# RISC-V Security Model (non-normative)

RISC-V Security Model Task Group

Version 0.3, 05/2024: 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 (editor), Paul Elliott, Ravi Sahita (co-editor), Robin Randhawa, Samuel Ortiz, Steve Wallach, Suresh Sugumar, Terry Wang, Victor Lu, Ved Shanbhogue, Yann Loisel

# Chapter 1. Introduction

This specification provides guidelines for building secure RISC-V systems using RISC-V security building blocks. It is aimed at developers of RISC-V technical specifications, as well as at designers of secure RISC-V systems.

A few example use cases are provided, which are based on commonly used security deployment models. These are not intended to be exhaustive but are common enough to represent a wide range of deployments of secure products. They are accompanied by use case specific security guidelines which are intended to help readers implement secure products for their specific use cases.

The examples may be extended over time as required.

The examples are not definitions of formal Protection Profiles (See: csrc.nist.gov/glossary/term/protection_profile). Formal protection profiles are typically provided by third party certification bodies for different ecosystems. The guidelines provided within the examples in this specification are intended to help readers adapt RISC-V security features to meet security requirements of commonly used third party protection profiles.

RISC-V is currently not intending to create a security certification programme. This specification is provided as non-normative guidance for developing secure RISC-V systems which are certifiable within existing third party security certification programmes. As such, there is no RISC-V proof of concept or RISC-V testing associated with this specification.

This specification does not contain threat modelling or security assessment of individual RISC-V technical specifications. Individual RISC-V technical specifications are expected to use the Security Model as a guide to develop their own specific security analysis, including formal threat modeling where appropriate. For this purpose, all guidelines in this document are labelled to enable referencing from other specifications. Specific security analysis in the context of a RISC-V technical specification may require testing and a proof of concept as per normal RISC-V development processes for RISC-V technical specifications.

Security is an evolving area where new use cases and new threats can emerge at any time. This specification represents the RISC-V security model and best practice as of the date of publication of this document.

New versions of this document may be developed and released as and when required.

## 1.1. Requirements and tracking

This is a non-normative specification. However, the specification makes formal recommendations where appropriate. Those are expressed as trackable requirements using the following format:

| ID# | Requirement |
|---|---|
| SR_CAT_NNN | The `SR_CAT` is a "Security Requirement CATegory" prefix that logically groups the requirements (e.g. SR_UPD denotes security requirements related to updates, and SR_ATT denotes security requirements related to attestation) and is followed by 3 digits - `NNN` - assigning a numeric ID to the requirement.<br><br>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 dependent specifications.

## 1.2. Relationship with external protection profiles

To assist with articulating security guidelines relevant to the included examples, this specification references external protection profiles as appropriate. Such references are not intended to mandate any specific implementations, but to provide guidelines on how RISC-V security building blocks may be used to comply with those protection profiles.

Typically, protection profiles cover some or all of:

- Security reference architectures and taxonomy
- Hardware and software security requirements
- Interfaces and programming models
- Reference firmware/software
- Certification programs
- Misc Processes and methodology

The following are examples of some external protection profiles that are referenced by this specification:

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

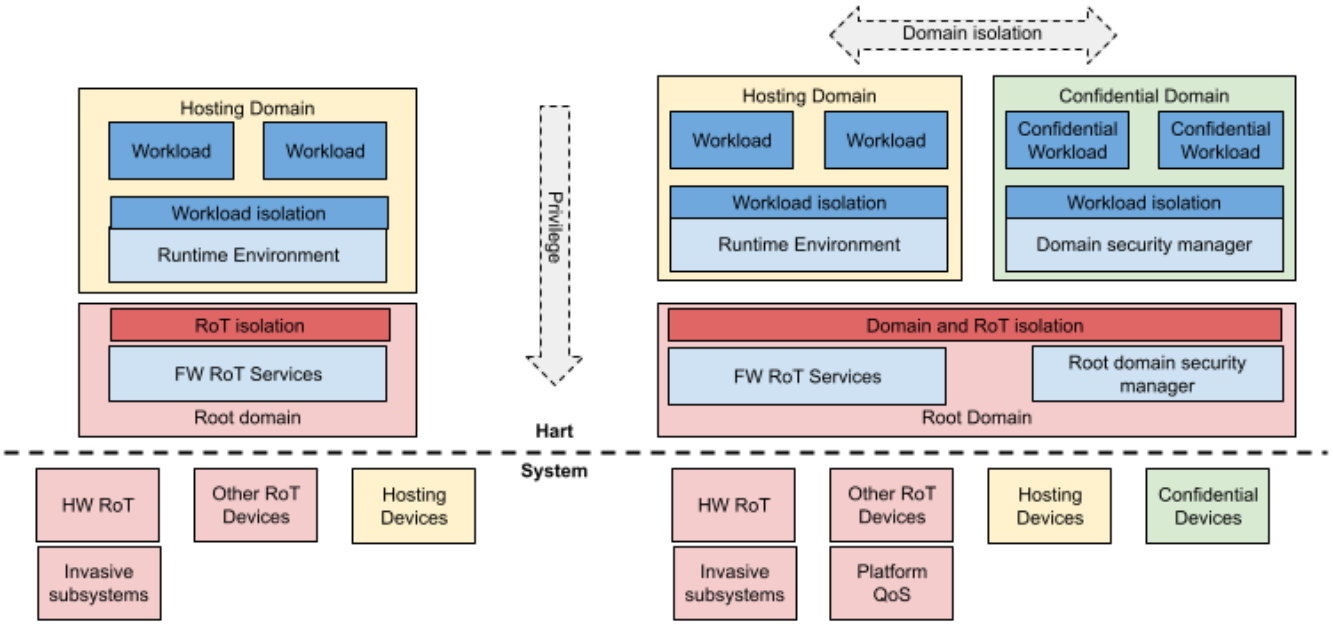


*Figure 1: Generic security reference model*

The figure above outlines a generic security reference model and taxonomy. This specification is oriented around this reference model. The model is not tied to any particular implementation.

Most systems are built with multiple software components each managing *assets* that need to be protected. These components are often sourced from multiple different supply chains. The security reference model uses the generic term *domain* to identify a logically isolated region, with at least some private resources and execution state not accessible to at least some other domain(s). The figure above uses two domains for illustrative purposes but some use-cases can require more. This specification does not imply or limit the number of domains or the type of use cases a RISC-V system can support. See Use case examples for more comprehensive and specific examples.

Logically isolated domains at the software level are typically reflected in logically isolated domains at the system hardware level, where hardware resources can be assigned or restricted to specific software domains. For example, device DMA transfers can be restricted to memory assigned to a particular domain.

### 2.1.1. Assets

Examples of assets include:

- Cryptographic keys and credentials
- User data
- Proprietary models
- Secret algorithms

In this specification, a *hardware provisioned asset* is an immutable asset provisioned in hardware by a security provisioning process, before a device is used in a production environment. For example, hardware provisioned keys or identities.

### 2.1.2. Trusted Computing Base (TCB)

The *Trusted Computing Base (TCB)* of any system function is the totality of protection mechanisms within a computer system, including hardware, firmware, and software - the combination responsible for enforcing a security policy.

The following are examples of software components that can be a part of some function's TCB:

- An operating system
- A hypervisor and a guest operating system
- A TEE security manager
- Hosting services such as orchestration and server provisioning software

### 2.1.3. Root of trust

A *root of trust (RoT)* is the foundation on which all secure operations of a system depend. A RoT is typically a combination of a minimal amount of hardware and software that has to be implicitly trusted by all system components.

A RoT supports fundamental security services, for example:

- Boot and attestation
- Security life cycle management
- Key derivations and sealing (sealing is defined in a later section)
- Security provisioning

Depending on use case and ecosystem requirements, a RISC-V 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 and possibly isolated trusted subsystem, which can provide stronger protection against physical and logical attacks.

The HW RoT acts as a *primary root of trust* on the system.

*It is common for secure systems to support multiple trust chains with their own root of trust. For example, a TPM can be a root of trust for UEFI boot flows within a runtime environment while a SIM can be a root of trust for user identity management.*

*For the purpose of this document, these should be treated as secondary roots of trust.*

*The HW RoT can manage the security life cycle of a secondary root of trust (booting etc).*

| ID# | Requirement |
|---|---|
| SR_ROT_001 | A complex secure system MUST implement a HW RoT |
| SR_ROT_002 | A simpler system MAY not require a HW RoT but one is still recommended |

An example of a complex system is one with multiple, out-of-order, cache-coherent harts.

