RISC-V Security Model (non-normative)

RISC-V Security Model Task Group

Version 0.1, 11/2023: This document is in development. Assume everything can change. See http://riscv.org/spec-state for details.

Table of Contents

Preamble

This document is in the Development state

Assume everything can change. This draft specification will change before being accepted as informative, so implementations made to this draft specification will likely not follow the future informative specification.

Copyright and license information

Contributors

This RISC-V specification has been contributed to directly or indirectly by (in alphabetical order): Ali Zhang, Andy Dellow, Carl Shaw, Colin O'Flynn, Dean Liberty, Dong Du, Deepak Gupta, Colin O'Flynn, Guerney Hunt, Luis Fiolhais, Manuel Offenberg, Markku Juhani Saarinen, Munir Geden, Mark Hill, Nicholas Wood, Paul Elliott, Ravi Sahita, Samuel Ortiz, Steve Wallach, Suresh Sugumar, Terry Wang, Victor Lu, Ved Shanbhogue, Yann Loisel

Chapter 1. Introduction

This specifications provides guidelines for how RISC-V systems can use RISC-V security building blocks to build secure systems for different use cases.

The guidelines are listed through a few example uses cases based on commonly used profiles. The guidelines are not intended to be exhaustive; it is expected that the principles described in the chosen examples are general enough to be applicable to other use cases as well. The examples may be extended over time as required.

1.1. Requirements and tracking

Where this specification makes formal recommendations, those are captured as trackable requirements using the following format:

ID#	Requirement
CAT_NNN	The `CAT` is a category prefix that logically groups the requirements and is followed by 3 digits - `NNN` - assigning a numeric ID to the requirement. The requirements use the key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" that are to be interpreted as described in RFC 2119 when, and only when, they appear in all capitals, as shown here. When these words are not capitalized, they have their normal English meanings.

A requirement or a group of requirements may be followed by non-normative text providing context or justification for the requirement. The non-normative text may also be used to reference sources that are the origin of the requirement.

Trackable requirements are intended for ease of reference across dependant specifications.

1.2. Relationship to external profiles

For the purpose of this specification, external profiles apply to existing ecosystems or segments, but do not generally mandate implementations or architectures. This specification does not aim to establish new profiles. Its main purpose is to provide guidelines for how RISC-V security building blocks can be used to build RISC-V products which can comply with existing profiles.

Some profiles cover some or all of:

- Security reference architectures and taxonomy
- Hardware and software security requirements
- Interfaces and programming models
- Protection profiles and certification programmes
- Reference firmware/software

Other profiles are focussed on processes and methodology.

Examples of external profiles include:

Profile	Description
Global Platforms (GP)	Trusted execution environments(TEE) and trusted firmware for mobile, connected clients, and IoT. Secure element (SE) for tamper resistant storage of and operations on cryptographic secrets. SESIP certification. globalplatform.org/
Platform Security Architecture (PSA)	Platform security requirements for connected devices. PSA Certified. www.psacertified.org/
Trusted computing group (TCG)	Trusted platform module (TPM) and Device identifier composition engine (DICE) for trusted platforms. TCG certification. trustedcomputinggroup.org/
Confidential computing consortium	Common principles and protocols for protecting data in use (confidential computing). confidentialcomputing.io/
NIST	Widely used US standards for security processes, protocols and algorithms. Examples for the purposes of this specification: NISTIR 8259 - IoT device cybersecurity capability SP800-207 - Zero Trust Architecture www.nist.gov/

This is not an exhaustive list, more examples can be found in the reference section of this specification.

Chapter 2. RISC-V security model overview

The aim of this chapter is to define common taxonomies and principles for secure systems as used in the rest of this and other RISC V specifications. It is divided into the following sections:

- Reference model
  Defines a set of generic hardware and software subsystems used in examples and use cases to describe secure systems.
- Adversarial model
  Defines common attack types on secure systems, and identifies RISC-V extensions which can aid mitigation.
- Ecosystem security objectives
  Defines common security features and functional guidelines, used to deploy trustworthy devices in an ecosystem.

