Zero Trust Secure RISC-V System

Secure Systems
Research Center

tii.ae

Innovation for a better world

Contents

Securing our digital future with Zero Trust

Introduction: The need for end-to-end system security

The world is becoming increasingly more connected at a staggering rate with the rise of the Internet of Things (IoT), autonomous cars, drones, robots, and industrial control systems that are hooked up to the wireless network. The recent pandemic has also accelerated the trends in remote/ hybrid work and study. At the same pace, more malicious actors are discovering more innovative ways to penetrate devices and computer systems through various techniques like network attacks, software supply chain attacks, ransomware, side-channel attacks, etc.

In the recent past, the bulk of security breaches has involved compromising IT systems for ransomware attacks that steal or lock information and demand money from victims. The relatively new kind of attacks exploit hardware vulnerabilities in modern microprocessors, the popular ones being Spectre/ Meltdown[1] which caused millions of computers to be vulnerable to rogue applications to read unauthorized application/ user data.

Of particular note are new software supply chain attacks[2] that target software developers and suppliers by infecting legitimate software to distribute malware. The recent Solar Winds[3] breach used a routine software update to slip malicious code into known software and then used it as a vehicle for a massive cyberattack. There have also been some reports of hardware supply chain attacks[4] where someone or a company in the supply chain can install a malicious microchip on a circuit board used to build a computer and other network components. Using this chip, the attacker can eavesdrop on data or obtain remote access to the corporate infrastructure.

The early generation of IoT devices was rushed to market with only preliminary considerations of how they might be protected against hackers or securely updated against new threats. Many of these early devices are not updateable after the fact. Consequently, they are a popular target for hackers eager to create large-scale botnets for launching distributed denial of service attacks such as the Mirai botnet[5] that compromised over 600,000 routers, digital video recorders, and cameras. This has given rise to a secondary industry of IoT security gateways designed to detect and block malicious activity outside poorly secured appliances like lighting controllers, crockpots, TV set-top boxes, and cameras.

Security researchers have identified how similar coordinated assaults on smart heaters, air conditioners, and other power-intensive equipment could bring down the power grid.[6] Indeed hackers remotely compromised the IT systems for managing the power grid in Ukraine in 2015, bringing the entire grid offline in some regions for several days.[7] One medical device company recalled 500,000 pacemakers after discovering a significant vulnerability.[8]

Hence there is an utmost necessity to secure all our cyber assets (computers, phones, IoT, etc) and cyber-physical and autonomous assets (those that interact with the physical world and take actions without much human intervention, ex: self-driving cars, drones, industrial robots, etc) in an end-to-end manner covering all aspects of the system stack from applications, operating system, hypervisor, firmware, SoC, peripherals, down to the motherboard.

1 "Meltdown and Spectre". https://meltdownattack.com/.
2 "Software Supply Chain Attack". https://learn.microsoft.com/en-us/microsoft-365/security/intelligence/supply-chain-malware.
3 "Solar Winds Breach". https://www.techtarget.com/whatis/feature/SolarWinds-hack-explained-Everything-you-need-to-know.
4 "Hardware Supply Chain Attack". https://theintercept.com/2019/01/24/computer-supply-chain-attacks/.
5 "The Mirai Botnet Explained: How IoT Devices Almost Brought down the Internet | CSO Online." Accessed March 16, 2022. https://www.csoonline.com/article/3258748/the-mirai-botnet-explained-how-teen-scammers-and-cctv-cameras-almost-brought-down-the-internet.html.
6 Greenberg, Andy. "Hacked Water Heaters Could Trigger Mass Blackouts Someday." Wired. Accessed June 13, 2022. https://www.wired.com/story/water-heaters-power-grid-hack-blackout/.
7 "Ukraine Power Grid Hack." https://en.wikipedia.org/wiki/Ukraine_power_grid_hack
8 Hern, "Hacking Risk Leads to Recall of 500,000 Pacemakers Due to Patient Death Fears."

Security Vulnerabilities

Below is a list of common cyber-physical attacks that target autonomous cyber-physical systems exploiting their inherent flaws or vulnerabilities in the design, here are some example scenarios:

Fault Injection Attack

A drone delivery service is attempting to deliver high-value cargo to a customer. An attacker's objective is to hijack the drone and force it to land in another location to sell the cargo in the underground economy. Possible attack strategies include spoofing the sensors, jamming GPS or optical sensors, injecting fake visual location data, or a complete takeover using the control protocol. The data from successful attacks will help train machine-learning algorithms to ease future attacks.

Firmware Update Attack

New firmware updates and security bug fixes are distributed to computer systems via Firmware Over the Air (FOTA) mechanisms via the network. An attacker's objective is to attempt to corrupt the firmware image or prevent the update from happening, so the system is never up-to-date in terms of the latest firmware and is thereby vulnerable to attacks.

Code Reuse Attack

Code-reuse attacks are software exploits in which an attacker directs control flow through existing code with malicious intent. For example, return-oriented programming is an effective code-reuse attack in which short code sequences ending in a return instruction (return or ret is a pseudo-instruction that is expanded to jalr zero, 0(ra) for RISC-V architecture) are found within existing binaries and executed in arbitrary order by taking control of the stack and hijack the normal program flow to execute carefully crafted machine instructions. These techniques take advantage of software flaws, such as out-of-bound buffer writes or code pointer overwrites, to alter the control flow of the software run by the core.

Software Supply Chain Attack

In the past, enterprises would craft their applications from scratch. This was a time-consuming and slow process. Digital leaders like Google, Amazon, and Netflix managed to dominate their industries thanks to a more iterative and faster pace of development. These new practices took advantage of open-source software components as a starting point for adding new value. Over the last several years, attackers have discovered ways to compromise these software building blocks to attack high-value targets through software supply chain attacks. Software supply chain attacks have grown 300% in 2021. [9]

Hardware Supply Chain Attack

Now, attackers are extending these same tactics into the hardware used to secure software. Both white-hat researchers and hackers have developed various tactics and tools.[10] A new firmware-level compromise called MoonBounce can compromise systems at the hardware level that is not detectable by traditional OS security scans.[11] Lenovo recently released an emergency security update to prevent hardware attacks that could affect the boot sector on over 100 models.[12] As a result, MITRE's Common Attack Pattern Enumeration and Classification database added a specific category for vulnerabilities in 2020.[13] A recent Ponemon Institute survey found that 64% of enterprises were planning to take steps to improve security at the hardware level, and 85% consider the hardware and firmware security a high or very high priority.[14]

[9] VentureBeat. "Report: Software Supply Chain Attacks Increased 300% in 2021," January 27, 2022. https://venturebeat.com/2022/01/27/report-software-supply-chain-attacks-increased-300-in-2021/.
[10] Tortuga Logic. "A History of Hardware Security and What It Means for Today's Systems," March 22, 2022. https://tortugalogic.com/history-of-hardware-security/.
[11] "MoonBounce: The Dark Side of UEFI Firmware." Accessed May 13, 2022. https://securelist.com/moonbounce-the-dark-side-of-uefi-firmware/105468/.
[12] "Hackers Can Infect >100 Lenovo Models with Unremovable Malware. Are You Patched? | Ars Technica." Accessed May 13, 2022. https://arstechnica.com/information-technology/2022/04/bugs-in-100-lenovo-models-fixed-to-prevent-unremovable-infections/.
[13] Tortuga Logic. "Reducing Hardware Security Risk," July 1, 2020. https://semiengineering.com/reducing-hardware-security-risk/.
[14] "Intel Study: Secure Systems Start with Hardware," https://download.intel.com/newsroom/2022/corporate/secure-systems-hardware-study.pdf

Security Challenges

For several decades system designers and attackers have been playing this cat-and-mouse game: waiting for a new attack to manifest and then trying to find mitigation and release in the next product line or bug fixes patches (ex: microcode patches from Intel/ AMD), and it's a never-ending game. We need a radically new and comprehensive approach to secure cyber-physical computing systems and protect against current and future cyber-physical attacks. It is important to develop new security approaches that consider the interaction among hardware, software, and communication systems so they can be hardened end-to-end. This includes creating a framework for understanding and defending against autonomous security risks across all types of infrastructure, including fleets of cars, automated warehouses, construction sites, farms, and smart cities.

What is Zero Trust?

Zero Trust is a term coined by Forrester Research in 2010 that refers to a proactive and pervasive approach to network security designed to minimize uncertainty. It shifts the paradigm from trust-based on physical connectivity or proximity to a new model that involves always authenticating and verifying every access.

Zero-trust security emerged as a recognition that traditional approaches to securing the perimeter of enterprises, governments, and services do not work well as we move towards distributed and decentralized architectures in the cloud.

The Zero Trust paradigm allows security teams to plan for the possibility that vulnerabilities may exist throughout a chain of interactions among multiple systems, such as across several cloud services, data processes, storage services, and networks. The fundamental concept is to never trust and always verify the provenance of each request. Another basic principle is to assume that a breach has already occurred, so it is essential to limit the blast radius of any breach. Figure 1 summarises the various principles.

1 **Fail Safely and Securely:** Ensure that error conditions don't leave secrets around.
2. **Complete Mediation:** Check every single access to confirm legitimacy.
3. **Rule of Least Privilege:** Minimize any hardware agent's privileges and minimize privilege creep.
4. **Separation of Duty:** Make agents have their own purpose on the designs.
5. **Least Common Mechanism:** Separate out security functions from others.
6. **Secure the Weakest Link:** Protect the design's weakest part.
7. **Defense in Depth:** Build multiple walls.
8. **Simplicity:** Invent simpler architectures.
9. **Psychological Acceptability:** Make security mechanisms easy to use and acceptable to customers.

