NIST IR 8320

Hardware-Enabled Security:

Enabling a Layered Approach to Platform Security for Cloud and Edge Computing Use Cases

Michael Bartock
Murugiah Souppaya
Ryan Savino
Tim Knoll
Uttam Shetty
Mourad Cherfaoui
Raghu Yeluri
Akash Malhotra
Don Banks
Michael Jordan
Dimitrios Pendarakis
J. R. Rao
Peter Romness
Karen Scarfone

This publication is available free of charge from:
https://doi.org/10.6028/NIST.IR.8320

National Institute of Standards and Technology
U.S. Department of Commerce

NIST IR 8320

Hardware-Enabled Security:

Enabling a Layered Approach to Platform Security for Cloud and Edge Computing Use Cases

Michael Bartock
Murugiah Souppaya
Computer Security Division
Information Technology Laboratory

Don Banks
Arm Advanced Technology Group
San Jose, CA

Ryan Savino
Tim Knoll
Uttam Shetty
Mourad Cherfaoui
Raghu Yeluri
Intel Data Platforms Group
Santa Clara, CA

Michael Jordan
Dimitrios Pendarakis
J. R. Rao
IBM Systems and IBM Research
Poughkeepsie and Yorktown Heights, NY

Peter Romness
Cisco
McLean, VA

Akash Malhotra
AMD Product Security and Strategy Group
Austin, TX

Karen Scarfone
Scarfone Cybersecurity
Clifton, VA

This publication is available free of charge from:
https://doi.org/10.6028/NIST.IR.8320

May 2022



U.S. Department of Commerce
Gina Raimondo, Secretary

National Institute of Standards and Technology
Laurie E. Locascio, NIST Director and Undersecretary of Commerce for Standards and Technology

National Institute of Standards and Technology Interagency or Internal Report 8320
94 pages (May 2022)

This publication is available free of charge from:
https://doi.org/10.6028/NIST.IR.8320

Certain commercial entities, equipment, or materials may be identified in this document in order to describe an experimental procedure or concept adequately. Such identification is not intended to imply recommendation or endorsement by NIST, nor is it intended to imply that the entities, materials, or equipment are necessarily the best available for the purpose.

There may be references in this publication to other publications currently under development by NIST in accordance with its assigned statutory responsibilities. The information in this publication, including concepts and methodologies, may be used by federal agencies even before the completion of such companion publications. Thus, until each publication is completed, current requirements, guidelines, and procedures, where they exist, remain operative. For planning and transition purposes, federal agencies may wish to closely follow the development of these new publications by NIST.

Organizations are encouraged to review all draft publications during public comment periods and provide feedback to NIST. Many NIST cybersecurity publications, other than the ones noted above, are available at https://csrc.nist.gov/publications.

Submit comments on this publication to: hwsec@nist.gov

National Institute of Standards and Technology
Attn: Applied Cybersecurity Division, Information Technology Laboratory
100 Bureau Drive (Mail Stop 2000) Gaithersburg, MD 20899-2000

All comments are subject to release under the Freedom of Information Act (FOIA).

Reports on Computer Systems Technology

The Information Technology Laboratory (ITL) at the National Institute of Standards and Technology (NIST) promotes the U.S. economy and public welfare by providing technical leadership for the Nation's measurement and standards infrastructure. ITL develops tests, test methods, reference data, proof of concept implementations, and technical analyses to advance the development and productive use of information technology. ITL's responsibilities include the development of management, administrative, technical, and physical standards and guidelines for the cost-effective security and privacy of other than national security-related information in federal information systems.

Abstract

In today's cloud data centers and edge computing, attack surfaces have shifted and, in some cases, significantly increased. At the same time, hacking has become industrialized, and most security control implementations are not coherent or consistent. The foundation of any data center or edge computing security strategy should be securing the platform on which data and workloads will be executed and accessed. The physical platform represents the first layer for any layered security approach and provides the initial protections to help ensure that higher-layer security controls can be trusted. This report explains hardware-enabled security techniques and technologies that can improve platform security and data protection for cloud data centers and edge computing.

Keywords

confidential computing; container; hardware-enabled security; hardware security module (HSM); secure enclave; trusted execution environment (TEE); trusted platform module (TPM); virtualization.

Disclaimer

Any mention of commercial products or reference to commercial organizations is for information only; it does not imply recommendation or endorsement by NIST, nor does it imply that the products mentioned are necessarily the best available for the purpose.

Acknowledgments

The authors thank everyone who contributed their time and expertise to the development of this report, including:

- From AMD: David Kaplan and Kathir Nadarajah
- From Arm: Yuval Elad, Nicholas Wood, Joanna Farley, Paul Howard, Dan Handley, Stuart Yoder, Erik Jacobson, and Mark Knight
- From Cisco: Jeff Schutt, Scott Phuong, Charlie Hsu, Timothy Wilson-Johnston, and Justin Salerno

This publication is available free of charge from: https://doi.org/10.6028/NIST.IR.8320

- From IBM: Jonathan Bradbury, James Bottomley, Debapriya Chatterjee, Chris Engel, Ken Goldman, Guerney Hunt, Kathryn Ignaszewski, Hani Jamjoom, Elaine Palmer, Harmeet Singh, and Balaram Sinharoy
- From Intel Corporation: Ravi Sahita, Alex Eydelberg, Sugumar Govindarajan, Kapil Sood, Jeanne Guillory, David Song, Scott Raynor, Scott Huang, Matthew Areno, Charlie Stark, Subomi Laditan, Kamal Natesan, Haidong Xia, Jerry Wheeler, Dhinesh Manoharan, and John Pennington
- From Red Hat: Luke Hinds and Mark Bohannon for contributing content related to Keylime

Audience

The primary audiences for this report are security professionals, such as security engineers and architects; system administrators and other information technology (IT) professionals for cloud service providers; and hardware, firmware, and software developers who may be able to leverage hardware-enabled security techniques and technologies to improve platform security for cloud data centers and edge computing.

Trademark Information

All registered trademarks or trademarks belong to their respective organizations.

Patent Disclosure Notice

NOTICE: ITL has requested that holders of patent claims whose use may be required for compliance with the guidance or requirements of this publication disclose such patent claims to ITL. However, holders of patents are not obligated to respond to ITL calls for patents and ITL has not undertaken a patent search in order to identify which, if any, patents may apply to this publication.

As of the date of publication and following call(s) for the identification of patent claims whose use may be required for compliance with the guidance or requirements of this publication, no such patent claims have been identified to ITL.

No representation is made or implied by ITL that licenses are not required to avoid patent infringement in the use of this publication.

This publication is available free of charge from: https://doi.org/10.6028/NIST.IR.8320

Table of Contents

1 Introduction 1
2 Hardware Platform Security Overview 3
3 Platform Integrity Verification 5
3.1 Hardware Security Module (HSM) 5
3.2 The Chain of Trust (CoT) 6
3.3 Supply Chain Protection 7
4 Software Runtime Protection Mechanisms 8
4.1 Return Oriented Programming (ROP) and Call/Jump Oriented Programming (COP/JOP) Attacks 8
4.2 Address Translation Attacks 8
4.3 Memory Safety Violations 9
4.4 Side-Channel Attacks 10
5 Data Protection and Confidential Computing 12
5.1 Memory Isolation 12
5.2 Application Isolation 13
5.3 VM Isolation 13
5.4 Cryptographic Acceleration 14
6 Remote Attestation Services 15
6.1 Platform Attestation 15
6.2 Remote TEE Attestation 17
7 Cloud Use Case Scenarios Leveraging Hardware-Enabled Security 19
7.1 Visibility to Security Infrastructure 19
7.2 Workload Placement on Trusted Platforms 19
7.3 Asset Tagging and Trusted Location 21
7.4 Workload Confidentiality 22
7.5 Protecting Keys and Secrets 24
8 Next Steps 26
References 27

List of Appendices

Appendix A— Vendor-Agnostic Technology Examples 35
A.1 Platform Integrity Verification 35

This publication is available free of charge from: https://doi.org/10.6028/NIST.IR.8320

A.1.1 UEFI Secure Boot (SB) 35
A.2 Keylime 36
Appendix B— Intel Technology Examples 37
B.1 Platform Integrity Verification 37
B.1.1 The Chain of Trust (CoT) 37
B.1.2 Supply Chain Protection 41
B.2 Software Runtime Protection Mechanisms 42
B.2.1 Return Oriented Programming (ROP) and Call/Jump Oriented Programming (COP/JOP) Attacks 42
B.2.2 Address Translation Attacks 42
B.3 Data Protection and Confidential Computing 44
B.3.1 Memory Isolation 44
B.3.2 Application Isolation 45
B.3.3 VM Isolation 46
B.3.4 Cryptographic Acceleration 46
B.3.5 Technology Example Summary 47
B.4 Remote Attestation Services 48
B.4.1 Intel Security Libraries for the Data Center (ISecL-DC) 48
B.4.2 Technology Summary 48
Appendix C— AMD Technology Examples 49
C.1 Platform Integrity Verification 49
C.1.1 AMD Platform Secure Boot (AMD PSB) 49
C.2 Data Protection and Confidential Computing 49
C.2.1 Memory Isolation 49
C.2.2 VM Isolation 50
Appendix D— Arm Technology Examples 52
D.1 Platform Integrity Verification 52
D.1.1 Arm TrustZone Trusted Execution Environment (TEE) for Armv8-A 52
D.1.2 Arm Secure Boot and the Chain of Trust (CoT) 55
D.1.3 Platform Security Architecture (PSA) Functional APIs 57
D.1.4 Platform AbstRaction for SECurity (Parsec) 59
D.2 Software Runtime Protection Mechanisms 61

This publication is available free of charge from: https://doi.org/10.6028/NIST.IR.8320

D.2.1 Return Oriented Programming (ROP) and Jump Oriented Programming (JOP) Attacks 61
D.2.2 Memory Safety Violations 62
D.2.3 Arm Mitigations Against Side-Channel Attacks 64
D.3 Data Protection and Confidential Computing 66
D.3.1 Arm Confidential Compute Architecture (CCA) 66
D.3.2 Arm Cryptographic Acceleration 71
Appendix E— Cisco Technology Examples 72
E.1 Platform Integrity Verification 72
E.1.1 Cisco Platform Roots of Trust 72
E.1.2 Cisco Chain of Trust (CoT) 73
E.2 Cisco Supply Chain Protection 73
E.3 Cisco Software Runtime Protections 73
E.4 Cisco Data Protection and Confidential Computing 74
E.5 Cisco Platform Attestation 74
E.6 Cisco Visibility to Security Infrastructure 74
E.7 Cisco Workload Placement on Trusted Platforms 74
Appendix F— IBM Technology Examples 75
F.1 Platform Integrity Verification 75
F.1.1 Hardware Security Module (HSM) 75
F.1.2 IBM Chain of Trust (CoT) 75
F.2 Software Runtime Protection Mechanisms 76
F.2.1 IBM ROP and COP/JOP Attack Defenses 76
F.3 Data Protection and Confidential Computing 76
F.3.1 IBM Memory Isolation Technology 76
F.3.2 IBM Application Isolation Technology 77
F.3.3 IBM VM Isolation Technology 77
F.3.4 IBM Cryptographic Acceleration Technology 78
F.4 Remote Attestation Services 78
F.4.1 IBM Platform Attestation Tooling 78
F.4.2 IBM Continuous Runtime Attestation 78
Appendix G— Acronyms and Abbreviations 79
Appendix H— Glossary 85

This publication is available free of charge from: https://doi.org/10.6028/NIST.IR.8320

List of Figures

Figure 1: Notional Example of Remote Attestation Service 16
Figure 2: Notional Example of TEE Attestation Flow 18
Figure 3: Notional Example of Orchestrator Platform Labeling 20
Figure 4: Notional Example of Orchestrator Scheduling 21
Figure 5: Notional Example of Key Brokerage 23
Figure 6: Notional Example of Workload Image Encryption 23
Figure 7: Notional Example of Workload Decryption 24
Figure 8: Firmware and Software Coverage of Existing Chain of Trust Technologies 40
Figure 9: Arm Processor with TrustZone 53
Figure 10: Boot-Time and Run-Time Firmware 56
Figure 11: Root World (Monitor), Realm World, and Isolation Boundaries 68

This publication is available free of charge from: https://doi.org/10.6028/NIST.IR.8320

1 Introduction

In data centers and edge computing, there are three significant forces that impact security: (1) the introduction of billions of connected devices and increased adoption of the cloud have significantly increased attack surfaces; (2) hacking has become industrialized with sophisticated and evolving techniques to compromise data; and (3) solutions composed of multiple technologies from different vendors result in a lack of coherent and consistent implementations of security controls. Given these forces, the foundation for a data center or edge computing security strategy should have a consolidated approach to comprehensively secure entire systems, including hardware platforms, on which workloads and data are executed and accessed.

In the scope of this document, the *hardware platform* is a server (e.g., application server, storage server, virtualization server) in a data center or edge compute facility. The server's hardware platform, also called the *server platform*, represents the first part of the layered security approach. *Hardware-enabled security*—security with its basis in the hardware platform—can provide a stronger foundation than one offered by software or firmware, which has a larger attack surface and can be modified with relative ease. Hardware root of trust (RoT) can present a smaller attack surface if implemented with a small codebase. Existing security implementations can be enhanced by providing a base-layer, immutable hardware module that chains software and firmware verifications from the hardware all the way to the application space or specified security control. In that manner, existing security mechanisms can be trusted even more to accomplish their security goals without compromise, even when there is a lack of physical security or attacks originate from the software layer.

This report explains hardware-based security techniques and technologies that can improve server platform security and data protection for cloud data centers and edge computing. The rest of this report covers the following topics:

- Section 2 provides an overview of hardware platform security.
- Section 3 discusses the measurement and verification of platform integrity.
- Section 4 explores software runtime attacks and protection mechanisms.
- Section 5 considers protecting data in use, also known as confidential computing.
- Section 6 examines remote attestation services, which can collate platform integrity measurements to aid in integrity verification.
- Section 7 describes a number of cloud use case scenarios that take advantage of hardware-enabled security.
- Section 8 states the next steps for this report and how others can contribute.
- The References section lists the cited references for this report.
- Appendix A describes vendor-agnostic technology examples.
- Appendices B through F describe technology examples from Intel, AMD, Arm, Cisco, and IBM, respectively.
- Appendix G lists the acronyms and abbreviations used in the report.

This publication is available free of charge from: https://doi.org/10.6028/NIST.IR.8320

- Appendix H provides a glossary of selected terms used in the report.

As technology and security capabilities evolve, NIST is continuously seeking feedback from the community on the content of the report and soliciting additional technology example contributions from other companies.

Although this document does not address other platforms like laptops, desktops, mobile devices, or Internet of Things (IoT) devices, the practices in this report can be adapted to support those platforms and their associated use cases.

Please send your feedback and comments to hwsec@nist.gov.

This publication is available free of charge from: https://doi.org/10.6028/NIST.IR.8320

2 Hardware Platform Security Overview

The data center threat landscape has evolved in recent years to encompass more advanced attack surfaces with more persistent attack mechanisms. With increased attention being applied to high-level software security, attackers are pushing lower in the platform stack, forcing security administrators to address a variety of attacks that threaten the platform firmware and hardware. These threats can result in:

- Unauthorized access to and potential extraction of sensitive platform or user data, including direct physical access to dual in-line memory modules (DIMMs)
- Modification of platform firmware, such as that belonging to the Unified Extensible Firmware Interface (UEFI)/Basic Input/Output System (BIOS), Board Management Controller (BMC), Manageability Engine (ME), Peripheral Component Interconnect Express (PCIe) device, and various accelerator cards
- Supply chain interception through the physical replacement of firmware or hardware with malicious versions
- Access to data or execution of code outside of regulated geopolitical or other boundaries
- Circumvention of software and/or firmware-based security mechanisms

For example, LoJax, discovered in August 2018, manifests itself in UEFI malware, allowing it to continuously persist in the firmware layer despite operating system (OS) reinstallations, and thus remain invisible to standard kernel-based virus scans [1]. These attacks can be devastating to cloud environments because they often require server-by-server rebuilds or replacements, which can take weeks. Although still rare, these attacks are increasing as attackers become more sophisticated.