2.1. Reference model

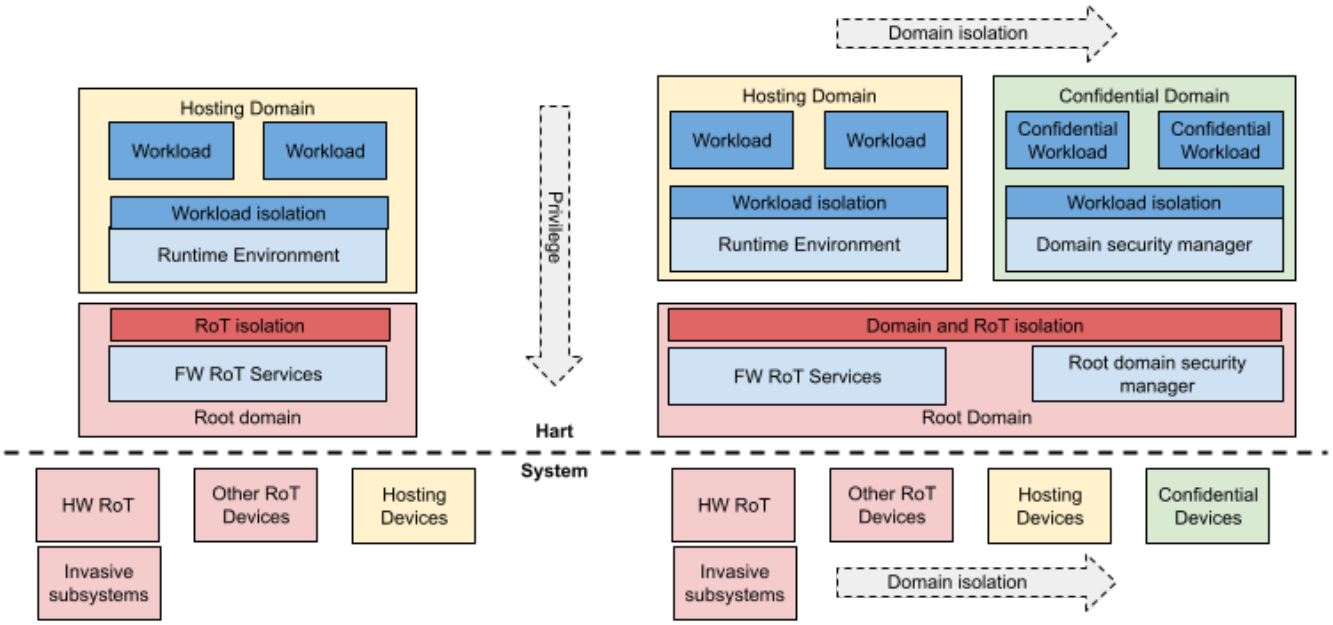

Figure 1: Generic security reference model

The figure above outlines a generic security reference model. It is not intended to describe any particular implementation and only aims to define a common taxonomy for the purpose of this and other RISC-V specifications.

Most systems are made up of different software components, often from different supply chains, each managing *assets* that need to be protected.

Examples of assets include:

- Cryptographic keys and credentials
- User data
- Proprietary models

- Secret algorithms

Assets can be protected by *isolation*. Isolation separates different software components on a Hart to protect *resources*:

- Memory
- Execution state

Examples of isolation mechanisms include:

- Privilege based isolation
  More privileged software is able to enforce security guarantees for less privileged software.
- Physical memory isolation
  More privileged software controls memory access for less privileged software.
- Domain isolation
  Separates resources so that a domain cannot access or modify resources assigned to a different domain without consent, regardless of privilege level.
- Virtualization
  Virtualization creates and manages *virtual resources* - compute, memory, devices - independent of actual physical hardware. A system, or individual domains, can be virtualized.

Isolation reduces dependencies between components, and reduces the amount of software that needs to be trusted, with the aim of minimizing the amount of software that needs to be trusted by a *workload* - the *trusted compute base (TCB)* of the workload.

The minimum amount of software that always has to be trusted on any system is its *root of trust (RoT)*. A RoT includes fundamental security services, for example:

- Boot and attestation
- Security lifecycle management
- Key derivations and sealing
- Security provisioning

Depending on use case and ecosystem requirements, a RoT can be:

- Hart firmware (FW RoT)
- A dedicated trusted subsystem (HW RoT), supporting a FW RoT

Using a HW RoT moves critical functions and assets off a Hart to a dedicated trusted subsystem, which can provide stronger protection against physical or logical attack than a complex Hart.

ID#	**Requirement**
CAT_NNN	A FW RoT SHOULD be the only software in machine mode (M).

In this document, the terms "FW RoT" and "HW RoT" will be used as defined above. The term "RoT" on its own can be used where a rule or a rational applies to

either model.

It is common for secure systems to support multiple trust chains and multiple roots of trust. For example, a TPM can be a root of trust for UEFI boot flows within a runtime environment, and a SIM can be a root of trust for user identity management within application security services. They are usually dedicated trusted subsystems with their own specifications and certification processes.

For the purpose of this document, these can be treated as secondary roots of trust. The HW RoT governs the security platform itself and acts as a primary root of trust on the system.

The full set of software a workload must trust is its *trusted compute base (TCB)*. A TCB can include other software. For example:

- On a vertically integrated embedded system: An operating system
- On a virtualized system: A hypervisor and a guest operating system
- In a data centre: A hypervisor and a guest OS, as well as other hosting services such as orchestration and server provisioning software

On complex systems the TCB can grow large and get difficult to certify and attest.

Domain isolation enables confidential workloads to be separated from complex hosting software, including other workloads. The TCB of a confidential workload can be reduced to a domain security manager in a confidential domain, and the RoT, while allowing the main runtime environment in a separate hosting domain to remain in control of resource management.

Domain isolation use cases include:

- Platform security services - for example: secure storage, user identity management, payment clients, DRM clients
- Hosted confidential third party workloads

Isolation policy needs to extend to devices:

- Physical memory access control for device initiated transactions
- Virtual memory translation for virtualized device transactions
- Interrupt management across privilege and domain boundaries

These policies can be enforced by system level hardware, controlled by Hart firmware.

Finally, *invasive subsystems* include any system or hart feature which could break security guarantees, either directly or indirectly. For example:

- External debug
- Event counters and performance monitoring
- Power and timing management

- RAS (*reliability, accessibility, serviceability*)

ID#	Requirement
CAT_NNN	Invasive subsystems MUST be controlled, or moderated, by a RoT.

2.2. Adversarial model

For the purpose of this specification, the main goal of an adversary is to gain unauthorized access to *resources* - memory, memory mapped devices, and execution state. For example, to access sensitive assets, to gain privileges, or to affect the control flow of a victim.

