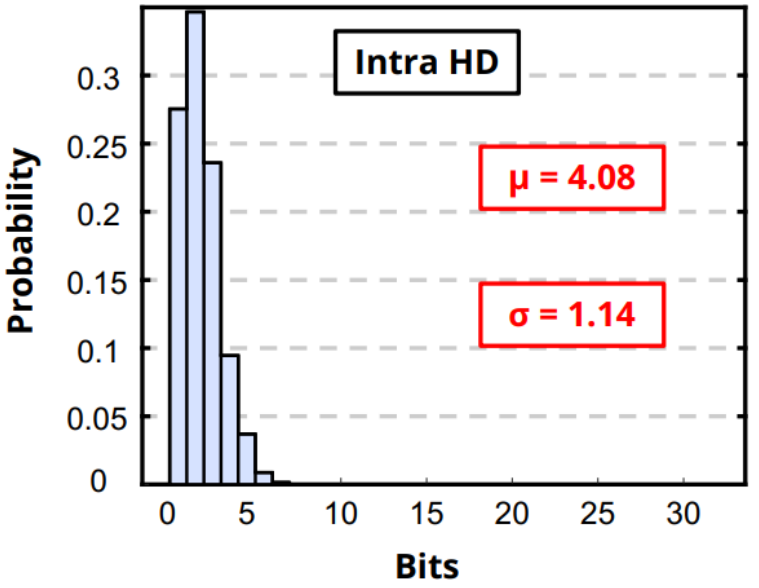
5. Peripherals (12/14) PUF in FPGA

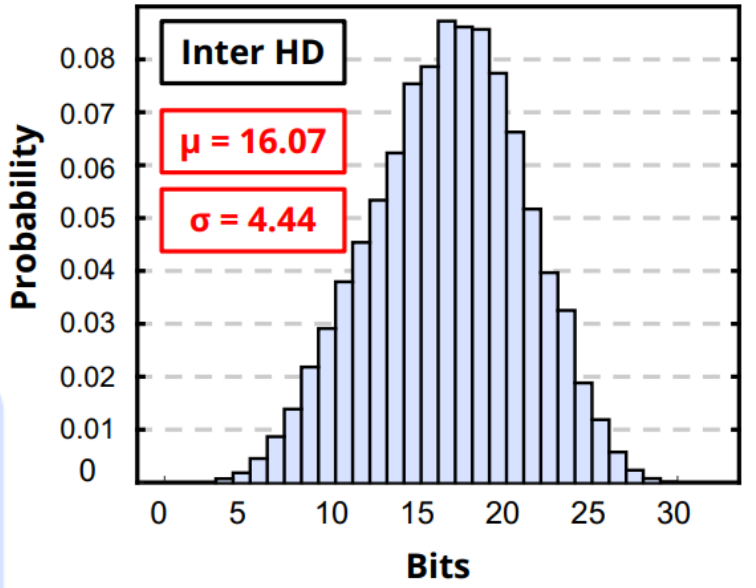
Conventional RO-based **PUF**:



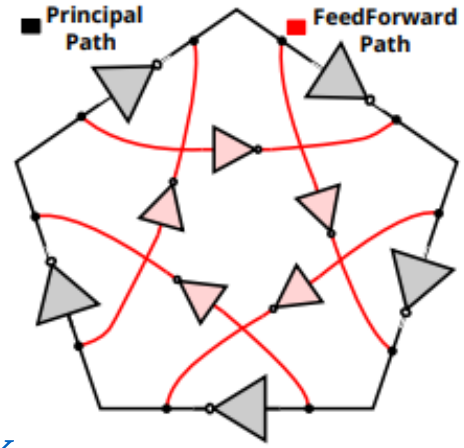
Passed the intra/inter-die tests:





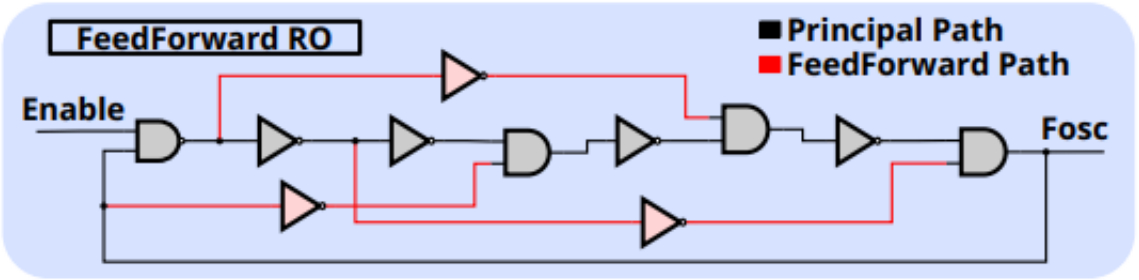


5-stage ***F***eedforward ***RO*** (***FRO***):



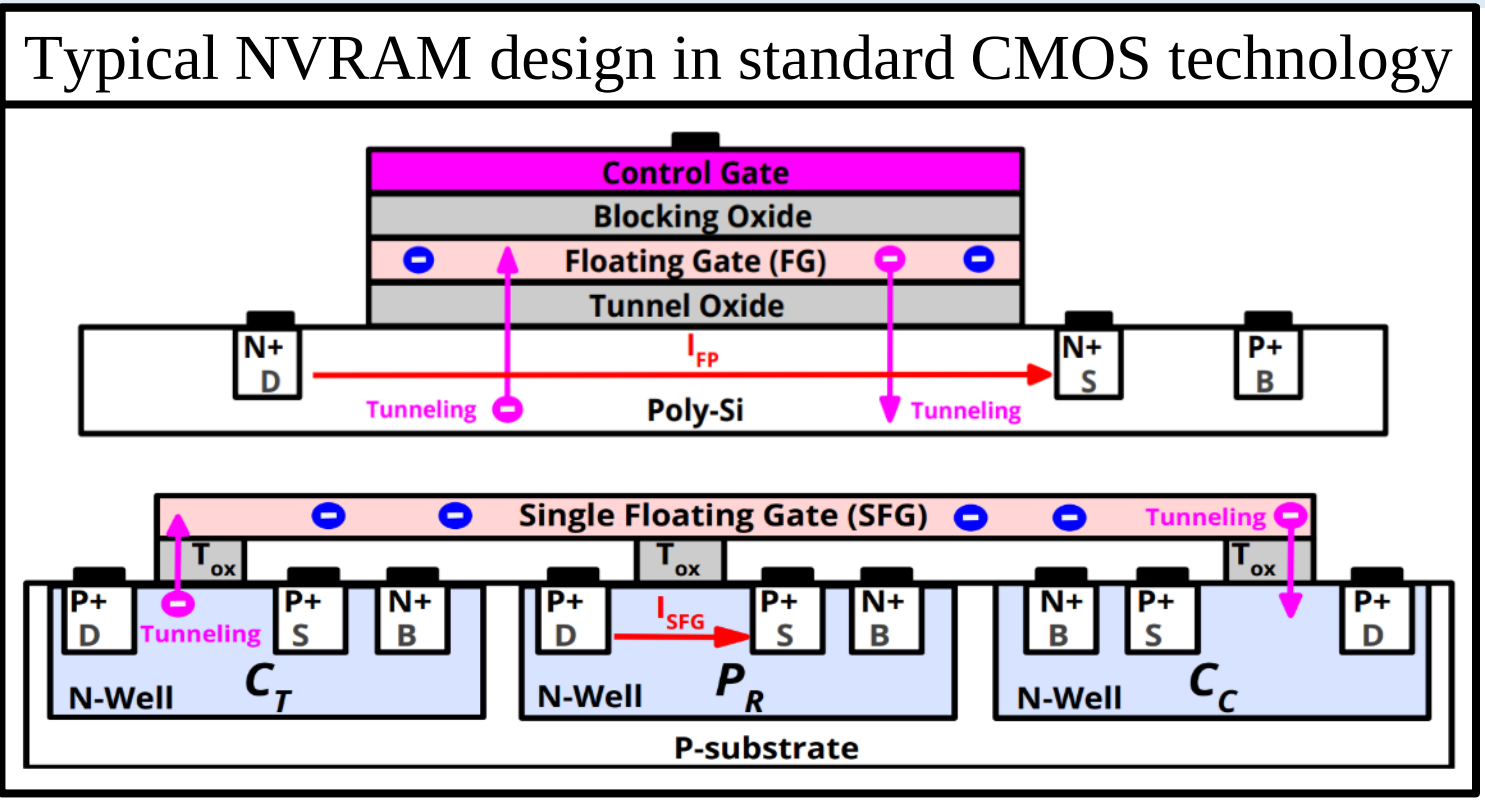
[*Link*]