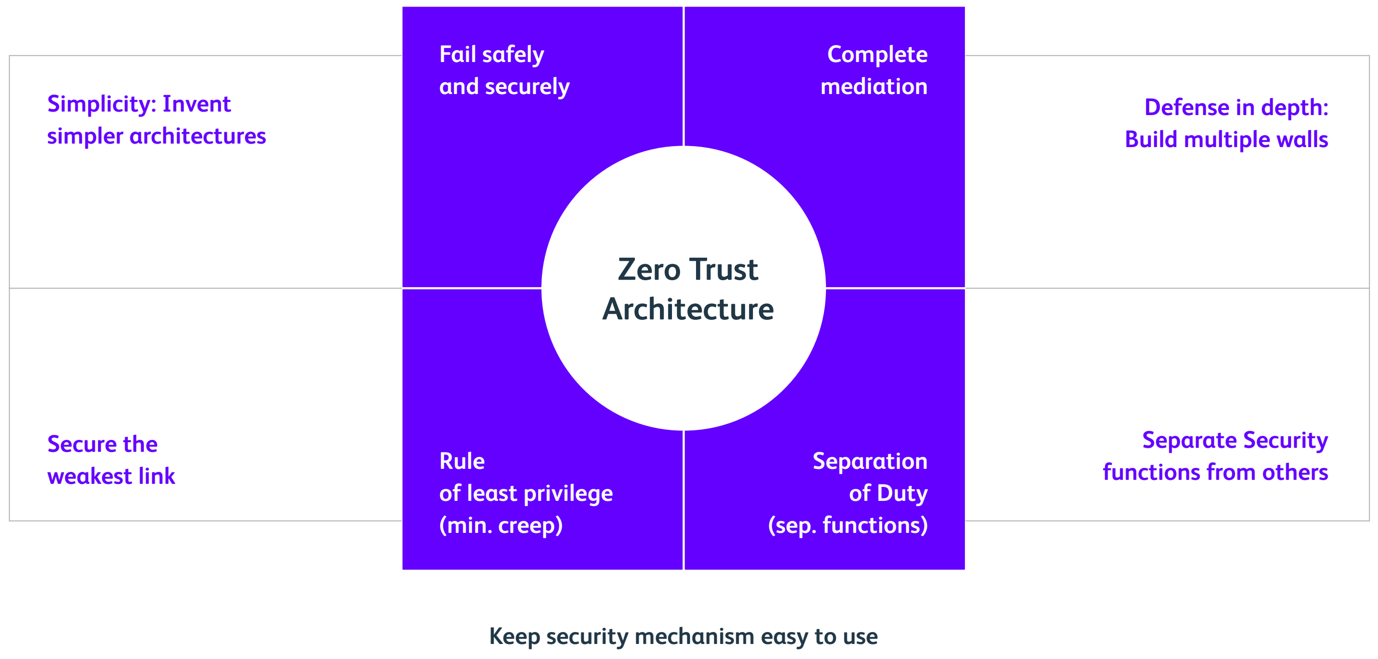

Simplicity: Invent simpler architectures
Fail safely and securely
Complete mediation
Defense in depth: Build multiple walls
Zero Trust Architecture
Secure the weakest link
Rule of least privilege (min. creep)
Separation of Duty (sep. functions)
Separate Security functions from others
Keep security mechanism easy to use

Figure 1: Zero Trust Principles Overview

In this paper, we present how the above zero trust approach can be extended and applied to hardware and software design that starts with the assumption that breaches could or already have occurred at each level of the hardware/software systems stack. This mindset can help mitigate the impact of attacks that have not yet been discovered.

Intel's[15] Zero Trust approach to architecting silicon is a great starting point for security researchers and engineers to appreciate and understand how and where can they apply these 9 principles for improving the trust and security of the systems and products they build.

[15] "A Zero Trust Approach to Architecting Silicon." https://www.intel.com/content/www/us/en/newsroom/opinion/zero-trust-approach-architecting-silicon.html#gs.3nfbi8

What is RISC-V?

The RISC-V (Reduced Instruction Set Computer version 5) architecture is a free and open ISA (instruction set architecture) that is quickly becoming the third most crucial architecture behind Intel and ARM. RISC-V architecture is governed by a non-profit organization called RISC-V International and it accelerates RISC-V adoption by working with the member community. Since the architecture is new and free from the legacy burden the possibilities to adapt to any applications/ use cases are enormous, and we can witness RISC-V being used from tiny IoT sensors to edge devices to autonomous robots to space satellites.

RISC-V International develops the ISA specifications for RISC-V and has a dedicated security team made up of volunteers from various parts of the world, to identify security vulnerabilities, propose mitigation strategies and produce specifications for security features and enhancements. Table 1 summarizes the roadmap of some of the existing and upcoming RISC-V security features.

We at Technology Innovation Institute (TII)/ Secure Systems Research Center (SSRC) are contributing to development of security features such as Confidential Computing, Control Flow Integrity, etc. We are also chairing a Trusted Computing Group[16], which performs research and defines specifications for supporting confidential computing for various device profiles including IoT, Edge, and Cloud.

[16] "RISC-V Trusted Computing SIG." https://lists.riscv.org/g/sig-trusted-computing

High level summary of existing and planned Security features inside RISC-V Processor Core

Vulnerability	Core Feature	Threat Mitigation	RISC-V Specification Approved?
Malicious programs accessing other programs memory	PMP	Memory access control policy for isolation among programs	Yes
	MMU	Memory protection via multi-level page tables	Yes
	MPU	Can be used on devices without MMU	Yes
Rogue I/O devices accessing illegal memory	IOPMP	Memory access control policy	Yes
	IOMMU	Memory protection via multi-level page tables for I/O devices	ETA ~Q4 2022
Malicious programs accessing the sensitive memory of application software	TEE	Providing a hardware-based isolated environment to keep sensitive data	Yes
	Confidential Computing	Providing multiple hardware-based isolated environments to keep various sensitive data's confidentiality and integrity protected plus encryption of memory to prevent physical memory attacks	ETA ~Q4 2022
Malicious programs exploiting software defects, pointer, memory vulnerabilities to leak secrets	CHERI	Capabilities based mechanism for pointer protection, fine-grained memory protection, and fine-grained software compartmentalization	TBD
Malicious programs performing control flow hijack via software buffer overflow, COP, JOP attacks	CFI	Via shadow stack and new ISA for function call labels	ETA ~Q4 2022
Timing, power and other side-channel attacks to steal sensitive data	Side-channel Safety	Microarchitecture review and implementation that is side-channel safe. fence.t & sec.flush are 2 instructions proposed for this	TBD

Table 1: Existing and planned Security features inside RISC-V Processor Core

PMP = Physical Memory Protection
MMU = Memory Management Unit
IOPMP = IO PMP
IOMMU = IO MMU
TEE = Trusted Execution Environment
TZ = Trust Zone from ARM
Realms = Confidential Computing from ARM
CHERI = Capability Hardware Enhanced RISC Instructions
CFI = Control Flow Integrity
H-Extn = Hypervisor Extension
PQC = Post Quantum Crypto
COP = Call Oriented Programming
JOP = Jump Oriented Programming

Applying Zero Trust to RISC-V Systems

TII/ SSRC security researchers along with our research partners have been exploring ways to systematically weave Zero Trust capabilities into every level of silicon hardware/ software design, by following the "Trust nobody" philosophy. Zero Trust requires the ability to (1) detect vulnerabilities when they occur (2) resist known attacks, (3) isolate the element that causes the vulnerability, (4) recover from the attack, and (5) reconfigure the system so that it can continue its mission without or despite a vulnerable component. We shall describe some of the key techniques we are currently working on, categorized into various levels of system stack: 1) Platform level, 2) SoC level, 3) CPU level, and 4) Software level.

1 Zero Trust at the Platform level

1a Component-to-Component Mutual Trust

Zero Trust principles used: Verify Explicitly, Assume Breach

Today's approach to platform design assumes components (ex: GPS, motor controller, PCI devices, etc) on the platform/ motherboard by default to be trusted based on the physical connectivity and often messages arriving on a hardware bus are also assumed to be legitimate. This approach leaves behind serious vulnerabilities if malicious actors find ways to penetrate the hardware supply chain.

One promising countermeasure is to adopt secure collaboration between components in hardware similar to how TLS and HTTPS secure web transactions, where components can mutually identify each other, authenticate each other, establish secure connections and measure the firmware. This could protect the intellectual property in the silicon industry and mitigate physical attacks and illegal firmware updates.

We are exploring various approaches to establish secure sessions between various components on the platform and we found there are already two industry standards available to establish secure communication between two endpoints (say a platform and a component) such as the Distributed Management Task Force's (DMTF) Security Protocol and Data Model[17] (SPDM) specification and the PCI-SIG Integrity and Data Encryption[18] (IDE) specification to establish secure communication at physical buses or links, which we would like to leverage and optimize to support our component-to-component mutual trust establishment. In this scheme, the platform does not communicate with any on-system components/ chips until it establishes trust as shown in Figure 2. This could allow system designers to securely integrate off-the-shelf or custom components into their systems.

The protocol running on the platform (initiator) challenges any new hardware component (responder) connected to the system. For example, if someone replaced a camera, GPS receiver, or motor in a drone, the platform would verify their authenticity using this protocol using public key cryptography certificates.

The components mentioned above are typically active components that might have some compute capacity and key storage to actively participate in the above-mentioned message exchange protocols. We are also actively looking to adapt these multi tenant host VM environments, to low-energy passive components such as low-power sensors that lack computing horsepower for performing cryptographic operations, storage, etc.

[17] "Security Protocol and Data Model Specification." https://www.dmtf.org/sites/default/files/standards/documents/DSP0274_1.0.1.pdf

[18] "Integrity and Data Encryption." https://pcisig.com/sites/default/files/files/PCIe%20Security%20Webinar_Aug%202020_PDF.pdf

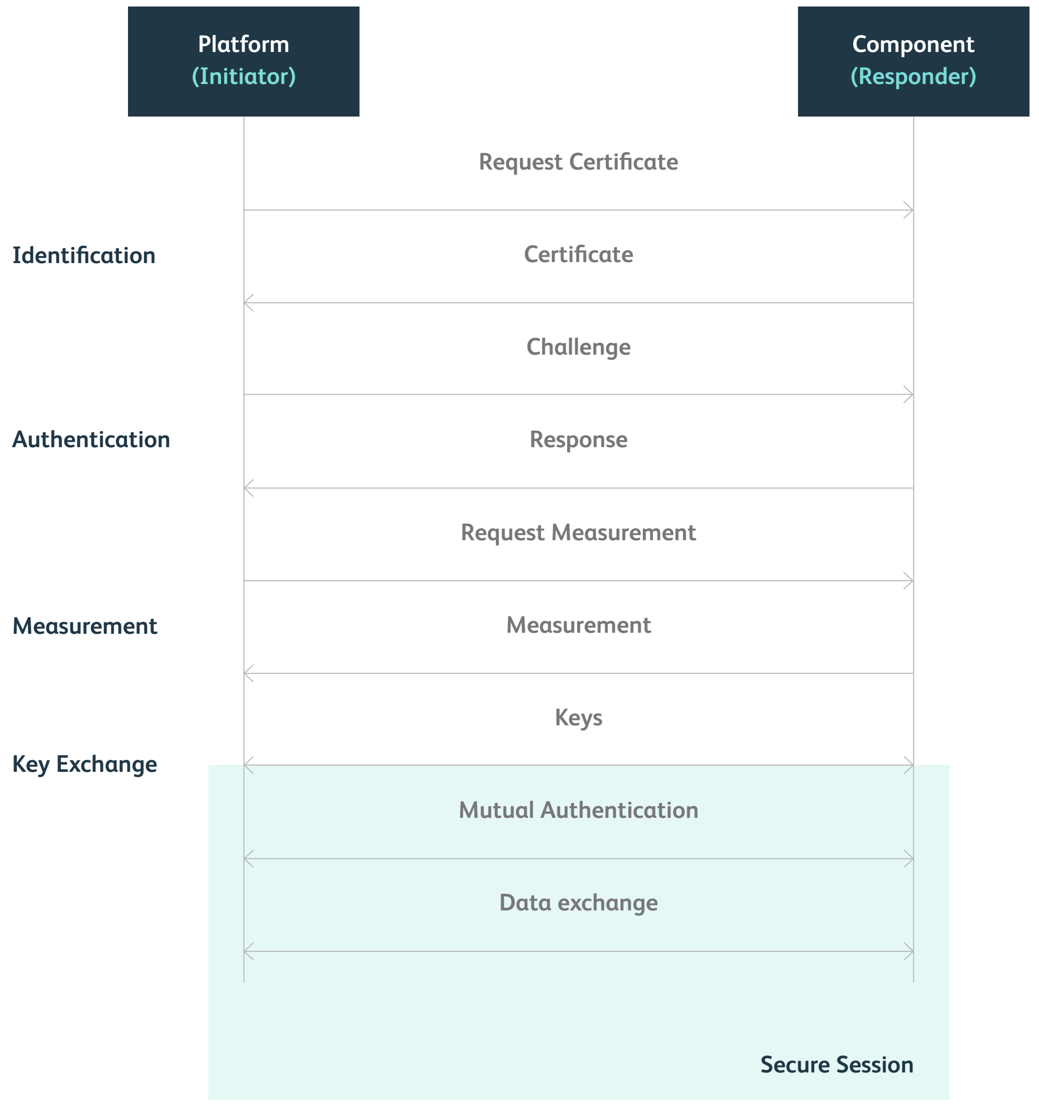


Figure 2: Two endpoints establish mutual authentication and encrypted communication

1b Efficient Redundancy for Fault Tolerance

Zero Trust principles used:
Fail Securely

As per the report from International Technology Roadmap for Semiconductors (ITRS), the reliability of Integrated Circuits (IC) has become a key challenge owing to the technological issues posed by the shrinking process nodes, sensitivity to external influences such as radiation-related effects (radioactive decay or cosmic rays), high temperature, electromigration, process variation, transistor aging (the process of silicon transistors developing flaws over time as they are used, degrading performance and reliability, and eventually failing altogether), etc. Hence, building reliable and fault-tolerant systems that are immune to manufacturing defects and to transient errors is key to achieving Zero Trust.