In general, adversaries capable of mounting the following broad classes of attacks should be considered by system designers:

- Logical
  The attacker and the victim are both processes on the same system.
- Physical
  The victim is a process on a system, and the attacker has physical access to the same system. For example: probing, interposers, glitching, and disassembly.
- Remote
  The victim is a process on a system, and the attacker does not have physical or logical access to the system. For example, radiation or power fluctuations, or protocol level attacks on connected services.

Attacks can be direct or indirect:

- Direct
  An adversary gains direct access to a resource belonging to the victim. For example: direct access to a memory location or execution state, or direct control of the control flow of a victim.
- Indirect
  An adversary can access or modify the content of a resource by a side channel. For example: by analyzing timing patterns of an operation by a victim to reveal information about data used in that operation, or launching row-hammer style memory attacks to affect the contents of memory owned by the victim.
- Chained
  An adversary is able to chain together multiple direct and indirect attacks to achieve a goal. For example, use a software interface exploit to affect a call stack, and use that to take redirect the control flow of a victim.

This specification is primarily concerned with ISA level mitigations against logical attacks.

Physical or remote attacks in general need to be addressed at system, protocol or governance level, and may require additional non-ISA mitigations. However, some ISA level mitigations can also help provide some mitigation against physical or remote attacks and this is indicated in the tables below.

The required level of protection can vary depending on use case. For example, a HW RoT may have stronger requirements on physical resistance than other parts of an SoC.

Finally, this specification does not attempt to rate attacks by severity, or by adversary skill level. Ratings tend to depend on use case specific threat models and requirements.

2.2.1. Logical

ID#	Attack	Type	Description	Current RISC-V mitigations	Planned RISC-V mitigations
CAT_NNN	Unrestricted access	Direct Logical	Direct access to unauthroized resources in normal operation.	• RISC-V privilege levels • RISC-V isolation (for example: PMP/sPMP, MTT, supervisor domains) • RISC-V hardware virtualization (H extension, MMU)	
CAT_NNN	Transient execution attacks	Chained Logical	Attacks on speculative execution implementations.	Known (documented) attacks except Spectre v1 are specific to particular micro-architectures, and RISC-V systems are not expected to be vulnerable to those. This is an evolving area of research. For example: Spectre and meltdown papers www.intel.com/content/www/us/en/developer/topic-technology/software-security-guidance/processors-affected-consolidated-product-cpu-model.html[Intel security guidance] developer.arm.com/documentation/#cf-navigationhierarchiesproducts=Arm%20Security%20Center,Speculative%20Processor%20Vulnerability[Arm speculative vulnerability]	Fence.t could mitigate against Spectre v1.

ID#	Attack	Type	Description	Current RISC-V mitigations	Planned RISC-V mitigations
CAT_NNN	Interface abuse	Chained Logical	Abusing interfaces across privilege or isolation boundaries, for example to elevate privilege or to gain unauthorized access to resources.	• RISC V privilege levels • RISC-V isolation	High assurance cryptography
CAT_NNN	Event counting	Direct Logical	For example, timing processes across privilege or isolation boundaries to derive information about confidential assets.	• Data-independent timing instructions • Performance counters restricted by privilege and isolation boundaries (sscofpmf, smcntrpmf)	
CAT_NNN	Redirect control flow	Chained Logical	Unauthorized manipulation of call stacks and jump targets to redirect a control flow to code controlled by an attacker.	• Shadow stacks (Zicfiss) • Landing pads (Zicfilp)	

2.2.2. Physical and remote

ID#	Attack	Type	Description	RISC-V recommendations
CAT_NNN	Analysis of physical leakage	Direct or indirect Physical or remote	For example, observing radiation, power line patterns, or temperature.	• Implement robust power management and radiation control • Data Independent Execution Latency (Zkt, Zvkt)

ID#	Attack	Type	Description	RISC-V recommendations
CAT_NNN	Physical memory manipulation	Direct Logical or physical	• Row-hammer type software attacks to manipulate nearby memory cells • Using NVDIMM, interposers, or physical probing to read, record, or replay physical memory • Physical attacks on hardware shielded locations to extract hardware provisioned assets	• Implement robust memory error detection, cryptographic memory protection, or physical tamper resistance • Supervisor domain ID, privilege level, or MTT attributes, could be used to derive memory encryption contexts at domain or workload granularity • Provide a degree of tamper resistance
CAT_NNN	Boot attacks	Chained Logical or physical	• Glitching to bypass secure boot • Retrieving residual confidential memory after a system reset	• Implement robust power management • Implement cryptographic memory protection with at least boot freshness
CAT_NNN	Subverting supply chains	Remote	Infiltration or collusion to subvert security provisioning chains, software supply chains and signing processes, hardware supply chains, attestation processes, development processes (for example, unfused development hardware or debug authorizations)	Deploy appropriate governance, accreditation, and certification processes for an ecosystem.

2.3. Ecosystem security objectives

Ecosystem security objectives identify a set of common features and mechanisms that can be used to enforce and establish trust in an ecosystem.

These features are defined here at a functional level only. Technical requirements are typically use case specific and defined by external certification programmes.

In some cases RISC-V non-ISA specifications can provide guidance or protocols. This is discussed more in use case examples later in this specification.

2.3.1. Secure identity

ID#	Requirement
CAT_NNN	A security platform MUST be securely identifiable

Identifies the immutable part of the security platform - immutable hardware, configurations, and firmware. Immutable components cannot change after completed security provisioning (see also security lifecycle management).