Proposed FRO-based **PUF** in FPGA:

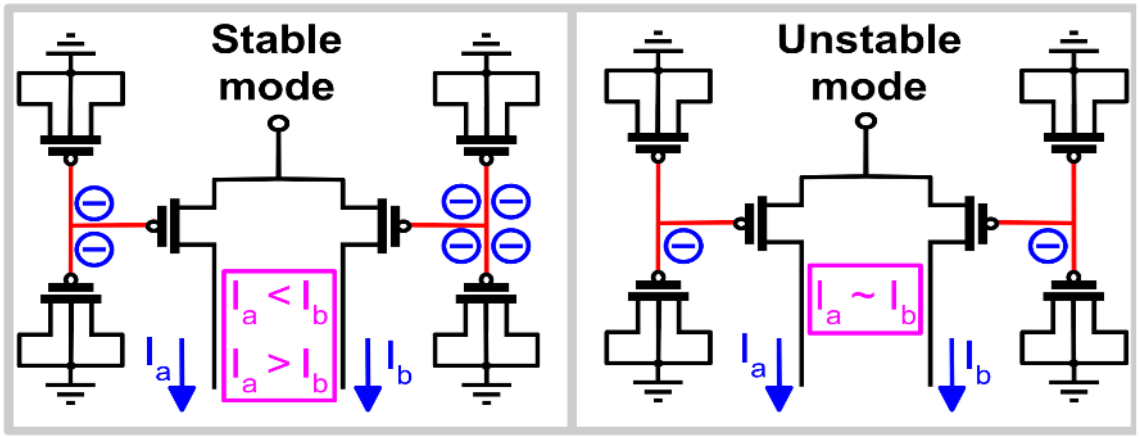


5. Peripherals (13/14) NVRAM

Typical NVRAM design in standard CMOS technology

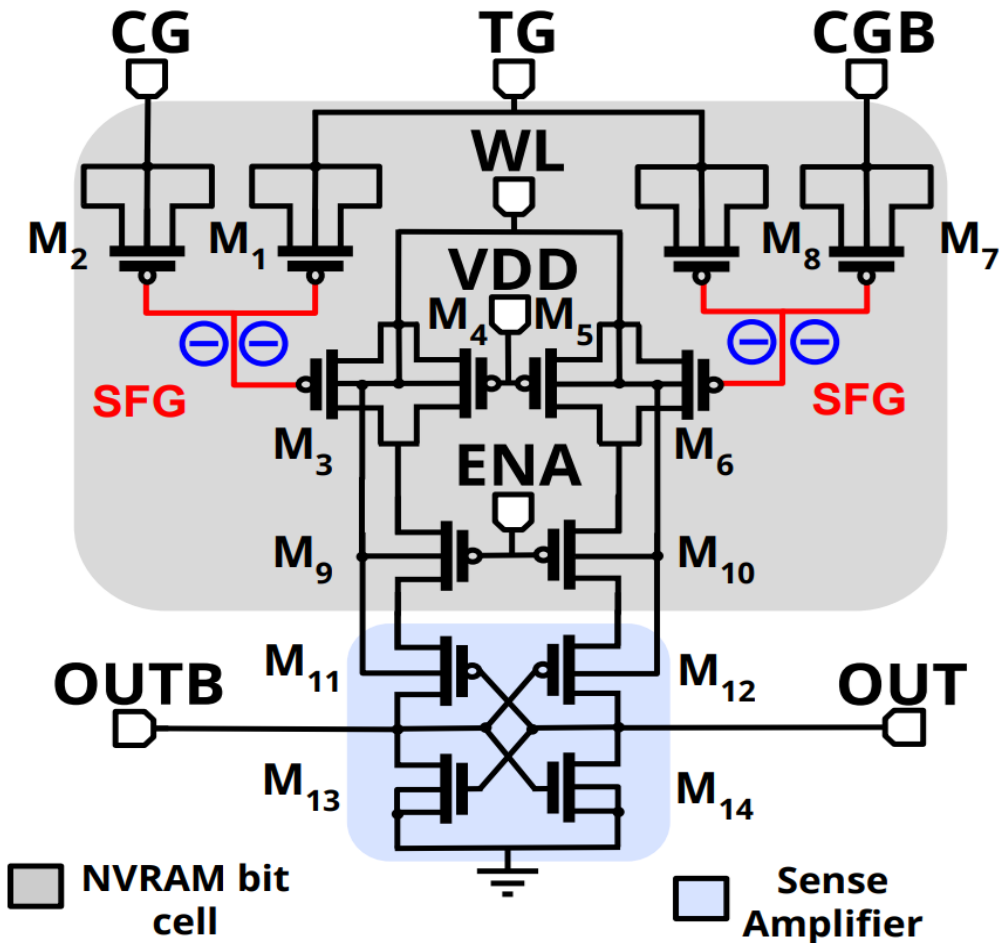


Simplified operating modes:



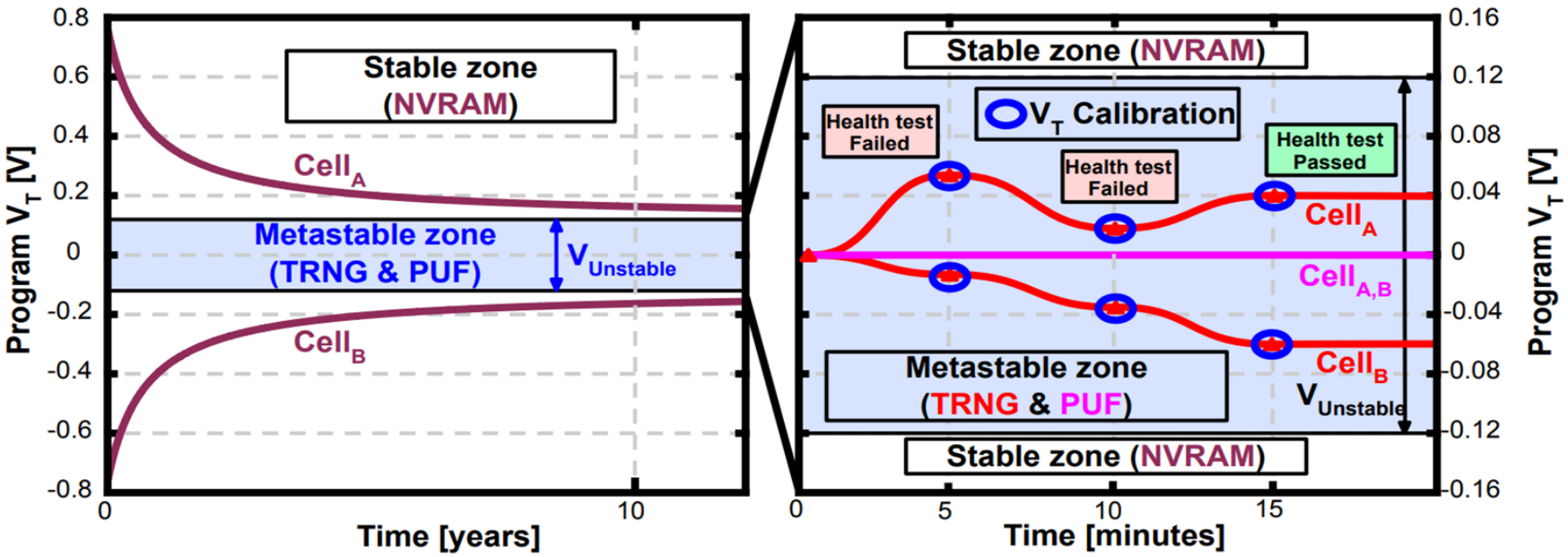
[Link]