An example of a simpler system is one with a single, in-order, hart such as in a microcontroller.

*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 rationale applies to either model.*

### 2.1.4. Isolation

Assets can be protected by *isolation*. Isolation reduces dependencies between components, and reduces the amount of software that needs to be trusted.

Isolation protects *resources*:

- Memory and memory mapped devices
- *Execution state*, including Hart register 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
  Software in one domain cannot access or modify resources assigned to a different domain (without consent), regardless of privilege level.
  (Higher privileged software in one domain cannot access resources assigned to a lower privileged software in a different domain)
- Virtualization
  Virtualization creates and manages *virtual resources* - compute, memory, devices - independent of actual physical hardware. A system, or individual domains, can be virtualized.

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.

Examples of confidential workloads include:

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

### 2.1.5. Device assignment

Isolation policy needs to extend to device assignment:

- 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.

### 2.1.6. Invasive subsystems

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

- External debug
- Power and timing management
- RAS (*reliability, accessibility, serviceability*)

| ID# | Requirement |
|---|---|
| SR_INV_001 | Invasive subsystems MUST be controlled, or moderated, by a RoT. |
| SR_INV_002 | Invasive subsystems SHOULD be enabled separately for M-mode & non-M-mode software. |
| SR_INV_003 | Invasive subsystems SHOULD be enabled separately for individual domains |

### 2.1.7. Event counters

Event counters are commonly used for performance management and resource allocation on systems.

However, they can pose a security risk. For example, a workload can maliciously attempt to infer another workload's secrets by monitoring that other workload's operation. The victim workload can be at the same, lower, or higher privilege than the malicious workload.

| ID# | Requirement |
|---|---|
| SR_PMU_001 | Lower privileged software MUST NOT be able to monitor higher privileged software. |
| SR_PMU_002 | Software in one domain MUST NOT be able to monitor software in a different domain, without consent. |

### 2.1.8. Platform quality of service

More complex systems, such as server platforms, can provide *platform quality of service (QoS)* features beyond the capabilities of basic event counters. Platform QoS features include any Hart and system hardware and firmware aimed at managing access to shared physical resources across workloads while

minimizing contention.

For example:

- Memory bandwidth management
- Cache allocation policies across workloads, including workload prioritization
- Hart allocation policies across workloads

These types of features rely on monitoring the resource utilization of workloads, similar to event counters, and on the optimization of resource allocation policies.

| ID# | Requirement |
|---|---|
| SR_QOS_001 | Lower privileged software MUST NOT be able to observe QoS events or attributes concerning higher privileged software. |
| SR_QOS_002 | Software in one domain MUST NOT be able to observe QoS events or attributes concerning a different domain, without consent. |

### 2.1.9. Denial of service

The RISC-V security model is primarily concerned with protection of assets. It is not concerned with providing service guarantees.

For example, a hosting environment is free to apply its own resource allocation policy to relevant workloads. This can include denying service to some workloads.

| ID# | Requirement |
|---|---|
| SR_DOS_001 | Lower privileged software MUST NOT be able to deny service to higher privileged software, or other isolated workloads at the same privilege level. |
| SR_DOS_002 | Software in one domain SHOULD NOT be able to deny service to software in a different domain |

Higher privileged software must always be able to enforce its own resource management policy without interference, including scheduling, resource assignment and revocation policies.

Similarly, a hosting domain owning resource allocation and host management across a system normally has to be able to enforce its own policies across domains. Including denying service. But other domains should not be able to deny service to the hosting domain, or to other domains.

## 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.

At an implementation level there can be further distinctions, for example the degree of proximity required to execute a remote or a physical attack as defined above. However, this document does not make any finer grained distinctions other than logical, physical and remote.

Attacks can be direct, indirect or chained:

- Direct
  An adversary gains direct access to a resource belonging to the victim. For example: direct access to the victim's memory or execution state, or direct control of the victim's control flow.
- Indirect
  An adversary can use a side channel to access or modify the content of a resource owned by the victim. 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, using a software interface exploit to affect the call stack such that control flow is redirected to the adversary's code.

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 can 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 could 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 |
|---|---|---|---|---|---|
| SR_LGC_001 | Unrestricted access | Direct Logical | Unauthorized direct access to resources in normal operation. | • RISC-V privilege levels<br>• RISC-V isolation (for example: PMP/ePMP, sPMP, MTT, supervisor domains)<br>• RISC-V hardware enforced virtualization (H extension, MMU) | CHERI |

| ID# | Attack | Type | Description | Current RISC-V mitigations | Planned RISC-V mitigations |
|---|---|---|---|---|---|
| SR_LGC_002 | Transient execution attacks | Chained Logical | Attacks on speculative execution implementations. | Known (documented) attacks, except Spectre v1, are specific to particular micro-architectures. Micro-architecture for RISC-V systems is implementation specific, but must not introduce such vulnerabilities.<br><br>This is an evolving area of research.<br><br>For example:<br>Spectre and meltdown papers<br>Intel security guidance<br>Arm speculative vulnerability | Fence.t, or similar future extensions, may at least partially mitigate against Spectre v1. |
| SR_LGC_003 | 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<br>• RISC-V isolation | High assurance cryptography |
| SR_LGC_004 | Event counting | Direct Logical | For example, timing processes across privilege or isolation boundaries to derive information about confidential assets. | • Data-independent timing instructions<br>• Performance counters restricted by privilege and isolation boundaries (sscofpmf, smcntrpmf) | |
| SR_LGC_005 | 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)<br>• Landing pads (Zicfilp) | CHERI |
| SR_LGC_006 | Memory safety | Logical | Unauthorized access to resources within an isolated component. For example, pointer or allocation errors (temporal memory safety), or buffer overflows (spatial memory safety). | RISC-V pointer masking (J-extension)<br>Shadow stacks (Zicfiss)<br>Landing pads (Zicfilp)<br><br>Memory safe programming, for example:<br>https:/www.cisa.gov/sites/default/files/2023-12/CSAC_TAC_Recommendations-Memory-Safety_Final_20231205_508.pdf | Architectural sandboxing, such as HFI.<br>Capability based architecture, such as CHERI. |

### 2.2.2. Physical and remote

| ID# | Attack | Type | Description | RISC-V recommendations |
|---|---|---|---|---|
| SR_PHY_001 | Analysis of physical leakage | Direct or indirect<br>Physical or remote | For example, observing radiation, power line patterns, or temperature. | • Implement robust power management and radiation control<br>• Data Independent Execution Latency (Zkt, Zvkt) |

| ID# | Attack | Type | Description | RISC-V recommendations |
|---|---|---|---|---|
| SR_PHY_002 | Physical memory manipulation | Direct<br>Logical or physical | • Row-hammer type software attacks to manipulate nearby memory cells<br>• Using NVDIMM, interposers, or physical probing to read, record, or replay physical memory<br>• Physical attacks on hardware shielded locations to extract hardware provisioned assets | • Implement robust memory error detection, cryptographic memory protection, or physical tamper resistance<br>• Supervisor domain ID, privilege level, or MTT attributes, may be used to derive memory encryption contexts at domain or workload granularity<br>• Provide a degree of tamper resistance |
| SR_PHY_003 | Boot attacks | Chained<br>Logical or physical | • Glitching to bypass secure boot<br>• Retrieving residual confidential memory after a system reset | • Implement robust power management<br>• Implement cryptographic memory protection with at least boot freshness |
| SR_PHY_004 | 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 programs.

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 |
|---|---|
| SR_IDN_001 | A secure platform MUST be securely identifiable |

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

A *secure identity* is an element capable of generating a cryptographic signature which can be verified by a remote party. This is usually an asymmetric key pair, but symmetric signing schemes can also be used. Secure identities are typically used as part of an attestation process.

A secure identity's scope and uniqueness is use case dependent. For example, a secure identity can be:

- 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

A secure identity can be directly hardware provisioned, or derived from other hardware provisioned assets.