Workloads subject to specific regulations or containing sensitive data present additional security challenges for multi-tenant clouds. While virtualization and containers significantly benefit efficiency, adaptability, and scalability, these technologies consolidate workloads onto fewer physical platforms and introduce the dynamic migration of workloads and data across platforms. Consequently, cloud adoption results in a loss of customer visibility and control over the platforms that host virtualized workloads and data, and introduces the usage of third-party infrastructure administrators. Cloud providers and cloud adopters follow a shared responsibility model, where each party has responsibility for different aspects of the overall implementation. Cloud providers can expose information related to infrastructure security and platform capability in order to provide their tenants with security assurances. Furthermore, cloud providers often have data centers that span multiple geopolitical boundaries, subjecting workload owners to complicated legal and regulatory compliance requirements from multiple countries. Hybrid cloud architectures, in particular, utilize multiple infrastructure providers, each with its own infrastructure configurations and management.

Without physical control over or use of confidential computing features or visibility into platform configurations, conventional security best practices and regulatory requirements become difficult or impossible to implement. With regulatory structures like the General Data Protection Regulation (GDPR) introducing high-stakes fines for noncompliance, having visibility and control over where data may be accessed is more important than ever before. Top concerns

This publication is available free of charge from: https://doi.org/10.6028/NIST.IR.8320

among security professionals include the protection of workloads from general security risks, the loss or exposure of data in the event of a data breach, and regulatory compliance.

Existing mitigations of threats against cloud servers are often rooted in firmware or software, making them vulnerable to the same attack strategies. For example, if the firmware can be successfully exploited, the firmware-based security controls can most likely be circumvented in the same fashion. Hardware-enabled security techniques can help mitigate these threats by establishing and maintaining *platform trust*—an assurance in the integrity of the underlying platform configuration, including hardware, firmware, and software. By providing this assurance, security administrators can gain a level of visibility and control over where access to sensitive workloads and data is permitted. Platform security technologies that establish platform trust can provide notification or even self-correction of detected integrity failures. Platform configurations can automatically be reverted back to a trusted state and give the platform resilience against attack.

To achieve the necessary security controls, an RoT can be leveraged as a starting point that is implicitly trusted. Hardware-based controls can provide a foundation for establishing platform integrity assurances. Combining these functions with a means of producing verifiable evidence that these integrity controls are in place and have been executed successfully is the basis of creating a trusted platform. Minimizing the footprint of this RoT translates to reducing the number of modules or technologies that must be implicitly trusted. This substantially reduces the attack surface.

Platforms that secure their underlying firmware and configuration provide the opportunity for trust to be extended higher in the software stack. Verified platform firmware can, in turn, verify the OS boot loader, which can then verify other software components all the way up to the OS itself and the hypervisor or container runtime layers. The transitive trust described here is consistent with the concept of the *chain of trust (CoT)*—a method where each software module in a system boot process is required to measure the next module before transitioning control.

Rooting platform integrity and trust in hardware security controls can strengthen and complement the extension of the CoT into the dynamic software category. There, the CoT can be extended even further to include data and workload protection. Hardware-based protections through CoT technology mechanisms can form a layered security strategy to protect data and workloads as they move to multi-tenant environments in a cloud data center or edge computing facility.

In addition, there are other hardware platform security technologies that can protect data at rest, in transit, and in use by providing hardware-accelerated disk encryption or encryption-based memory isolation. Many of these capabilities can help mitigate threats from speculative execution and side-channel attacks. By using hardware to perform these tasks, the attack surface is mitigated, preventing direct access or modification of the required firmware. Isolating these encryption mechanisms to dedicated hardware can allow performance to be addressed and enhanced separately from other system processes as well. An example of hardware-based isolation is discussed later in the document.

This publication is available free of charge from: https://doi.org/10.6028/NIST.IR.8320

3 Platform Integrity Verification

A key concept of trusted computing is verification of the underlying platform's integrity. Platform integrity is typically comprised of two parts:

- **Cryptographic measurement of software and firmware.** In this report, the term *measurement* refers to calculating a cryptographic hash of a software or firmware executable, configuration file, or other entity. If there is any change in an entity, a new measurement will result in a different hash value than the original [2]. By measuring software and firmware prior to execution, the integrity of the measured modules and configurations can be validated before the platform launches or before data or workloads are accessed. These measurements can also act as cryptographic proof for compliance audits.
- **Firmware and configuration verification.** When firmware and configuration measurements are made, local or remote attestations can be performed to verify if the desired firmware is actually running and if the configurations are authorized [3]. Attestation can also serve as the foundation for further policy decisions that fulfill various cloud security use case implementations. For instance, encryption keys can be released to client workloads if a proof is performed that the platform is trusted and in compliance with policies.

In some cases, a third part is added to platform integrity:

- **Firmware and configuration recovery.** If the verification step fails (i.e., the attestations do not match the expected measurements), the firmware and configuration can automatically be recovered to a known good state, such as rolling back firmware to a trusted version. The process by which these techniques are implemented affects the overall strength of the assertion that the measured and verified components have not been accidentally altered or maliciously tampered. Recovery technologies allow platforms to maintain resiliency against firmware attacks and accidental provisioning mistakes [4].

There are many ways to measure platform integrity. Most technologies center around the aforementioned concept of the CoT. In many cases, a hardware security module is used to store measurement data to be attested at a later point in time. The rest of this section discusses hardware security modules and various CoT technology implementations.

3.1 Hardware Security Module (HSM)

A *hardware security module (HSM)* is "a physical computing device that safeguards and manages cryptographic keys and provides cryptographic processing" [5]. Cryptographic operations such as encryption, decryption, and signature generation/verification are typically hosted on the HSM device, and many implementations provide hardware-accelerated mechanisms for cryptographic operations.

A *trusted platform module (TPM)* is a special type of HSM that can generate cryptographic keys and protect small amounts of sensitive information, such as passwords, cryptographic keys, and cryptographic hash measurements. [3] The TPM is a standalone device that can be integrated with server platforms, client devices, and other products. One of the main use cases of a TPM is

This publication is available free of charge from: https://doi.org/10.6028/NIST.IR.8320

to store digest measurements of platform firmware and configuration during the boot process. Each firmware module is measured by generating a digest, which is then extended to a TPM platform configuration register (PCR). Multiple firmware modules can be extended to the same PCR, and platform-specific specifications like the Trusted Computing Group (TCG) PC Client Platform Firmware Profile provide guidelines for which firmware measurements are encompassed by each PCR [6].

TPMs also host functionality to generate binding and signing keys that are unique per TPM and stored within the TPM non-volatile random-access memory (NVRAM). The private portion of this key pair is decrypted inside the TPM, making it only accessible by the TPM hardware or firmware. This can create a unique relationship between the keys generated within a TPM and a platform system, restricting private key operations to the platform firmware that has ownership and access to the specified TPM. Binding keys are used for encryption/decryption of data, while signing keys are used to generate/verify cryptographic signatures. The TPM provides a random number generator (RNG) as a protected capability with no access control. This RNG is used in critical cryptographic functionality as an entropy source for nonces, key generation, and randomness in signatures [6].

There are two versions of TPMs: 1.2 and 2.0. The 2.0 version supports additional security features and algorithms [6]. TPMs also meet the National Institute of Standards and Technology (NIST) Federal Information Processing Standard (FIPS) 140 validation criteria and support NIST-approved cryptographic algorithms [7].

3.2 The Chain of Trust (CoT)

The *chain of trust (CoT)* is a method for maintaining valid trust boundaries by applying a principle of transitive trust. Each firmware module in the system boot process is required to measure the next module before transitioning control. Once a firmware module measurement is made, it is recommended to immediately extend the measurement value to a root of trust for storage, such as an HSM register, for attestation at a later point in time [6]. The CoT can be extended further into the application domain, allowing for files, directories, devices, peripherals, etc. to be measured and attested.

Every CoT starts with an RoT module. It can be composed of different hardware and firmware components. For several platform integrity technologies, the RoT core firmware module is rooted in immutable read-only memory (ROM) code. However, not all technologies define their RoTs in this manner [6]. The RoT is typically separated into components that verify and measure. The core RoT for verification (CRTV) is responsible for verifying the first component before control is passed to it. The core RoT for measurement (CRTM) is the first component that is executed in the CoT and extends the first measurement to the TPM. The CRTM can be divided into a static portion (SCRTM) and dynamic portion (DCRTM). The SCRTM is composed of elements that measure firmware at system boot time, creating an unchanging set of measurements that will remain consistent across reboots except for volatile attributes like date and time. The DCRTM allows a CoT to be established without rebooting the system, permitting the RoT for measurement to be reestablished dynamically.

An RoT that is built with hardware protections will be more difficult to change, while an RoT that is built solely in firmware can be flashed and modified. An immutable hardware RoT

This publication is available free of charge from: https://doi.org/10.6028/NIST.IR.8320

possesses the risk of not being able to be patched. The goal should be to keep the RoT as small as possible. In the context of this publication, firmware is a specific class of computer software that provides the low-level control for a device's specific hardware.

Various platform integrity technologies build their own CoTs. Please refer to the following technology examples in the appendices for more information:

- UEFI Secure Boot (SB)
- Intel Trusted Execution Technology (TXT)
- Intel Boot Guard
- Intel Platform Firmware Resilience (PFR)
- Intel Technology Example Summary
- AMD Platform Secure Boot (AMD PSB)
- Arm TrustZone Trusted Execution Environment (TEE) for Armv8-A
- Arm Secure Boot and the Chain of Trust (CoT)
- Cisco Platform Roots of Trust
- IBM Chain of Trust (CoT)

3.3 Supply Chain Protection

Organizations are increasingly at risk of supply chain compromise, whether intentional or unintentional. Managing cyber supply chain risks requires, in part, ensuring the integrity, quality, and resilience of the supply chain, its products, and its services. Cyber supply chain risks may include counterfeiting, unauthorized production, tampering, theft, and insertion of malicious or otherwise unexpected software and hardware, as well as poor manufacturing and development practices in the cyber supply chain [8] [9] [10].

Special technologies have been developed to help ascertain the authenticity and integrity of platform hardware, including its firmware and configuration. These technologies help ensure that platforms are not tampered with or altered from the time that they are assembled at the manufacturer site to the time that they arrive at a customer data center ready for installation. Verification of these platform attributes is one aspect of securing the supply chain.[1] Some technologies include an additional feature for locking the boot process or access to these platforms until a secret is provided that only the customer and manufacturer know.

Please refer to the following technology examples in the appendices for more information:

- Intel Transparent Supply Chain (TSC)
- Intel PFR with Protection in Transit (PIT)
- Cisco Supply Chain Protection

[1] For more information on supply chain security, see the National Cybersecurity Center of Excellence (NCCoE) Supply Chain Assurance project page at https://www.nccoe.nist.gov/supply-chain-assurance.

This publication is available free of charge from: https://doi.org/10.6028/NIST.IR.8320

4 Software Runtime Protection Mechanisms

This section describes various software runtime attacks and protection mechanisms.

4.1 Return Oriented Programming (ROP) and Call/Jump Oriented Programming (COP/JOP) Attacks

ROP attacks focus on utilizing buffer overflows and targeted memory overwrites of return addresses in the stack. Attackers redirect return flows by corrupting addresses on the data stack to be locations in already-executable code. These small selected sequences of code called *gadgets* result in malicious modifications to the system or the invocation of normally unauthorized operations. A common example is a call to the shell executable within the system interface [11].

COP/JOP attacks are similar to ROP attacks, relying on gadget building blocks. They target indirect jump instructions at the end of a gadget, many of which are intentionally emitted by the compiler. However, a jump gadget performs a one-directional control flow transfer to its target, as opposed to ROP, where gadgets return control back to the stack. This can make it difficult for attackers to regain control after executing their gadgets, but solutions to this problem, such as the one presented in [11], are beginning to appear.

Applications can utilize a parallel stack, known as the *shadow stack*, to help mitigate software attacks that attempt to modify the control flow. Utilizing special hardware, the shadow stack is used to store a copy of return addresses; the address is checked against the normal program stack on return operations. If the content differs, an exception is generated, which can help prevent malicious code from gaining control of the system with techniques such as ROP. In this way, shadow stack hardware can help mitigate some of the most common and exploitable types of software bugs.

Several defenses and preventative measures have been developed within industry to accommodate ROP and COP/JOP attacks, including:

- Intel Control-Flow Enforcement Technology (Intel CET)
- Arm Pointer Authentication Code (PAC)
- Arm Branch Target Identification (BTI)
- IBM ROP and COP/JOP Attack Defenses

4.2 Address Translation Attacks

Commodity OSs rely on virtual memory protection models enabled via paging enforced by the processor memory management unit (MMU). OSs isolate process and kernel memory using page tables managed by systems software, with access permissions such as user/supervisor and read/write/execute (RWX). Process and kernel memory accesses are via virtual addresses that are mapped to physical memory addresses via address translation structures. These structures used for address translation are critical to enforcing the isolation model.

This publication is available free of charge from: https://doi.org/10.6028/NIST.IR.8320

Modern OSs are single address space kernels (as opposed to micro-kernels), which provide good performance but have a large attack surface. A vulnerability in the kernel or driver can be leveraged to escalate privileges of a malicious process. Kernel read/write (RW) primitives can be leveraged with Write-What-Where vulnerabilities exploited from flaws discovered in kernel code and/or drivers.

Heuristic defense mechanisms such as Page Table randomization can be bypassed with information leaks achieved via malicious RW primitives. Such information leaks are performed by chaining together a set of system calls (*syscalls*). For example, one syscall can allocate RWX pool memory, and a second can exploit an arbitrary memory write to overwrite the address translation structures. Two types of attacks can utilize this methodology for nefarious purposes. First, an attacker can redirect a virtual address in use to attacker-controlled contents (many times set up in user-space memory). Second, an attacker can create a malicious alias mapping that references desired physical memory with attacker-chosen permissions (e.g., RW access to a page via an alias mapping that was originally read-only). It is important for address translation protection mechanisms to block both of these types of attacks.

In addition to protecting the integrity of address translation structures, processors can also detect and block any execution or data access setup by lower-privilege code from a higher-privilege access. These protections establish boundaries, requiring code to execute with only the necessary permissions and forcing elevated permission requests when needed.

Several defenses and preventative measures have been developed within industry to accommodate address translation attacks, including the following:

- Intel Hypervisor Managed Linear Address Translation (HLAT)
- Intel Supervisor Mode Execution Prevention (SMEP) and Supervisor Mode Access Prevention (SMAP)
- AMD Supervisor Mode Execution Prevention (SMEP) and Supervisor Mode Access Prevention (SMAP)

4.3 Memory Safety Violations

Approximately 70 percent of the vulnerabilities addressed through security updates each year are memory safety issues [12]. They are especially common in programs written in languages such as C and C++ that expose pointers. These code bugs can be exploited by attackers to reveal data, including keys and other secrets. There are both temporal and spatial memory safety violations. Some common examples of both types are as follows:

- **use-after-free**: a program continues to use allocated memory after releasing it (*temporal*)
- **use-out-of-scope**: a program uses memory from another program scope not within its current scope (*temporal*)
- **use-before-initialization**: a program accesses memory that has been allocated but not yet initialized (*temporal*)
- **bounds violations, buffer overflow**: a program accesses memory beyond the bounds of an allocated buffer/data structure (*spatial*)

This publication is available free of charge from: https://doi.org/10.6028/NIST.IR.8320

Hardware-based technologies have been introduced or are being developed in the industry to address memory safety violations. Both require companion software support.