Proposed NVRAM design without using special layer(s)

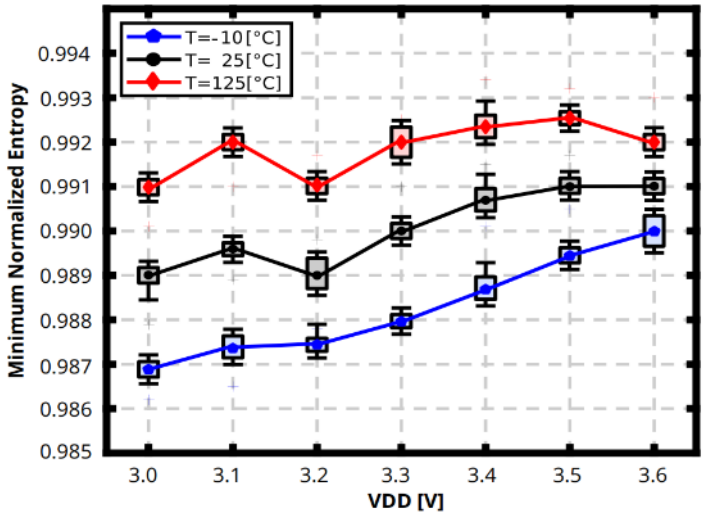


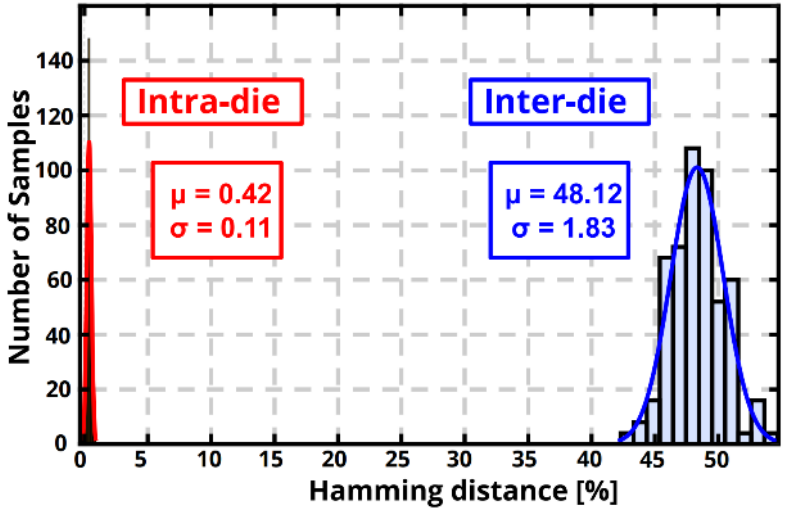
5. Peripherals (14/14) NVRAM

The physical phenomenon of the proposed NVRAM cell

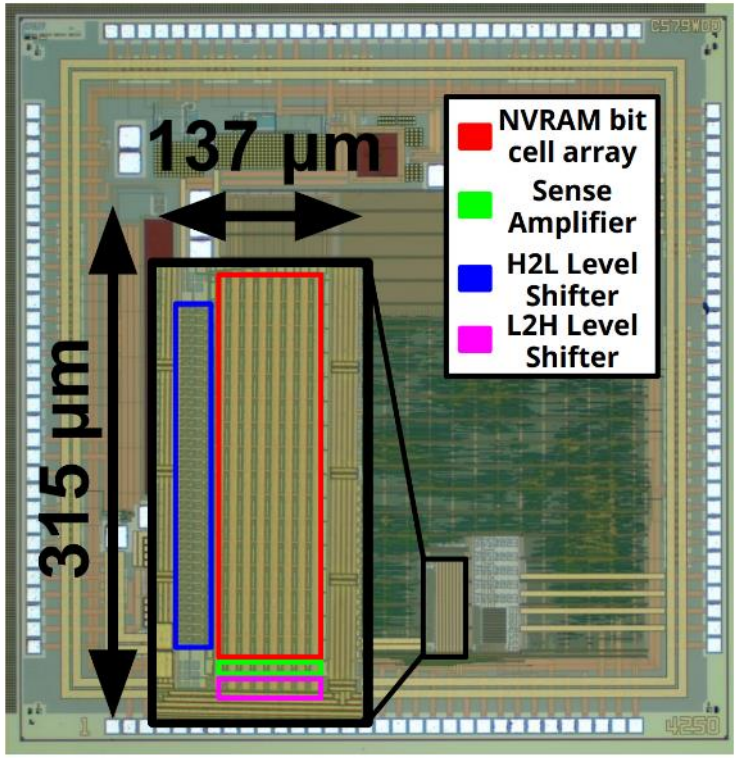


PVT tests:





The chip passed intra/inter-die tests with PVT variations.



Die tests at typical condition of 3.3V@25°C

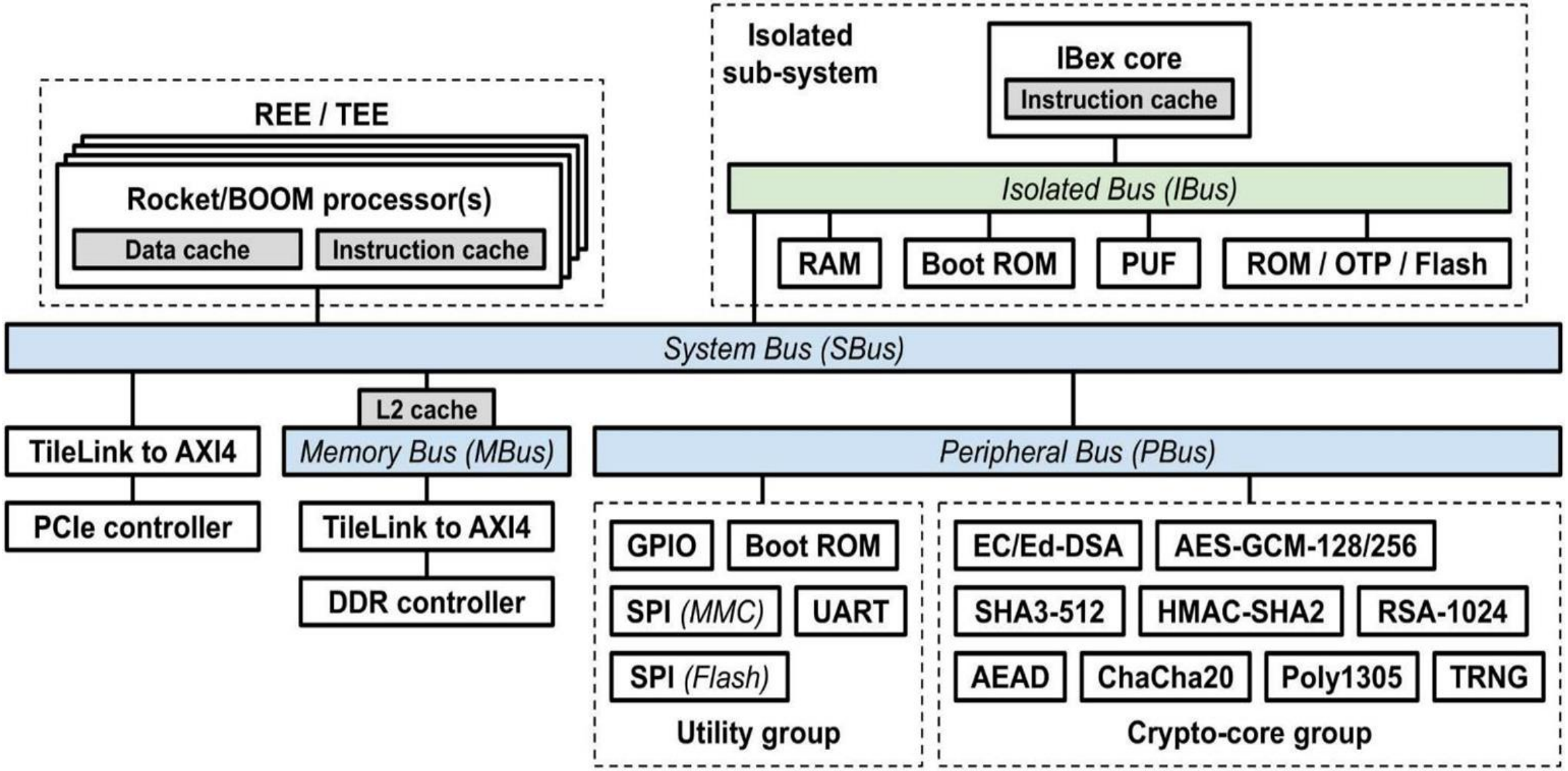
[Link]