### 2.3.2. Security life cycle

| ID# | Requirement |
|---|---|
| SR_LFC_001 | A secure system MUST manage a security life cycle. |

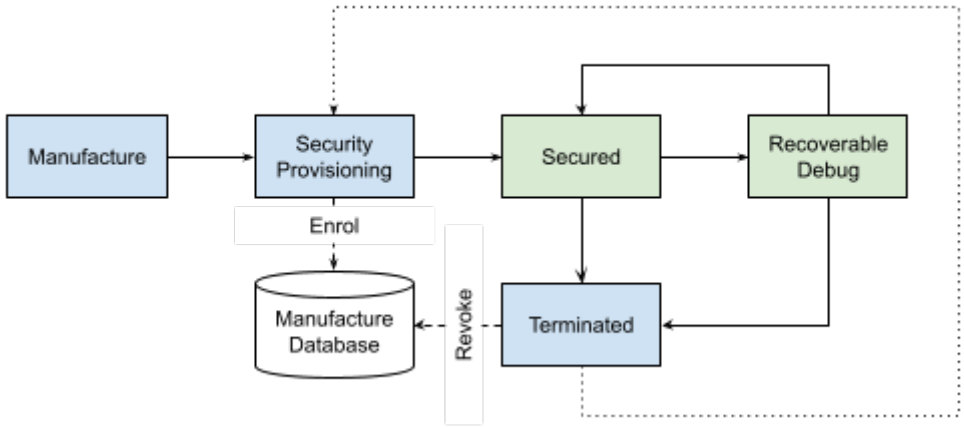


*Figure 2: Generic security life cycle*

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

It can be extended as indicated below to cover additional security provisioning steps such as device onboarding, device activation, user management, and RMA (Return Merchandize Authorization) processes. These are use case or ecosystem specific and out of scope of this specification.

For the purpose of this specification, *revealing debug* includes any HW or FW debug capability which:

- Could break security guarantees or could expose assets
- Is not part of an attested trust contract with a relying party

Examples of revealing debug include revealing logging, external debug or boundary scans, dedicated debug builds of software components, or enabling self-hosted debug for a component.

Depending on use case, an attested software component can include debug capabilities managed through an ecosystem defined governance process - *trusted debug*. For example, self-hosted debug enabled following an ecosystem specific authorization process. In this case the debug capability, and the associated governance, is part of the trust contract with a relying party.

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

- Manufacture - The system may not yet be locked down and has no hardware provisioned assets

- Security provisioning - The process of provisioning hardware provisioned assets
  Depending on ecosystem requirement, security provisioning may 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 - hardware provisioned assets are locked (immutable), only authorized software can be used, and revealing debug is not enabled.
  Additional specific provisioning stages can take place in this state - for example network onboarding and device activation, App/Device attestation or user identity management. This is out of scope of this specification.
- Recoverable debug - part of the system is in a revealing debug state
  At least the RoT is not compromised and hardware provisioned secrets remain protected.
  This state is both attestable and recoverable. For example, revealing debug is enabled for a domain without compromising another domain or any RoT 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 may support re-provisioning from a terminated state, for example following repair/RMA. This can be viewed as equivalent to starting over from the security provisioning state, and creates a new instance with a new secure identifier.

| ID# | Requirement |
|---|---|
| SR_LFC_002 | Hardware provisioned assets MUST only be accessible while the system is in secured state, or a recoverable debug state. |
| SR_LFC_003 | Derived assets MUST only be available if a component is in secured state. |

For example, returning garbage or some known test and debug value when attempting to read a hardware provisioned asset, unless the system is in a secured state, or a recoverable debug state. Derived assets would then also become unavailable in these states, though test and debug versions may be available.

| ID# | Requirement |
|---|---|
| SR_LFC_004 | Hardware provisioned assets MUST only be accessible while the system is in a secured state, or a recoverable debug state (with the recoverable debug state in attestation evidence). |
| SR_LFC_005 | 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.

| ID# | Requirement |
|---|---|
| SR_LFC_006 | Revealing debug MUST be reflected in attestation. |

*Attestable states* are ones where the RoT and hardware provisioned assets are not compromised by debug and a valid attestation can be generated reflecting that state:

- Secured
- Recoverable debug

In other states the system is not able to generate a valid attestation key. It is still *indirectly attestable* as

any generated attestation will not be signed correctly and can be rejected by a relying party.

Trusted debug is part of a trust contract with a relying party and is application specific. The presence of trusted debug can be determined indirectly by a relying party through other attested properties, for example measurements.

### 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 |
|---|---|
| SR_ATT_001 | A confidential service, and all software and hardware components it depends on, MUST be attestable. |

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

- Verify the security state of a confidential service
- Verify the security state of all software and hardware a confidential 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. Eg : 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 |
|---|---|
| SR_ATT_002 | A secure platform attestation MUST be signed by a HW RoT, if present, or else by a FW RoT |
| SR_ATT_003 | A secure platform attestation MUST be signed using a hardware provisioned (directly or derived) secure identity |
| SR_ATT_004 | A layered attestation MAY be signed by lower privileged software, itself attested by a security platform attestation |
| SR_ATT_005 | Layered attestations MUST be cryptographically bound such that a relying party can determine that they:<br>• Were generated on the same system<br>• Are fresh. |

*Software interfaces should only support either direct attestation or layered attestation workflows, never both, to prevent impersonation attacks.*

### 2.3.4. Authorized software

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

| ID# | Requirement |
|---|---|
| SR_AUT_001 | A system in secured or recoverable debug states MUST only load authorized software. |
| SR_AUT_002 | A system in security provisioning state SHOULD only load authorized software. |

Two complementary processes can be used to authorize software:

- Measurement
  In the context of this document, a measurement is a record of a present state of the system, which can be used by a remote party to verify the security state of the system. It is typically a cryptographic fingerprint, such as a running hash of memory combined with security lifecycle state and other attributes. Although depending on use case other kinds of measurements can be used.
- 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 relying 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 |
|---|---|
| SR_MSM_001 | A secure platform MUST be measured. |
| SR_MSM_002 | A secure 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 |
|---|---|
| SR_MSM_003 | A system MUST only use authorizations from trusted authority. |

- Direct verification requires a signed image authorization from a trusted authority before loading

an image
For example, a signed image, or a separately signed authorization message.

- Indirect verification requires a signed authorization from a trusted authority for migrating assets bound to a previously measured state
For example, a signed provisioning message.

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

| ID# | Requirement |
|---|---|
| SR_MSM_004 | 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 mutable 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 |
|---|---|
| SR_UPD_001 | 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 |
|---|---|
| SR_UPD_002 | 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 |
|---|---|
| SR_UPD_003 | Updates affecting a security platform SHOULD be deferred. |
| SR_UPD_004 | Updates MAY be live if live update capability, and suitable governance, is part of an already attested trust contract between a relying 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 |
|---|---|
| SR_UPD_005 | System updates MUST be monotonic |
| SR_UPD_006 | System updates SHOULD be robust against update failures |

Earlier versions could be carrying known vulnerabilities, or could be able to 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 |
|---|---|
| SR_UPD_007 | 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 |
|---|---|
| SR_ISO_001 | Isolated software components SHOULD be supported |

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

| ID# | Requirement |
|---|---|
| SR_ISO_002 | 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. Sealing

Sealing is the process of protecting confidential assets on a system, typically using sealing keys derived in different ways for different use cases as discussed in this section. For example, from a hardware provisioned root key, from a boot state (measurements, security life cycle state), or provisioned at runtime by a remote provisioning system.

Sealing can be:

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

| ID# | Requirement |
|---|---|
| SR_SLG_001 | Sealed assets SHOULD only be possible to unseal in a secured state |

For example, local sealing key derivations should take the security life cycle state of the system into account. And remote sealing key provisioning should always attest the system before releasing unsealing credentials or keys.

Local sealing can be:

- Direct
  Direct sealing binds assets to sealing keys derived by a RoT.
- Layered
  Layered sealing enables delegation of some sealing key derivations to lower privileged software.

| ID# | Requirement |
|---|---|
| SR_SLG_002 | Locally sealed assets MUST only be possible to unseal on the same physical instance of a system that they were sealed on. |

For example, using sealing keys derived from a hardware provisioned *hardware unique key (HUK)*.

| ID# | Requirement |
|---|---|
| SR_SLG_003 | Locally sealed assets bound to a boot measurement MUST only be possible to unseal if that measurement has not changed, or the system has received an authorized update. |

See system updates

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 enables access restrictions on software components executing on a hart, as well as on device accesses. 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

**See section 1.2 (Privilege Levels) of the Privileged ISA specification.**

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

**See chapter 8 (Hypervisor Extension) of the Privileged ISA specification.**

The 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 partitionining 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/Smepmp, and sPMP are discussed later in this chapter.