Fault-tolerant system design techniques have been around for decades, for example in airplane, space, and medical industries, enabling a system to continue its intended operation, possibly at a reduced level, rather than failing completely, when some part of the system fails.

Some popular fault-tolerant techniques:

1 Triple Modular Redundancy (TMR) in which 3 redundant modules (SoCs on the platform) execute the same code in parallel and the result is votted by a majority-voting aggregator to produce a single output, and If any one of the three modules malfunctions or fails, the other two modules can correct the fault thereby continuing the intended operation without failing. It comes at a cost of 3x power, size, weight, etc, which is not practical for autonomous robots such as drones for example, as its usable battery capacity determines the usefulness of the drone.
2 Dual Modular Redundancy (DMR) that uses only 2 redundant modules executing the same code in parallel and a voting aggregator to detect a potential malfunction or failure saves on power, size, weight, etc., but cannot recover from a fault. Some existing fault recovery mechanisms are
   a Checkpointing with roll-back technique: instead of comparing outputs of each module at every instruction, it only takes periodic snapshots of the good state of execution, so whenever a fault has encountered the execution can roll-back to a recent snapshot. Implementing such a mechanism for real-time systems can be quite challenging and may even harm functionality; e.g., performing a roll-back can create a delay that may cause drones to crash while recovering from the fault.
   b Roll-forward technique[19]: allows both modules to continue executing their task speculatively, and power up temporally a third module that can judge which of the other two modules is faulty. Once the fault is detected, the faulty module would roll-forward to the state of the other module, thereby recovering from the fault.

We are exploring approaches to leveraging and optimizing some of the above techniques and also coming up with new schemes to meet the requirements for systems such as drones, having real-time constraints and resource constraints such as power, weight, battery capacity, etc. In order to minimize the overheads of DMR such as 1) additional latency caused by roll-back, and 2) the need for a third temporary module, we are considering approaches, such as combining the roll-back and roll-forward techniques together.

[17] "Roll-forward recovery." https://ieeexplore.ieee.org/document/494475

2 Zero Trust at the SoC level

2a Dynamic Hierarchical ML-based Hardware Security

Zero Trust principles used:
Fail Securely

With the pace at which cyberattacks are deployed, the software/ firmware ecosystem is well adapted to respond to security threats dynamically by the periodic/ on-demand release of software upgrades (e.g., the updates that we receive for iPhones, Android phones) for security mitigations. But on the hardware side, this flexibility is limited. Hardware is designed and once shipped is expected to last for 10 of years while continuously defending against security attacks, which is unrealistic hardware is usually static and unmodifiable (exceptions being microcode updates on X86 systems that have some capability to fix hardware security bugs).

We propose a new mechanism by adding sensors and control units to each component of the SoC, and a global control unit with ML algorithms to 1) dynamically evaluate if a component is under attack and 2) mitigate the attack by changing the structure and/or operation mode of the system so that it could continue to function under the attack.

As an analogy, the human central nervous system operates in a hierarchical nature, for example: when touching a hot object then an immediate autonomous response is generated to remove the hand from the hot object, and only later the information is processed in the brain to analyze the situation. Similarly, we use a hierarchical structure that allows fast, and efficient responses to immediate threats, and the use of more sophisticated algorithms for a deeper holistic analysis.

As described in Figure 3, the following are the building blocks for this approach

- Sensors – provide information regarding the component at run-time; e.g., performance and power counters, reliability counters
- Local Control Unit – collects information from the sensors, validates measurements with respect to the current state and detect any deviations, and take emergency actions upon any violations. The sensor sampling rate can be adjusted, for example, based on the speed of a drone, and the nature of the sampled signals.
- Global Control Unit – Analyses information received from all local control units, and executes system-level policies and actions. As long as information retrieved from the local sensors agrees with the predicted global state, the system considers a normal mode of operation. If there is a disagreement with the predicted global state, it can decide to allow the operation to continue and to learn it as a new valid state, or assume an emergency situation and start the recovery mechanism.

All these blocks are interconnected with each other via a dedicated bus for exchanging commands and data messages.

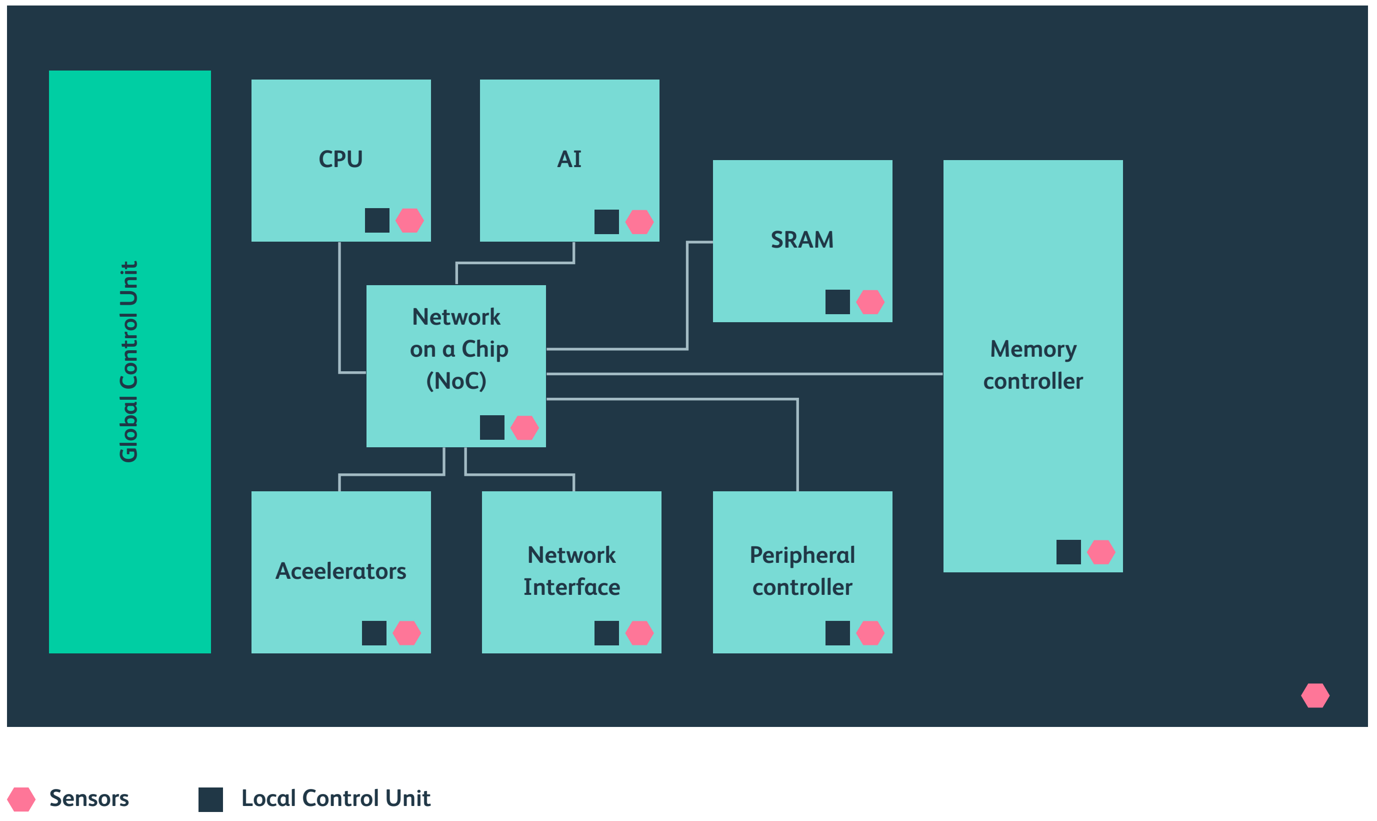


Figure 3: Global and Local Control units with sensors on the SoC

2b Logic Locking

Zero Trust principles used:
Assume Breach, Verify Explicitly, Fail Securely, Secure Weak Link

The globalization of integrated circuit (IC) supply chain and the emergence of threats, such as intellectual property (IP) piracy, reverse engineering, and hardware Trojans, have forced semiconductor companies to revisit the trust in the supply chain. Logic locking is emerging as a popular and effective countermeasure against these threats.

Logic locking as described in Figure 4, is a technique for locking the description of a chip design (design netlist), which can only be unlocked with a unique chip-specific key. Specifically, the design is modified to add new inputs that expect a “logic locking key”, to be applied to unlock the original full chip functionality. This secret key must be loaded into non-volatile memory after the chip has been received post-fabrication by the designer/ company that designed the chips, as described in Figure 5. Without the proper key loaded into the chip, it's basically a non-functional chip. This makes it harder for a third party to clone the design or for a contract manufacturer to profit from making extra copies of the design.

Logic locking can protect fabless chip design companies that outsource fabrication to third-party chip foundries. For example, an unscrupulous engineer or hacker that breaks into these systems might reverse engineer the chip blueprint or copy and pirate the chip or critical blocks. Another concern is that these individuals may also tamper with the chip design to inject stealthy circuitry to launch a malicious attack using a hardware trojan. A third possibility is a fab may produce extra copies to sell on the grey market. Logic locking ensures that only authorized users can activate chips. It can also make it more challenging to reverse engineer the chip to pirate the design or insert meaningful hardware trojans.