Outline

1. Introduction
2. TPM and TEE
3. Why RISC-V?
4. Proposed System
5. Peripherals
6. Result
7. Conclusion

6. Result (1/14) FPGA implementation

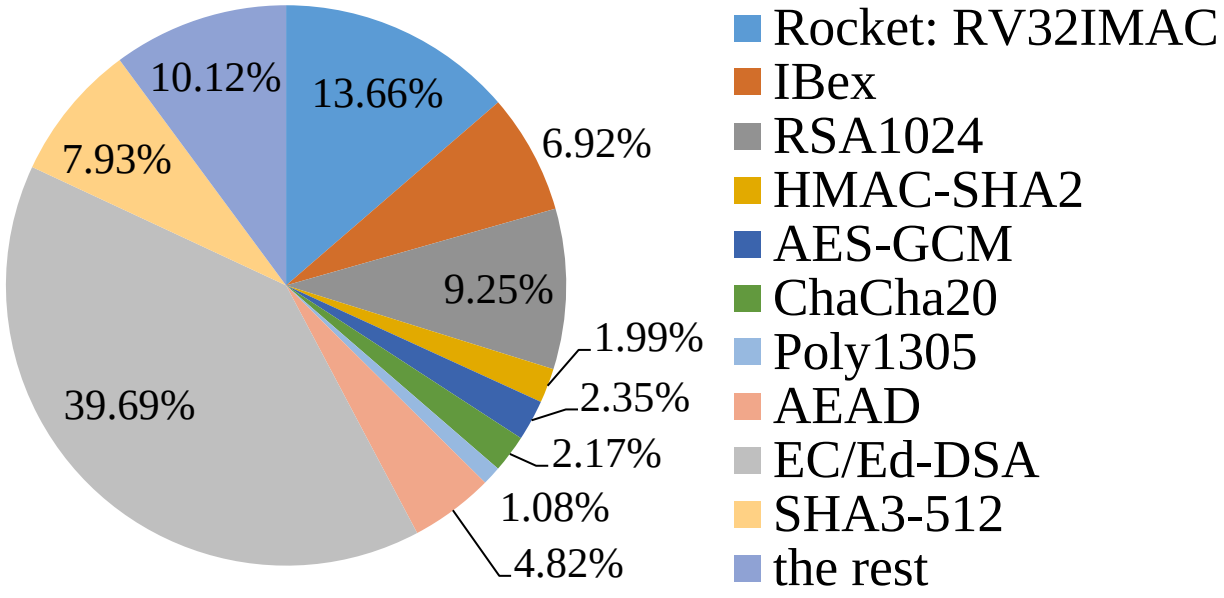
The proposed design was tested with the VC707 FPGA board.



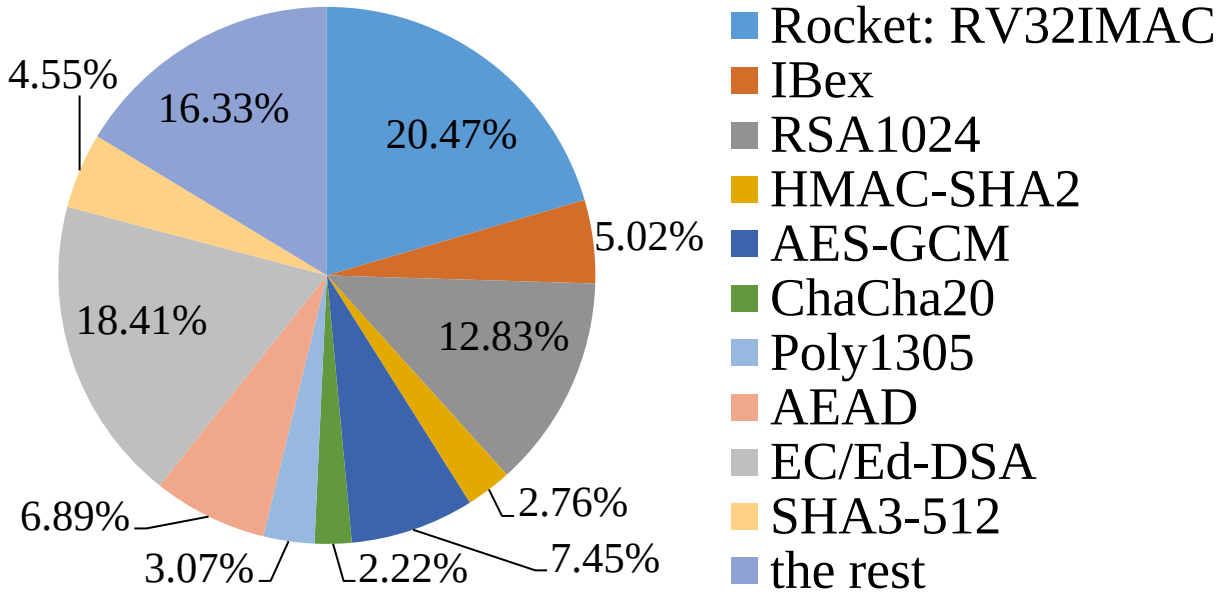
6. Result (2/14) FPGA implementation

SoC resources utilization pie chart: implementation on VC707 FPGA

LUT Ultilization



Register Utilization



***Note**: "*the rest*" means all the buses, TRNG,
and utility-group peripherals such as GPIO, SPI, boot ROM, etc.

6. Result (3/14) Self-test software

Initial test and drivers for using crypto-cores were developed

```
TEE-HW FSBL:       2022-04- 8-13:55:24-aa000a4
Using ZSBL DTB
Got TL_CLK: 50000000
Got NUM_CORES: 1
Got TIMEBASE: 1000000


Welcome to TEE-HW Bootloader

Press 'z' to run ALL      hardware tests
Press '1' to run SHA-3    hardware test
Press '2' to run ED25519  hardware test
Press '3' to run AES      hardware test
Press '4' to run RNG      hardware test
Press '6' to run CHACHA   hardware test
Press '7' to run POLY     hardware test
Press '8' to run CP_AEAD  hardware test
Press '9' to run AES GCM  hardware test
Press 'a' to run HMAC SHA hardware test
Press 'b' to run RSA      hardware test
Press 'd' to run DHRYSTONE test
Press 'e' to run AES GCM 1MB hardware test
Press 'f' to run HMACSHA512 1MB hardware test
Press 'h' to run ED\EC hardware test
Press ENTER to boot Linux   1


/ 81ff8200 <- 0001ff82kB / 00020000kB
Booting payload
```