### 3.1.3. PMA

**See section 3.6 (Physical Memory Attributes) of the Privileged ISA specification.**

*Physical memory attributes (PMA)* are intended to capture inherent properties of the underlying hardware. For example, read-only ROM regions, or non-cachable device regions. Often PMA can be fixed at design time or at boot, but sometimes runtime PMA can be required.

A separate hardware checker - *PMA checker* - enforces PMA rules at runtime once a physical address is known. PMA rules are always checked on every physical access, and typically configured by region.

### 3.1.4. PMP

**See section 3.7 (Physical Memory Protection) of the Privileged ISA specification.**

*Physical memory protection (PMP)* enables M-mode to access-control physical memory for supervisor and U modes (with or without H-extension).

| ID# | Requirement |
|---|---|
| SR_PMP_001 | 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.

### 3.1.5. Smepmp

**See the PMP Enhancements for memory access and execution prevention on Machine mode specification.**

Smepmp 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.

| ID# | Requirement |
|---|---|
| SR_PMP_002 | If PMP is supported then Smepmp MUST be supported. |

### 3.1.6. sPMP

**See the RISC-V S-mode Physical Memory Protection (SPMP) specification.**

*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 |
|---|---|
| SR_PMP_003 | If MMU is not supported, sPMP SHOULD be used to protect S-mode from lower privilege levels. |

### 3.1.7. MMU

**See sections 4.3 to 4.6 (Page-Based Virtual-Memory Systems) of the Privileged ISA specification.**

*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 |
|---|---|
| SR_MMU_001 | Either PMP/Smepmp or MTT MUST be used to protect M-mode from lower privilege levels. |
| SR_MMU_002 | if the Sv extension is supported then 1st-stage page tables MUST used to protect the S-Mode Supervisor domain from accesses made by U-Mode. |
| SR_MMU_003 | if the H extension is supported 1st-stage and/or G-stage page tables MUST used to protect Supervisor domain H/S-mode from lower privilege levels. |

### 3.1.8. MTT

See the RISC-V Supervisor Domains Access Protection specification.

The *memory tracking table (MTT)* is a memory structure managed by machine mode that is used to track 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 |
|---|---|
| SR_MTT_001 | Either PMP/Smepmp or MTT MUST be used to protect M-mode from lower privilege levels |
| SR_MTT_002 | MTT configurations MUST only be directly accessible to machine mode |

*The M-Mode resident software responsible for managing context switches and communication between supervisor domains is called the Root Domain. An MTT can be sufficient for protecting the Root Domain by enabling M-mode to ensure that its own resources are never assigned to any another domain. The use of PMP/Smepmp will add yet further protections for M-mode, such as the ability to implement temporal isolation boundaries within M-mode (to protect early boot code, for example), or to prevent itself from accessing or executing from memory assigned to lower privilege levels (privilege escalation).*

### 3.1.9. Supervisor domains

See the RISC-V Supervisor Domains Access Protection specification.

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

A supervisor domain is an S-Mode compartment that is physically isolated from other supervisor domains. The memory, execution state and devices belonging to a supervisor domains are isolated from other supervisor domains. This isolation of supervisor domains and the context switching between them is managed by M-mode firmware.

A supervisor domain is identified at an architecture level by a *supervisor domain id (SDID)* CSR, managed by M-mode firmware.

| ID# | Requirement |
|---|---|
| SR_SUD_001 | PMP/Smepmp 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 |
| --- | --- |
| SR_SUD_002 | A system supporting supervisor domains MUST support supervisor domain extensions for interrupts (Smsdia) and SHOULD support supervisor domain extensions for external debug (TBD). |

#### Supervisor domains: Interrupts, External debug and Performance counters

See chapter 6 (Smsdia) of the RISC-V Supervisor Domains Access Protection specification.

See the RISC-V External Debug Security Extension specification.

Browse the tech-privileged mailing list for relevant background on 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

### 3.1.10. IOPMP

See the RISC-V IOPMP specification.

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

| ID# | Requirement |
| --- | --- |
| SR_IOP_001 | A system which supports PMP/Smepmp, or sPMP, MUST implement IOPMP for device access control unless the system supports IOMTT.<br><br>Depending in system design, IOMTT can enforce the same access control policies as IOPMP. |
| SR_IOP_002 | IOPMP configurations MUST only be directly accessible to machine mode. |

*IOPMP defines multiple "models" for different system configurations. Unless specified differently in the use cases in this specification, system designers are free to choose any IOPMP model.*

### 3.1.11. IOMTT

See the RISC-V Supervisor Domains Access Protection specification.

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

| ID# | Requirement |
|---|---|
| SR_IOM_001 | A system which supports MTT MUST implement IOMTT for access-control for device-initiated memory accesses. |
| SR_IOM_002 | IOMTT configurations MUST only be directly accessible to machine mode. |
| SR_IOM_003 | A system which implements IOMTT MAY also implement IOPMP to access-control device-initiated access to M-mode memory. |

*IOMTT can also be sufficient for protecting Root devices in the sense that M-mode can enforce that its own resources are never assigned to another domain. Use of IOPMP or similar still adds further protections. For example, a system may require that Root devices are not able to access memory assigned to TEE domain.*

### 3.1.12. IOMMU

See the RISC-V IOMMU specification.

IOMMU is a system level component performing memory address translation from IO Virtual Addresses to Physical Addresses thereby allowing devices to access virtual memory locations. It complements the MMU.

| ID# | Requirement |
|---|---|
| SR_IOM_004 | Systems supporting MMU SHOULD also support IOMMU |
| SR_IOM_005 | Systems supporting IOMMU MUST also enforce physical memory access control for M-mode memory against device-initiated transactions (IOMTT or IOPMP). |

## 3.2. Software enforced memory tagging

See the RISC-V Pointer Masking specification.

*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 the 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". Pointer masking implementations should use the proposed RISC-V pointer masking extension (Smmpm, Smnpm, Ssnpm).

With software based memory tagging the access rules encoded in tags are enforced by software, such as the compiler and the application runtime.

See also hardware enforced memory tagging below.

## 3.3. Control flow integrity

See the RISC-V Control Flow Integrity specification.

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

**See the RISC-V Cryptography Extension specification.**

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 the ISA level. This can both help reduce the TCB of an isolated component and also 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. However, the entropy source ISA specification does contain general recommendations and references.

| ID# | Requirement |
|---|---|
| SR_CPT_001 | RISC-V systems SHOULD support either scalar or vector cryptographic ISA extensions |
| SR_CPT_002 | 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.

### 3.4.1. Post quantum cryptography

**See the RISC-V Specification for Post-quantum Cryptography specification.**

The *RISC-V Post Quantum Cryptography* initiative aims to specify ISA extensions that enhance performance and implementation efficiency for contemporary public-key cryptography, with a focus on standard Post-Quantum Cryptography algorithms like Kyber, Dilithium, and others. The ISA design and evaluation prioritize the requirements of real-world networked devices, ensuring that the Post-Quantum Cryptography (PQC) extensions effectively complement existing scalar and vector cryptography extensions.

### 3.4.2. High assurance cryptography

*See the RISC-V Specification for High Assurance Cryptography

The High Assurance Cryptography task group will create instruction set extensions (ISEs) that facilitate higher levels of assurance than the existing Scalar and Vector Crypto ISEs. One initial focus will be on full-rounds vector AES extensions that allow (do not prevent) effective side-channel resistant implementations and that may perform better than the existing round-based instructions, with future work on other algorithms. A second intimately related focus area will be ISEs that manage secret keys — not restricted to just AES keys — in ways that better protect them from unauthorized users and from side-channel analysis.

## 3.5. Capability based architecture

### 3.5.1. CHERI

See the RISC-V Specification for CHERI Extensions specification.

CHERI - an ISA technique that uses capability-based memory protection for spatial and temporal memory safety, compartmentalization, and control-flow enforcement. Source code has to be recompiled to capture memory safety properties inherent in the source language.