The first, *memory tagging* (also known as coloring, versioning, or tainting) is discussed in [13] and [14]. It is a probabilistic lock-and-key approach to detecting memory violation bugs. With a tag size of 4, the probability of detecting a bug is 94 percent; with a tag size of 8, the probability is 99.6 percent. Memory tagging is expected to be generally applicable to 64-bit software written in C and C++. Use in mixed-language environments, e.g., C/C++ code and interacting with just-in-time (JIT) compiled or interpreted languages, is also expected to benefit. Applicability to software in other languages will vary. Since memory tagging imposes no changes to standard C/C++ application binary interfaces (ABIs), incremental deployment is possible.

The second is *capability-based hardware systems,* which enable software to efficiently implement fine-grained memory protection and scalable software compartmentalization by providing strong, non-probabilistic, efficient mechanisms to support the principles of least privilege and intentional use in the execution of software at multiple levels of abstraction, preventing and mitigating memory safety vulnerabilities.

Hardware capability technology combines references to memory locations—pointers—with limits on how the references can be used. These limits relate to both the address ranges and the functionality that the references can be used to access. This combined information is called a *capability*. It is constructed so that it cannot be forged by software. Replacing pointers with capabilities in a program vastly improves memory safety.

The benefit of hardware capability technology goes beyond memory safety. This is because capabilities can be used as a building block for more fine-grained compartmentalization of software. This could result in inherently more robust software that is resistant to attack. A powerful feature of compartmentalization is that even if one compartment is compromised by an attacker, the attacker cannot break out of the compartment to access any other information or to take overall control of the computing system.

In addition to changes to hardware, capability-based security requires re-architecting how code is designed. Code must be written and compiled in a different way to take advantage of the novel hardware features and to achieve a more secure result.

Several defenses and preventative measures have been developed within industry to accommodate memory safety violations, including the following:

- Arm Privileged Access Never (PAN)
- Arm User (EL0) Execute Never (UXN) and Privileged (EL1/EL2) Execute Never (PXN)
- Arm Memory Tagging Extension (MTE)
- Arm Hardware Enforced Capability-Based Architecture (Morello and CHERI)

4.4 Side-Channel Attacks

The vulnerability underlying cache timing side-channel attacks is that the pattern of allocations into the cache of a central processing unit (CPU), and, in particular, which cache sets have been

This publication is available free of charge from: https://doi.org/10.6028/NIST.IR.8320

used for the allocation, can be determined by measuring the time taken to access entries that were previously in the cache or to access entries that have been allocated. This may leak information about the pattern of cache allocations that could be read by other, less privileged software.

The new feature of speculation-based cache timing side-channels is their use of speculative memory reads. Speculative memory reads are common in high-performance CPUs. By performing speculative memory reads to cacheable locations beyond an architecturally unresolved branch (or other change in program flow), the result of those reads can themselves be used to form the addresses of further speculative memory reads. These speculative reads cause allocations of entries into the cache whose addresses are indicative of the values of the first speculative read. This becomes an exploitable side-channel if untrusted code is able to control the speculation in such a way that it causes a first speculative read of a location that would not otherwise be accessible to that untrusted code. The effects of the second speculative allocation within the caches can be measured by that untrusted code.

Processor designs have evolved to meet these threats by adding additional instructions along with firmware and software support to mitigate this class of attack. Defenses and preventative measures developed within industry to accommodate side-channel attacks include the following:

- Arm Mitigations Against Side-Channel Attacks

This publication is available free of charge from: https://doi.org/10.6028/NIST.IR.8320

5 Data Protection and Confidential Computing

With the increase in adoption of customer-based cloud services, virtualization has become a necessity in cloud data center infrastructure. Virtualization simulates hardware for multiple cloud workloads. Each workload is isolated from others so that it has access to only its own resources, and each workload can be completely encapsulated for portability [15] [16]. Conventional virtual machines (VMs) have an isolated kernel space running all aspects of a workload alongside the kernel. Today, the virtualized environment has been extended to include containers and full-featured workload orchestration engines. Containers offer application portability by sharing an underlying kernel, which drastically reduces workload-consumed resources and increases performance.

While containers can provide a level of convenience, vulnerabilities in the kernel space and shared layers can be susceptible to widespread exploitation, making security for the underlying platform even more important. With the need for additional protection in the virtualized workspace, an emphasis has been placed on encrypting data both at rest and while in use. *At-rest* encryption provides protection for data on disk. This typically refers to an unmounted data store and protects against threats such as the physical removal of a disk drive. Protecting and securing cloud data while *in use*, also referred to as *confidential computing*, utilizes hardware-enabled features to isolate and process encrypted data in memory so that the data is at less risk of exposure and compromise from concurrent workloads or the underlying system and platform [17]. This section describes technologies that can be leveraged for providing confidential computing for cloud and edge.

A *trusted execution environment (TEE)* is an area or enclave protected by a system processor. Sensitive secrets like cryptographic keys, authentication strings, or data with intellectual property and privacy concerns can be preserved within a TEE, and operations involving these secrets can be performed within the TEE, thereby eliminating the need to extract the secrets outside of the TEE. A TEE also helps ensure that operations performed within it and the associated data cannot be viewed from outside, not even by privileged software or debuggers. Communication with the TEE is designed to only be possible through designated interfaces, and it is the responsibility of the TEE designer/developer to define these interfaces appropriately. A good TEE interface limits access to the bare minimum required to perform the task.

5.1 Memory Isolation

There are many technologies that provide data protection via encryption. Most of these solutions focus on protecting the respective data while at rest and do not cover the fact that the data is decrypted and vulnerable while in use. Applications running in memory share the same platform hardware and can be susceptible to attacks either from other workloads running on the same hardware or from compromised cloud administrators. There is a strong desire to secure intellectual property and ensure that private data is encrypted and not accessible at any point in time, particularly in cloud data centers and edge computing facilities. Various hardware technologies have been developed to encrypt content running in platform memory.

This publication is available free of charge from: https://doi.org/10.6028/NIST.IR.8320

Please refer to the following technology examples in the appendices for more information:

- Intel TME and Multi-Key Total Memory Encryption (Intel MKTME)
- AMD Secure Memory Encryption (SME)/Transparent Secure Memory Encryption (TSME)
- Arm Realm Memory Isolation and Protection
- Arm External Memory (DRAM) Encryption and Integrity with CCA
- IBM Memory Isolation Technology

5.2 Application Isolation

Application isolation utilizes a TEE to help protect the memory reserved for an individual application. The trust boundary associated with the application is restricted to only the CPU. Future generations of these techniques will allow entire applications to be isolated in their own enclaves rather than only protecting specific operations or memory. By using separate application enclaves with unique per-application keys, sensitive applications can be protected against data exposure, even to malicious insiders with access to the underlying platform. Implementations of application isolation will typically involve customer application developers integrating a toolkit within the application layer, and it is the developers' responsibility to ensure secure TEE design.

Please refer to the following technology examples in the appendices for more information:

- Intel Software Guard Extensions (SGX)
- Arm Confidential Compute Architecture (CCA)
- Arm TrustZone Trusted Execution Environment (TEE) for Armv8-A
- Arm Realm Memory Isolation and Protection
- IBM Application Isolation Technology

5.3 VM Isolation

As new memory and execution isolation technologies become available, it is more feasible to isolate entire VMs. VMs already enjoy a degree of isolation due to technologies like hardware-assisted virtualization, but the memory of each VM remains unencrypted. Some existing memory isolation technologies require implicit trust of the virtual machine manager (VMM). Isolation technologies in future platform generations will remove the VMM from the trust boundary and allow full encryption of VM memory with per-VM unique keys, protecting the VMs from not only malicious software running on the hypervisor host but also rogue firmware.

VM isolation can be used to help protect workloads in multi-tenant environments like public and hybrid clouds. Isolating entire VMs translates to protection against malicious insiders at the cloud provider, or malware exposure and data leakage to other tenants with workloads running on the same platform. Many modern cloud deployments use VMs as container worker nodes. This provides a highly consistent and scalable way to deploy containers regardless of the underlying physical platforms. With full VM isolation, the virtual workers hosting container

This publication is available free of charge from: https://doi.org/10.6028/NIST.IR.8320

workloads can be effectively isolated without impacting the benefits of abstracting the container from the underlying platform.

Please refer to the following technology examples in the appendices for more information:

- Intel Trust Domain Extensions (Intel TDX)
- AMD Secure Encrypted Virtualization (SEV)
- Arm Confidential Compute Architecture (CCA)
- IBM VM Isolation Technology

5.4 Cryptographic Acceleration

Encryption is quickly becoming more widespread in data center applications as industry adopts more standards and guidelines regarding the sensitivity of customer data and intellectual property. Because cryptographic operations can drain system performance and consume large amounts of compute resources, the industry has adopted specialized hardware interfaces called *cryptographic accelerators*, which offload cryptographic tasks from the main processing unit onto a separate coprocessor chip. Cryptographic accelerators often come in the form of pluggable peripheral adapter cards.

Please refer to the following technology examples in the appendices or technology vendor websites for more information:

- Intel Advanced Encryption Standard New Instructions (AES-NI)
- Intel QuickAssist Technology (QAT) with Intel Key Protection Technology (KPT)
- AMD Advanced Encryption Standard
- Arm Cryptographic Acceleration
- IBM Cryptographic Acceleration Technology

This publication is available free of charge from: https://doi.org/10.6028/NIST.IR.8320

6 Remote Attestation Services

Measuring a server's firmware/configuration and extending these measurements to a root of trust for storage and reporting can help keep track of which firmware is running on a platform. Some platform integrity technologies can even perform local attestation and enforcement of firmware and configuration on a server. However, data centers are usually made up of thousands of servers, and manually keeping track of them and their respective firmware is an overwhelming task for an operator. A remote service can address this by collating server information and measurement details. Cryptographic signatures can be used to ensure the integrity of transferred measurement data. Furthermore, the remote service can be used to define allowlist policies, specifying which firmware versions and event measurements are acceptable for servers in a particular data center environment. This service would verify or attest each server's collected data against these policies, feeding the results into a policy orchestrator to report, alert, or enforce rules based on the events.

A remote attestation service can provide additional benefits besides verifying server firmware. Specifying allowlist policies for specific firmware versions can allow data center administrators to easily invalidate old versions and roll out new upgrades. In some cases, certain hardware technologies and associated capabilities on platforms can be discoverable by their specific event log measurements recorded in an HSM. The information tracked in this remote attestation service can even be exposed through the data center administration layer directly to the enterprise user. This would give endpoint customers hardware visibility and the ability to specify firmware requirements or require platform features for the hardware on which their services are running.

The key advantage to remote attestation is the enforcement of compliance across all hardware systems in a data center. The ability to verify against a collective allowlist as opposed to a local system enforcing a supply chain policy provides operators more flexibility and control in a cryptographically secured manner. These enforcement mechanisms can even be combined to provide stronger security policies.

6.1 Platform Attestation

Figure 1 shows a remote attestation service (AS) collecting platform configurations and integrity measurements from data center servers at a cloud service provider (CSP) via a trust agent service running on the platform servers. A cloud operator is responsible for defining allowlisted trust policies. These policies should include information and expected measurements for desired platform CoT technologies. The collected host data is compared and verified against the policies, and a report is generated to record the relevant trust information in the AS database.

This publication is available free of charge from: https://doi.org/10.6028/NIST.IR.8320

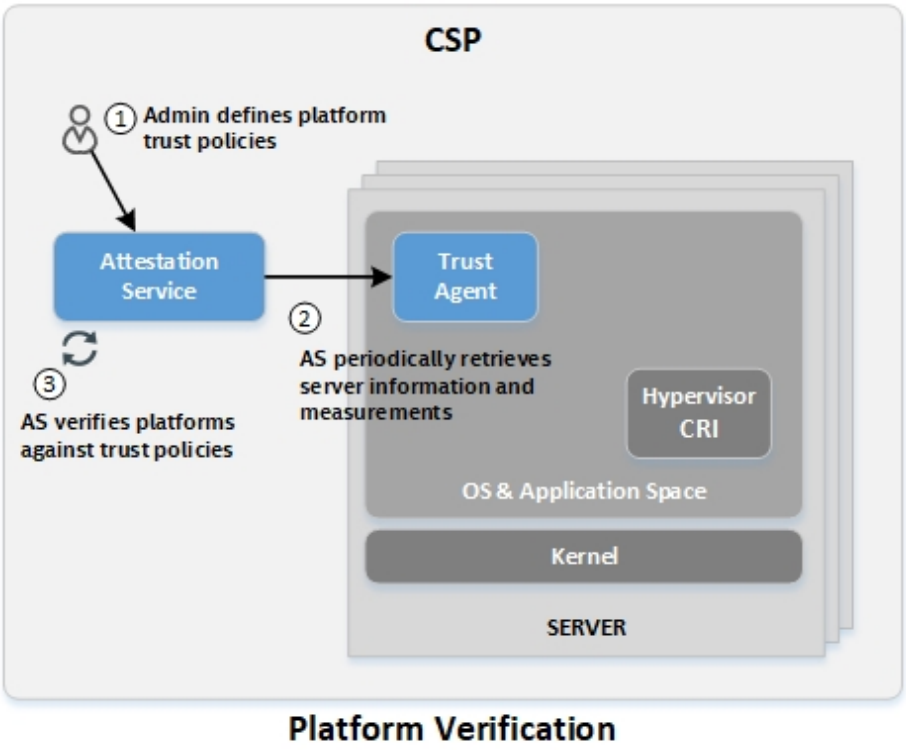


Figure 1: Notional Example of Remote Attestation Service

Platform attestation can be extended to include application integrity or the measurement and verification of the hypervisor container runtime interface (CRI) and applications installed on bare-metal servers. During boot time, an application agent on the server can measure operator-specified files and directories that pertain to particular applications. An allowlist trust policy can be defined to include these expected measurements, and this policy can be included in the overall trust assessment of the platform in the remote AS. By extending measurements to a platform TPM, applications running on the bare-metal server can be added to the CoT. The components of the trust agent and application agent can be added to the policy and measured alongside other applications to ensure that the core feature elements are not tampered with. For example, a typical Linux implementation of the application agent could run inside initrd, and measurements made on the filesystem could be extended to the platform TPM.

An additional feature commonly associated with platform trust is the concept of *asset tagging*. *Asset tags* are simple key value attributes that are associated with a platform like location, company name, division, or department. These key value attributes are tracked and recorded in a central remote service, such as the AS, and can be provisioned directly to a server through the trust agent. The trust agent can then secure these attribute associations with the host platform by writing hash measurement data for the asset tag information to a hardware security chip, such as the platform TPM NVRAM. The asset tag hash is then retrieved by the AS as part of the TPM quote and included in the platform trust report evaluation.

This publication is available free of charge from: https://doi.org/10.6028/NIST.IR.8320

Please refer to the following technology examples in the appendices for more information:

- Intel Security Libraries for the Data Center (ISecL-DC)
- Remote Attestation Service - Project Veraison (VERificAtIon of atteStatiON)
- IBM Platform Attestation Tooling

6.2 Remote TEE Attestation

There are instances when the high assurance that the output of the processing in a TEE can be trusted should be conveyed to a relying party. This is achieved thanks to a TEE attestation flow. *Remote TEE attestation* involves the generation of a verifiable cryptographic evidence by the TEE. The evidence is then sent to a relying party, which can validate the signature of the evidence. If the signature is valid, the relying party concludes that the remote code is running in a genuine TEE.

An evidence usually contains the measurement of the TEE, as well as data related to the authenticity of the TEE and the compliant version of it. The measurement is a digest of the content of the TEE (e.g., code and static data). The measurement obtained at build time is typically known to the relying party and is compared against a measurement contained in the evidence that is actively taken during runtime. This allows the relying party to determine that the remote code has not been tampered with. An evidence may also contain the enclave's developer signature and platform trusted computing base (TCB) information.