Tests at FSBL *(M-mode)*
before boot into Linux

Drivers and tests *(U-mode)* after boot into Linux

```
# ls
aes_gcm_driver.ko      edec_test_384          rsa_driver.ko
aes_gcm_test           edec_test_521          rsa_test
chacha_driver.ko       edec_test_full         testdriver.ko
chacha_poly_driver.ko  edec_test_sign         testdriver_write.ko
chacha_poly_test       hmac_sha_driver.ko     tests.ke
chacha_test            hmac_sha_test          tls_client
edec_driver.ko         keystone-driver.ko     tls_client.o
edec_test              poly_driver.ko         tls_server
edec_test_256          poly_test              tls_server.o
#
```

6. Result (4/14) Self-test software

Some examples of self-test software

HMAC-SHA2

```
f1bd938f54b2f392f7b4e811544c65e9
65c0885d54ed4b2c50732032abd953a4
df89de58ee61514ed075f8e590ac3d2e
38d7489e28742abcae67242212a4b33d

Time: 0s 0ms 92us

Hardware 384:

9d72af3c8cf178c12e6069bdfd645b09
e2a2245e3850d3c595def9f13670511b
7c3472789dfae890c52c3d4d88891148

Time: 0s 0ms 86us

Hardware 256:

61cc4af0e0e6c428fdc1b4e890777c7c
e2fa0dacf7a21ce31028721aebdcbc2d

Time: 0s 0ms 83us
```

6.16× to 7.62 × faster

AES-GCM

```
Begin AES GSM hardware test:


Software:

e2006eb42f5277022d9b19925bc419d7
a592666c925fe2ef718eb4e308efeaa7
c5273b394118860a5be2a97f56ab7836

5ca597cdbb3edb8d1a1151ea0af7b436



Time: 0s 0ms 397us
```

```
Hardware:

e2006eb42f5277022d9b19925bc419d7
a592666c925fe2ef718eb4e308efeaa7
c5273b394118860a5be2a97f56ab7836

5ca597cdbb3edb8d1a1151ea0af7b436


Time: 0s 0ms 53us
```