# Chapter 4. Use case examples

This chapter provides a selection of non-exhaustive example security usecases based on commonly used deployment security models. The examples may be extended over time as required.

## 4.1. Generic system without supervisor domains

### 4.1.1. Overview

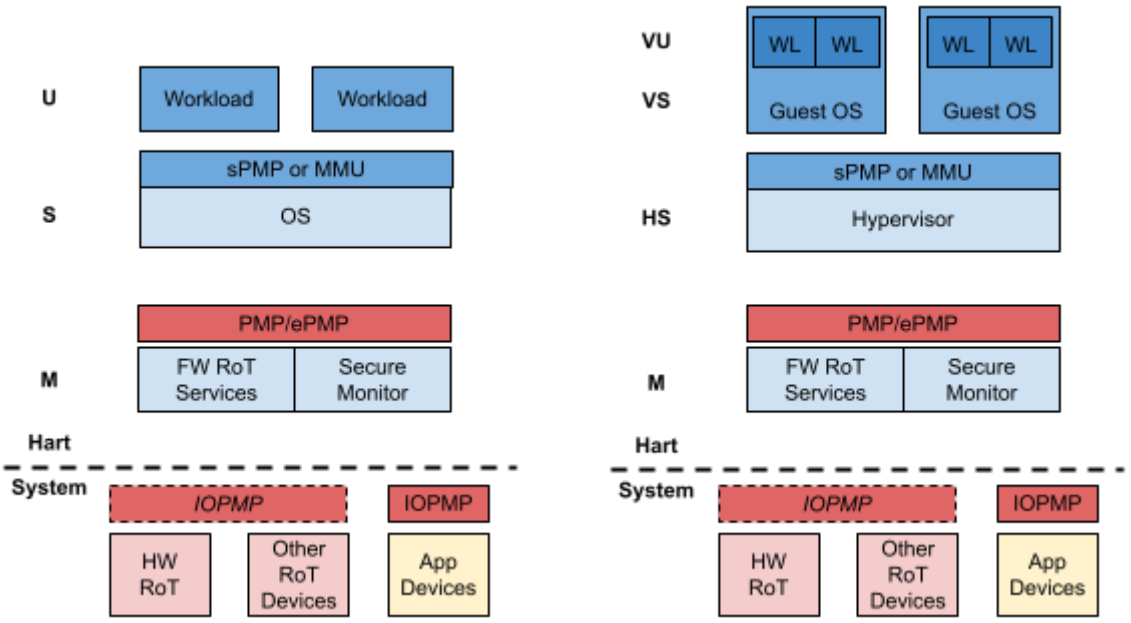


*Figure 3: Generic vertically integrated system*

A generic vertically integrated system can be either virtualized or non-virtualized.

M-mode hosts a FW RoT. An OS or a Hypervisor in S or HS mode controls applications or guests. Guests and applications execute in U or VS/VU modes and trust the OS or Hypervisor to provide isolation guarantees.

A system level HW RoT is recommended.

*The RISC-V architecture also caters for systems with just M and U modes, commonly used in embedded systems, helper cores, and similar use cases. On secure systems not supporting S-mode a FW RoT has to share M-mode with an OS. RISC-V does not exclude such implementations - for example, implementations using a certified OS and FW RoT, or using a HW RoT to isolate sensitive code and assets (physical isolation). There is no current mechanism in RISC-V for isolation within M-mode itself other than temporal boundaries*

*To minimize the TCB of the FW RoT RISC-V recommends that secure systems implement S mode, and deprivilege non-RoT firmware such as an OS or non-security services.*

### 4.1.2. Isolation model

| ID# | Requirement |
|---|---|
| SR_GEN_001 | PMP/Smepmp, MUST be used to isolate M-mode from lower privilege levels. |

| ID# | Requirement |
| --- | --- |
| SR_GEN_002 | sPMP or MMU MUST be used to protect Supervisor mode from lower privilege levels (with or without H-extension) |

See PMP and Smepmp

See sPMP

See MMU

The sPMP is typically used in relatively static deployments such as determinism sensitive use cases like automotives.

The MMU is typically required for Linux based systems and for virtualized systems.

Either MMU or sPMP can be used both with or without the hypervisor extension. For example, the hypervisor extension with sPMP can support static partitioning hypervisors commonly used in automotive. A single stage MMU can be used without the hypervisor extension for full Linux support.

### 4.1.3. Root of Trust

See reference model.

| ID# | Requirement |
| --- | --- |
| SR_GEN_004 | A secure system SHOULD implement a HW RoT |

### 4.1.4. Authorized Boot

Multiple models can be used to ensure a secure system can only run authorized software.

See authorized software.

### 4.1.5. Attestation

Multiple models can be used to prove to a relying party that a secure system is in a trustworthy state.

See attestable services.

### 4.1.6. Sealing

Multiple models can be used to protect assets if a system is not in a trustworthy state.

See sealing.

### 4.1.7. Device access control

For the purpose of this specification, a device can be a logical device. A physical device can present one or more logical devices, each with its own (logical) control interface.

Isolation guarantees provided to software also apply to device initiated transaction.

| ID# | Requirement |
|---|---|
| SR_GEN_005 | IOPMP or IOMTT MUST be used to guarantee that devices assigned to lower privilege levels cannot access resources assigned to M-mode. |
| SR_GEN_006 | IOPMP, or IOMTT with IOMMU, MUST be used to enforce access rules for devices assigned to user applications or guests on a virtualized system. |

On a non-virtualized system, user devices can be managed by the OS which can enforce access rules for user applications.

On a virtualized system, devices can be virtualized and assigned to guests by the hypervisor configuring MMU and IOMMU translation rules.

*IOMTT can also be sufficient for protecting Root devices in the sense that M-mode can enforce that its own resources are never assigned to another domain. Use of IOPMP or similar could still add further protections. For example, a system may require that Root devices cannot access memory assigned to Confidential domain.*

### 4.1.8. Debug and performance management

See security life cycle.
See enhanced RISC-V external debug security

| ID# | Requirement |
|---|---|
| SR_GEN_007 | External debug MUST only be enabled by HW RoT (M-mode external debug) or by FW RoT (non M-mode external debug). |
| SR_GEN_008 | External debug SHOULD be enabled separately for M-mode & non-M-mode software. |
| SR_GEN_009 | Self-hosted debug MAY be used for debug of non M-mode software. |
| SR_GEN_010 | Self-hosted debug MUST only be enabled by a higher privileged component. |

For example, external debug can be enabled for non-M-mode software without affecting M-mode (recoverable debug). And an S-mode OS can enable self-hosted debug for a user application without affecting other applications or S-mode itself.

| ID# | Requirement |
|---|---|
| SR_GEN_011 | FW RoT MAY disable self-hosted debug for all non M-mode software. |

For example, disable self-hosted debug in a production system for certification reasons.

| ID# | Requirement |
|---|---|
| SR_GEN_012 | External debug MUST only be enabled following system reset (part of measuring) of the affected component, moderated by a RoT. |
| SR_GEN_013 | Revealing self-hosted debug MUST only be enabled following reboot (part of measuring) of the affected component. |
| SR_GEN_014 | Trusted self-hosted debug MAY be enabled at runtime (after measuring) of the affected component, to an application specific governance process. |

Guarantees the system remains attestable.

| ID# | Requirement |
|---|---|
| SR_GEN_015 | Lower privileged software MUST NOT be able to monitor higher privileged software. |

| ID# | Requirement |
|---|---|
| SR_GEN_016 | Software in one domain MUST NOT be able to monitor software in a different domain, without consent. |

Prevents using event counters to monitor across application or privilege boundaries. Event counters can be managed by higher privileged software as part of context switching across boundaries.