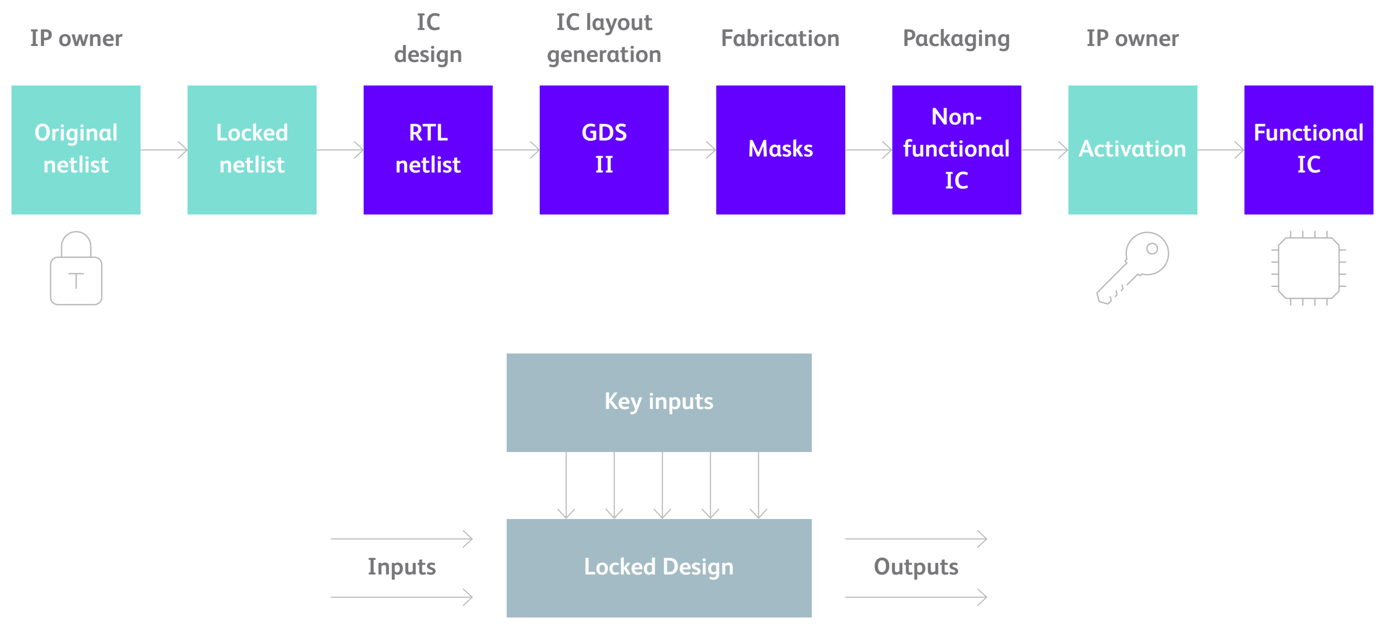
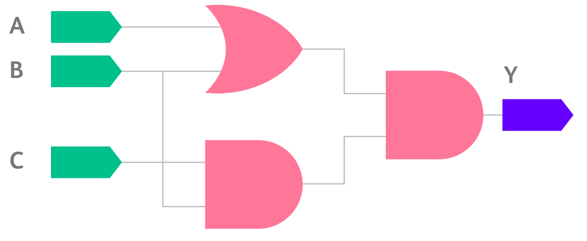
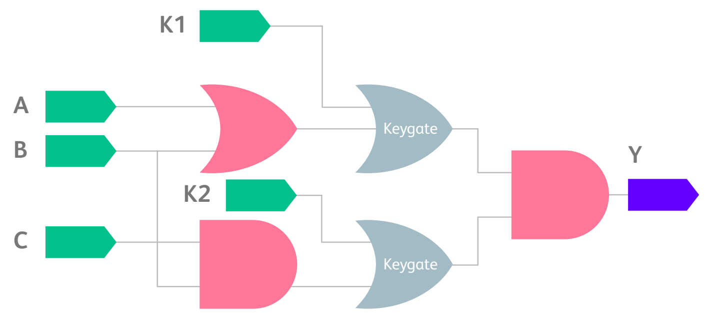


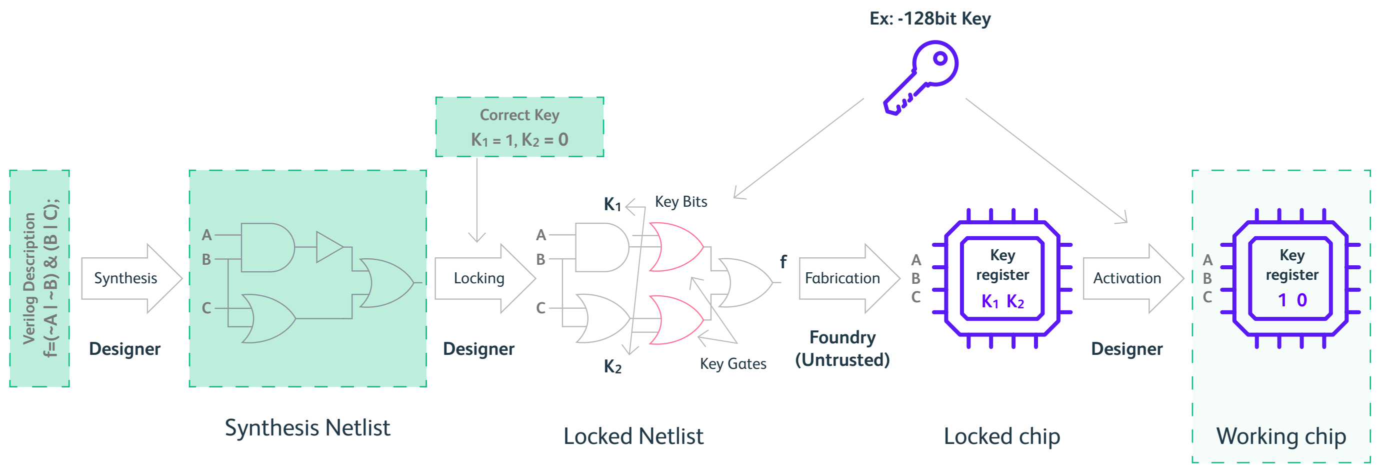
Figure 4: Logic Locking Overview



Original circuit



Logic locked circuit with key-input K1 and K2 fed to key-gates



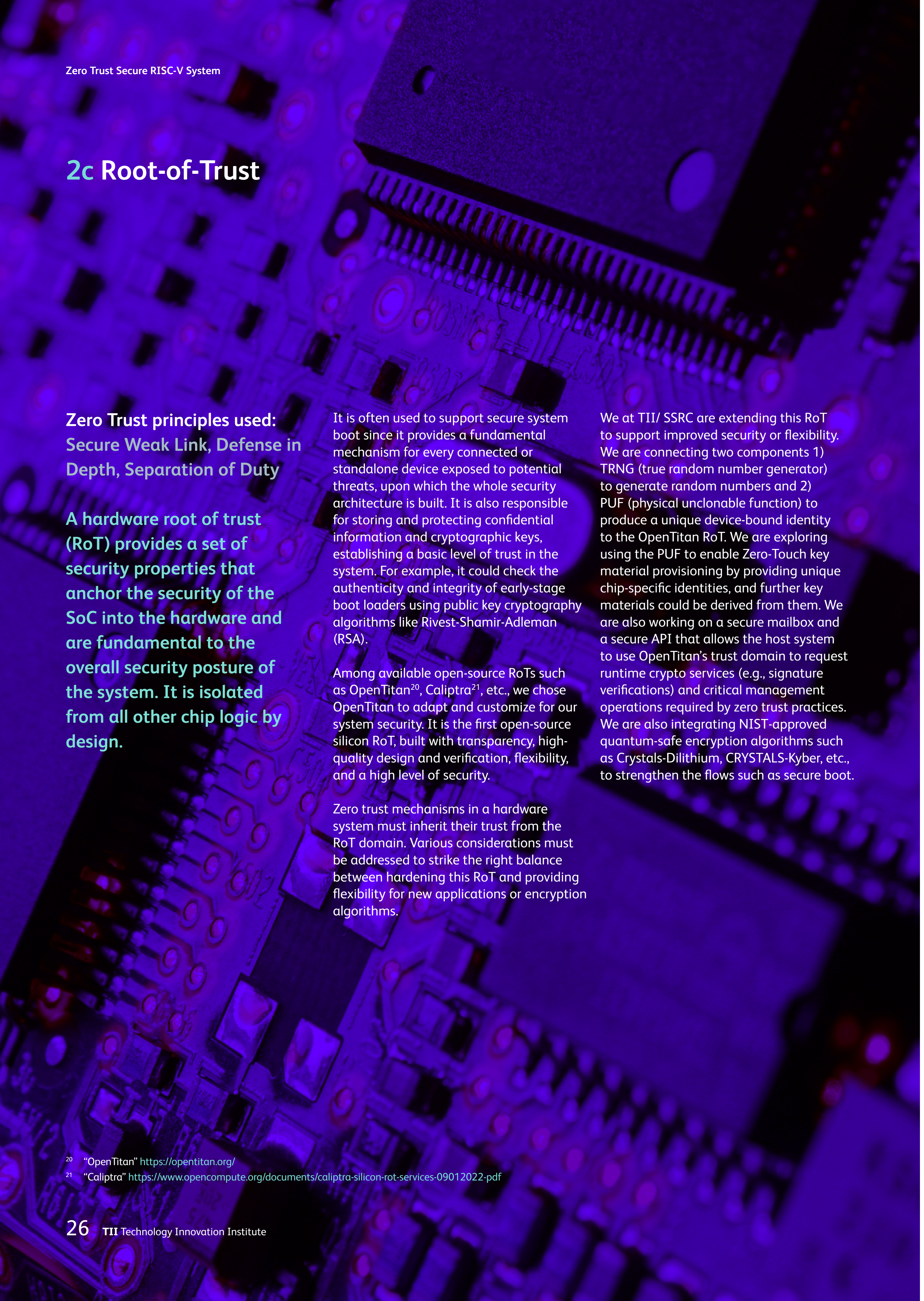
⬚ Potentially Secure Assets 🟩 Secure Assets

Figure 5: Logic Locking Flow

Existing logic locking approaches incur implementation costs, including increased area, power consumption, and performance. Owing to the overhead, it may be desirable to only apply logic locking to critical blocks in a targeted fashion rather than the entire design. For example, designers may apply logic locking to security-critical blocks or locations in the chip that would cripple chip functionality most effectively.

Chip designers need to consider the security of the key across the entire lifecycle - design, activation, and operation of the protected chip. For example, 1) hackers may attempt to retrieve the key by analyzing the chip blueprint stolen from a fab or a working chip procured from the marketplace, or 2) an adversary might simulate the locked design netlist to prune away the incorrect keys and use machine learning techniques to hone these attacks, or 3) an adversary may also probe signals to find the secret key directly, which are practically infeasible due to the enormous time it takes to break the key brute force.

Logic locking, when implemented correctly, is a proactive and strong defense at chip designers' disposal in mitigating chip supply chain vulnerabilities such as Trojans, IP piracy, and chip overbuilding. We have implemented a proof-of-concept for this defense in our SoC by locking several critical design blocks on the chip. We are also enhancing our logic locking tool with the following new capabilities - 1) the ability to choose the security-sensitive blocks to lock and how much to lock depending on the threat model and PPA (power, performance, area) targets of the project, and 2) creating chip specific keys by utilizing the on-chip PUF (physical unclonable function) to limit the impact of a leaked key.



2c Root-of-Trust

Zero Trust principles used: Secure Weak Link, Defense in Depth, Separation of Duty

A hardware root of trust (RoT) provides a set of security properties that anchor the security of the SoC into the hardware and are fundamental to the overall security posture of the system. It is isolated from all other chip logic by design.