A *secure identity* is one capable of generating a cryptographic signature which can be verified by a remote party. Usually an asymmetric key pair, but sometimes symmetric signing schemes can be used). It is typically used as part of an attestation process.

Its scope and uniqueness depends on use case. For example:

- Unique to a system
- Shared among multiple systems with the same immutable security properties (group based anonymization)
- Anonymized using an attestation protocol supporting a third party anonymization service

It can be directly hardware provisioned, or derived from other hardware provisioned assets.

2.3.2. Security lifecycle

ID#	Requirement
CAT_NNN	A secure system MUST manage a security lifecycle.

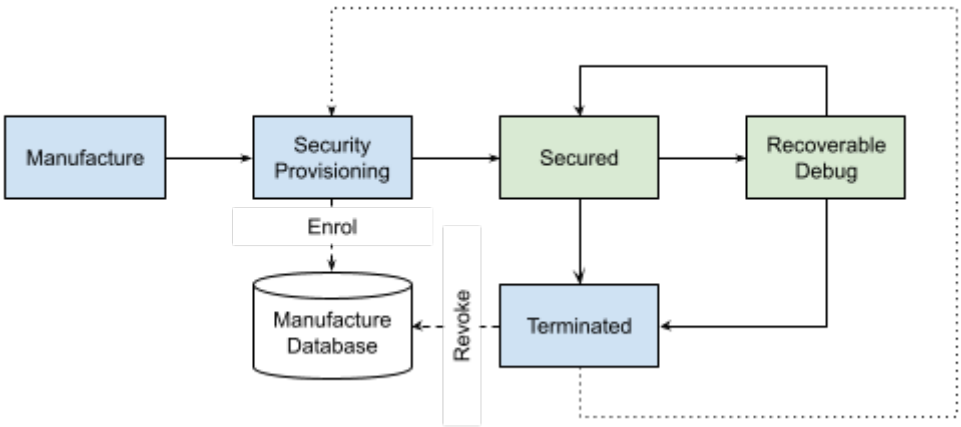

Figure 2: Generic security lifecycle

A security lifecycle reflects the trustworthiness of a system during its lifetime and reflects the lifecycle state of hardware provisioned assets.

It can be extended as indicated below to cover additional security provisioning steps such as device

onboarding, device activiation, user management, and RMA processes. These are use case or ecosystem specific and out of scope of this specification.

For the purpose of this specification, a minimum security lifecycle includes at least the following states:

- Manufacture - The system may not yet be fully locked down and has no hardware provisioned assets
- Security provisioning - The process of provisioning hardware provisioned assets
  Depending on ecosystem requirement, security provisioning could be performed in multiple stages through a supply chain and may require additional sub-states. These types of application specific extensions are out of scope of this specification.
- Secured - the system is fully locked down and has all its hardware provisioned assets
  Additional application specific provisioning stages can take place in this state - for example network onboarding and device activation, TSS/App/Device attestation or user identity management. This is out of scope of this specification.
- Recoverable debug - part of the system is in a debug state
  At least trusted security services or a hardware root of trust are not compromised, and hardware provisioned secrets remain protected.
  This state is both attestable and recoverable. For example, debug is enabled for a security domain without compromising another security domain or any trusted security services.
- Terminated - any system change which could expose hardware provisioned assets

  Typically hardware provisioned assets are made permanently inaccessible and revoked before entering this state. This also protects any derived assets such as attestation and sealing keys.

A system could support re-provisioning from a terminated state, for example following repair. This is equivalent to starting over from the security provisioning state and creates a new instance with a new secure identifier.

ID#	Requirement
CAT_NNN	Hardware provisioned assets MUST only be accessible while the system is in secured state, or a recoverable debug state.
CAT_NNN	Derived assets MUST only be available if a component is in secured state.

A derived asset in this context is any asset derived from hardware provisioned assets. For example attestation keys, or sealing keys for a supervisor domain.

2.3.3. Attestable services

For the purpose of this specification a confidential service can be any isolated component on a system. For example, a hosted confidential workload, or an isolated application security service.

ID#	Requirement
CAT_NNN	A confidential service, and all software and hardware components it depends on, MUST be attestable.

Attestation allows a remote reliant party to determine the trustworthiness of a confidential service before submitting assets to it.

- Verify the security state of a confidential service
- Verify the security state of all software and hardware a conidential service depends on
- Establish an attested secure connection to a confidential service

Attestation can be direct or layered.

- Direct
  The whole system can be defined by a single security platform attestation. For example, can be used in vertically integrated connected IoT devices and edge devices.
- Layered
  Enables parts of the attestation process to be delegated to lower privileged components.

Direct and layered attestation are discussed in more detail in use case examples later in this specification.

ID#	Requirement
CAT_NNN	A security platform attestation MUST be signed by a HW RoT, if present, or by trusted security services
CAT_NNN	A security platform attestation MUST be signed using a hardware provisioned (directly or derived) secure identity
CAT_NNN	A layered attestation MAY be signed by lower privileged software, itself attested by a security platform attestation
CAT_NNN	A layered attestation MUST be signed by a secure identity cryptographically bound (for example, hash locked) to a fresh security platform attestation

Care needs to be taken in attestation interface design. For example, software interfaces should only support either direct attestation or layered attestation workflows, never both, to prevent impersonation.

2.3.4. Authorized software

Running unauthorized software can compromise the security state of the system.

ID#	Requirement
CAT_NNN	A system in secured state MUST only load authorized software.