## 4.2. Global Platform TEE

### 4.2.1. Overview

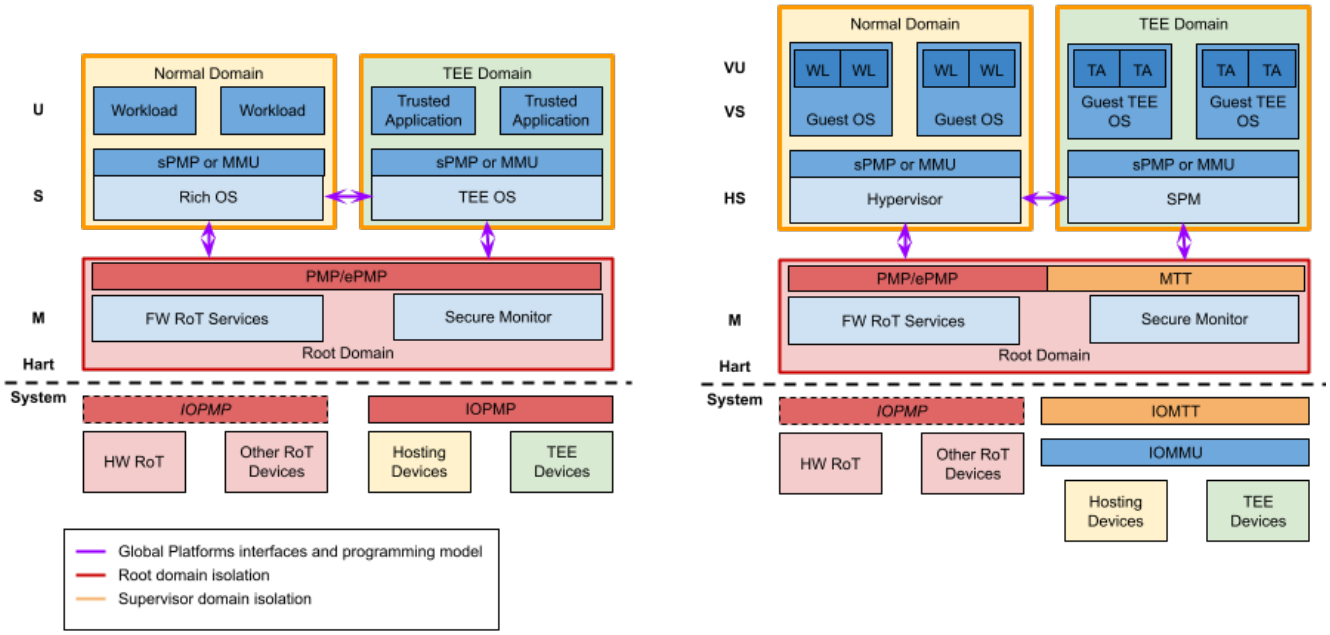


*Figure 4: Global platform TEE use cases*

Global platform defines technical standards, interface specifications and programming models, open source firmware, and certification programs for *trusted execution environments (TEE)*.

A TEE is an isolated environment providing security services. TEE services can be available to software on multiple Harts. For example:

- Payment clients
- DRM clients and content protection
- Secure storage
- User identity management
- Attestation services

The TEE model divides software into physically isolated domains:

- Normal domain
  Typically hosting a *rich OS* (for example, RTOS or Linux), and user applications.
- TEE domain
  Hosts a *TEE OS* (domain security manager) and *trusted applications (TA)*.
- Root domain

Hosts RoT firmware, including a secure monitor.

The TEE OS is primarily responsible for isolation of TA, and for providing root of trust services, within the TEE domain.

The OS in Normal domain typically controls scheduling on the system, across all Harts available to it. To interact with TA services in TEE domain, the OS in Normal domain interacts with a TEE OS through a secure monitor in Root domain.

The secure monitor is responsible for context switching and isolation across domain boundaries, including event management.

For the purpose of this specification, TEE deployment models can be separated as:

- Static partitioning TEE
  A single TEE provides security services to Normal domain. TA are typically installed at boot by RoT FW and TEE OS, though Global Platform does also define protocols for installation of TA at runtime. System configuration and resource allocation can be mostly static, making the system more deterministic.

  *Use case examples:* edge devices and IoT, automation, and automotive.
- Virtualized TEE
  On a virtualized system, TEE can also be virtualized. In this case a *secure partition manager* in TEE domain is responsible for isolation of multiple TEE guests (for example, an OEM TEE and separate third party TEE). This model can also support more dynamic resource allocation.

  *Use case examples:* mobile clients, and automotive.

### 4.2.2. Isolation model

A Global Platform TEE requires the following isolation guarantees:

| ID# | Requirement |
|---|---|
| SR_TEE_001 | Root domain MAY access resources assigned to any domain, but SHOULD prevent itself from unintended access to resources assigned to a different domain (privilege escalation). |
| SR_TEE_002 | No other domains can access resources assigned to Root domain |
| SR_TEE_003 | Resources assigned to TEE domain MUST NOT be accessible to Normal domain |
| SR_TEE_004 | Resources assigned to Normal domain MUST be accessible to Normal domain (r/w/x), and to TEE domain (r/w) (default sharing rule) |
| SR_TEE_005 | Resources assigned to a single TA, or a guest TEE, MUST not be accessible by a different TA, or guest TEE, without consent. |

In the standard GP TEE model, each TA is expected to be a self-contained unit providing a specific security service, either to Normal domain or to other TA. All communications are implemented through secure channels managed by the TEE OS or SPM.

Sharing of memory between TA is generally discouraged. But there are mechanisms to do so in specific use cases. For example, sharing media buffers in a secure media path. Such policies are enforced by SPM or TEE OS.

Processes in Normal domain can share memory assigned to Normal domain when interacting with a TA in TEE world (default sharing rule). Such shared memory can be cached when context switching

between Normal and TEE domains.

RISC-V hardware enforced isolation mechanisms can be used as follows to meet those guarantees:

See supervisor domains. See PMP and Smepmp See sPMP See MMU See MTT

| ID# | Requirement |
|---|---|
| SR_TEE_006 | PMP/Smepmp, or MTT, MUST be used to isolate Root domain from other domains. |
| SR_TEE_007 | Supervisor domains MUST be used to enforce isolation between Normal and TEE domains. |

| ID# | Requirement |
|---|---|
| SR_TEE_008 | For a static partition TEE, sPMP or MMU MUST be used to enforce isolation between TA in TEE domain. |

| ID# | Requirement |
|---|---|
| SR_TEE_009 | For a virtualized TEE, hypervisor extension MUST be supported |
| SR_TEE_010 | For a virtualized TEE, sPMP or MMU MUST be used to enforce isolation between guest TEE, and between TA within a TEE. |

### 4.2.3. Root of Trust

See reference model.

| ID# | Requirement |
|---|---|
| SR_TEE_011 | A TEE based system SHOULD implement a HW RoT |

### 4.2.4. Authorized boot

See authorized software.

TEE boot is typically based on:

- Measured and verified local boot (direct or indirect)
- Sealing, to protect TEE production assets

The process can involve multiple stages (layered boot).

### 4.2.5. Attestation

See attestable services.

Static partition TEE attestation is typically based on a direct security platform attestation.

| ID# | Requirement |
|---|---|
| SR_TEE_012 | A direct security platform attestation MUST cover at least:<br>• TEE domain<br>• Root domain<br>• Boot state of all trusted subsystems |

Virtualized TEE attestation can be layered, for performance or separation of concern. For example:

- A security platform attestation, signed by a RoT, covering trusted subsystems, Root domain, and SPM
- Separate guest TEE attestation(s) signed by SPM

### 4.2.6. Sealing

See sealing.

In the Global Platform security model, SPM or TEE OS typically provide local trusted storage, key management, and cryptographic services to TA and guest TEE. These services support local sealing of TA or guest TEE assets, and minimize exposure of cryptographic materials.

| ID# | Requirement |
|---|---|
| SR_TEE_013 | Local sealing for a TA, or a TEE guest, MUST be unique to TEE domain and to a physical instance of a system. |
| SR_TEE_014 | Local sealing for a TA, or a TEE guest, SHOULD also be unique to the TEE guest or the TA. |
| SR_TEE_015 | Local sealing MAY be layered. |

For example:

- TEE domain unique sealing keys derived by a RoT from a hardware unique key
- TA, or guest TEE, unique sealing keys derived by TEE OS or SPM from a TEE domain unique sealing key

### 4.2.7. Device access control

For the purpose of this specification, a device can be a logical device. A physical device can present one or more logical devices, each with its own (logical) control interface.