The evidence may also contain the public key part of a key pair generated inside the TEE or a secure hash of the public key if there is a limitation on the size of the evidence. In the latter case, the public key must be communicated along with the evidence. The public key allows the relying party to wrap secrets that it wants to send to the TEE. This capability allows the relying party to provision secrets directly to the TEE without needing to trust any other software running on the server.

Figure 2 shows an example TEE attestation flow.

This publication is available free of charge from: https://doi.org/10.6028/NIST.IR.8320

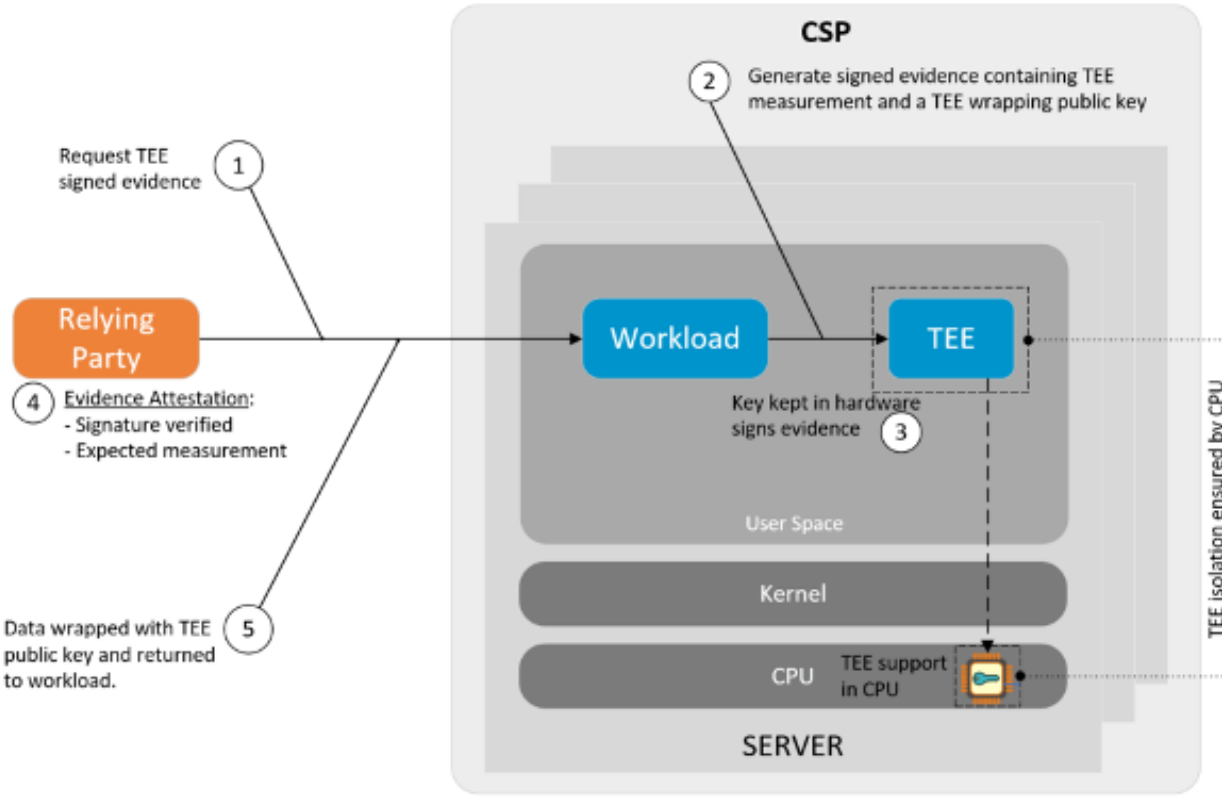


Figure 2: Notional Example of TEE Attestation Flow

Please refer to the following technology example in the appendices for more information:

- Arm Secure Boot and the Chain of Trust (CoT)
- IBM Continuous Runtime Attestation

This publication is available free of charge from: https://doi.org/10.6028/NIST.IR.8320

7 Cloud Use Case Scenarios Leveraging Hardware-Enabled Security

This section describes a number of cloud use case scenarios that take advantage of the hardware-enabled security capability and trust attestation capability integrated with an operator orchestration tool to support various security and compliance objectives.

7.1 Visibility to Security Infrastructure

A typical attestation includes validation of the integrity of platform firmware measurements. These measurements are unique to a specific BIOS/UEFI version, meaning that the attestation report provides visibility into the specific firmware version currently in use, in addition to the integrity of that firmware. Attestation can also include hardware configuration and feature support information, both by attesting feature support directly and by resulting in different measurements based on which platform integrity technologies are used.

Cryptographically verifiable reports of platform integrity and security configuration details (e.g., BIOS/UEFI versions, location information, application versions) are extremely useful for compliance auditing. These attestation reports for the physical platform, or VM/processes in the case of confidential computing, can be paired with workload launch or key release policies, providing traceability to confirm that data and workloads have only been accessed on compliant hardware in compliant configurations with required security technologies enabled.

7.2 Workload Placement on Trusted Platforms

Platform information and verified firmware/configuration measurements retained within an attestation service can be used for policy enforcement in countless use cases. One example is orchestration scheduling. Cloud orchestrators, such as Kubernetes and OpenStack, provide the ability to label server nodes in their database with key value attributes. The attestation service can publish trust and informational attributes to orchestrator databases for use in workload scheduling decisions. Figure 3 illustrates this.

This publication is available free of charge from: https://doi.org/10.6028/NIST.IR.8320

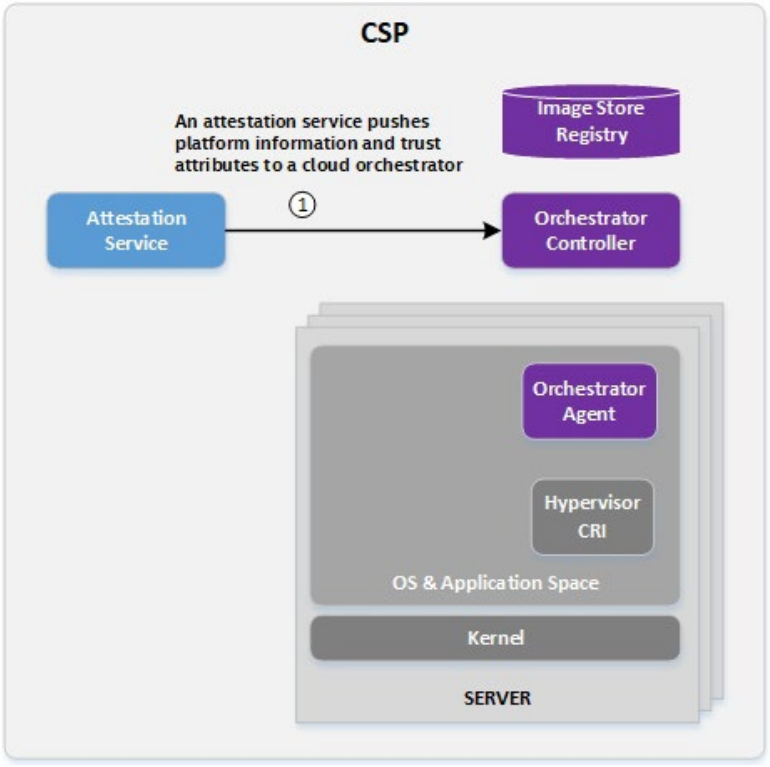


Figure 3: Notional Example of Orchestrator Platform Labeling

In OpenStack, this can be accomplished by labeling nodes using custom traits. Workload images can be uploaded to an image store containing metadata that specifies required trait values to be associated with the node that is selected by the scheduling engine. In Kubernetes, nodes can be labeled in etcd via node selector or node affinity. Custom resource definitions (CRDs) can be written and plugged into Kubernetes to receive label values from the attestation service and associate them with nodes in the etcd. When a deployment or container is launched, node selector or node affinity attributes can be included in the configuration yaml to instruct Kubernetes to only select nodes that have the specified labels. Other orchestrator engines and flavors can be modified to accommodate a similar use case. Figure 4 illustrates how an orchestrator can be configured to only launch workloads on trusted platforms or platforms with specified asset tag attributes.

This publication is available free of charge from: https://doi.org/10.6028/NIST.IR.8320

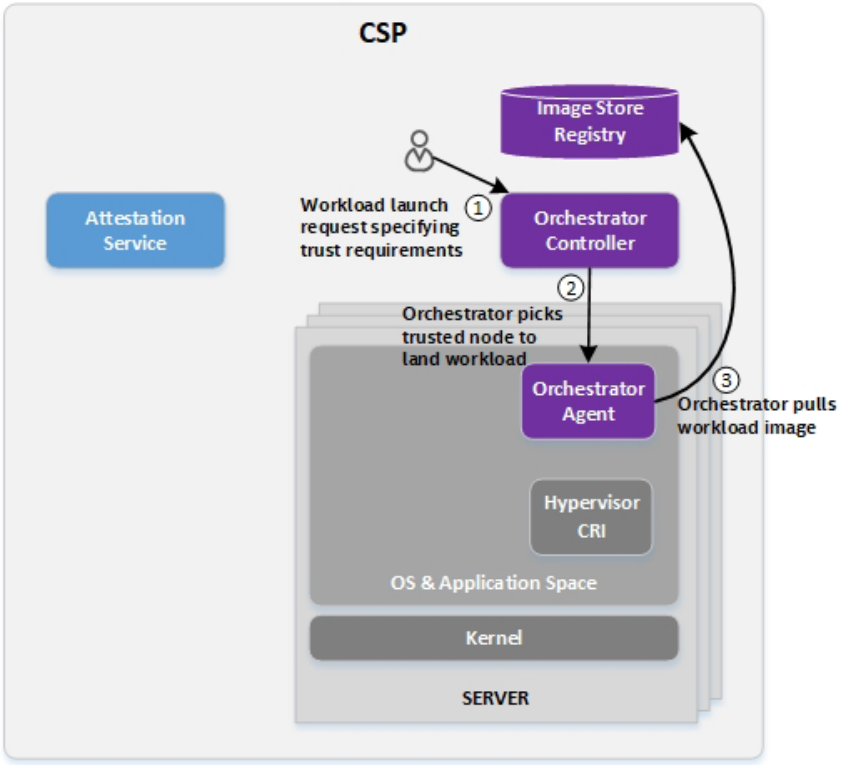


Figure 4: Notional Example of Orchestrator Scheduling

7.3 Asset Tagging and Trusted Location

Trusted geolocation is a specific implementation of the aforementioned trusted asset tag feature used with platform attestation. Key attribute values specifying location information are used as asset tags and provisioned to server hardware, such as the TPM. In this way, location information can be included in platform attestation reports and therefore consumed by cloud orchestrators, infrastructure management applications, policy engines, and other entities [18]. Orchestration using asset tags can be used to segregate workloads and data access in a wide variety of scenarios. Geolocation can be an important attribute to consider with hybrid cloud environments subject to regulatory controls like GDPR, for example. Violating these constraints by allowing access to data outside of specific geopolitical boundaries can trigger substantial penalties.

In addition to location, the same principle can apply to other sorts of tag information. For example, some servers might be tagged as appropriate for storing health information subject to Health Insurance Portability and Accountability Act (HIPAA) compliance. Data and workloads requiring this level of compliance should only be accessed on platforms configured to meet those compliance requirements. Other servers may be used to store or process information and workloads not subject to HIPAA requirements. Asset tags can be used to flag which servers are appropriate for which workloads beyond a simple statement of the integrity of those platforms. The attestation mechanisms help ensure that the asset tag information is genuine, preventing easy subversion.

Outside of specific regulatory requirements, an organization may wish to segregate workloads by department. For example, human resources and finance information could be restricted to

This publication is available free of charge from: https://doi.org/10.6028/NIST.IR.8320

platforms with different security profiles, and big data workloads could be required to run on platforms tagged for performance capabilities. For cloud orchestration platforms that do not natively support discovery or scheduling of workloads based on specific platform features, asset tags can provide a mechanism for seamlessly adding such a capability. For example, workloads that require Intel SGX can be orchestrated to only run on platforms that support the SGX platform feature, even if the cloud platform does not natively discover support for SGX. The open-ended user-configurable asset tag functionality allows virtually any level of subdivision of resources for business, security, or regulatory needs.

7.4 Workload Confidentiality

Customers who place their workloads in the cloud or the edge accept that their workloads are secured by their service providers, typically without insight or knowledge as to what security mechanisms are in place. The ability for end users to encrypt their workload images can provide at-rest cryptographic isolation to protect customer data and intellectual property. Key control is integral to the workload encryption process. While it is preferable to transition key storage, management, and ownership to the endpoint customer, an appropriate key release policy must be defined that includes a guarantee from the service provider that the utilized hardware platform and firmware are secure and uncompromised.

There are several key management solutions (KMSs) in production that provide services to create and store keys. Many of these are compliant with the industry-standardized Key Management Interoperability Protocol (KMIP) or Public Key Cryptography Standards (PKCS) #11 over a network connection and can be deployed within customer enterprises. The concept is to provide a thin layer on top of the KMS called a *key broker*, as illustrated in Figure 5, that applies and evaluates policies to requests that come into the KMS. Supported requests to the key broker include key creation, key release policy association, and key request by evaluating associated policies. The key release policy can be any arbitrary set of rules that must be fulfilled before a key is released. The policy for key release is open-ended and meant to be easily extendible, but for the purpose of this discussion, a policy associated with platform trust is assumed.

This publication is available free of charge from: https://doi.org/10.6028/NIST.IR.8320

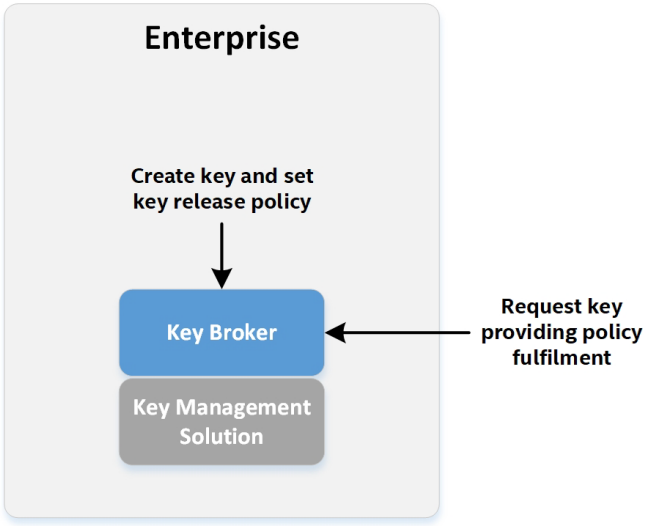


Figure 5: Notional Example of Key Brokerage

Once the key policy has been determined, a KMS-created and managed key can be used to encrypt a workload image, as shown in Figure 6. The enterprise user may then upload the encrypted image to a CSP orchestrator image store or registry.