It is often used to support secure system boot since it provides a fundamental mechanism for every connected or standalone device exposed to potential threats, upon which the whole security architecture is built. It is also responsible for storing and protecting confidential information and cryptographic keys, establishing a basic level of trust in the system. For example, it could check the authenticity and integrity of early-stage boot loaders using public key cryptography algorithms like Rivest-Shamir-Adleman (RSA).

Among available open-source RoTs such as OpenTitan[20], Caliptra[21], etc., we chose OpenTitan to adapt and customize for our system security. It is the first open-source silicon RoT, built with transparency, high-quality design and verification, flexibility, and a high level of security.

Zero trust mechanisms in a hardware system must inherit their trust from the RoT domain. Various considerations must be addressed to strike the right balance between hardening this RoT and providing flexibility for new applications or encryption algorithms.

We at TII/ SSRC are extending this RoT to support improved security or flexibility. We are connecting two components 1) TRNG (true random number generator) to generate random numbers and 2) PUF (physical unclonable function) to produce a unique device-bound identity to the OpenTitan RoT. We are exploring using the PUF to enable Zero-Touch key material provisioning by providing unique chip-specific identities, and further key materials could be derived from them. We are also working on a secure mailbox and a secure API that allows the host system to use OpenTitan's trust domain to request runtime crypto services (e.g., signature verifications) and critical management operations required by zero trust practices. We are also integrating NIST-approved quantum-safe encryption algorithms such as Crystals-Dilithium, CRYSTALS-Kyber, etc., to strengthen the flows such as secure boot.

[20] "OpenTitan" https://opentitan.org/
[21] "Caliptra" https://www.opencompute.org/documents/caliptra-silicon-rot-services-09012022-pdf

2d Concolic Testing for Security Verification

Zero Trust principles used: Verify Explicitly, Assume Breach, Defense in Depth

Chip designers are increasingly building modern system-on-chip designs by leveraging various pre-verified hardware IPs (intellectual property - examples IP: CPU, GPU, memory controller, etc. available with source code) from third parties, to reduce design and verification costs and reduce time to market. However, the growing reliance on these third-party vendors increasingly affects the security and trustworthiness of these systems. For example, a vulnerability in an IP could be exploited to insert backdoor trojans or launch an attack. Various approaches are being explored to detect and inhibit SoC vulnerabilities systematically.

One concern is that untrusted components could engage in various malicious activities such as denial-of-service, disruption of functionality, leaking sensitive information, reducing battery life, altering sensitive messages, etc., which could lead to catastrophic outcomes.

Current industrial practices based on fuzzing and penetrated tests used to detect such vulnerabilities incur significant drawbacks, including needing an expert. Another approach that uses commercial functional verification frameworks is also being explored using simulation or formal analysis. However, simulation-based systems struggle to detect corner-case vulnerabilities, which an adversary can exploit. Formal tools struggle to support full-scale SoC since they suffer from state space explosion.

We are currently exploring a promising approach that leverages an efficient, transformative infrastructure based on concolic testing for detecting exploitable SoC security bugs and violations at the source code level. Concolic testing, a combination of "concrete" plus "symbolic," is a semi-formal symbolic execution-based test generation methodology used to generate tests to cover a small fraction of corner cases and rare functional scenarios.

We are exploring ways to support parallel and sequential executions and infer the security implications of clock cycles on hardware designs. One benefit of this approach is that it forms one concrete path at a time, so it does not suffer from the scalability issue of formal testing.

As described in Figure 6, our concolic testing framework accepts as input 1) RTL (register transfer language) source code for hardware designs and 2) security properties extracted from threat model and security objectives and automatically produces test cases and test results. Based on the test results the engineer will be able to identify and rectify security vulnerabilities.

Let's take an example to put things in perspective:

Inputs to Concolic Testing Engine:

Security Properties extracted from Threat Model & Security Objective + Assertions + RTL Design

Asset	Crypto keys in internal registers
Threat	Leakage of secret asset i.e., unencrypted plain text can be retrieved by an attacker. This violates the confidentiality property of the secure assets of SoC design.
Trigger Condition	Asynchronous reset applied to crypto engine
Security Objective	Internal register values shall be cleared after any asynchronous reset

Table 2: Input to Concolic Testing Engine

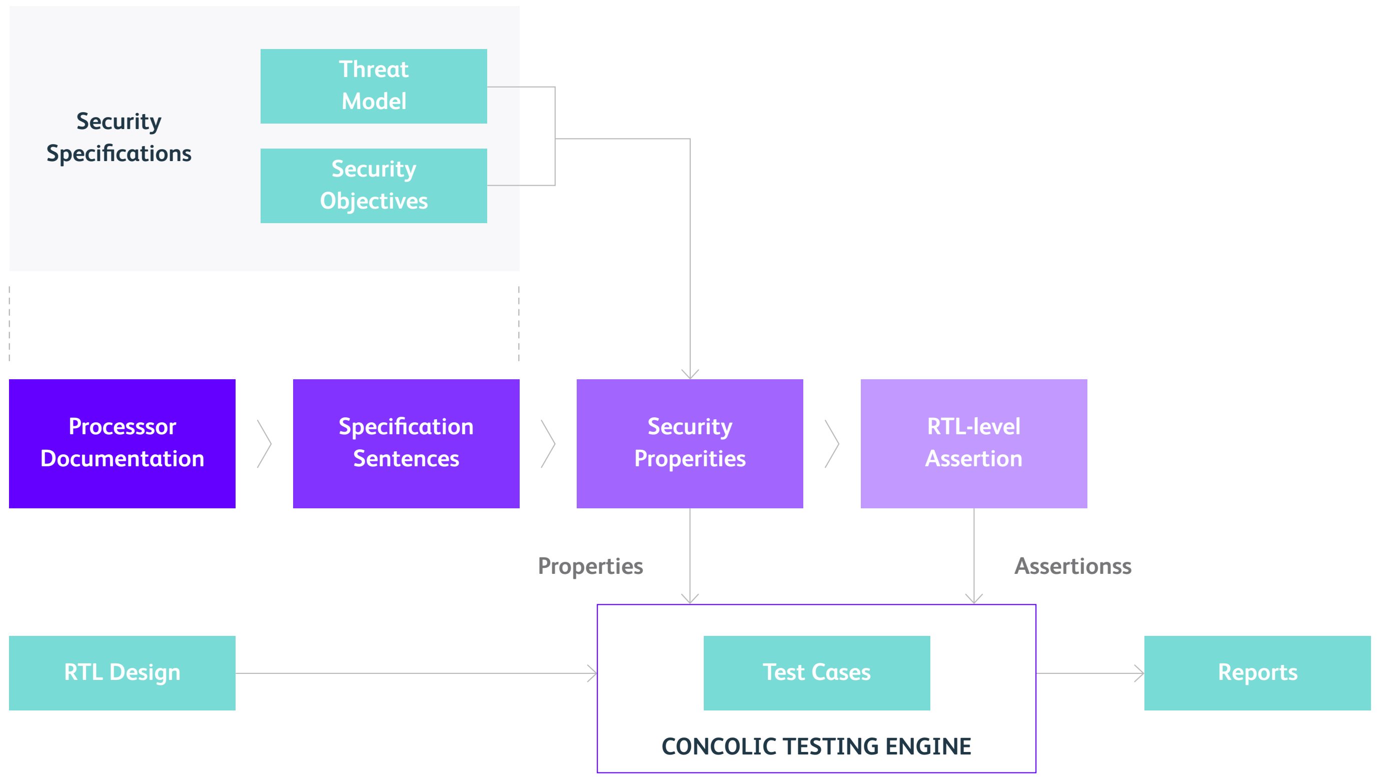


Figure 6: RTL-level Concolic Testing Framework

Output from Concolic Testing Engine:

Security Reports

Security Weakness	Weaknesses in this category are related to system power, voltage, current, temperature, clocks, system state saving/restoring, and resets at the platform and SoC level
Violation Type	Information leakage
Hardware CWE[22] Category	Power, Clock, Thermal, and Reset Concerns (CWE-1206[23])
Test Case	Apply asynchronous reset and read contents of the target register where the key is stored. If the value read is previous content, then the bug is confirmed
Bug Description	Registers are not cleared after asynchronous reset
RTL Assertion	<<reg1>> r1 = 31'h0000;

Table 3: Output from Concolic Testing Framework

[22] "Common Weakness Enumeration" https://cwe.mitre.org/
[23] "CWE-1206" https://cwe.mitre.org/data/definitions/1206.html

2e Future-proofing against Quantum Computer Attacks

Zero Trust principles used: Assume Breach, Secure Weak Link, Defense in Depth

Quantum computers are still in their early development. One concern is that more capable hardware will be able to take advantage of new quantum algorithms to crack popular cryptography techniques. For example, Shor's algorithm could break integer factorization and discrete logarithm-based cryptography techniques like RSA and ECC. Grover's algorithm imposes a similar threat to symmetric cryptography. For today's ubiquitous RSA encryption algorithm, a conventional computer would need about 300 trillion years to crack a 2,048-bit RSA digital key. But with Shor's algorithm on a quantum computer powered by 4,099 qubits would need just 10 seconds[24]. A 1000 logical qubit quantum computer would be a reality by end of this decade.

NIST has initiated a process of standardizing several post-quantum cryptographic (PQC)[25] algorithms that are proven to be resilient to new quantum attacks. As an example, CRYSTALS-Dilithium[26] is a digital signature scheme that is strongly secure under chosen message attacks based on the hardness of lattice problems, and not based on integer factoring problem which is susceptible to quantum computer attacks. The security notion means that an adversary having access to a signing oracle cannot produce a signature of a message whose signature he hasn't yet seen, nor produce a different signature of a message that he already saw signed.

We at TII/SSRC are working on enabling post-quantum secure boot on top of OpenTitan RoT. This includes modifying the boot flow and integrating a hardware accelerator for the CRYSTALS-Dilithium algorithm inside the RoT as shown in Figure 7. Via a secure mailbox, we expose APIs for PQC signing, public-key encryption, etc., to the host system. In the future when NIST approves more PQC candidates we shall add support for them and share it back with the community.

[24] "Quantum computers could crack today's encrypted messages." https://www.cnet.com/tech/computing/quantum-computers-could-crack-todays-encrypted-messages-thats-a-problem/#:~:text=For%20today's%20ubiquitous%20RSA%20encryption,just%2010%20seconds%2C%20Wood%20said.