The security guarantees also apply to device initiated accesses, for example DMA and interrupts.

| ID# | Requirement |
|---|---|
| SR_TEE_016 | A static partition TEE MUST use IOPMP to enforce access rules for devices. |
| SR_TEE_017 | A virtualized TEE MUST use IOMTT and IOMMU to enforce access rules for devices assigned to Normal or TEE domains, and SHOULD use IOPMP to enforce access rules for Root devices. |

For a static partition TEE, domain level granularity can be sufficient as device access within TEE and Normal domains is governed by TEE OS and the rich OS respectively. It can be implemented using IOPMP. Policy can be controlled by boot configuration, by a HW or FW RoT.

For a virtualized TEE, IOMTT enforces supervisor domain level access rules (physical isolation). IOMMU enforces guest and TA level access rules (virtualization), supporting device assignment to a guest TEE or a TA.

*IOMTT can also be sufficient for protecting Root devices in the sense that M-mode can enforce that its own resources are never assigned to another domain. Use of IOPMP or similar could still add further protections. For example, a system may require that Root devices cannot be used to access memory assigned to Confidential domain.*

### 4.2.8. System integration

In the case of a Global Platform TEE system a rich OS in Normal domain is free to schedule services, including TEE services, on any Hart available to it. The number and make-up of supervisor domains can be known, and a simple convention can be used for common identification (SDID value, see supervisor domains) of Normal, TEE, and Root domains across multiple Harts in a system.

System integration in this context involves providing *security attributes* on a system interconnect, tagging all transactions (CPU or system agent initiated) to either Root, Normal, or TEE domains.

Possible use cases include:

- Tweaking cryptographic memory protection (uniqueness)
- Tagging interrupts, debug accesses, or coherent memory accesses
- Device assignment (IOPMP/IOMTT integration), static or dynamic

The attributes can be derived, for example, from SDID and privilege level, or from PMA.

For some use cases security attributes can be extended to reflect finer granularity, for example for cryptographic memory protection with TA granularity.

### 4.2.9. Debug and performance management

See security life cycle.
See enhanced RISC-V external debug security

| ID# | Requirement |
|---|---|
| SR_TEE_018 | External debug MUST be enabled separately for Root domain. |
| SR_TEE_019 | External debug MUST be enabled separately for each supervisor domain. |
| SR_TEE_020 | External debug MUST only be enabled by a HW RoT (Root domain external debug) or by Root domain (supervisor domain external debug). |
| SR_TEE_021 | Self-hosted debug MAY be used for debug within a supervisor domain. |
| SR_TEE_022 | Self-hosted debug MUST only be enabled by a higher privileged component. |

For example, within normal domain an S-mode or VS-mode OS can enable self-hosted debug for a user application. Or an HS-mode hypervisor can enable self-hosted debug for a VS-mode guest. Only Root domain should enable self-hosted debug for an S-mode OS or an HS mode hypervisor.

Within TEE domain a TEE OS can enable self-hosted debug for a TA. An SPM can enable self-hosted debug for guest TEE. Only Root domain should enable self-hosted debug of SPM (virtualized) or TEE OS (non-virtualized).

A machine mode monitor can enable external debug of individual supervisor domains without affecting M-mode, or any other supervisor domain.

| ID# | Requirement |
|---|---|
| SR_TEE_023 | Root domain MAY disable self-hosted debug for a whole domain. |

For example, for all of TEE domain on a production system, for certification reasons.

| ID# | Requirement |
|---|---|
| SR_TEE_024 | External debug MUST only be enabled following system reset (part of measuring) of the affected component. |
| SR_TEE_025 | Revealing self-hosted debug MUST only be enabled following reboot (part of measuring) of the affected component. |
| SR_TEE_026 | Trusted self-hosted debug MAY be enabled at runtime (after measuring) of the affected component, to an application specific governance process. |

Guarantees the system remains attestable.

See event counters

## 4.3. Confidential computing on RISC-V (CoVE)

### 4.3.1. Overview

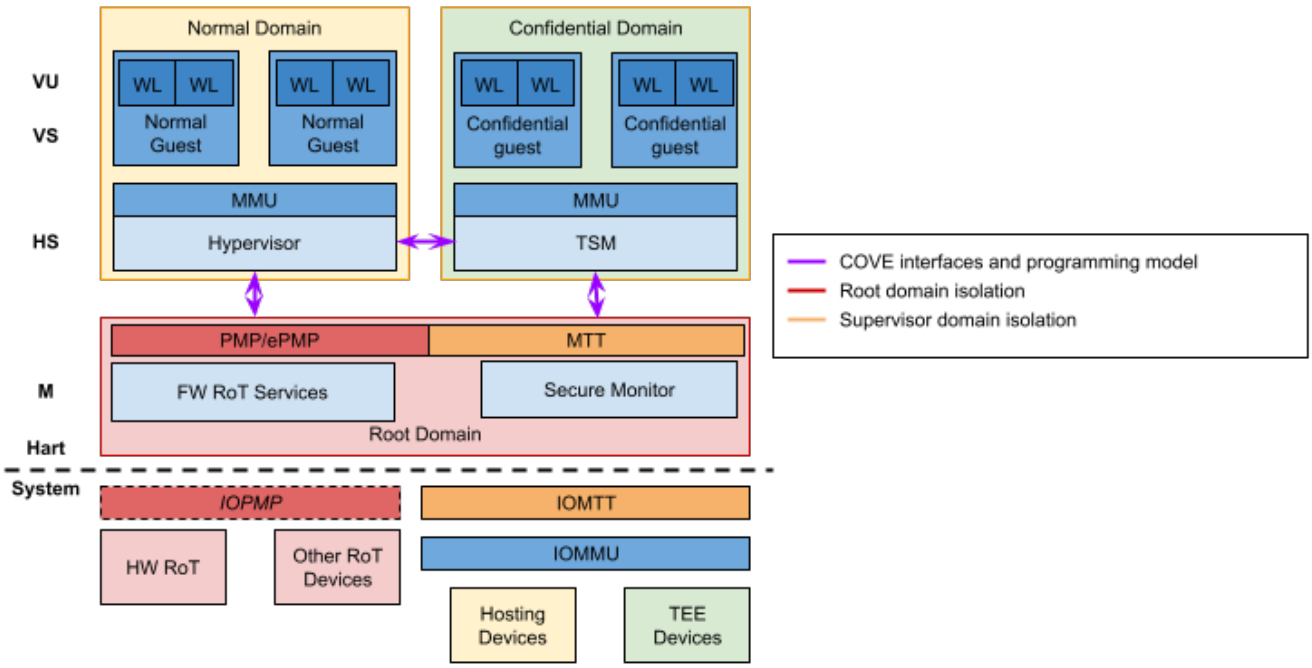


*Figure 5: Confidential compute use case*

In hosting environments, tenant workloads rely on isolation primitives that are managed by host privileged software. This can lead to a large TCB for tenants which may include, for example, a hypervisor, orchestration services, and host management services. It may also include other tenants exploiting vulnerabilities in complex hosting software.

Confidential compute aims to achieve a minimal and certifiable TCB for *confidential workloads*.

*CoVE (Confidential VM Extensions)* specification defines a confidential compute platform for RISC-V systems, including interfaces and programming models, covering life cycle management, attestation, resource management and devices assignment, for confidential workloads. It is based on principles defined by Confidential Computing Consortium. Reference firmware for CoVE is being developed as part of the RISC-V Software Ecosystem project.

CoVE is primarily aimed at cloud hosting of confidential workloads. But the underlying isolation model could potentially be used in other use cases, such as some mobile clients or edge devices.

CoVE divides software into physically isolated domains:

- Normal domain
  Typically hosting a hypervisor, and Normal guests and services.
- Confidential domain
  Hosts a domain security manager (*trusted security manager, TSM*) and confidential guests.
- Root domain
  Hosts RoT firmware, including a secure monitor.

The TSM is primarily responsible for isolation of confidential workloads, and for providing RoT services, within the Confidential domain.

A hypervisor in Normal domain typically controls scheduling and resource assignment on the system across all Harts available to it, including for confidential workloads. It interacts with the TSM through the secure monitor in Root domain to manage confidential workloads.

The secure monitor is responsible for context switching and isolation across domain boundaries, including event management.