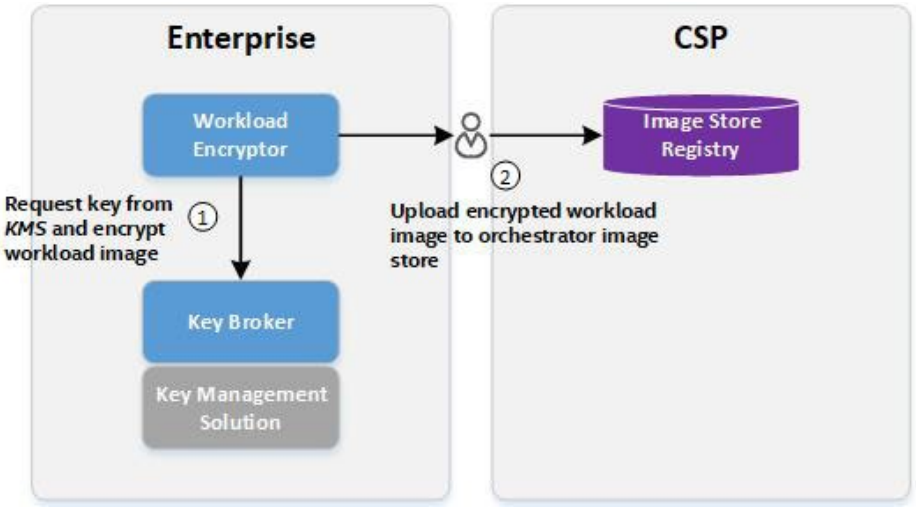


Figure 6: Notional Example of Workload Image Encryption

The key retrieval and decryption process is the most complex piece of the workload confidentiality story, as Figure 7 shows. It relies on a secure key transfer between the enterprise and CSP with an appropriate key release policy managed by the key broker. The key is released to the workload server to decrypt the encrypted workload and launch it. The policy for key release discussed here is based on platform trust and the valid proof thereof. The policy can also dictate a requirement to wrap the key using a public wrapping key, with the private portion of the wrapping key only known to the hardware platform within the CSP.

This publication is available free of charge from: https://doi.org/10.6028/NIST.IR.8320

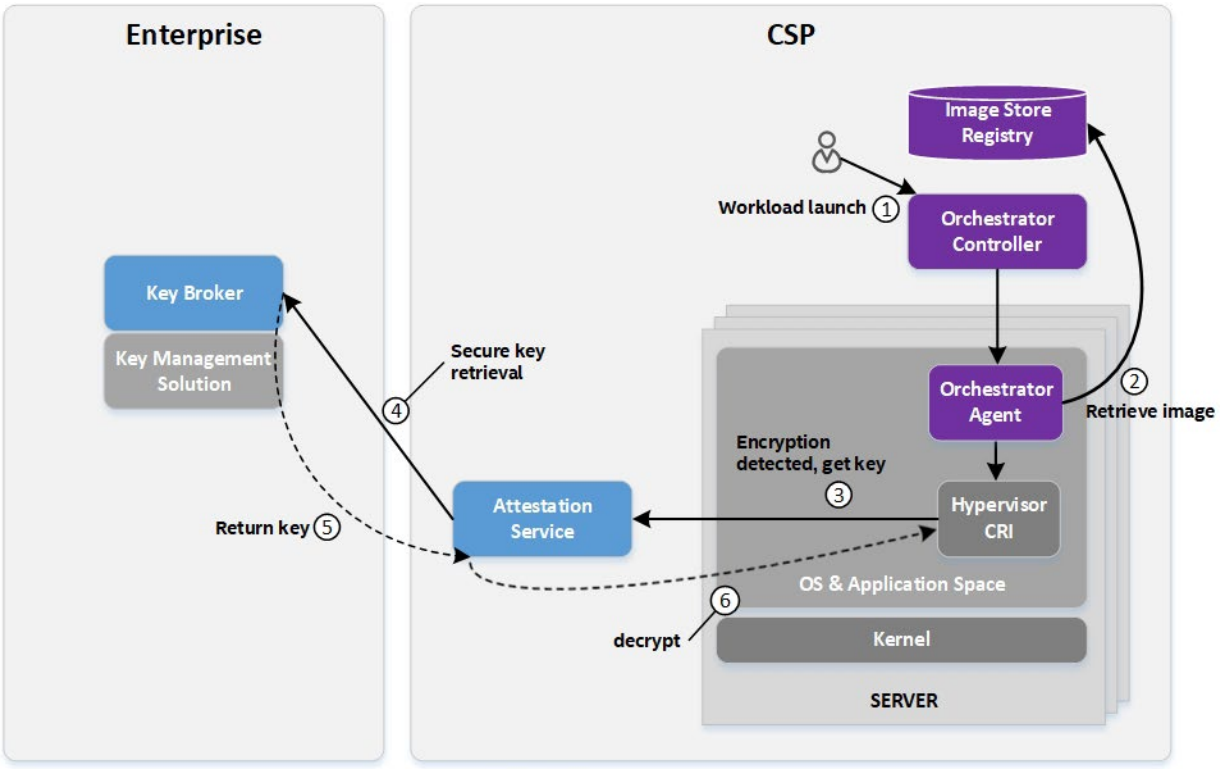


Figure 7: Notional Example of Workload Decryption

When the runtime node service receives the launch request, it can detect that the image is encrypted and make a request to retrieve the decryption key. This request can be passed through an attestation service so that an internal trust evaluation for the platform can be performed. The key request is forwarded to the key broker with proof that the platform has been attested. The key broker can then verify the attested platform report and release the key back to the CSP and node runtime services. At that time the node runtime can decrypt the image and proceed with the normal workload orchestration. The disk encryption kernel subsystem can provide at-rest encryption for the workload on the platform.

7.5 Protecting Keys and Secrets

Cryptographic keys are high-value assets in workloads, especially in environments where the owner of the keys is not in complete control of the infrastructure, such as public clouds, edge computing, and network functions virtualization (NFV) deployments. In these environments, keys are typically provisioned on disk as flat files or entries in configuration files. At runtime, workloads read the keys into random access memory (RAM) and use them to perform cryptographic operations like data signing, encryption/decryption, or Transport Layer Security (TLS) termination.

Keys on disk and in RAM are exposed to conventional attacks like privilege escalation, remote code execution, and input buffer mismanagement. Keys can also be stolen by malicious administrators or be disclosed because of operational errors. For example, an improperly protected VM snapshot can be used by a malicious agent to extract keys.

This publication is available free of charge from: https://doi.org/10.6028/NIST.IR.8320

An HSM can be attached to a server and used by workloads to store keys and perform cryptographic operations. This results in keys being protected at rest and in use. In this model, keys are never stored on disk or loaded into RAM. If attaching an HSM to a server is not an option, or if keys are needed in many servers at the same time, an alternative option is to use a network HSM. Workloads send the payload that needs cryptographic processing over a network connection to the network HSM, which then performs the cryptographic operations locally, typically in an attached HSM.

An HSM option is not feasible in some environments. Workload owners may not have access to a cloud or edge environment in order to attach their HSM to a hardware server. Network HSMs can suffer from network latency, and some workloads require an optimized response time. Additionally, network HSMs are often provided as a service by the cloud, edge, or NFV providers and are billed by the number of transactions. Cost is often a deciding factor for using a provider network HSM.

This publication is available free of charge from: https://doi.org/10.6028/NIST.IR.8320

8 Next Steps

NIST is seeking feedback from the community on the content of the report and soliciting additional technology example contributions from other companies. The report is intended to be a living document that will be frequently updated to reflect advances in technology and the availability of commercial implementations and solutions. This can help raise the bar on platform security and evolve the use cases.

Please send your feedback and comments on this report to hwsec@nist.gov.

NIST is also working on other publications on hardware-enabled security as part of the NCCoE Trusted Cloud project. More information on the project and links to the other publications are available at https://www.nccoe.nist.gov/projects/trusted-cloud-vmware-hybrid-cloud-iaas-environments.

This publication is available free of charge from: https://doi.org/10.6028/NIST.IR.8320

References

[1] Barrett B (2018) *Russia's Elite Hackers Have a Clever New Trick That's Very Hard to Fix*, Wired. Available at https://www.wired.com/story/fancy-bear-hackers-uefi-rootkit

[2] Intel Corporation (2021) *Intel® Trusted Execution Technology (Intel® TXT) – Software Development Guide – Measured Launched Environment Developer's Guide, Revision 017*. Available at https://www.intel.com/content/www/us/en/software-developers/intel-txt-software-development-guide.html

[3] Regenscheid AR (2014) BIOS Protection Guidelines for Servers. (National Institute of Standards and Technology, Gaithersburg, MD), NIST Special Publication (SP) 800-147B. https://doi.org/10.6028/NIST.SP.800-147B

[4] Regenscheid AR (2018) Platform Firmware Resiliency Guidelines. (National Institute of Standards and Technology, Gaithersburg, MD), NIST Special Publication (SP) 800-193. https://doi.org/10.6028/NIST.SP.800-193

[5] Barker EB, Barker WC (2019) Recommendation for Key Management: Part 2 – Best Practices for Key Management Organizations. (National Institute of Standards and Technology, Gaithersburg, MD), NIST Special Publication (SP) 800-57 Part 2, Rev. 1. https://doi.org/10.6028/NIST.SP.800-57pt2r1

[6] Trusted Computing Group (2019) *Trusted Platform Module Library Specification, Family "2.0"*. Available at https://trustedcomputinggroup.org/work-groups/trusted-platform-module/

[7] National Institute of Standards and Technology (2001) Security Requirements for Cryptographic Modules. (U.S. Department of Commerce, Washington, DC), Federal Information Processing Standards Publication (FIPS) 140-2, Change Notice 2 December 03, 2002. https://doi.org/10.6028/NIST.FIPS.140-2

[8] Diamond T, Grayson N, Paulsen C, Polk T, Regenscheid A, Souppaya M, Brown C (2020) Validating the Integrity of Computing Devices: Supply Chain Assurance. (National Institute of Standards and Technology, Gaithersburg, MD). Available at https://www.nccoe.nist.gov/supply-chain-assurance

[9] Boyens JM, Paulsen C, Moorthy R, Bartol N (2015) Supply Chain Risk Management Practices for Federal Information Systems and Organizations. (National Institute of Standards and Technology, Gaithersburg, MD), NIST Special Publication (SP) 800-161. https://doi.org/10.6028/NIST.SP.800-161

[10] National Institute of Standards and Technology (2022) *Cybersecurity Supply Chain Risk Management*. Available at https://csrc.nist.gov/Projects/cyber-supply-chain-risk-management

This publication is available free of charge from: https://doi.org/10.6028/NIST.IR.8320

[11] Bletsch T, Jiang X, Freeh V, Liang Z (2011) Jump-Oriented Programming: A New Class of Code-Reuse Attack. Available at https://www.comp.nus.edu.sg/~liangzk/papers/asiaccs11.pdf

[12] Arm (2022) *Trusted Firmware-A Project*. Available at https://www.trustedfirmware.org/projects/tf-a/

[13] Arm (2022) *Morello*. Available at https://developer.arm.com/architectures/cpu-architecture/a-profile/morello

[14] Arm (2022) *Arm Morello System Development Platform Technical Reference Manual*. Available at https://developer.arm.com/documentation/102278/latest

[15] Scarfone KA, Souppaya MP, Hoffman P (2011) Guide to Security for Full Virtualization Technologies. (National Institute of Standards and Technology, Gaithersburg, MD), NIST Special Publication (SP) 800-125. https://doi.org/10.6028/NIST.SP.800-125

[16] Intel Corporation (2022) *Intel® Virtualization Technology (Intel® VT).* Available at https://www.intel.com/content/www/us/en/virtualization/virtualization-technology/intel-virtualization-technology.html

[17] Linux Foundation (2022) *What is the Confidential Computing Consortium?* Available at https://confidentialcomputing.io

[18] Bartock MJ, Souppaya MP, Yeluri R, Shetty U, Greene J, Orrin S, Prafullchandra H, McLeese J, Scarfone KA (2015) Trusted Geolocation in the Cloud: Proof of Concept Implementation. (National Institute of Standards and Technology, Gaithersburg, MD), NIST Interagency or Internal Report (IR) 7904. https://doi.org/10.6028/NIST.IR.7904

[19] Debian Wiki (2022) *Secure Boot*. Available at https://wiki.debian.org/SecureBoot

[20] Wilkins R, Richardson B (2013) *UEFI Secure Boot in Modern Computer Security Solutions* (Unified Extensible Firmware Interface Forum). Available at https://uefi.org/sites/default/files/resources/UEFI_Secure_Boot_in_Modern_Computer_Security_Solutions_2013.pdf

[21] RedHat (2021) *README* (for shim). Available at https://github.com/rhboot/shim

[22] Intel Corporation (2018) *Intel® Trusted Execution Technology (Intel® TXT) Overview*. Available at https://www.intel.com/content/www/us/en/support/articles/000025873/technologies.html

[23] Futral W, Greene J (2013) *Intel® Trusted Execution Technology for Server Platforms* (Apress, Berkeley, CA). Available at https://link.springer.com/book/10.1007/978-1-4302-6149-0

This publication is available free of charge from: https://doi.org/10.6028/NIST.IR.8320

[24] Linux Kernel Organization (2022) *Intel® TXT Overview*. Available at https://www.kernel.org/doc/Documentation/intel_txt.txt

[25] Intel Corporation (2022) *Transparent Supply Chain*. Available at https://www.intel.com/content/www/us/en/products/docs/servers/transparent-supply-chain.html

[26] Intel Corporation (2020) *Intel Highlights Latest Security Investments at RSA 2020*. Available at https://newsroom.intel.com/news-releases/intel-highlights-latest-security-investments-rsa-2020/

[27] Department of Defense (2018) Subpart 246.870, Contractors' Counterfeit Electronic Part Detection and Avoidance Systems, *Defense Federal Acquisition Regulation Supplement*. Available at https://www.acq.osd.mil/dpap/dars/dfars/html/current/246_8.htm#246.870-2

[28] Intel Corporation (2021) Intel 64 and IA-32 Architectures Software Developer's Manual Volume 1: Basic Architecture. Available at https://software.intel.com/content/www/us/en/develop/download/intel-64-and-ia-32-architectures-software-developers-manual-volume-1-basic-architecture.html

[29] Nichols S (2020) RIP ROP, COP, JOP? Intel to bring anti-exploit tech to market in this year's Tiger Lake chip family. (The Register, San Francisco, CA). Available at https://www.theregister.com/2020/06/15/intel_cet_tiger_lake

[30] Patel BV (2020) A Technical Look at Intel's Control-flow Enforcement Technology. (Intel Fellow Client Computing Group, Intel Corporation). Available at https://software.intel.com/content/www/us/en/develop/articles/technical-look-control-flow-enforcement-technology.html

[31] Patel BV (2016) Intel Releases New Technology Specifications to Protect Against ROP attacks. (Intel Corporation).

[32] Shanbhogue V, Gupta D, Sahita R (2019) Security Analysis of Processor Instruction Set Architecture for Enforcing Control-Flow Integrity. (Hardware and Architectural Support for Security and Privacy '19: Proceedings of the 8th International Workshop on Hardware and Architectural Support for Security and Privacy). https://doi.org/10.1145/3337167.3337175

[33] Intel Corporation (2021) Intel Architecture Instruction Set Extensions and Future Features Programming Reference. Available at https://software.intel.com/content/www/us/en/develop/download/intel-architecture-instruction-set-extensions-programming-reference.html

[34] Intel Corporation (2018) Intel Security Features and Technologies Related to Transient Execution Attacks. Available at

This publication is available free of charge from: https://doi.org/10.6028/NIST.IR.8320

https://software.intel.com/content/www/us/en/develop/articles/software-security-guidance/best-practices/related-intel-security-features-technologies.html

[35] Anvin HP (2012) Description x86: Supervisor Mode Access Prevention. Available at https://lwn.net/Articles/517251/

[36] Corbet J (2012) Supervisor Mode Access Prevention. Available at https://lwn.net/Articles/517475/

[37] Intel Corporation (2022) *Strengthen Enclave Trust with Attestation*. Available at https://www.intel.com/content/www/us/en/developer/tools/software-guard-extensions/attestation-services.html

[38] European Telecommunications Standards Institute (2022) *Network Functions Virtualisation (NFV)*. Available at https://www.etsi.org/technologies/nfv

[39] Intel Corporation (2020) Intel® Trust Domain Extensions. Available at https://software.intel.com/content/dam/develop/external/us/en/documents/tdx-whitepaper-v4.pdf

[40] Intel Corporation (2022) Intel AES New Instructions (Intel AES-NI). Available at https://www.intel.com/content/www/us/en/architecture-and-technology/advanced-encryption-standard-aes/data-protection-aes-general-technology.html

[41] Intel Corporation (2012) *Flexible Workload Acceleration on Intel Architecture Lowers Equipment Cost*. Available at https://www.intel.fr/content/dam/www/public/us/en/documents/white-papers/communications-quick-assist-paper.pdf

[42] Tadepalli, H (2017) Intel QuickAssist Technology with Intel Key Protection Technology in Intel Server Platforms Based on Intel Xeon Processor Scalable Family. (Intel Corporation).