Two complementary processes can be used to authorize software:

- Measuring
  A measurement is a cryptographic fingerprint, such as a running hash of memory contents and launch state.
- Verification

Verification is a process of establishing that a measurement is correct (expected)

A boot process is typically layered, allowing software to be measured and verified in stages. Different measurement and verification policies can be employed at different stages. This is discussed further in use case examples later in this specification. The properties discussed below still apply to each stage.

Measurements can be calculated at boot (boot state), and sometimes also dynamically at runtime (runtime state). Measuring runtime state can be used as a robustness feature to mitigate against unauthorized runtime changes of static code segments. It is out of scope of this specification, though the principles discussed below can still be applied.

Verification can be:

- Local
  A measurement is verified locally on the device.
- Remote
  A measurement is verified by a remote provisioning service, or a remote reliant party.

Verification can be:

- Direct
  The measurement is directly compared with an expected measurement from a signed authorization.
- Indirect
  The measurement is included in derivations of other assets, for example sealing keys, binding assets to a measured state.

ID#	Requirement
CAT_NNN	A security platform MUST be measured.
CAT_NNN	A security platform MUST be verified, either directly or indirectly, before launching services which depend on the security platform.

Verification ensures the system has loaded authorized software

ID#	Requirement
CAT_NNN	A system MUST only use authorizations from trusted signers.

- Direct verification requires a signed image autorization from a trusted signer
  For example, a signed image header, or a separately signed authorization message.
- Indirect verification requires a signed authorization for migrating assets bound to a measured state
  For example, a signed authorization message, or a signed provisioning message.

Either way, only authorizations from trusted signers should be used. For example, from a list of hardware provisioned or securely discovered trusted signers.

ID#	Requirement
CAT_NNN	Local verification MUST be rooted in immutable boot code.

For example, ROM or locked flash, or rooted in a HW RoT itself rooted in immutable boot code.

2.3.5. System updates

Over time, any non-immutable component may need updates to address vulnerabilities or functionality improvements. A system update can concern software, firmware, microcode, or any other updatable component on a system.

ID#	Requirement
CAT_NNN	All components on a system which are not immutable MUST be updatable.

Immutable components include at least immutable boot code. Some trusted subsystems can also include immutable software to meet specific security certification requirements.

System updates are typically layered so that updates can target only parts of a system and not a whole system. The properties discussed below still apply to any system update.

ID#	Requirement
CAT_NNN	A system update MUST be measured and verified before launch.

See Section 2.3.4.

A system update can be:

- Deferred
  The update can only be effected after a restart of at least the affected component, and all of its dependents.
- Live
  The update can be effected without restarting any dependent components.

ID#	Requirement
CAT_NNN	Updates affecting a security platform SHOULD be deferred.
CAT_NNN	Updates MAY be live if live update capability, and suitable governance, is part of an already attested trust contract between a reliant party and the system.

A system update changes the attested security state of the affected component(s), as well as that of all other components that depend on it. It can affect whether a dependent confidential service is still considered trustworthy or not, as well as affect any derived assets such as sealing keys.

ID#	Requirement
CAT_NNN	System updates MUST be monotonic
CAT_NNN	System updates SHOULD be robust against update failures

Earlier versions may be carrying known vulnerabilities, or may affect the safe operation of a system in other ways.

For example, using derived anti-rollback counters (counter tree) rooted in a hardware monotonic counter.

A system can still support recovery mechanisms, with suitable governance, in the case of update failures. For example, a fallback process or a dedicated recovery loader.

Success criteria for a system update are typically use case or ecosystem specific and out of scope of this specification. Examples include local watchdog or checkpoints, and network control through a secure update protocol, and a dedicated recovery loader.

ID#	Requirement
CAT_NNN	System updates, and authorization messages, SHOULD only be received from trusted sources.

A system update is itself always verified before being launched. Verifying the source as well can mitigate against attempts to inject adversary controlled data into a local update process. Including into protected memory regions.

2.3.6. Isolation

Complex systems include software components from different supply chains, and complex integration chains with different roles and actors. These supply chains and integration actors often share mutual distrust:

- Developed, certified, deployed and attested independently
- Protected from errors in, or abuse from, other components
- Protected from debugging of other components
- Contain assets which should not be available to other components

Use cases later in this specification provide examples of RISC-V isolation models.

ID#	Requirement
CAT_NNN	Isolated software components SHOULD be supported

An isolated component has private memory and private execution contexts not accessible to other components.

ID#	Requirement
CAT_NNN	Devices MUST not access memory belonging to an isolated component without permission

Isolation can also extend to other features, such as interrupts and debug.

2.3.7. Data sealing

Sealing is the process of protecting confidential data on a system.

ID#	Requirement
CAT_NNN	Sealed data MUST only be accessible to an isolated component

Sealing can be:

- Local
  Local sealing binds assets to a local device (hardware unique sealing).
- Remote
  Remote sealing binds assets to credentials provided by a remote provisioning service following successful attestation.

Local sealing can be:

- Direct
  Direct sealing binds assets to sealing keys derived by trusted security services, or a HW RoT.
- Layered
  Layered sealing enables delegation of some sealing key derivations to lower privileged software.

ID#	Requirement
CAT_NNN	Valid local sealing keys SHOULD only be generated in secured state.
CAT_NNN	Valid local sealing keys MAY be generated in a trusted debug state for unaffected software components.

Sealing is discussed further in use cases examples later in this document.

Chapter 3. RISC-V security building blocks

This chapter outlines brief descriptions of RISC-V security building blocks discussed in this specification, together with general guidelines and links to technical specifications.