[25] "NIST PQC algorithm candidates" https://csrc.nist.gov/projects/post-quantum-cryptography

[26] "CRYSTALS-Dilithium." https://pq-crystals.org/dilithium/

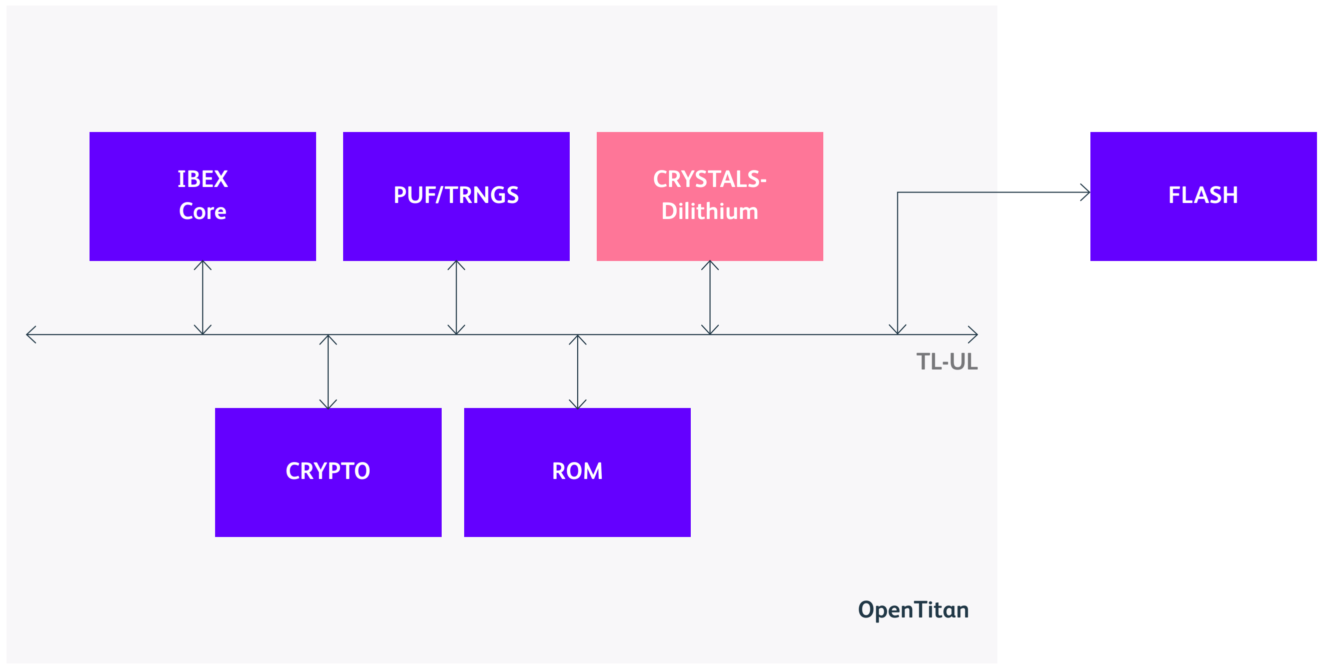


Figure 7: Integration of CRYSTALS-Dilithium accelerator inside OpenTitan RoT

2f Hardware Shielding

Zero Trust principles used: Fail Securely, Defense in Depth

Typical hardware attacks may involve efforts to probe chip logic or perform side-channel analysis to look for secrets. Existing mitigation strategies include randomizing logic, adding locking mechanisms, masking logic, and adding redundant operations. Speculative execution attacks put a dangerous new twist on information leakage through microarchitectural side channels. Ordinarily, programmers can reason about leakage based on the program's semantics and prevent said leakage by carefully writing the program to not pass secrets to covert channel-creating transmitter instructions, such as branches and loads. Speculative execution breaks this defense because a transmitter might miss-speculatively execute with a secret operand even if it can never execute with said operand in valid executions.

One promising alternative we are exploring is to prevent the signals from being exposed to the hacker (shielding). This is similar to how modern systems handle faulty components by hiding faults from the real world. A fault becomes a bug only if the outside world sees it. Similarly, an internal vulnerability can only be exploited if seen by hackers. This approach involves developing a set of primitives that prevent the signals that may be susceptible to side-channel attacks from being exposed externally.

The primitives need to strike the right balance between obfuscation and efficiency. For example, one primitive we have developed adds a random delay in processing, making it harder for an external observer to correlate between value and execution time.[27] This can protect the system against timing attacks. Another primitive we developed adds extra power or other measurable signals to chip emissions during sensitive operations, making it harder for hackers to find EMF (electromagnetic field) signals that correlate with activity. This can reduce the risk that power traces could be used to recover sensitive information.

We are developing new and improved primitives that help protect against these classes of attacks and a new security management unit integrated into the hardware to run these primitives efficiently. This approach will 1) improve the protection of the system to manage known security attacks and 2) provide an infrastructure to manage future security attacks which are not known at the design or even implementation time.

[27] [Men19] A. Mendelson, "Secure Speculative Core," 2019 32nd IEEE International System-on-Chip Conference (SOCC), 2019, pp. 426-431

3 Zero Trust at Processor (RISC-V) Level

3a Deep-Learning based Control Flow Integrity

Zero Trust principles used: Verify Explicitly, Defense in Depth, Simplicity, Fail Securely

Today, a lot of software is written in memory-unsafe languages, such as C and C++, which introduces memory corruption bugs. This makes software vulnerable to attack since attackers exploit these bugs to make the software misbehave. Modern Operating Systems (OSs) and microprocessors are equipped with security mechanisms to protect against some classes of attacks. However, these mechanisms cannot defend against all attack classes. In particular, Code Reuse Attacks (CRA), which re-uses pre-existing software for malicious purposes, is an important threat that is difficult to protect against.

Computer security exploits like Return-Oriented Programming (ROP)[28] and Jump-Oriented Programming (JOP) hijack the normal program flow to execute carefully crafted machine instructions. These techniques take advantage of software flaws, such as out-of-bound buffer writes or code pointer overwrites, to alter the control flow of the software run by the core.

Control-Flow Integrity[29] (CFI) detects this malicious code from redirecting normal program flow and helps mitigate it. CFI protects both the forward edge and the backward edge of the program control flow.

- **Forward edge protection:** Carefully crafted gadgets could be introduced to alter indirect branch targets in the victim code, for which landing pads or labels act as a mitigation technique that restricts indirect branch targets to limited addresses that are pre-labeled for forward edge integrity protection.
- **Backward edge protection:** Buffer overflow attack or code injection attack could corrupt the call stack thereby causing the victim code to return to invalid addresses, for which a shadow call stack is a popular mitigation mechanism for backward edge integrity protection.

[28] "Return Oriented Programming" https://en.wikipedia.org/wiki/Return-oriented_programming
[29] "A Survey of Hardware-based Control Flow Integrity" https://arxiv.org/ftp/arxiv/papers/1706/1706.07257.pdf

Comparison of some popular CFI techniques available

Technique	Forward Edge	Backward Edge	Memory Overhead	Runtime Overhead	Architectural Modifications
Code Pointer Integrity	Probabilistic	Probabilistic	Minor	Acceptable (for only code pointers)	Major (requires cryptographic accelerator)
Memory Tagging	Probabilistic	Probabilistic	Noticeable	Minor	Major (requires tag cache)
Shadow Stack + Landing pads	Fine grained	Total	Minor	Minor	Modest (requires ISA and MMU changes)

Table 4: Popular and existing CFI mechanisms [30]

[30] "Hardware CFI Techniques" https://docs.google.com/document/d/1QrqfWbEuY3X8yPlI-udMLo0YCLPxWZC2wsw2IJ_bohs/edit

RISC-V security community recently has recommended control-flow integrity support using shadow stacks and landing pads due to its low memory, runtime overheads with modest architectural extensions to the CPU (ISA extensions and compiler adaptations).

We are currently exploring to enhance this landscape further with a promising approach with deep learning as described in Figure 8, where control-flow related signals and microarchitectural (uArch) events from the RISC-V Core, data from sensors (power and performance counters) are extracted, and exported to a train a neural hardware monitor to detect malicious control flow patterns. Later on the actual target device, the neural monitor performs inferencing to detect faults and report to the Core, for the handling of control flow violations. We propose to redesign the RISC-V CPU decoder, the control pipeline, potential ISA extensions, and the compiler toolchain extensions.

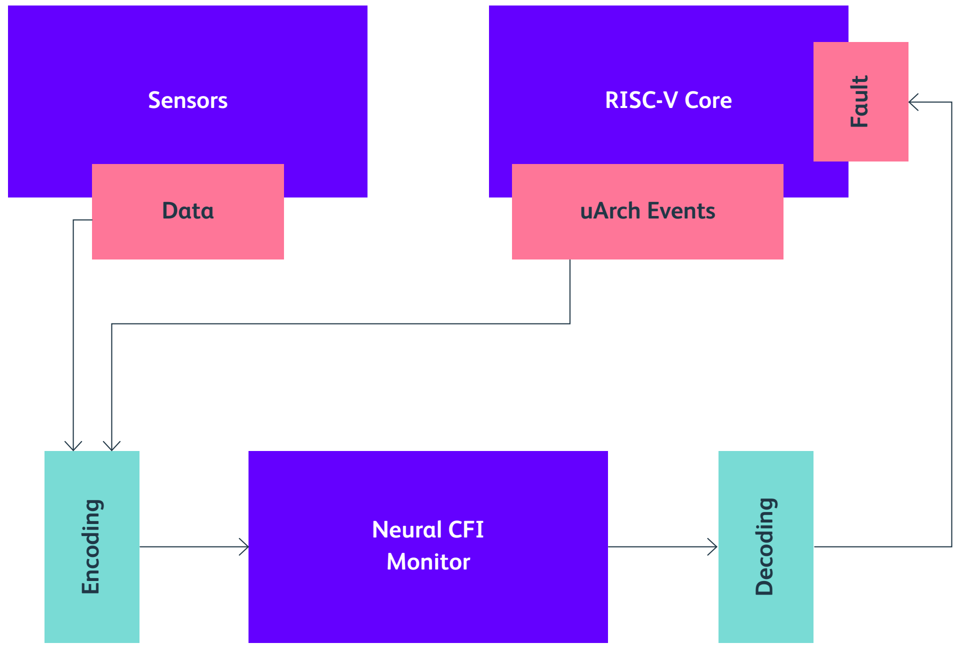


Figure 8: ML-based Control Flow Integrity Architecture

3b Hypervisor Support for Isolation

Zero Trust principles used:
Least Privilege, Separation of Duty, Defense in Depth

Virtualization is a technique that offers temporal and spatial isolation among various processes/ applications by using a software layer beneath the operating system, called a hypervisor. The hypervisor allows hosting one or more Virtual Machines (VMs) on the same platform. This approach improves security by making it easier to standardize application and OS code and enforcing isolation across apps and their data. This also makes it possible to efficiently share the underlying hardware to improve utilization and enhance software modularity and flexibility.

Virtualization technology plays a crucial role in several aspects of the zero-trust vision, such as 1) the rule of least privilege, 2) separation of duty and 3) defense in depth. A new privileged hypervisor mode for RISC-V enables the separation and segregation of security-critical tasks from non-critical ones to simplify implementing the rule of least privilege. It also makes it easier to decompose and compartmentalize different subsystems in specific virtual machines to help enforce separation of duty. It also adds a new layer of defense for the overall defense-in-depth strategy.

In Figure 9, we describe an overview of how legacy software architecture can be modularized with hypervisor capability, where each of the applications could be isolated from each other and the underlying hardware for security and reliability. For example, in a drone usecase let's say the networking proxy has crashed due to some fault injection attack, it will be contained within the networking VM and will not bring down the whole system, and a restart of the networking VM alone could fix the problem. The drone could be offline and hovering for a period of time until the network connection is restored again.