7.49× faster

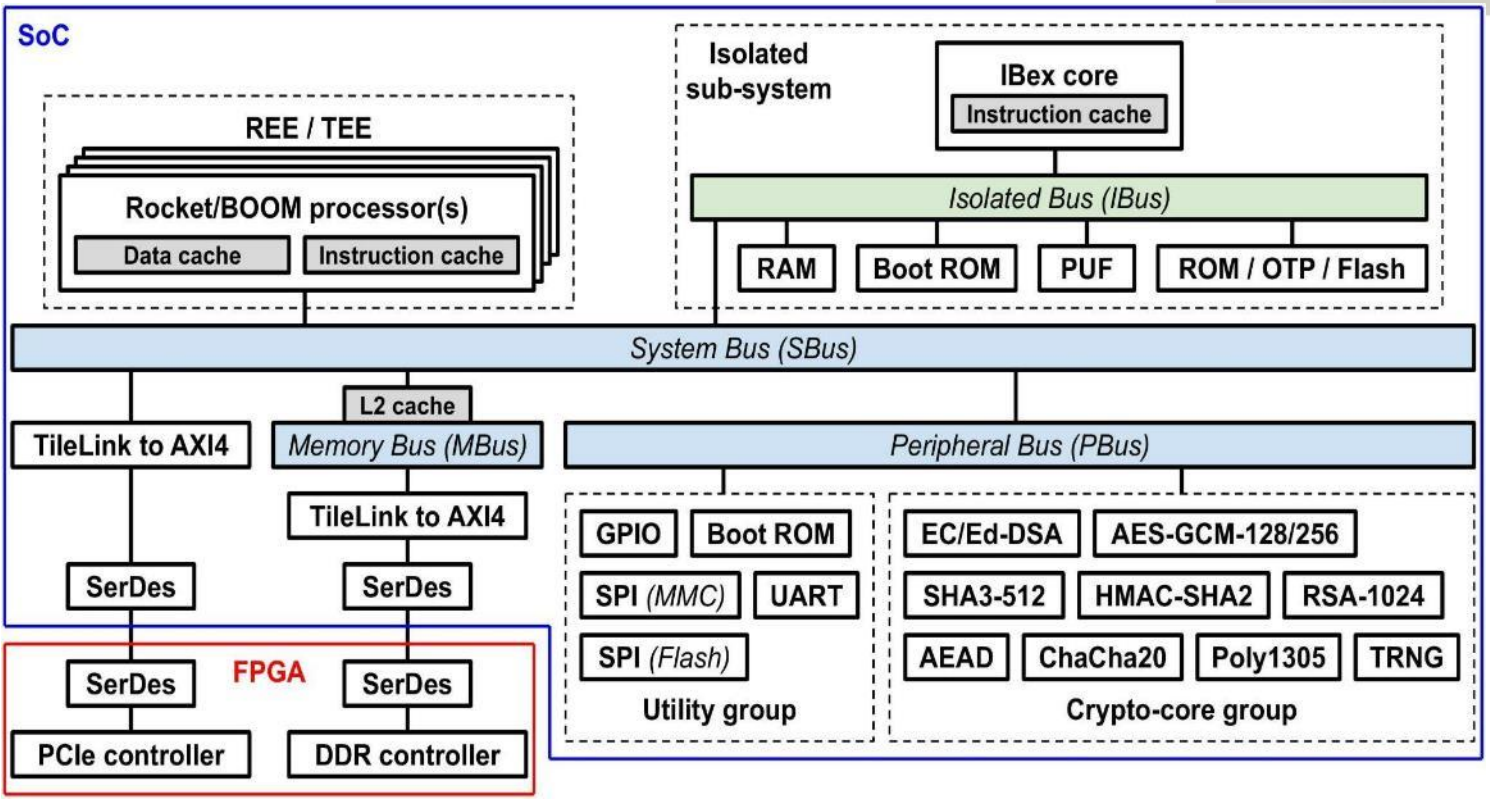
AEAD *(ChaCha+Poly)*

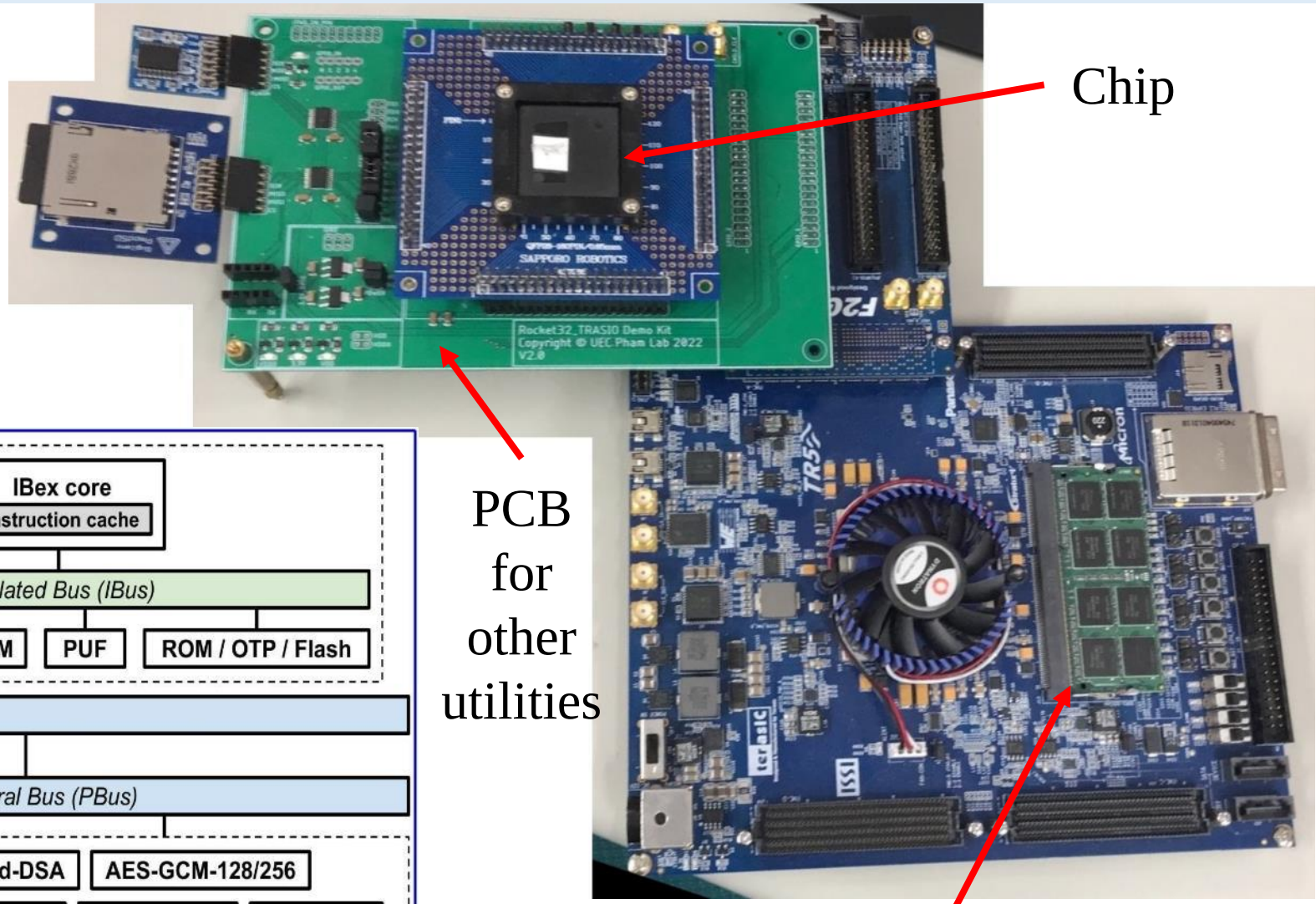
```
Hardware:


 Cipher:

d31a8d34648e60db7b86afbc53ef7ec2
a4aded51296e08fea9e2b5a736ee62d6
3dbea45e8ca9671282fafb69da92728b
1a71de0a9e060b2905d6a5b67ecd3b36
92ddbd7f2d778b8c9803aee328091b58
fab324e4fad675945585808b4831d7bc
3ff4def08e4b7a9de576d26586cec64b
6116000000000000000000000000000

 MAC:

1ae10b594f09e26a7e902ecbd0600691

 Time: 0s 0ms 43us
```

12.1× faster

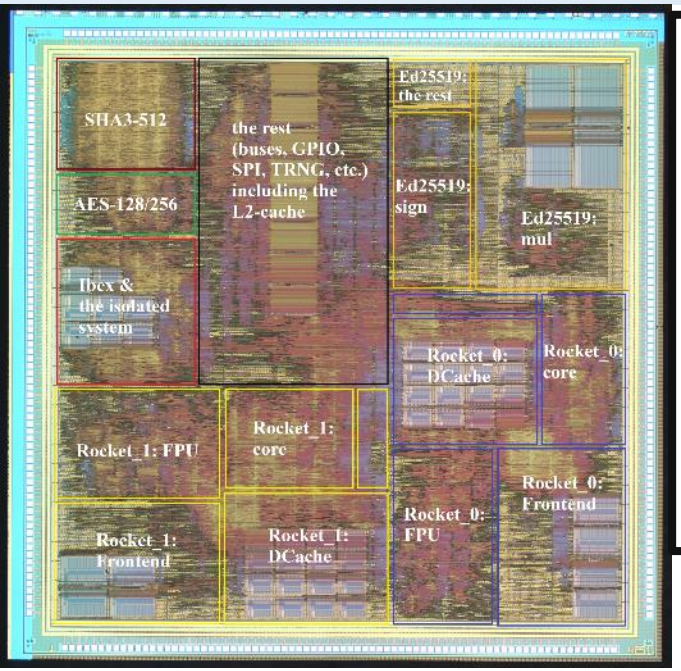
6. Result (5/14) VLSI implementation

To boot rich OS *(such as Linux)*, a big memory space is required.
→ Off-chip DDR is needed.
→ An FPGA will be used for DDR and PCIe hard IPs.



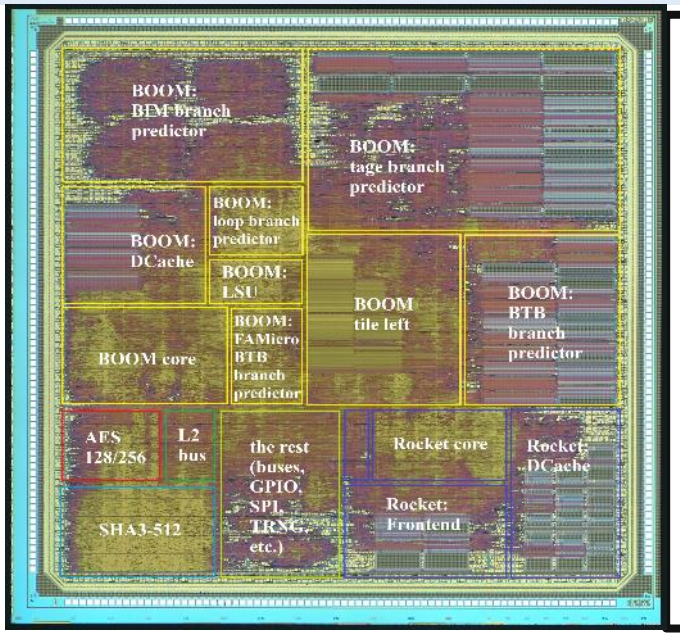


6. Result (6/14) VLSI implementation



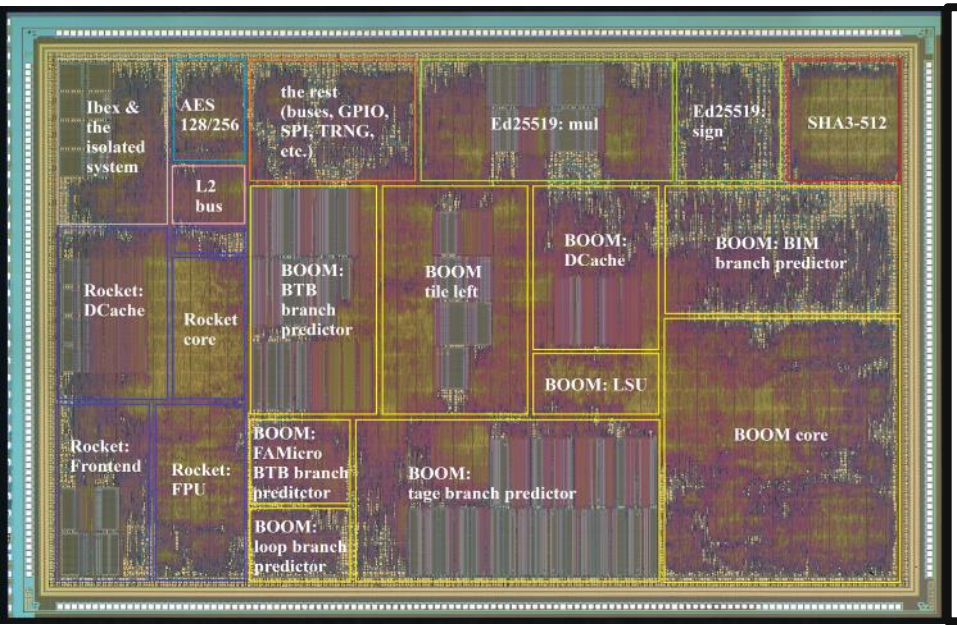
64-bit dual Rocket with crypto-cores and secure boot