### 4.3.2. Isolation model

Confidential workloads are provided the following isolation guarantees:

| ID# | Requirement |
|---|---|
| SR_CFC_001 | Root domain MAY access resources assigned to any domain, but SHOULD prevent itself from unintended access to resources assigned to a different domain (privilege escalation). |
| SR_CFC_002 | Resources assigned to Root domain MUST be private to Root domain |
| SR_CFC_003 | Resources assigned only to Confidential domain MUST not be accessible by Normal domain |
| SR_CFC_004 | Resources assigned only to Normal domain MUST not be accessible by Confidential domain |
| SR_CFC_005 | Resources MAY be assigned to both Normal and Confidential domains (sharing by consent). |
| SR_CFC_006 | Resources assigned to a single confidential workload MUST NOT be accessible by any other confidential workload |
| SR_CFC_007 | Resources MAY be assigned to multiple confidential workloads (sharing by consent) |

RISC-V hardware enforced isolation mechanisms can be used as follows to meet those guarantees:

See supervisor domains. See PMP and Smepmp See sPMP See MMU See MTT

| ID# | Requirement |
|---|---|
| SR_CFC_008 | PMP/Smepmp or MTT MUST be used to isolate Root domain from other domains. |
| SR_CFC_009 | Supervisor domains MUST be used to enforce isolation between Normal and Confidential domains. |

| ID# | Requirement |
|---|---|
| SR_CFC_010 | Hypervisor extension MUST be supported |
| SR_CFC_011 | MMU MUST be used to enforce isolation between Confidential guests within Confidential domain, and between workloads within guests. |

### 4.3.3. Root of trust

See reference model.

| ID# | Requirement |
|---|---|
| SR_CFC_012 | A CoVE system MUST implement a HW RoT |

### 4.3.4. Authorized Boot

See authorized software.

| ID# | Requirement |
|---|---|
| SR_CFC_013 | Confidential guests MUST not boot until at least the security platform has been verified:<br>• TSM in Confidential domain<br>• Root domain<br>• Boot state of all trusted subsystems |

Boot in a cloud hosting context is typically based on:

- Measured boot of a hosting platform, including Root domain and TSM
- Platform attestation and security provisioning (unsealing) by a remote provisioning system
- Launch and measurement of confidential workloads, only once the system has been unsealed

A *trusted platform module* (TPM) can be used to measure the security platform.

Measuring confidential guests can be done by TSM in Confidential domain.

The process can involve multiple stages (layered boot).

### 4.3.5. Attestation

See attestable services.

Virtualized TEE attestation can be layered, for performance or separation of concern. For example:

- A security platform attestation, signed by a RoT, covering trusted subsystems, Root domain, and SPM
- Separate guest TEE attestation(s) signed by SPM

See attestable services.

Attestation of confidential workloads is typically layered, for performance and separation of concern:

- A security platform attestation, signed by a hardware root of trust
- A confidential workload attestation, signed by TSM

| ID# | Requirement |
|---|---|
| SR_CFC_014 | A security platform attestation MUST cover at least:<br>• HW RoT<br>• TSM<br>• Root domain<br>• Boot state of all trusted subsystems |

### 4.3.6. Sealing

See sealing.

Sealing of confidential workloads is typically based on remote sealing, unsealing assets for a confidential workload following successful attestation by a remote provisioning system. This enables use cases such as:

- Shared assets across multiple instances of a confidential workload (scale or redundancy)
- Unsealing different sets of assets for different users of a service

TSM itself is typically stateless across reset and does not require any sealed assets of its own.

### 4.3.7. Device access control

For the purpose of this specification, a device can be a logical device. A physical device can present more than one logical devices, each with its own (logical) control interface.

The security guarantees also apply to device initiated accesses, for example DMA and interrupts.

| ID# | Requirement |
|---|---|
| SR_CFC_015 | IOMTT and IOMMU MUST be used to enforce access rules for devices assigned to Normal or Confidential domains. |
| SR_CFC_016 | IOPMP SHOULD be used to enforce access rules for Root devices. |
| SR_CFC_017 | IOPMP and IOMTT configurations MUST only be directly accessible by Root domain. |

IOMTT enforces supervisor domain level access rules (physical isolation). IOMMU enforces guest and TA level access rules (virtualization), supporting device assignment to a Confidential guest.

*IOMTT can also be sufficient for protecting Root devices in the sense that M-mode can enforce that its own resources are never assigned to another domain. Use of IOPMP or similar could still add further protections. For example, a system may require that Root devices cannot be used to access memory assigned to Confidential domain.*

### 4.3.8. System integration

In the case of a confidential compute system, hypervisor in Normal domain typically controls scheduling and resource assignment on the system across all Harts available to it. The number and make-up of supervisor domains can be known, and a simple convention can be used for common identification of Normal, Confidential, and Root domains across multiple Harts in a system.

System integration in this context involves providing *security attributes* on the interconnect, tagging all

transactions (CPU or system agent initiated) to either Root, Normal, or TEE domains.

Possible use cases include:

- Tweaking cryptographic memory protection (uniqueness)
- Tagging interrupts, debug accesses, or coherent memory accesses
- Device assignment (IOPMP/IOMTT integration), static or dynamic

The attributes can be derived, for example, from SDID and privilege mode.

For some use cases security attributes can be extended to reflect finer granularity, for example for cryptographic memory protection with confidential workload granularity.

### 4.3.9. Trusted device assignment

The goal of confidential compute is to provide a minimum TCB for a confidential service, and CPU isolation mechanisms discussed so far does that on a Hart.

But most confidential services also make use of devices, both on-chip and external. Device virtualization can guarantee exclusivity for devices assigned to a confidential workload - TSM can guarantee that a device assigned to a confidential workload cannot be accessed by:

- Any other confidential workload
- Any software in Normal domain

But the confidential workload still has to trust all intermediaries between the workload and the device, both physical and software. For example:

- Drivers
- Physical interconnects and device hardware interfaces

Secure access to devices is important in a number of use cases where a device performs work on assets owned by a confidential workload, such as accelerators.

The *TEE device interface security protocol (TDISP)* defined by PCIe provides a security architecture and protocols allowing a confidential workload to securely attest, manage and exchange data with a trusted device.

CoVE defines RISC-V support for TDISP. See:

pcisig.com/specifications/ github.com/riscv-non-isa/riscv-ap-tee-io

### 4.3.10. Debug and performance management

See security life cycle.
See enhanced RISC-V external debug security

| ID# | Requirement |
|---|---|
| SR_CFC_018 | External debug MUST be enabled separately for Root domain. |
| SR_CFC_019 | External debug MUST be enabled separately for each supervisor domain. |

| ID# | Requirement |
|---|---|
| SR_CFC_020 | External debug MUST only be enabled by a HW RoT (Root domain external debug) or by Root domain (supervisor domain external debug). |
| SR_CFC_021 | Self-hosted debug MAY be used for debug within a supervisor domain. |
| SR_CFC_022 | Self-hosted debug MUST only be enabled by a higher privileged component. |

For example, within normal domain an HS-mode hypervisor can enable self-hosted debug for a VS-mode guest. Only Root domain should enable self-hosted debug for the HS mode hypervisor.

Within Confidential domain the TSM can enable self-hosted debug for a confidential guest. Only Root domain should enable self-hosted debug of TSM.

A machine mode monitor can enable external debug of individual supervisor domains without affecting M-mode, or any other supervisor domain.

| ID# | Requirement |
|---|---|
| SR_CFC_023 | External debug MUST only be enabled following system reset (part of measuring) of the affected component. |
| SR_CFC_024 | Revealing self-hosted debug MUST only be enabled following reboot (part of measuring) of the affected component. |
| SR_CFC_025 | Trusted self-hosted debug MAY be enabled at runtime (after measuring) of the affected component, to an application specific governance process. |

Guarantees the system remains attestable.

See event counters

### 4.3.11. Platform QoS

See platform quality of service.

# 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.

ML-KEM (FIPS-203), ML-DSA (FIPS-204), and SLH-DSA (FIPS-205). ML-KEM defines a key-encapsulation mechanism used to establish a shared secret key over a public channel. ML-DSA and SLH-DSA defined digital signature schemes.

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

| ID# | Requirement |
|---|---|
| SR_CPT_003 | In applications which require a migration path to PQC algorithms, all immutable components SHOULD support PQC alternatives. |
| SR_CPT_004 | All mutable components MUST at least have a migration path to quantum safe cryptography. |

Immutable components, in particular immutable boot code, cannot be updated. To provide a full migration path for a system, immutable components need to support PQC alternatives.

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)