[43] Intel Corporation (2022) *Intel® Security Libraries for Datacenter (Intel® SecL-DC).* Available at https://intel-secl.github.io/docs/4.2/

[44] Kaplan D, Powell J, Woller T (2016) AMD Memory Encryption. (Advanced Micro Devices, Inc.). Available at https://developer.amd.com/wordpress/media/2013/12/AMD_Memory_Encryption_Whitepaper_v7-Public.pdf

[45] Kaplan D (2017) Protecting VM Register State with SEV-ES. (Advanced Micro Devices, Inc.). Available at https://www.amd.com/system/files/TechDocs/Protecting%20VM%20Register%20State%20with%20SEV-ES.pdf

[46] Advanced Micro Devices, Inc. (2020) AMD SEV-SNP: Strengthening VM Isolation with Integrity Protection and More. Available at

This publication is available free of charge from: https://doi.org/10.6028/NIST.IR.8320

https://www.amd.com/system/files/TechDocs/SEV-SNP-strengthening-vm-isolation-with-integrity-protection-and-more.pdf

[47] Arm (2022) *Arm Architecture Reference Manual for A-profile architecture*. Available at https://developer.arm.com/documentation/ddi0487/latest

[48] Arm (2020) *Trusted Base System Architecture for Armv8-A*. Available at https://developer.arm.com/documentation/den0021/f/?lang=en

[49] Arm (2022) *TrustZone for AArch64*. Available at https://developer.arm.com/documentation/102418/0100/What-is-TrustZone-?lang=en

[50] Parsec Project (2022) *Parsec.* Available at https://github.com/parallaxsecond/parsec

[51] Parsec Project (2019) *Authenticators*. Available at https://parallaxsecond.github.io/parsec-book/parsec_service/authenticators.html

[52] SPIFFE (2022) *SPIFFE Verifiable Identity Document (SVID)*. Available at https://spiffe.io/docs/latest/spiffe-about/spiffe-concepts/#spiffe-verifiable-identity-document-svid

[53] SPIFFE (2022) *SPIRE Concepts*. Available at https://spiffe.io/docs/latest/spire-about/spire-concepts/

[54] SPIFFE (2022) *SPIFFE Workload API*. Available at https://spiffe.io/docs/latest/spiffe-about/spiffe-concepts/#spiffe-workload-api

[55] Arm (2021) *Base System Architecture – Architecture Compliance Suite*. Available at https://github.com/ARM-software/bsa-acs

[56] Ubuntu (2021) *Firmware Test Suite (fwts)*. Available at https://wiki.ubuntu.com/FirmwareTestSuite/Reference

[57] Kerrisk M (2021) Unix sockets peer credential checks. Available at https://man7.org/linux/man-pages/man2/getsockopt.2.html

[58] Serebryany K, Stepanov E, Shlyapnikov A, Tsyrklevich V, Vyukov D (2018) *Memory Tagging and how it improves C/C++ memory safety*, Google. Available at https://arxiv.org/ftp/arxiv/papers/1802/1802.09517.pdf

[59] Serebryany K (2019) *ARM Memory Tagging Extension and How It Improves C/C++ Memory Safety*. Available at https://www.usenix.org/system/files/login/articles/login_summer19_03_serebryany.pdf

[60] Watson RNM et al. (2020) Capability Hardware Enhanced RISC Instructions: CHERI Instruction-Set Architecture (Version 8). University of Cambridge Computer

This publication is available free of charge from: https://doi.org/10.6028/NIST.IR.8320

Laboratory, Technical Report Number 951. Available at https://www.cl.cam.ac.uk/techreports/UCAM-CL-TR-951.pdf

[61] Arm (2022) *Arm Architecture Reference Manual Supplement – Morello for A-profile Architecture*. Available at https://developer.arm.com/documentation/ddi0606/latest

[62] Arm (2020) *Cache Speculation Side-channels, Version 2.5*. Available at https://developer.arm.com/documentation/102816/0205/

[63] Arm (2020) *Straight-line Speculation, Version 1.0*. Available at https://developer.arm.com/documentation/102825/0100/

[64] Arm (2021) *Arm v8.5-A/v9 CPU updates, Version 1.5*. Available at https://developer.arm.com/documentation/102822/0105/

[65] Arm (2022) *Introducing Arm Confidential Compute Architecture Version 2*. Available at https://developer.arm.com/documentation/den0125/latest

[66] Arm (2021) *The Realm Management Extension (RME), for Armv9-A*. Available at https://developer.arm.com/documentation/ddi0615/aa

[67] Arm (2021) *Arm Realm Management Extension (RME) System Architecture*. Available at https://developer.arm.com/documentation/den0129/aa

[68] Lundblade L, Mandyam G, O'Donoghue J (2022) The Entity Attestation Token (EAT). (Internet Engineering Task Force, RATS Working Group), IETF Internet-Draft draft-ietf-rats-eat-12. https://datatracker.ietf.org/doc/html/draft-ietf-rats-eat

[69] Trusted Computing Group (2018) Implicit Identity Based Device Attestation, Version 1.0. Available at https://trustedcomputinggroup.org/wp-content/uploads/TCG-DICE-Arch-Implicit-Identity-Based-Device-Attestation-v1-rev93.pdf

[70] Dyer JG, Lindemann M, Perez R, Sailer R, van Doorn L, Smith SW (2001) "Building the IBM 4758 secure coprocessor," in *Computer*, vol. 34, no. 10, pp. 57-66, Oct. 2001. https://ieeexplore.ieee.org/document/955100

[71] IBM (2022) *IBM Cloud Hyper Protect Crypto Services*. Available at https://www.ibm.com/cloud/hyper-protect-crypto

[72] Trusted Computing Group (2019) *TPM 2.0 Library*. Available at https://trustedcomputinggroup.org/resource/tpm-library-specification/

[73] Trusted Computing Group (2019) *Trusted Platform Module Library Part 1: Architecture*. Available at https://trustedcomputinggroup.org/wp-content/uploads/TCG_TPM2_r1p59_Part1_Architecture_pub.pdf

[74] Engel C (2020) "POWER9 Introduces Secure Boot to PowerVM," IBM Power Systems Community. Available at

This publication is available free of charge from: https://doi.org/10.6028/NIST.IR.8320

https://community.ibm.com/community/user/power/blogs/chris-engel1/2020/06/17/power9-introduces-secure-boot-to-powervm

[75] Heller D, Sastry N (2019) *OpenPower Secure and Trusted Boot, Part 2: Protecting System Firmware with OpenPOWER Secure Boot*, IBM. Available at https://developer.ibm.com/articles/protect-system-firmware-openpower/

[76] IBM (2019) *z/Architecture Principles of Operation, Thirteenth Edition*. Available at http://publibfp.dhe.ibm.com/epubs/pdf/a227832c.pdf

[77] Security Target for PR/SM for IBM z14 and IBM LinuxONE Systems, Version: 16.12 Revision: 1, Status: RELEASE, Last Update: 2018-06-20. (A newer version of this reference is available at https://commoncriteriaportal.org/files/epfiles/1133b_pdf.pdf.)

[78] IBM (2022) *IBM Hyper Protect Virtual Servers*. Available at https://www.ibm.com/products/hyper-protect-virtual-servers

[79] OpenPOWER Foundation (2021) *How to build and run Secure VM using Ultravisor on an OpenPOWER machine*. Available at https://github.com/open-power/ultravisor/wiki/How-to-build-and-run-Secure-VM-using-Ultravisor-on-a-OpenPOWER-machine

[80] OpenPOWER Foundation (2021) *Ultravisor*. Available at https://github.com/open-power/ultravisor

[81] Joseph EK (2020) *Technical Overview of Secure Execution for Linux on IBM Z*, IBM. Available at https://developer.ibm.com/blogs/technical-overview-of-secure-execution-for-linux-on-ibm-z/

[82] IBM (2006) *Secure Blue - Secure CPU Technology*. Available at https://researcher.watson.ibm.com/researcher/view_page.php?id=6904

[83] Williams P, Boivie R (2011) "Secure Blue++: CPU Support for Secure Executables," Trust 2011, 4th International Conference on Trusted Computing, June 22-24, 2011, Pittsburgh, Pennsylvania.

[84] Boivie R, Williams P (2011) "SecureBlue++: CPU Support for Secure Execution," 2nd Annual NSA Trusted Computing Conference, September 20-22, 2011, Orlando, Florida.

[85] IBM (2019) "IBM 4767-002 PCIe Cryptographic Coprocessor (HSM)." Available at https://public.dhe.ibm.com/security/cryptocards/pciecc2/docs/4767_PCIe_Data_Sheet.pdf

[86] IBM (2021) "IBM CEX7S / 4769 PCIe Cryptographic Coprocessor (HSM)." Available at https://public.dhe.ibm.com/security/cryptocards/pciecc4/docs/4769_Data_Sheet.pdf

This publication is available free of charge from: https://doi.org/10.6028/NIST.IR.8320

[87] IBM (2022) *CEX7S / 4769 Overview: IBM Systems cryptographic hardware products*. Available at https://www.ibm.com/security/cryptocards/pciecc4/overview

[88] IBM (2021) *Protected-key CPACF*. Available at https://www.ibm.com/docs/en/zos/2.3.0?topic=management-protected-key-cpacf

[89] *IBM TPM Attestation Client Server*. Available at https://sourceforge.net/projects/ibmtpm20acs/files/AttestProv.doc/download?use_mirror=phoenixnap

[90] *Integrity Measurement Architecture (IMA)*. Available at https://sourceforge.net/p/linux-ima/wiki/Home/

[91] Sailer R, Zhang X, Jaeger T, van Doorn L (2004) "Design and Implementation of a TCG-Based Integrity Measurement Architecture." *Proceedings of the 13th USENIX Security Symposium*, USENIX Association, Berkeley, CA, pp. 223-38. Available at https://www.usenix.org/legacy/publications/library/proceedings/sec04/tech/full_papers/sailer/sailer_html/index.html

This publication is available free of charge from: https://doi.org/10.6028/NIST.IR.8320

Appendix A—Vendor-Agnostic Technology Examples

This section describes vendor-agnostic technology examples that map back to the key concepts described in the various sections of the document.

A.1 Platform Integrity Verification

A.1.1 UEFI Secure Boot (SB)

"UEFI Secure Boot (SB) is a verification mechanism for ensuring that code launched by a computer's UEFI firmware is trusted" [19]. SB prevents malware from taking "advantage of several pre-boot attack points, including the system-embedded firmware itself, as well as the interval between the firmware initiation and the loading of the operating system" [20].

This publication is available free of charge from: https://doi.org/10.6028/NIST.IR.8320

The basic idea behind SB is to sign executables using a public-key cryptography scheme. The public part of a *platform key (PK)* can be stored in the firmware for use as a root key. Additional key exchange keys (KEKs) can also have their public portion stored in the firmware in what is called the *signature database*. This database contains public keys that can be used to verify different components that might be used by UEFI (e.g., drivers), as well as bootloaders and OSs that are loaded from external sources (e.g., disks, USB devices, network). The signature database can also contain *forbidden signatures*, which correspond to a revocation list of previously valid keys. The signature database is meant to contain the current list of authorized and forbidden keys as determined by the UEFI organization. The signature on an executable is verified against the signature database before the executable can be launched, and any attempt to execute an untrusted program will be prevented [19][20].

Before a PK is loaded into the firmware, UEFI is considered to be in *setup mode*, which allows anyone to write a PK or KEK to the firmware. Writing the PK switches the firmware into *user mode*. Once in user mode, PKs and KEKs can only be written if they are signed using the private portion of the PK. Essentially, the PK is meant to authenticate the platform owner, while the KEKs are used to authenticate other components of the distribution (distro), like OSs [20].

Shim is a simple software package that is designed to work as a first-stage bootloader on UEFI systems. It is a common piece of code that is considered safe, well-understood, and audited so that it can be trusted and signed using PKs. This means that firmware certificate authority (CA) providers only have to worry about signing shim and not all of the other programs that vendors might want to support [19]. Shim then becomes the RoT for all the other distro-provided UEFI programs. It embeds a distro-specific CA key that is itself used to sign additional programs (e.g., Linux, GRUB, fwupdate). This allows for a clean delegation of trust; the distros are then responsible for signing the rest of their packages. Ideally, shim will not need to be updated often, which should reduce the workload on the central auditing and CA teams [19].

A key part of the shim design is to allow users to control their own systems. The distro CA key is built into the shim binary itself, but there is also an extra database of keys that can be managed by the user—the so-called *Machine Owner Key (MOK)*. Keys can be added and removed in the MOK list by the user, entirely separate from the distro CA key. The mokutil utility can be used to help manage the keys from Linux OS, but changes to the MOK keys may only be confirmed

directly from the console at boot time. This helps remove the risk of OS malware potentially enrolling new keys and therefore bypassing SB [19].

On systems with a TPM chip enabled and supported by the system firmware, shim will extend various PCRs with the digests of the targets it is loading [21]. Certificate hashes are also extended to the TPM, including system, vendor, MOK, and shim denylisted and allowlisted certificate digests.

A.2 Keylime

"Keylime is an open source project hosted by the Cloud Native Computing Foundation (CNCF), a vendor-neutral forum with more than 145 user organizations using cloud native technologies to build their products and services around many of the open source projects, including, for example, Kubernetes. Keylime provides a highly scalable remote boot attestation and runtime integrity measurement solution. Keylime enables users to monitor remote nodes using a hardware based cryptographic root of trust.

Keylime was originally born out of the security research team in MIT's Lincoln Laboratory.

Keylime provides an end-to-end solution for bootstrapping hardware-rooted cryptographic trust for remote machines, the provisioning of encrypted payloads, and run-time system integrity monitoring. It also provides a flexible framework for the remote attestation using a TPM-based hardware root of trust. Users can create their own customized actions that will trigger when a machine fails its attested measurements.

Keylime's goal is to make TPM technology easily accessible to developers and users alike, without the need for a deep understanding of the lower levels of a TPM's operations. Amongst many scenarios, it is well suited to tenants who need to remotely attest machines not under their own full control (such as a customer of hybrid cloud or a remote edge/IoT device in an insecure physical tamper-prone location).

Keylime can be used to monitor an entire fleet of OpenShift worker nodes and take immediate action if any node is compromised. It measures trusted boot of machines and run-time integrity using the Linux Kernels Integrity Measurement Architecture."

This publication is available free of charge from: https://doi.org/10.6028/NIST.IR.8320

Appendix B—Intel Technology Examples

This section describes a number of Intel technology examples that map back to the key concepts described in the various sections of the document.

B.1 Platform Integrity Verification

B.1.1 The Chain of Trust (CoT)

B.1.1.1 Intel Trusted Execution Technology (TXT)

Intel Trusted Execution Technology (TXT) in conjunction with a TPM provides a hardware RoT available on Intel server and client platforms that enables "security capabilities such as measured launch and protected execution" [22]. TXT utilizes *authenticated code modules (ACMs)* that measure various pieces of the CoT during boot time and extend them to the platform TPM [2][22]. TXT's ACMs are chipset-specific signed binaries that are called to perform functions required to enable the TXT environment. An ACM is loaded into and executed from within the CPU cache in an area referred to as the *authenticated code RAM (AC RAM)*. CPU microcode, which acts as the *core root of trust for measurement (CRTM)*, authenticates the ACM by verifying its included digital signature against a manufacturer public key with its digest hard-coded within the chipset. The ACM code, loaded into protected memory inside the processor, performs various tests and verifications of chipset and processor configurations.