See also the reference use cases chapter of this specification for common examples of how RISC-V security building blocks can be combined.

3.1. Isolation

Isolation enable separation of software components executing on a Hart, as well as device assignment. RISC-V enables:

- Privilege based isolation
- Physical memory access control (hart and device-initiated accesses)
- Virtual memory management (hart and device virtualization)
- Hypervisor extension
- Supervisor domains

3.1.1. Privilege levels

Privileged ISA

Standard privilege levels - Machine mode (M), Supervisor mode (S), and User mode (U) - enable separation of more privileged software from less privileged software.

3.1.2. Hypervisor extension

Privileged ISA

Hypervisor extension supports standard supervisor level hypervisors. It extends S mode into Hypervisor-extended supervisor mode (HS), and a virtual supervisor mode (VS) for guests. It also extends U mode into standard user mode (U) and virtual user mode (VU).

Isolation of guests is enforced using two-stage address translation and protection. Two-stage address translation and protection is in effect in VS and VU modes.

Alternatively sPMP can be used instead of MMU to support static partition hypervisors, for example on systems with hard and deterministic real time requirements [Note -The sPMP for Hypervisor extension has not been specified yet].

MMU, PMP/ePMP, and sPMP are discussed later in this chapter.

3.1.3. PMP and ePMP

Privileged ISA

Physical memory protection (PMP) enables M-mode to access-control physical memory for supervisor or HS modes.

ID#	Requirement
CAT_NNN	PMP configurations MUST only be directly accessible to machine mode

Individual access controlled regions can be locked until the next system reset to create temporal isolation boundaries, such as protecting immutable boot code.

ePMP extends PMP protection by allowing machine mode to restrict its own access to memory allocated to lower privilege levels. This can be used to mitigate against privilege escalation attacks, for example.

ID#	Requirement
CAT_NNN	If PMP is supported then ePMP MUST be supported.

3.1.4. sPMP

github.com/riscv/riscv-spmp

Supervisor PMP (sPMP) enables supervisor mode to control physical memory access for U mode.

sPMP allows supervisor mode to restrict its own access to memory allocated to lower privilege levels. This can be used to mitigate against privilege escalation attacks, for example.

When combined with H-extension, sPMP can be nested so that the hypervisor can control memory allocations to its guests, and each guest can control its own memory allocations to its workloads.

ID#	Requirement
CAT_NNN	Either sPMP, 1st-stage or G-stage page tables MUST be used to protect Supervisor domain H/S-mode from lower privilege levels.

3.1.5. MMU

Privileged ISA

Memory management unit (MMU) enables address translation and protection for:

- Isolating an OS from workloads on a system without H-extension (one-stage translation)
- Isolating a hypervisor from a guest, on a system with H-extension (two-stage translation)

ID#	Requirement
CAT_NNN	Either PMP/ePMP or MTT MUST be used to protect M-mode from lower privilege levels.

3.1.6. Supervisor domains

github.com/riscv/riscv-smmtt

Supervisor domains allow software components on the same hart to be developed, certified, deployed and attested independently of each other.

A supervisor domain is a compartment above M-mode, physically isolated - memory, execution state, and devices - from other supervisor domains regardless of privilege level (below M-mode). Isolation and context switching between supervisor domains are managed by M-mode firmware.

ID#	Requirement
CAT_NN	Either PMP/ePMP or MTT MUST be used to enforce physical memory isolation boundaries for supervisor domains, and to protect machine mode from any supervisor domain.

PMP can be used for more static and deterministic use cases.

MTT can be used where more fine grained dynamic resource management across supervisor domain boundaries is required.

ID#	Requirement
CAT_NNN	A system supporting supervisor domains MUST support supervisor domain extensions for interrupts (Smsdia) and performance counters (TBD), and SHOULD support supervisor domain extensions for external debug (Smsdsd TBD).

Interrupts: github.com/riscv/riscv-smmtt
External debug: github.com/riscv-non-isa/riscv-external-debug-security
Performance counters:

These extensions enable management of interrupts, external debug, and performance counters across supervisor domain boundaries. M-mode firmware should context switch hart HPM event/counters to manage isolation of performance counters:

- External debug can be enabled for one supervisor domain without affecting other supervisor domains
- M-mode firmware manage interrupt routing and preemption across supervisor domain boundaries
- M-mode firmware can ensure that performance counters cannot be used by software in one supervisor domain to measure operations in other supervisor domains

3.1.7. MTT

github.com/riscv/riscv-smmtt

The *memory tracking table (MTT)* is a memory structure managed by machine mode, tracking memory ownership across supervisor domains. It is designed to enable fine grained dynamic

memory management across supervisor domain boundaries, with policy typically set by a hypervisor in a hosting domain responsible for resource management.

ID#	Requirement
CAT_NNN	Either PMP/ePMP or MTT MUST be used to protect M-mode from lower privilege levels
CAT_NNN	MTT configurations MUST only be directly accessible to machine mode

3.1.8. IOPMP

github.com/riscv-non-isa/iopmp-spec

IOPMP is a system level component providing physical memory access control for device-initiated transactions, complementing PMP and sPMP rules.

ID#	Requirement
CAT_NNN	A system which supports PMP/ePMP, or sPMP, MUST implement IOPMP for device access control.
CAT_NNN	IOPMP configurations MUST only be directly accessible to machine mode.