The RISC-V specification has 4 privilege modes 1) M-mode is the highest privilege mode where trusted firmware runs, 2) new HS-mode which is introduced to support a hypervisor, 3) S-mode to host an operating system and 4) U-mode for user/ applications. With hypervisor extension[31] the supervisor mode is modified to a hypervisor-extended supervisor mode (HS-mode), which is orthogonal to the new virtual supervisor mode (VS-mode) and virtual user mode (VU-mode), and therefore can easily accommodate different hypervisor architectures. The RISC-V security committee has ratified the hypervisor extensions specification 1.0 version and we have implemented the feature on a RISC-V CVA6[32] CPU and contributed it back to OpenHW Group for anyone to download and use. We are also actively contributing to developing the specifications for new features such as the RISC-V Advanced Interrupt Architecture (AIA) and RISC-V I/O Memory Management Unit (IOMMU).

[31] "RISC-V Hypervisor Extensions" https://lists.riscv.org/g/tech-privileged/topic/80346318
[32] "RISC-V CVA6 CPU" https://github.com/openhwgroup/cva6

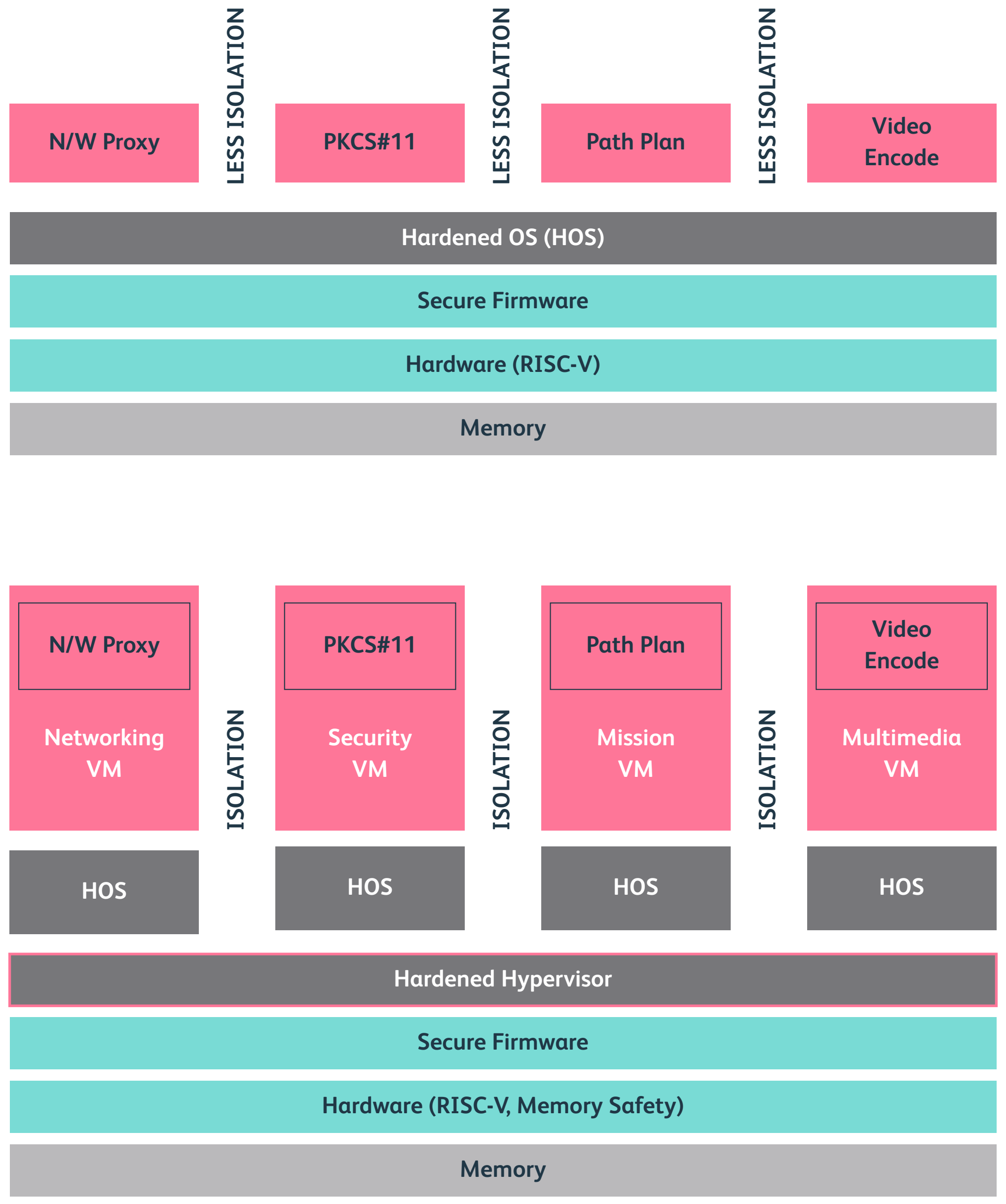


Figure 9: Hypervisor Isolation Architecture

3c Capabilities-based Hardware Architecture (CHERI)

Zero Trust principles used:
Least Privilege, Separation of Duty, Defense in Depth, Complete Mediation, Least Common Mechanism, Secure the Weakest Link

CHERI (Capability Hardware Enhanced RISC Instructions)[33] extends conventional hardware Instruction-Set Architectures (ISAs) with new architectural features to enable fine-grained memory protection and highly scalable software compartmentalization. The CHERI memory-protection features allow historically memory-unsafe programming languages such as C and C++ to be adapted to provide strong, compatible, and efficient protection against many currently widely exploited vulnerabilities. The CHERI scalable compartmentalization features enable the fine-grained decomposition of operating-system (OS) and application code, to limit the effects of security vulnerabilities in ways that are not supported by current architectures.

CHERI has the potential to radically change the way we build security on processors in the future. It automatically enforces several Zero Trust principles at the hardware level, including complete mediation, rule of least privilege, separation of duty, least common mechanism, secure the weakest link, and defense-in-depth. It could even defend against control flow attacks such as ROP/ JOP attacks that we had discussed in chapter 3.a., and hence some of the other techniques could become redundant with CHERI.

The industry has started implementing CHERI on research platforms, such as ARM's Morello program. The RISC-V security committee has newly created a CHERI special interest group to collaborate with security experts to analyze and explore possibilities of standardizing CHERI ISA into the RISC-V ISA.

We are exploring ways of combining 1) hardware virtualization, 2) CHERI architecture, 3) TEE (trusted execution environment), and 4) microkernels to balance tradeoffs in performance, power, code size, security, and programming complexity by utilizing the best from each architecture for improving RISC-V security even further.

[33] "CHERI Architecture"
https://www.cl.cam.ac.uk/research/security/ctsrd/cheri/cheri-risc-v.html#:~:text=CHERI%2DRISC%2DV%20is%20a,extended%20version%20of%20ARMv8%2DA.

4 Zero Trust at the Software level

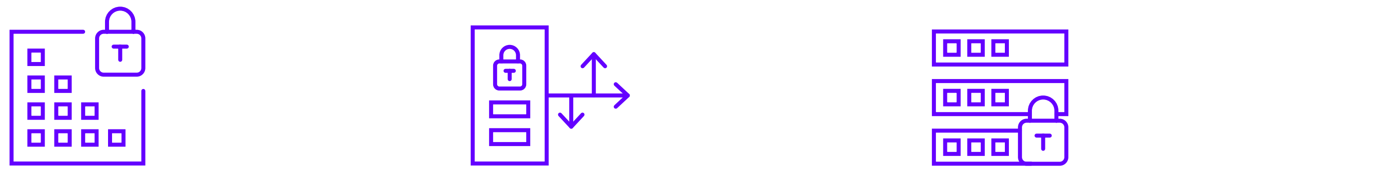
4a Confidential Computing

Zero Trust principles used: Least Privilege, Separation of Duty, Defense in Depth, Secure Weak Link

Data in all computer systems can be in transit (networking), at rest (storage), or in use (compute). Confidential Computing protects data in use by performing the computation in a hardware-based or virtualized Trusted Execution Environment (TEE). This protects data confidentiality, data integrity, and code integrity.

Existing encryption

Confidential computing



Data at rest

Encrypt unactive data when stored in blob storage, database, etc.

Data in transit

Encrypt data that is flowing between untrusted public or private networks

Data in use

Protect/encrypt data that is in use, while in RAM, and during computation

Figure 10: Data protection at rest, in transit and in use

Confidential computing can protect the security of data, the integrity of the data, and the code that processes the data, to ensure apps perform the correct computation. All three elements of data confidentiality, data integrity, and code integrity can help dramatically simplify the creation of zero-trust architectures, without any uncertainty from operating systems, hypervisors, or other applications.

The vulnerability of data-in-use has gotten the attention of attackers who now have been targeting data, including high-profile memory scraping, such as the Target breach in 2013, and CPU-side-channel attacks. In addition, the triton attack in 2017 and the Ukraine power grid attack in 2015 are only two of several high-profile attacks on data in use involving malware injection. The protection of data and applications during execution is increasingly important and must be part of the overall defense strategy.

This is why we need Confidential Computing.

Traditionally higher privileged OS/ Hypervisors with their right to manage resources also have full access to application memory. As a result, any attack on OS/ Hypervisor will open a doorway to application data. So the industry was focussing on hardening the OS/ Hypervisors. With confidential computing, we would like to change that paradigm, where the OS/ Hypervisors are only resource managers and are denied access to application memory. Now the application only needs to trust the hardware not OS/ Hypervisors thereby shrinking the TCB (Trusted Computing Base).

Intel demonstrated the first practical, confidential computing architecture with the Intel SGX and Intel TDX architectures. Arm followed with their ARM Realms technology built on top of prior ARM TrustZone technology and virtualization technology. AMD also offers a competing Confidential VM technology.

There are several recent research contributions to RISC-V such as Keystone which implements enclave memory isolation by leveraging the PMP (Physical Memory Protection) mechanism of RISC-V, which includes a set of paired registers to indicate physical memory regions as well as their access permissions. However, the number of enclaves in Keystone is restricted by the number of PMP regions (up to 16). In order to defend against physical attacks, Keystone leverages on-chip computing, which is costly due to the restricted on-chip RAM. CURE[34] adopts enclave ID-based access control for customizable enclaves. It utilizes a hardware arbiter to record contiguous physical memory regions of enclaves, which can only support 13 enclaves. Similarly, the enclave number of Sanctum35 is also restricted by the number of isolated DRAM regions. TIMBER-V[36] extends the RISC-V ISA to run an unlimited number of enclaves, but it incurs non-trivial overhead (25.2% on average) and does not consider memory integrity protection.

Penglai proposes new hardware extensions like Mountable Merkle Tree and Guarded Page Tables to achieve scalable protection (up to 1000 enclaves).

We at TII/ SSRC have collaborated with the RISC-V.org security horizontal committee to form a Trusted Computing Group to focus specifically on the problem of enabling confidential computing[37] on RISC-V platforms. We analyzed the threat model[38] and are in the process of defining a scalable and compatible confidential computing model. Initial proposals as described in Figure 11, prescribe a software-based architecture without any RISC-V instruction set extensions and following industry standard attestation protocols such as Internet Engineering Task Force (IETF) Remote ATtestation ProcedureS (RATS)[39] or Trusted Computing Group (TCG) DICE[40] attestation architecture. After performance evaluation, we would consider adding new ISA extensions and registers for this feature.