5.0×5.0-mm2 ROHM180nm on *2021/02*



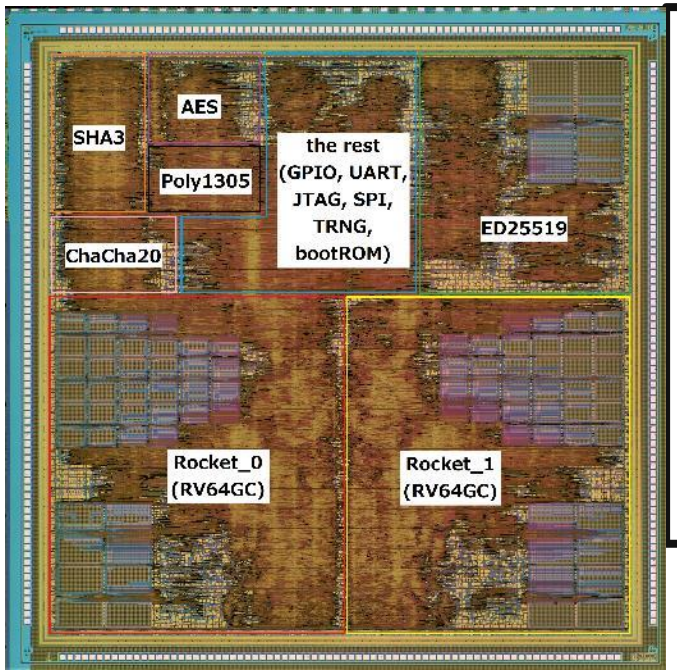
32-bit Rocket-Boom with crypto-cores and secure boot