3.1.9. IOMTT

github.com/riscv/riscv-smmtt

IOMTT is a system level component providing physical memory access control for device-initiated transactions, complementing MTT rules.

ID#	Requirement
CAT_NNN	A system which supports MTT MUST implement IOMTT for access-control for device-initiated memory accesses.
CAT_NNN	IOMTT configurations MUST only be directly accessible to machine mode.
CAT_NNN	A system which implements IOMTT MAY also implement IOPMP to access-control device-initiated access to M-mode memory.

3.1.10. IOMMU

github.com/riscv-non-isa/riscv-iommu

IOMMU is a system level component providing virtual memory access-control for device-initiated transactions, complementing MMU translation rules.

ID#	Requirement
CAT_NNN	Systems supporting MMU SHOULD also support IOMMU

ID#	Requirement
CAT_NNN	Systems supporting IOMMU MUST also enforce physical memory access control for M-mode memory against device-initiated transactions (e.g. via IOMTT, IOPMP or equivalent).

3.2. Software enforced memory tagging

github.com/riscv/riscv-j-extension

Memory tagging (MT), is a technique which can improve the memory safety of an application. A part of the effective address of a pointer can be masked off, and used as a tag indicating intended ownership or state of a pointer. The tag can be used to track accesses across different regions, as well as protecting against pointer misuse such as "use after free". The pointer masking should use the proposed J-extension pointer masking extension (Smpm, Smnpm, Ssnpm).

With software based memory tagging the access rules encoded in tags are enforced by software (compiler).

See also hardware enforced memory tagging below.

3.3. Control flow integrity

github.com/riscv/riscv-cfi

Control-flow Integrity (CFI) capabilities help defend against Return-Oriented Programming (ROP) and Call/Jump-Oriented Programming (COP/JOP) style control-flow subversion attacks, where an attacker attempts to modify return addresses or call/jump address to redirect a victim to code reused by the attacker.

These attack methodologies use code sequences in authorized modules, with at least one instruction in the sequence being a control transfer instruction that depends on attacker-controlled data either in the return stack or in memory used to obtain the target address for a call or jump. Attackers stitch these sequences together by diverting the control flow instructions (e.g., JALR, C.JR, C.JALR), from their original target address to a new target via modification in the return stack or in the memory used to obtain the jump/call target address.

Risc-V provides two defenses:

- Shadow stacks (Zicfiss) - protect return addresses on call stacks
- Labeled Landing pads (Zicfilp) - protect target addresses in jumps and branches

3.4. Cryptography

github.com/riscv/riscv-crypto

Risc-V includes ISA extensions in the following cryptographic areas:

- Scalar cryptography

- Vector cryptography
- Entropy source (scalar)

Risc-V cryptographic extensions are aimed at supporting efficient acceleration of cryptographic operations at ISA level. This can both help reduce the TCB of an isolated component, and avoid hardware bottlenecks (for example, system level cryptographic subsystems).

The entropy source extension provides an ISA level interface to a hardware entropy source. Entropy source requirements can depend on use case or ecosystem specific requirements and Risc-V does not provide any entropy source technical specification. But the entropy source ISA specification does contain general recommendations and references.

ID#	Requirement
CAT_NNN	Risc-V systems SHOULD support either scalar or vector cryptographic ISA extensions
CAT_NNN	The entropy source ISA extension MUST be supported if either scalar or vector cryptographic ISA extensions are supported.

It is not necessary to support both scalar and vector operations, as a scalar operation can be viewed as a vector of size 1.

3.5. Roadmap

3.5.1. Capability based architecture

- Cheri
- Capstone

3.5.2. Hardware enforced memory tagging

Hardware enforced memory tagging extends software based memory tagging (see above) by moving enforcement of tagged rules to hardware.

3.5.3. HFI

Hardware-assisted fault isolation (HFI) aims to provide lightweight in-process isolation to mitigate against errors in one process compromising other processes within the same workload.

3.5.4. Lightweight isolation

- TBD

3.5.5. System level isolation

- WorldGuard

3.5.6. Cryptography enhancements

- PQ
- High assurance computing (HAC)

Chapter 4. Use case examples

4.1. Basic non-virtualized system

4.1.1. Overview

4.1.2. Secure and Verified Boot

4.1.3. Isolation model

4.1.4. Device access control

4.1.5. Sealing

4.1.6. Attestation

4.2. Basic virtualized system

4.2.1. Overview

4.2.2. Isolation model

4.2.3. Device access control

4.2.4. Sealing

4.2.5. Attestation

4.3. Global Platforms TEE

4.3.1. Overview

4.3.2. Isolation model

4.3.3. Device access control

4.3.4. Sealing

4.3.5. Attestation

4.4. Confidential computing on RISC-V (CoVE)

4.4.1. Overview

4.4.2. Measured Boot

4.4.3. Isolation model

4.4.4. Physical threat protection

4.4.5. Device access control

4.4.6. Trusted device assignment

4.4.7. Sealing

4.4.8. Attestation

4.4.9. Debug, QoS and Performance Monitoring

4.5. Additional examples

(Variations on the above)

Android pKVM

Chapter 5. Cryptography

RISC-V supports a number of ISA-level extensions aimed at improving performance for cryptographic operations (scalar and vector). They also include an ISA-level entropy source, and guidelines for data independent execution latency.

See cryptography
See github.com/riscv/riscv-crypto

Current ISA level cryptographic extensions work at round level. With the data independent execution latency properties, they can provide some mitigation against some side-channel attacks, such as cache timing attacks. They may not defend fully against some differential power analysis, for example.