[34] "CURE: A Security Architecture with CUstomizable and Resilient Enclaves" https://www.usenix.org/system/files/sec21summer_bahmani.pdf
[35] "Sanctum Secure Processor" https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/costan
[36] "TIMBER-V: Tag-Isolated Memory Bringing Fine-grained Enclaves to RISC-V" https://www.ndss-symposium.org/ndss-paper/timber-v-tag-isolated-memory-bringing-fine-grained-enclaves-to-risc-v/
[37] "RISC-V Confidential Computing Overview" https://docs.google.com/presentation/d/1D6O9VopMXBrPmwZW-IKnXRrwJcVnXlU6ZC_Zkf9qh5E/edit#slide=id.p
[38] "RISC-V Confidential Computing Threat Model" https://docs.google.com/document/d/1TXiuy4ac3hQmEKvtTtM5aFVHLnNKCrYxeRZFYPRq2Xw/edit#heading=h.9yggln8khyd0
[39] "Remote ATtestation ProcedureS" https://datatracker.ietf.org/wg/rats/about/
[40] "DICE Attestation Architecture" https://trustedcomputinggroup.org/wp-content/uploads/TCG_DICE_Attestation_Architecture_r22_02dec2020.pdf

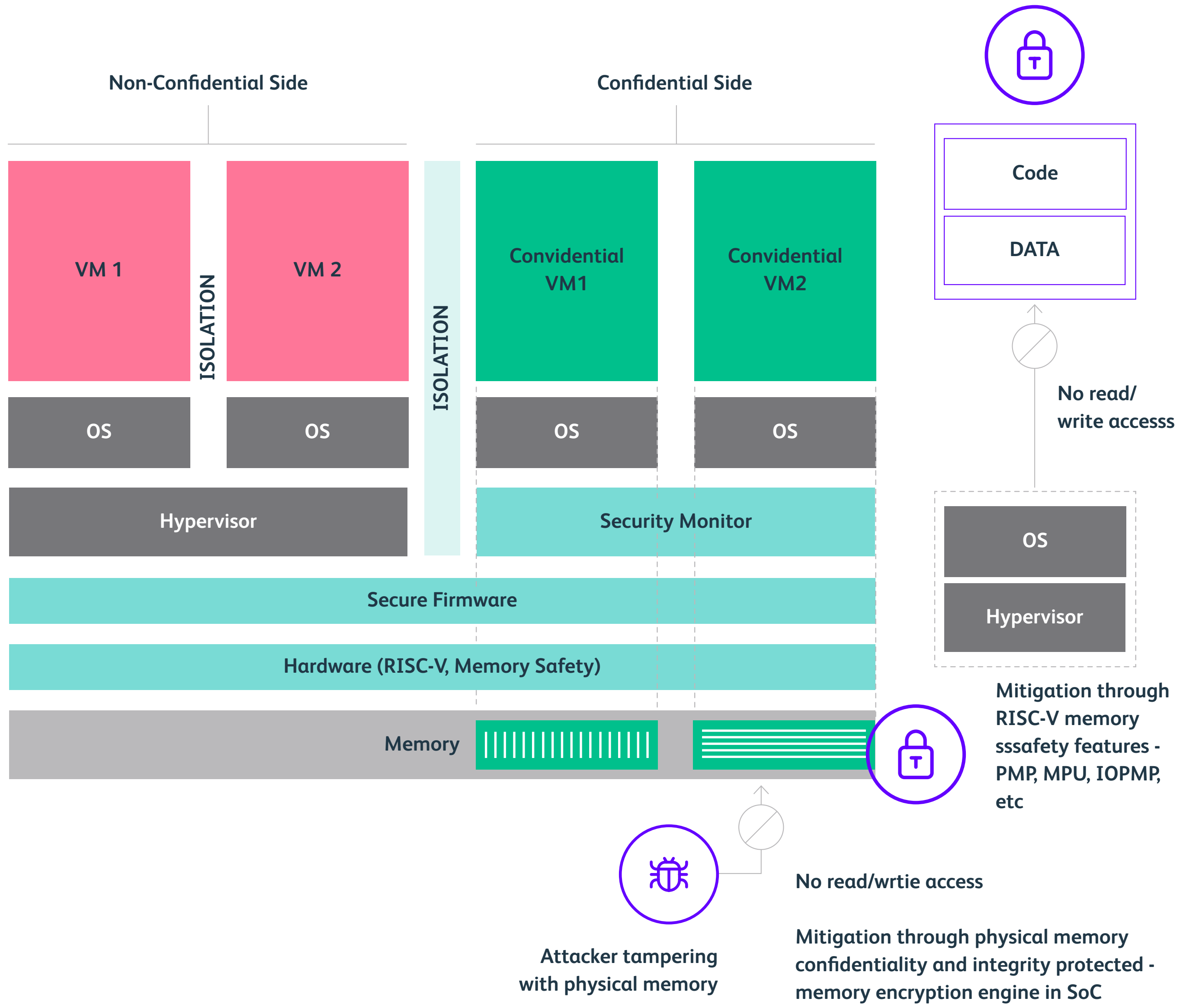

Non-Confidential Side
Confidential Side
VM 1
VM 2
ISOLATION
ISOLATION
Convidential VM1
Convidential VM2
OS
OS
OS
OS
Hypervisor
Security Monitor
Secure Firmware
Hardware (RISC-V, Memory Safety)
Memory
Code
DATA
No read/ write accesss
OS
Hypervisor
Mitigation through RISC-V memory sssafety features - PMP, MPU, IOPMP, etc
No read/wrtie access
Attacker tampering with physical memory
Mitigation through physical memory confidentiality and integrity protected - memory encryption engine in SoC

Figure 11: RISC-V Confidential Computing Proposed Architecture

Internet-of-Things (IoT) special case: IoT devices such as SIM cards, cameras, drones, smart homes, etc, are resource constrained in terms of features (example: lack of paged virtual memory/ MMU, security privilege rings such as supervisor mode, etc.) power, thermals, battery capacity, weight, etc., which brings significant challenges to implementing/ enhancing system security. Protection of data at the source of creation, which are these IoT devices, is gaining importance nowadays. We are exploring new and innovative ways to architect a lightweight implementation of confidential computing for the IoT class of devices, produce an IoT confidential computing specification with the community and standardize it at RISC-V International.

4b Capabilities-based Software Architecture (CHERI)

Zero Trust principles used: Least Privilege, Separation of Duty, Defense in Depth, Complete Mediation, Least Common Mechanism, Secure the Weakest Link

As previously mentioned, efforts like CHERI are exploring ways to implement better fine-grained security controls at the hardware level. Software-based capabilities complement these to enhance zero-trust architectures using fine-grained compartmentalization at the OS level. This approach needs to consider minimizing the impact and maximizing control across capabilities in middleware and OS libraries.

For decades, designers have used software capabilities to implement fine-grained security controls in microkernels. Now researchers are exploring ways to retrofit existing software stacks to leverage CHERI, for example, 1) CheriRTOS[41] provides efficient and scalable task isolation, fast and secure inter-task communication, fine-grained memory safety, and real-time guarantees, using CHERI hardware capabilities as the sole protection mechanism in embedded systems, 2) CAP-VMs[42] implements isolation and sharing at a finer granularity using CHERI hardware capabilities.

We are exploring ways to combine hardware capabilities like CHERI and advanced software techniques to compartmentalize software stacks at multiple levels better. We believe this will make configuring and enforcing different security policies/ guarantees more efficient.

[41] Xia et al., "CheriRTOS." https://www.cl.cam.ac.uk/research/security/ctsrd/pdfs/201810-iccd2018-cheri-rtos.pdf
[42] "CAP-VM." https://www.usenix.org/system/files/osdi22-sartakov.pdf

A More Secure Future

For many decades, Security was considered an afterthought. With the regular onslaught of cyber attacks on our networks and systems and their consequences on data privacy leading to financial losses, now security has taken the driver's seat in the industry. The cat-and-mouse game between system designers and attackers: waiting for a new attack to manifest and then trying to find mitigation, is a never-ending game, and so system designers need to take two steps forward to retain an advantage over attackers. Several leading companies including TII are adopting a security-first mindset and investing in Zero Trust research to secure our digital future in the years to come.

The foundation for system security starts with hardware, as software alone cannot defend against all threats. The root-of-trust is a key building block that provides an anchor point in the hardware to build levels of security and trust. The adoption of quantum-resistant cryptography and incorporation into the secure boot and other features are essential to data privacy and security. Confidential Computing designed for cloud security is now gaining momentum in the edge/IoT market segment as well and has its own challenges to deal with due to the resource-constrained nature of these devices. Leveraging the power of deep learning to harden security features such as control flow integrity and anomaly detection is also a growing area of research. CHERI is another promising capability that in the next 5 years, has the potential to hit the mainstream and even make several other security features redundant.

TII/ SSRC and its research partners for the next 2-3 years would perform research and development on the security areas as described in this whitepaper. We are contributing to RISC-V International, OpenHW.org, and confidentialcomputing.io organizations, to collaboratively advance RISC-V system security and incorporating these technologies into our research SoC named "Al Saqr" to enhance the security posture for autonomous cyber-physical systems, for example, drones systems. The deliverables such as RISC-V CPU ISA extensions, design, and code shall be made available as open source so that the rest of the security community shall reap benefits from our work.

Please reach out if you would like to partner with us on this journey.

Co-authors

Avi Mendelson	mendlson@technion.ac.il	Technion Israel Institute of Technology
Abdulhadi Shoufan	abdulhadi.shoufan@ku.ac.ae	Khalifa University
Kais Belwafi	kais.belwafi@ku.ac.ae	Khalifa University
Hussam Al Hamadi	hussam.alhamadi@ku.ac.ae	Khalifa University
Ashfaq Ahmed	ashfaq.ahmed@ku.ac.ae	Khalifa University
Freddy Gabbay	freddy.gabbay@gmail.com	Technion Israel Institute of Technology
Marco Solieri	marco.solieri@unimore.it	University of Modena and Reggio Emilia
Sandro Pinto	sandro.pinto@dei.uminho.pt	University of Minho
Ozgur Sinanoglu	ozgursin@nyu.edu	New York University Abu Dhabi
Rafail Psiakis	rafail.psiakis@tii.ae	Technology Innovation Institute
Jari Lukkarila	jari.lukkarila@tii.ae	Technology Innovation Institute
Kanad Basu	kanad.basu@utdallas.edu	University of Texas at Dallas
Mohamed Hassan	mohamed.hassan@mcmaster.ca	McMaster University
Andrea Acquaviva	andrea.acquaviva@unibo.it	University of Bologna
Andrea Bartolini	a.bartolini@unibo.it	University of Bologna
Francesco Barchi	francesco.barchi@unibo.it	University of Bologna
Emanuele Parisi	emanuele.parisi@unibo.it	University of Bologna
Davide Rossi	davide.rossi@unibo.it	University of Bologna
Suresh Sugumar	suresh.sugumar@tii.ae	Technology Innovation Institute
Luca Benini	luca.benini@unibo.it	University of Bologna
Marko Bertogna	marko.bertogna@unimore.it	University of Modena and Reggio Emilia
Shreekant (Ticky) Thakkar	shreekant.thakkar@tii.ae	Technology Innovation Institute

Technology Innovation Institute LLC
P.O. Box 9639
Abu Dhabi, UAE

tii.ae