5.0×5.0-mm2 ROHM-180nm on *2021/06*



64-bit Rocket-Boom with crypto-cores & secure boot

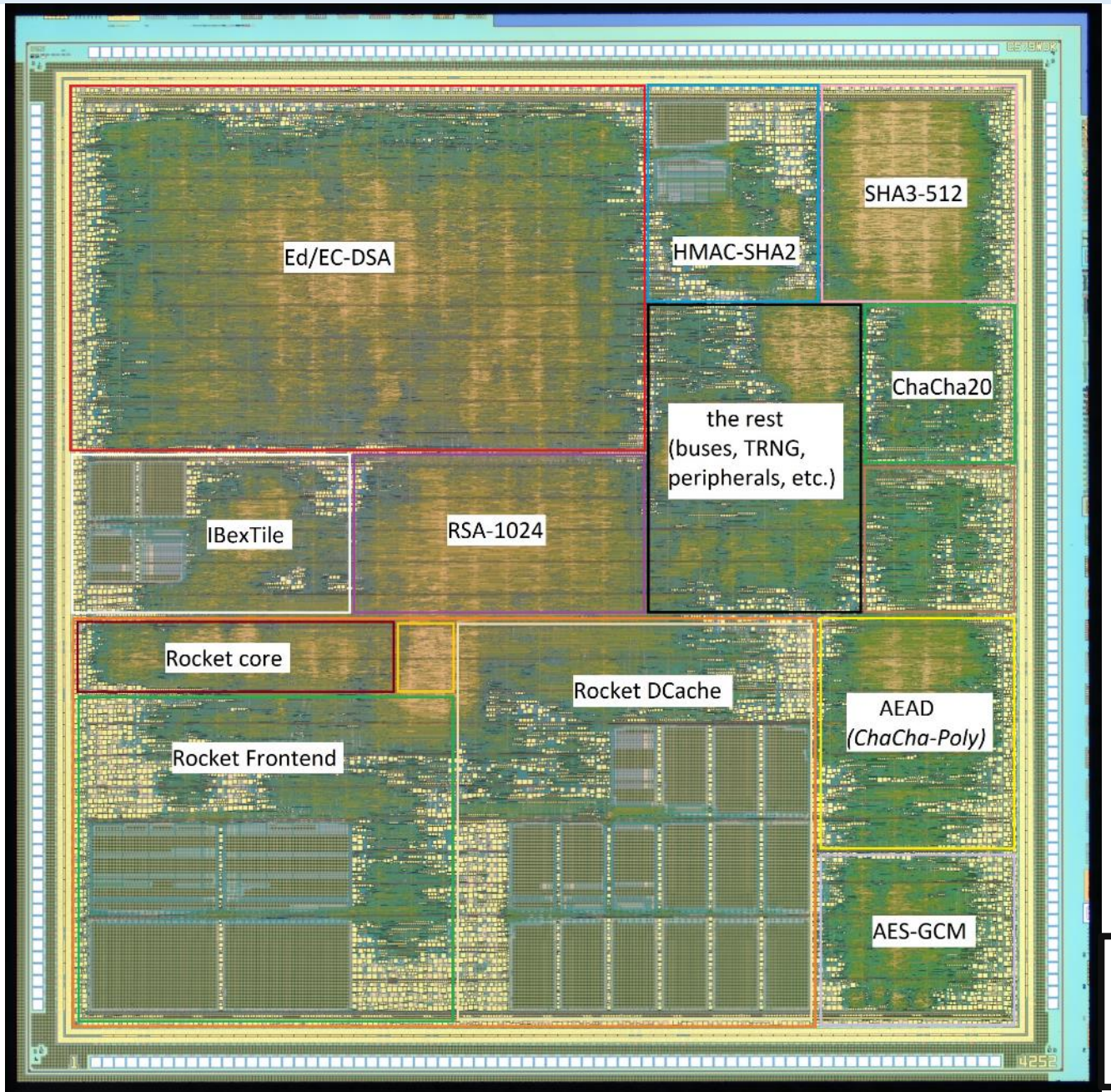
5.0×7.5-mm2 ROHM180nm on *2021/06*



64-bit dual Rocket with crypto-cores

5.0×5.0-mm2 ROHM-180nm on *2021/09*

6. Result (7/14) VLSI implementation

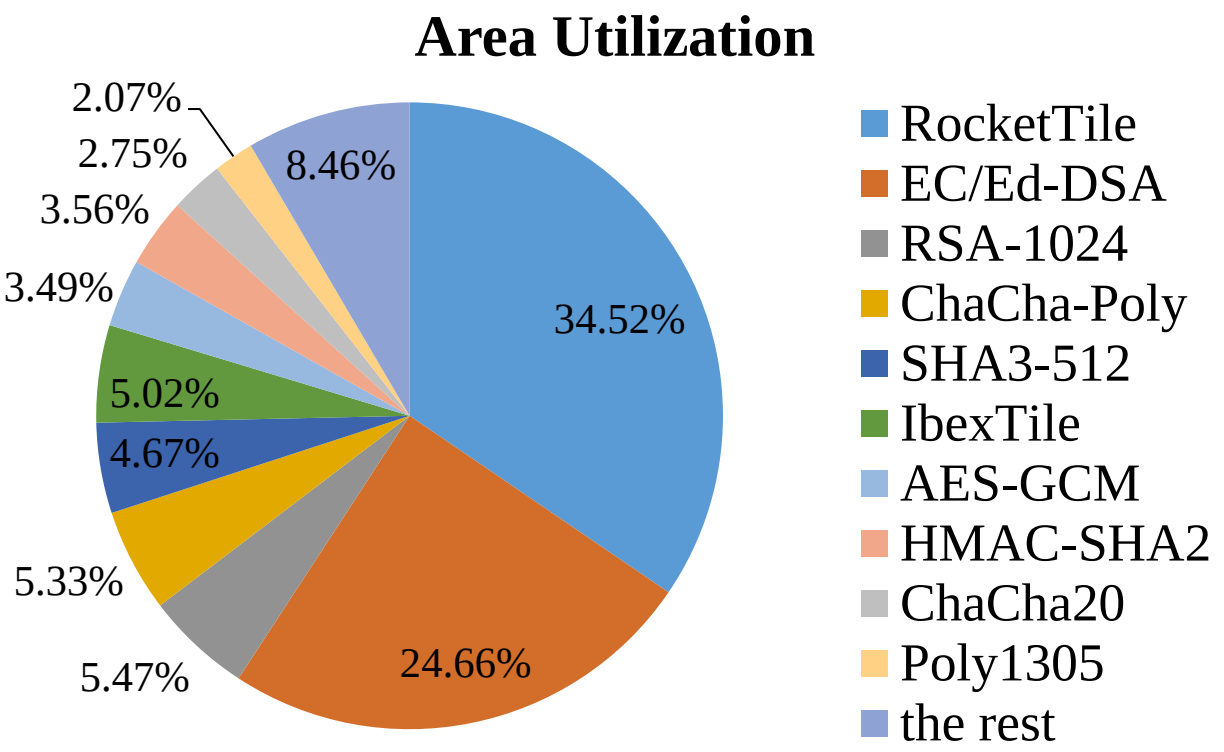


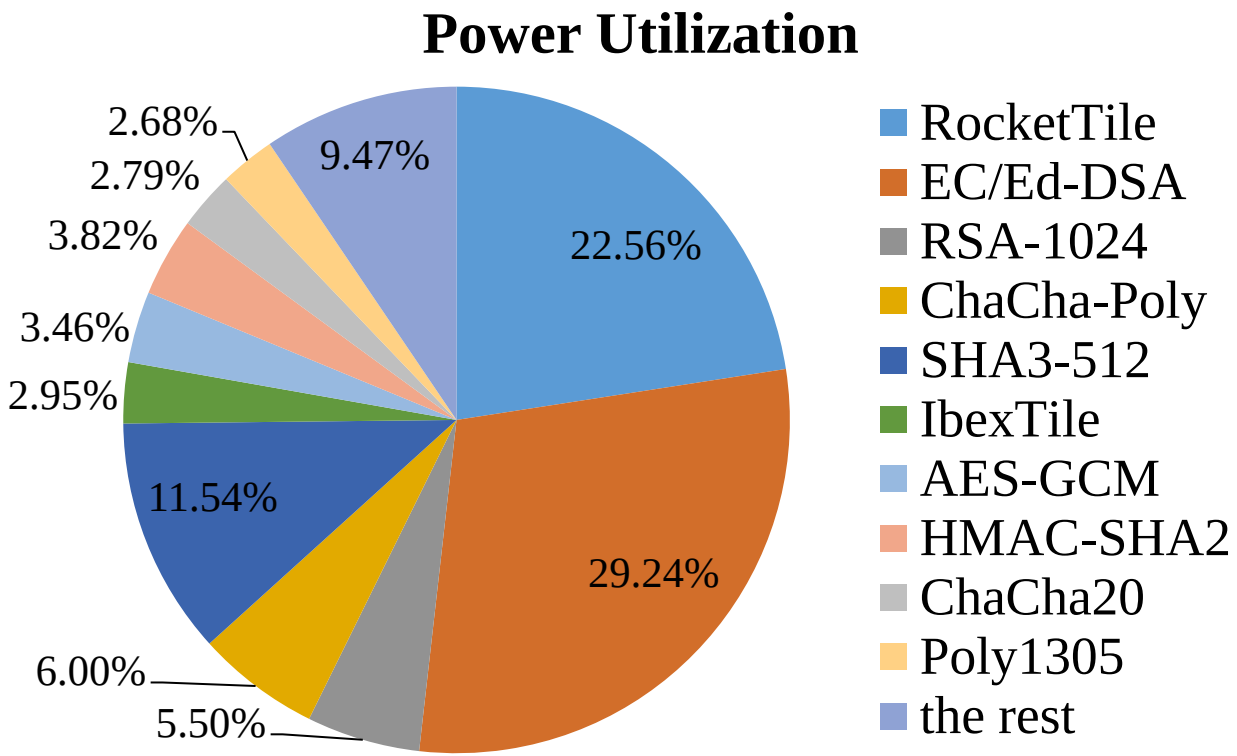
5.0×5.0-mm2 ROHM-180nm on *2022/02*	
Core	Rocket (×1)
ISA	RV32IMAC
Cache	$I =16KB and $D = 16KB
Crypto-cores: TRNG, RSA, AES-GCM, SHA3, HMAC-SHA2, ChaCha20, Poly1305, AEAD, and EC/Ed-DSA	
#Gate	1,535,403
#Cell	466,882
Area (μm^2)	20,799,437
Density	71.43%
Power (mW)	1,992
Fmax (MHz)	71
#MOSFET	7,982,582

32-bit Rocket with TLS-1.3 crypto-cores and secure boot

6. Result (8/14) VLSI implementation

SoC resources utilization pie chart: implementation on ROHM180nm





***Note**: "*the rest*" means all the buses, TRNG,
and utility-group peripherals such as GPIO, SPI, boot ROM, etc.

6. Result (9/14) 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 [11, 12] (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 [9] (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.*

6. Result (10/14) 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-co[2]		
ITUS [11, 12] (2019)	Baseline: Dual-Rock		
	+ CAU		
	+ KMU		
	+ CAU + KMU	+10,066 (40.88%)	+56,699 (76.35%)
HECTOR-V [9] (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%)

ITUS: secure boot by all hardware modules.
This work: crypto-cores just for accelerating the boot flow, not a hard requirement.

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

6. Result (11/14) Comparison

Even including crypto-cores, this work still smaller.

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

	esign	Registers Overhead (+%)	LUTs Overhead (+%)
This work (202[illegible])	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 [11,12] (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 [9] (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.*

6. Result (12/14) 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]		
	+ IBex[1] + crypto-cores[2]		
ITUS [11,12] (2019)	Baseline: Dual-Rocket		
	+ CAU		
	+ KMU		
	+ CAU + KMU		
HECTOR-V [9] (2021)	Baseline: Single-lowRISC		
	with RI5CY	N/A	+11,581 (20.89%)
	with Remus		+13,303 (23.99%)
	with Frankenstein		

[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.

6. Result (13/14) 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 (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 [9] (2021)	Baseline: Single-lowRISC		55,443
	with RI5CY	N/A	+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.*

6. Result (14/14) Comparison

TABLE 5.13: 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 [8]	HECTOR-V [9]	WorldGuard [10]	ITUS [11, 12]	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.

Outline

1. Introduction
2. TPM and TEE
3. Why RISC-V?
4. Proposed System
5. Peripherals
6. Result
7. Conclusion

7. Conclusion (1/1) Summary

Key Achievements

1. **TEE-HW with cryptographic accelerations:** using the framework, 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 bootloader program is flexible and can be updated.

3. **Silicon-proof TEE-HW chips:** ROHM-180nm chips were made for the TEE-HW with isolated RoT; and the measurements and tests were done.

4. **FPGA and VLSI implementations:** the proposed system can work on both FPGA and VLSI. All the cryptographic primitives, such as TRNG and PUF, have their equivalent in FPGA.

THANK YOU

2023/10/27