Work is on-going to define ISA-level *high assurance cryptography (HAC)*. This work includes defining full-round operations to increase side-channel resistance; adding operations supporting *post-quantum cryptography (PQC)*; and adding ISA-level privilege-based key management.

Cryptographic requirements depend on target ecosystem, as well as on varying regulatory requirements in different geographic regions. This chapter summarizes commonly used cryptographic guidance for secure systems, provided as guidance for development of RISC-V specifications and RISC-V based secure systems.

5.1. PQC readiness

Quantum safe cryptography is an evolving area of research. For example, see: csrc.nist.gov/projects/post-quantum-cryptography.

NIST has already standardized three PQ algorithms: Kyber (FIPS-203), Dilithium (FIPS-204), and SPHINCS+ (FIPS-205). Kyber is used for general encryption, and Dilihtium and SPHINCS+ are used for digital signatures.

RISC-V systems and specifications must at least support a migration path towards use of PQC.

ID#	Requirement
CAT_NNN	All immutable components MUST use quantum safe cryptography
CAT_NNN	All mutable components MUST at least have a migration path to quantum safe cryptography.

For example, immutable boot code must use a quantum safe algorithm for verification of a mutable boot stage, as an immutable boot stage cannot be updated.

Mutable stages can be updated, and can provide a migration path to quantum safe cryptography. For example, system designers should consider protocols, governance, and storage requirements for upgrading hardware provisioned assets to PQC versions.

5.2. Cryptographic algorithms and guidelines

The following resources provide general cryptographic guidance applicable to most western jurisdictions: csrc.nist.gov/Projects/Cryptographic-Standards-and-Guidelines www.cnss.gov/CNSS/issuances/Memoranda.cfm

In particular, for most new systems:

- Public identifier: 512 bits (for example, hash of a public key)
- Counter used as identifier: 64 bits
- Block cipher: AES-256
- Hash function: SHA-512, or SHA-3
- Message authentication: HMAC-SHA-512
- Asymmetric signing/encryption: RSA-3072, or ECC-384 (see PQC readiness)

Some legacy use cases may require use of other algorithms, such as SHA-256 or AES-128. In these cases, wherever possible, an upgrade path should be supported. For example, allocating sufficient storage to accommodate larger sizes in future updates.

Some use cases, such as cryptographic memory protection, may sometimes use specialized algorithms for performance in a constrained use case. These are not discussed here but should have similar properties to the ones listed above, but with different trade-offs.

For Chinese markets, equivalent *ShangMi (SM)* algorithm support is required. In particular:

- SM2: Authentication (ECC based)
- SM3: Hash function (256-bit)
- SM4: Block cipher

See gmbz.org.cn/main/index.html

RISC-V cryptographic ISA extensions also include support for ShangMi algorithms (SM3 and SM4)

Some Shang-Mi algorithms are also described in ISO specifications.

Other specific markets also require regional cryptographic algorithms, for example Russian Ghost. RISC-V cryptographic ISA extensions currently do not directly support Russia specific algorithms.

Appendix A: References

1. https://www.intel.com/content/www/us/en/newsroom/opinion/zero-trust-approach-architecting-silicon.html
2. https://www.forrester.com/blogs/tag/zero-trust/
3. https://docs.microsoft.com/en-us/security/zero-trust/
4. https://github.com/riscv/riscv-crypto/releases
5. https://github.com/riscv/riscv-platform-specs/blob/main/riscv-platform-spec.adoc
6. https://www.commoncriteriaportal.org/files/ppfiles/pp0084b_pdf.pdf
7. https://docs.opentitan.org/doc/security/specs/device_life_cycle/
8. https://nvlpubs.nist.gov/nistpubs/ir/2021/NIST.IR.8320-draft.pdf
9. https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-193.pdf
10. https://www.rambus.com/security/root-of-trust/rt-630/
11. https://docs.opentitan.org/doc/security/specs/
12. https://trustedcomputinggroup.org/work-groups/dice-architectures/
13. https://ieeexplore.ieee.org/iel7/8168766/8203442/08203496.pdf
14. https://dl.acm.org/doi/10.1145/168619.168635
15. https://dl.acm.org/doi/abs/10.1145/3342195.3387532
16. https://github.com/riscv/riscv-debug-spec/blob/master/riscv-debug-stable.pdf
17. https://csrc.nist.gov/csrc/media/events/non-invasive-attack-testing-workshop/documents/08_goodwill.pdf
18. https://www.iso.org/standard/60612.html
19. https://ieeexplore.ieee.org/document/6176671
20. https://tches.iacr.org/index.php/TCHES/article/view/8988
21. https://ieeexplore.ieee.org/abstract/document/1401864
22. https://www.electronicspecifier.com/products/design-automation/increasingly-connected-world-needs-greater-security
23. https://www.samsungknox.com/es-419/blog/knox-e-fota-and-sequential-updates
24. https://docs.microsoft.com/en-us/windows/security/threat-protection/intelligence/supply-chain-malware
25. https://dl.acm.org/doi/10.1145/3466752.3480068
26. https://arxiv.org/abs/2111.01421
27. https://www.nap.edu/catalog/24676/foundational-cybersecurity-research-improving-science-engineering-and-institutions
28. https://trustedcomputinggroup.org/work-groups/dice-architectures/

Bibliography

- [1] The RISC-V Instruction Set Manual Volume II: Privileged Architecture Document Version 20211203 (link)