The ACMs needed to initialize the TXT environment are the BIOS and the Secure Initialization (SINIT) ACMs. Both are typically provided within the platform BIOS image. The Secure Initialization Authenticated Code Module (SINIT ACM) can be provisioned on disk as well [2][23]. The BIOS ACM is responsible for measuring the BIOS firmware to the TPM and performs additional BIOS-based security operations. The latest version of TXT converged with Intel Boot Guard Technology labels this ACM as the Startup ACM to differentiate it from the legacy BIOS ACM. The SINIT ACM is used to measure the system software or OS to the TPM, and it "initializes the platform so the OS can enter the secure mode of operation" [23].

When the BIOS startup procedures have completed, control is transitioned to the OS loader. In a TXT-enabled system, the OS loader is instructed to load a special module called Trusted Boot before loading the first kernel module [23]. Trusted Boot (tboot) is an open-source, pre-kernel/virtual machine manager (VMM) module that integrates with TXT to perform a measured launch of an OS kernel/VMM. The tboot design typically has two parts: a preamble and the trusted core. The tboot preamble is most commonly executed by the OS loader but can be loaded at OS runtime. The tboot preamble is responsible for preparing SINIT input parameters and is untrusted by default. It executes the processor instruction that passes control to the CPU microcode. The microcode loads the SINIT into AC RAM, authenticates it, measures SINIT to the TPM, and passes control to it. SINIT verifies the platform configuration and enforces any present Launch Control Policies, measuring them and tboot trusted core to the TPM. The tboot trusted core takes control and continues the CoT, measuring the OS kernel and additional modules (like initrd) before passing control to the OS [24].

Intel TXT includes a policy engine feature that provides the capability to specify known good platform configurations. These *Launch Control Policies* (LCPs) dictate which system software is

This publication is available free of charge from: https://doi.org/10.6028/NIST.IR.8320

permitted to perform a secure launch. LCPs can enforce specific platform configurations and tboot trusted core versions required to launch a system environment [23].

B.1.1.2 Intel Boot Guard

Intel Boot Guard provides a hardware RoT for authenticating the BIOS. An original equipment manufacturer (OEM) enables Boot Guard authentication on the server manufacturing line by permanently fusing a policy and OEM-owned public key into the silicon. When an Intel processor identifies that Boot Guard has been enabled on the platform, it authenticates and launches an ACM. The ACM loads the initial BIOS or Initial Boot Block (IBB) into the processor cache, authenticates it using the fused OEM public key, and measures it into the TPM.

If the IBB authenticates properly, it verifies the remaining BIOS firmware, loads it into memory, and transfers execution control. The IBB is restricted to this limited functionality, which allows it to have a small enough size to fit in the on-die cache memory of Intel silicon. If the Boot Guard authentication fails, the system is forced to shut down. When the Boot Guard execution completes, the CoT can continue for other components by means of SB. TXT can be used in conjunction with these technologies to provide a dynamic trusted launch of the OS kernel and software.

Because Boot Guard is rooted in permanent silicon fuses and authenticates the initial BIOS from the processor cache, it provides resistance from certain classes of physical attacks. Boot Guard also uses fuses to provide permanent revocation of compromised ACMs, BIOS images, and input polices.

B.1.1.3 Intel Platform Firmware Resilience (PFR)

Intel Platform Firmware Resilience (PFR) technology is a platform-level solution that creates an open platform RoT based on a programmable logic device. It is designed to provide firmware resiliency (in accordance with NIST SP 800-193 [4]) and comprehensive protection for various platform firmware components, including BIOS, Server Platform Services Firmware (SPS FW), and BMCs. PFR provides the platform owner with a minimal trusted compute base (TCB) under full platform-owner control. This TCB provides cryptographic authentication and automatic recovery of platform firmware to help guarantee correct platform operation and to return to a known good state in case of a malicious attack or an operator error such as a failed update.

NIST SP 800-193 [4] outlines three guiding principles to support the resiliency of platforms against potentially destructive attacks:

- **Protection:** Mechanisms for ensuring that platform firmware code and critical data remain in a state of integrity and are protected from corruption, such as the process for ensuring the authenticity and integrity of firmware updates
- **Detection:** Mechanisms for detecting when platform firmware code and critical data have been corrupted
- **Recovery:** Mechanisms for restoring platform firmware code and critical data to a state of integrity in the event that any such firmware code or critical data is detected to have

This publication is available free of charge from: https://doi.org/10.6028/NIST.IR.8320

been corrupted or when forced to recover through an authorized mechanism. Recovery is limited to the ability to recover firmware code and critical data.

In addition, NIST SP 800-193 [4] provides guidance on meeting those requirements via three main functions of a Platform Root of Trust:

- **Root of Trust for Update,** which is responsible for authenticating firmware updates and critical data changes to support platform protection capabilities; this includes signature verification of firmware updates as well as rollback protections during update.
- **Root of Trust for Detection,** which is responsible for firmware and critical data corruption detection capabilities.
- **Root of Trust for Recovery,** which is responsible for recovery of firmware and critical data when corruption is detected or when instructed by an administrator.

PFR is designed to support NIST guidelines and create a resilient platform that is able to self-recover upon detection of attack or firmware corruption. This includes verification of all platform firmware and configuration at platform power-on time, active protection of platform non-volatile memory at runtime, and active protection of the Serial Peripheral Interface (SPI flash) and system management bus (SMBus). PFR functionality also incorporates monitoring the platform component's boot progress and providing automatic firmware recovery to a known good state upon detection of firmware or configuration corruption. PFR achieves this goal by utilizing a Field-Programmable Gate Array (FPGA) to establish an RoT.

PFR technology defines a special pre-boot mode (T-1) where only the PFR FPGA is active. Intel Xeon processors and other devices that could potentially interfere with the boot process, such as the Platform Controller Hub (PCH)/Manageability Engine (ME) and BMC, are not powered. Boot-critical firmware, like the BIOS, ME, and BMC, is cryptographically verified during T-1 mode. In case of corruption, a recovery event is triggered, and the corrupted firmware in the active regions of the SPI flash is erased and restored with a known-good recovery copy. Once successful, the system proceeds to boot in a normal mode, leveraging Boot Guard for static RoT coverage.

The PFR FPGA RoT leverages a key hierarchy to authenticate data structures residing in SPI flash. The key hierarchy is based on a provisioned Root Key (RK) stored in the NVRAM of the FPGA RoT and a Code Signing Key (CSK) structure, which is endorsed by the RK, stored in the SPI flash, and used for the signing of lower-level data structures. The PFR FPGA uses this CSK to verify the digital signature of the Platform Firmware Manifest, which describes the expected measurements of the platform firmware. The PFR FPGA RoT verifies those measurements before allowing the system to boot. When a recovery is needed, either because measurements do not match the expected value or because a hang is detected during system bootup, the PFR FPGA RoT uses a recovery image to recover the firmware. The recovery image and any update images are stored in a compressed capsule format and verified using a digital signature.

Each platform firmware storage is divided into three major sections: Active, Recovery, and Staging. The Recovery regions, as well as the static parts of the Active regions, are write-protected from other platform components by the PFR FPGA RoT. The Staging region is open to the other platform components for writing in order to provide an area to place digitally signed

This publication is available free of charge from: https://doi.org/10.6028/NIST.IR.8320

and compressed update capsules, which are then verified by the PFR FPGA RoT before being committed to the Active or Recovery regions. The Recovery copy can be updated in T-1 mode once the PFR FPGA has verified the digital signature of the update capsule and confirmed that the recovery image candidate is bootable.

B.1.1.4 Intel Technology Example Summary

There are several technologies that provide different levels of platform integrity and trust. Individual technologies do not provide a complete CoT. When used in combination, they can provide comprehensive coverage all the way up to the OS and VMM layer. Figure 8 outlines the firmware and software coverage of each existing CoT technology example.

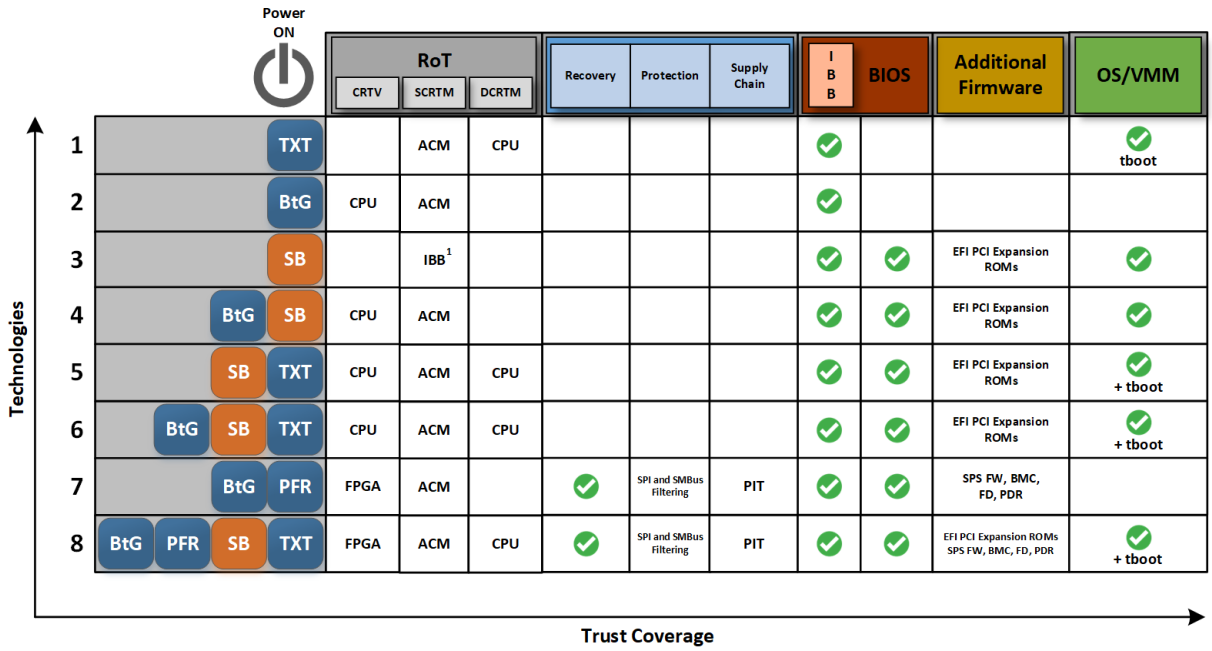


Figure 8: Firmware and Software Coverage of Existing Chain of Trust Technologies

Figure 8 identifies the components of each technology that make up the RoT in their own respective chains and also shows a rough outline of the firmware and software coverage of each technology.

Because many technologies are available, it can be difficult to decide on the correct combination for deployment. Figure 8 illustrates the possible combinations of technologies that extend measurements to a TPM for platform integrity attestation. Note that each combination includes at least one hardware technology to ensure an RoT implementation. A complementary option for extending the CoT up through the OS can also be provided. Including only the hardware technologies would break the CoT by supplying integrity measurements for only pre-OS firmware. Using only SB will use firmware as the RoT that does not have hardware security protections and is much more susceptible to attack. By enabling both parts, the CoT can be extended from a hardware RoT into the OS and beyond.

This publication is available free of charge from: https://doi.org/10.6028/NIST.IR.8320

These combinations will help ensure that appropriate measurements are extended to a TPM for integrity attestation and can prevent a server from booting if specific security modules are compromised. The attestation mechanisms provided by these technologies give cryptographic proof of the integrity of measured components, which can be used to provide visibility into platform security configurations and prove integrity. Note the combination of UEFI SB with TXT in Figure 8. This combination provides the UEFI SB signature verification capability on top of the tboot integrity measurement in the OS/VMM layer.

In addition to attestation, PFR provides verification of platform firmware and automatic recovery of compromised firmware to known good versions. PFR works with any combination of CoT technologies, providing a defense and resilience against firmware attack vectors. Combining a hardware-based firmware resilience technology like PFR with a hardware-based CoT configuration is part of a layered security strategy.

B.1.2 Supply Chain Protection

B.1.2.1 Intel Transparent Supply Chain (TSC)

"Intel Transparent Supply Chain (TSC) is a set of policies and procedures implemented at ODM factories that enable end-users to validate where and when every component of a platform was manufactured" [25]. "Intel TSC tools allow platform manufacturers to bind platform information and measurements using [a TPM]. This allows customers to gain traceability and accountability for platforms with component-level reporting" [26].

Intel TSC provides the following key features [25]:

- Digitally signed statement of conformance for every platform
- Platform certificates linked to a discrete TPM, providing system-level traceability
- Component-level traceability via a direct platform data file that contains integrated components, including a processor, storage, memory, and add-in cards
- Auto Verify tool that compares the snapshot of the direct platform data taken during manufacturing with a snapshot of the platform components taken at first boot
- Firmware load verification
- Conformity with Defense Federal Acquisition Regulation Supplement (DFARS) 246.870-2 [27]

B.1.2.2 Intel PFR with Protection in Transit (PIT)

In addition to the platform protection, detection, and recovery features, PFR also offers protection in transit (PIT) or supply chain protection. Platform lockdown requires that a password be present in the PFR FPGA as well as a radio frequency (RF) component. The password is removed before platform shipment and must be replaced before the platform will be allowed to power up. With platform firmware sealing, the PFR FPGA computes hashes of platform firmware in the PCH and BMC attached flash devices, including static and dynamic regions, and stores them in an NVRAM space before shipment. Upon delivery, the PFR FPGA

This publication is available free of charge from: https://doi.org/10.6028/NIST.IR.8320

will recompute the hashes and report any mismatches to ensure that the firmware has not been tampered with during system transit.

B.2 Software Runtime Protection Mechanisms

B.2.1 Return Oriented Programming (ROP) and Call/Jump Oriented Programming (COP/JOP) Attacks

B.2.1.1 Intel Control-Flow Enforcement Technology (Intel CET)

Intel CET is an instruction set extension to implement control flow integrity and defend against ROP and COP/JOP style subversion attacks. ROP and similarly COP/JOP have been the prevalent attack methodology for stealth exploit writers targeting vulnerabilities in programs. [28]

Intel CET prevents this class of exploits by providing the following capabilities:

- Shadow stack – return address protection to defend against ROP
- Indirect branch tracking – free branch protection to defend against COP/JOP

"CET introduces a shadow stack system to detect and thwart the stack manipulation required by ROP" [29]. This second stack is used exclusively for control transfer operations and is designed to be protected from application code memory accesses while keeping track of CPU stored copies of the return addresses [30]. "When CET is enabled, a CALL instruction pushes the return address into a shadow stack in addition to its normal behavior of pushing return address into the normal stack (no changes to traditional stack operation). The return instructions (e.g. RET) pops return address from both shadow and traditional stacks, and only transfers control to the popped address if return addresses from both stacks match. […] The page table protections for shadow stack are also designed to protect the integrity of the shadow stack by preventing unintended or malicious switching of shadow stack and/or overflow and underflow of shadow stack." [31]

"CET also adds an indirect branch tracking capability to provide software the ability to restrict COP/JOP attacks." [30] This ENDBRANCH instruction is a new addition to Intel Instruction Set Architecture (ISA). It marks legal targets for an indirect branch or jump, forcing the CPU to generate an exception for unintended or malicious operations [31].

"Intel has been actively collaborating with Microsoft and other industry partners to address control-flow hijacking by using Intel's CET technology to augment the previous software-only control-flow integrity solutions. Intel's CET, when used properly by software, is a big step in helping to prevent exploits from hijacking the control-flow transfer instructions." [31] A security analysis of Intel CET is published in [32].

B.2.2 Address Translation Attacks

B.2.2.1 Intel Hypervisor Managed Linear Address Translation (HLAT)

Hypervisor managed linear address translation (HLAT) is a capability to enable Intel Virtualization Technology (Intel VT-x)-based security monitors to enforce runtime protection

This publication is available free of charge from: https://doi.org/10.6028/NIST.IR.8320

and integrity assertions on OS-managed page tables. This helps protect kernel assets, as well as in-band security agents and agent-monitored assets from OS page-table attacks.

"[HLAT] is intended to be used by a Hypervisor/Virtual Machine Monitor (VMM) to enforce guest linear translation (to guest physical mappings). When combined with the existing Extended Page Table (EPT) capability, HLAT enables the VMM to ensure the integrity of combined guest linear translation (mappings and permissions) cached by the processor TLB, via a reduced software TCB managed by the VMM." [33] In this fashion, the VMM-enforced guest translations are more protected from alterations by untrusted system software adversaries. [33]

"This feature is intended to augment the security functionality for a type of Virtual Machine Monitor (VMM) that may use legacy EPT read/write/execute (XWR) permission bits (bits 2:0 of the EPTE) as well as the new user-execute (XU) access bit (bit 10 of the EPTE) to ensure the integrity of code/data resident in guest physical memory assigned to the guest OS. EPT permissions are also used in these VMMs to isolate memory; for example, to host a Secure Kernel (SK) that can manage security properties for the general purpose kernel (GPK). For such usages, it is important that the VMM ensure that the guest linear address mappings which are used by the General Purpose Kernel to refer to the EPT monitored guest physical pages are access-controlled as well." [33]

"VMMs could enforce the integrity of these specific guest linear to guest physical mappings (paging structures) by using legacy EPT permissions to mark the guest physical memory containing the relevant guest paging structures as read-only. The intent of marking these guest paging structures as read-only is to ensure an invalid mapping is not created by guest software. However, such page-table edit control techniques are known to cause very high overheads due to the requirement that the VMM must monitor all paging contexts created by the (Guest) operating system. HLAT enables a VMM to enforce the integrity of guest linear mappings without this high overhead." [33]

HLAT utilizes a processor mechanism that implements an alternate Intel Itanium architecture (IA) paging structure managed in guest physical memory by a Secure Kernel. This paging structure contains guest linear to guest physical translations that the VMM/Secure Kernel wants to enforce.

Additionally, to accommodate legacy page-table monitoring approaches, HLAT defines two new EPT control bits in EPT leaf entries. A "Paging-Write" control bit specifies which guest physical pages hold HLAT or legacy IA paging structures. This allows the processor to use the Paging-Write as permission to perform A/D bit writes, instead of the software W permission in the EPTE. A "Verify Paging-Write" control bit specifies which guest physical pages should only be referenced via translation (guest) paging structures marked as Paging-writable under EPT [33].

B.2.2.2 Intel Supervisor Mode Execution Prevention (SMEP) and Supervisor Mode Access Prevention (SMAP)

Supervisor Mode Execution Prevention (SMEP) and Supervisor Mode Access Prevention (SMAP) are opt-in capabilities that can be used by systems software (such as the kernel) to harden the privilege separation between user-mode and kernel-mode. These capabilities further

This publication is available free of charge from: https://doi.org/10.6028/NIST.IR.8320

enforce the user/supervisor properties specified via address translation mechanisms by mitigating malicious code execution or malicious use of data setup by processes executing in user-mode.

Intel OS Guard, also known as SMEP, helps prevent execution out of untrusted application memory while operating at a more privileged (supervisor) level. "[When] enabled, the operating system will not be allowed to directly execute application code, even speculatively. This makes branch target injection attacks on the OS substantially more difficult by forcing the attacker to find gadgets within the OS code. It is also more difficult for an application to train OS code to jump to an OS gadget. All major operating systems enable SMEP support by default." [34]

SMAP is a security feature that helps prevent unauthorized kernel consumption of data accessible to user space [35]. An enabling SMAP bit in the CR4 control register will cause a page fault to be triggered when there is any attempt to access user-space memory while running in a privileged mode. When access to user space memory is needed by the kernel, a separate AC flag is toggled to allow the required access [36]. "Two new instructions (STAC and CLAC) are provided to manipulate that flag relatively quickly." When the AC flag is set in protection mode under normal operating circumstances, SMAP blocks a whole class of exploits where the kernel is fooled into reading from (or writing to) user-space memory by mistake. SMAP also allows for the early discovery of kernel bugs where developers dereference user space pointers directly from the kernel [36].

B.3 Data Protection and Confidential Computing

B.3.1 Memory Isolation

B.3.1.1 Intel TME and Intel Multi-Key TME (Intel MKTME)

Intel Total Memory Encryption (Intel TME) provides the capability to encrypt the entire physical memory of a system. This capability is typically enabled in the very early stages of the boot process with a small change to the BIOS. Once this change is configured and locked, all data on the external memory buses of a CPU and any additional DIMMs will be encrypted using 128-bit keys utilizing the NIST standard AES-XTS algorithm. The encryption key used for Intel TME uses a hardware RNG implemented in the Intel CPU, and the keys are not accessible by software or by using external interfaces to the CPU. The architecture is flexible and will support additional memory protection schemes in the future. Intel TME is intended to support unmodified existing system and application software. The overall performance impact of TME is likely to be relatively small and highly dependent on workload.

Intel Multi-Key Total Memory Encryption (Intel MKTME) builds on Intel TME and adds support for multiple encryption keys. The CPU implementation supports a fixed number of encryption keys, and software can configure a CPU to use a subset of available keys. Software manages the use of keys and can use each of the available keys for encrypting any page of the memory. Thus, Intel MKTME allows page granular encryption of memory. By default, Intel MKTME uses the Intel TME encryption key unless explicitly specified by software.

In addition to supporting a CPU-generated ephemeral key (not accessible by software or by using external interfaces to a CPU), Intel MKTME also supports software-provided keys. Software-provided keys are particularly useful when used with nonvolatile memory, when combined with

This publication is available free of charge from: https://doi.org/10.6028/NIST.IR.8320

attestation mechanisms or used with key provisioning services. An OS may be enabled to take additional advantage of the Intel MKTME capability, both in native and virtualized environments. When properly enabled, Intel MKTME is available to each guest OS in a virtualized environment, and the guest OS can take advantage of Intel MKTME in the same ways as a native OS.

B.3.2 Application Isolation

B.3.2.1 Intel Software Guard Extensions (SGX)

Intel Software Guard Extensions (SGX) is a set of instructions that increases the security of application code and data. Developers can partition security-sensitive code and data into an *SGX enclave*, which is executed in a CPU protected region. The developer creates and runs SGX enclaves on server platforms where only the CPU is trusted to provide attestations and protected execution environments for enclave code and data. SGX also provides an enclave remote attestation mechanism. This mechanism allows a remote provider to verify the following [37]:

1. The enclave is running on a real Intel processor inside an SGX enclave.
2. The platform is running at the latest security level (also referred to as the *TCB version*).
3. The enclave's identity is as claimed.
4. The enclave has not been tampered with.

Once all of this is verified, the remote attester can then provision secrets into the enclave. SGX enclave usage is reserved for Ring-3 applications and cannot be used by an OS or BIOS driver/module.

SGX removes the privileged software (e.g., OS, VMM, System Management Mode devices) and unprivileged software (e.g., Ring-3 applications, VMs, containers) from the trust boundary of the code running inside the enclave, enhancing security of sensitive application code and data. An SGX enclave trusts the CPU for execution and memory protections. SGX encrypts memory to help protect against memory bus snooping and cold boot attacks for enclave code and data in host Dynamic Random Access Memory (DRAM). SGX includes ISA instructions that can be used to handle Enclave Page Cache page management for creating and initializing enclaves.

SGX relies on the system UEFI BIOS and OS for initial provisioning, resource allocation, and management. However, once an SGX enclave starts execution, it is running in a cryptographically isolated environment separate from the OS and BIOS.

SGX can allow any application (whole or part of) to run inside an enclave and puts application developers in control of their own application security. However, it is recommended that developers keep the SGX code base small, validate the entire system (including software side channel resistance), and follow other secure software development guidelines.

SGX enclaves can be used for applications ranging from protecting private keys and managing security credentials to providing security services. In addition, industry security standards, like European Telecommunications Standards Institute (ETSI) Network Functions Virtualization (NFV) Security (ETSI NFV SEC) [38], have defined and published requirements for Hardware

This publication is available free of charge from: https://doi.org/10.6028/NIST.IR.8320

Mediated Execution Enclaves (HMEEs) for the purposes of NFV, 5G, and edge security. SGX is an HMEE.

B.3.3 VM Isolation

B.3.3.1 Intel Trust Domain Extensions (Intel TDX)

Intel Trust Domain Extensions (Intel TDX) introduces new architectural elements to deploy hardware-isolated VMs called trust domains (TDs). Intel TDX is designed to isolate VMs from the VMM/hypervisor and any non-TD software on the platform to protect TDs from a broad range of software. TDX is built using a combination of Virtual Machine Extensions, (VMX) ISA extensions, MKTME technology, and a CPU-attested software module called the TDX-SEAM module. TDX isolates VMs from many hardware threats and most software-based threats, including from the VMM and other CSP software. TDX helps give the cloud tenant control of its own data security and intellectual property protection. TDX does this while maintaining the CSP role of managing resources and cloud platform integrity.

The TDX solution provides the following capabilities to TDs to address the security challenges:

- Memory and CPU state confidentiality and integrity to help keep the sensitive IP and workload data secure from most software-based attacks and many hardware-based attacks. The workload now has a tool that supports excluding the firmware, software, devices, and operators of the cloud platform from the TCB. The workloads can use this tool to foster more secure access to CPU instructions and other CPU features. The workload can have this ability irrespective of the cloud infrastructure used to deploy the workload.
- Remote attestation enables a relying party (either the owner of the workload or a user of the services provided by the workload) to establish that the workload is running on a TDX-enabled platform located within a TD prior to providing that workload data. Remote attestation aims to allow the owners and customers of the service to digitally determine the version of the TCB they are relying on to help secure their data. The VMM remains the platform resource manager, and TDs should not cause denial of service to the VMM. Defending TDs against denial of service by the VMM is not a goal.

TDX also augments defense of the TD against limited forms of attacks that use physical access to the platform memory, such as offline, DRAM analysis (example: cold-boot attacks), and active attacks of DRAM interfaces, including capturing, modifying, relocating, splicing, and aliasing memory contents [39]. The VMM continues to be the resource manager, and TDs do not have privileges to deny service to the VMM.

B.3.4 Cryptographic Acceleration

B.3.4.1 Intel Advanced Encryption Standard New Instructions (Intel AES-NI)

Intel AES New Instructions (Intel AES-NI) is an encryption instruction set that improves hardware performance of the Advanced Encryption Standard (AES) algorithm and accelerates data encryption. Intel AES-NI consists of seven new instructions that accelerate encryption and decryption and improve key generation and matrix manipulation, all while aiding in carry-less

This publication is available free of charge from: https://doi.org/10.6028/NIST.IR.8320

multiplication. This minimizes application performance concerns inherent in conventional cryptographic processing and helps provide enhanced security by addressing side channel attacks on AES associated with conventional software methods of table lookups [40].

AES is the most widely used standard for protecting network traffic, personal data, and corporate IT infrastructures. By implementing certain intensive sub-steps of the AES algorithm into the hardware, Intel AES-NI strengthens and accelerates execution of the AES application [40].

B.3.4.2 Intel QuickAssist Technology (QAT) with Intel Key Protection Technology (KPT)

Intel QuickAssist Technology (QAT) is a high-performance hardware accelerator for performing cryptographic, security, and compression operations. Applications like VMs, containers, and Function as a Service call Intel QAT using industry-standard OpenSSL, TLS, and Internet Protocol Security (IPsec) interfaces to offload symmetric and asymmetric cryptographic operations. Cloud, multi-tenancy, NFV, edge, and 5G infrastructures and applications are best suited for QAT for all types of workloads, including software-defined networks, content delivery networks, media, and storage [41].

Intel Key Protection Technology (KPT) helps enable customers to secure their keys to be used with QAT through a bring-your-own-key paradigm. KPT allows customers to deliver their own cryptographic keys to the QAT device in the target platform where their workload is running. KPT-protected keys are never in the clear in host DRAM or in transit. The customers encrypt their workload key (e.g., RSA private key for Nginx) using KPT inside their HSMs. This encrypted workload key is delivered to the target QAT platform, where it is decrypted immediately prior to use. KPT provides key protection at rest, in transit, and while in use [42].

B.3.5 Technology Example Summary

Cloud infrastructure creates improvements in the efficiency, agility, and scalability of data center workloads by abstracting hardware from the application layer. This introduces new security concerns as workloads become multi-tenant, attack surfaces become shared, and infrastructure administrators from the cloud operator gain access to underlying platforms. Isolation techniques provide answers to these concerns by adding protection to VMs, applications, and data during execution, and they represent a crucial layer of a layered security approach for data center security architecture.

Various isolation techniques exist and can be leveraged for different security needs. Full memory isolation defends a platform against physical memory extraction techniques, while the same technology extended with multiple keys allows individual VMs or platform tenants to have uniquely encrypted memory. Future generations of these technologies will allow full memory isolation of VMs, protecting them against malicious infrastructure insiders, multi-tenant malware, and more. Application isolation techniques allow individual applications to create isolated enclaves that require implicit trust in the platform CPU and nothing else and that have the ability to provide proof of the enclave to other applications before data is sent.

This publication is available free of charge from: https://doi.org/10.6028/NIST.IR.8320

B.4 Remote Attestation Services

B.4.1 Intel Security Libraries for the Data Center (ISecL-DC)

ISecL-DC is an open-source remote attestation implementation of a set of building blocks that utilize Intel Security features to discover, attest, and enable critical foundation security and confidential computing use-cases. This middleware technology provides a consistent set of application programming interfaces (APIs) for easy integration with cloud management software and security monitoring and enforcement tools. ISecL-DC applies the remote attestation fundamentals described in this section and standard specifications to maintain a platform data collection service and an efficient verification engine to perform comprehensive trust evaluations. These trust evaluations can be used to govern different trust and security policies applied to any given workload, as referenced in the workload scheduling use case in Section 7.2. In future generations, the product will be extended to include TEE attestation to provide assurance and validity of the TEE to enable confidential computing [43].

B.4.2 Technology Summary

Platform attestation provides auditable foundational reports for server firmware and software integrity and can be extended to include the location of other asset tag information stored in a TPM, as well as integrity verification for applications installed on the server. These reports provide visibility into platform security configurations and can be used to control access to data and workloads. Platform attestation is performed on a per-server basis and typically consumed by cloud orchestration or a wide variety of infrastructure management platforms.

TEE attestation provides a mechanism by which a user or application can validate that a genuine TEE enclave with an acceptable TCB is actually being used before releasing secrets or code to the TEE. Formation of a TEE enclave is performed at the application level, and TEE attestations are typically consumed by a user or application requiring evidence of enclave security before passing secrets.

These different attestation techniques serve complementary purposes in a cloud deployment in the data center or at the edge computing facility.

This publication is available free of charge from: https://doi.org/10.6028/NIST.IR.8320

Appendix C—AMD Technology Examples

This section describes a number of AMD technology examples that map back to the key concepts described in the various sections of the document.

C.1 Platform Integrity Verification

C.1.1 AMD Platform Secure Boot (AMD PSB)

AMD Platform Secure Boot (AMD PSB) provides a hardware RoT to authenticate the initial Platform BIOS code during the boot process of the server. Manufacturers of server systems, like OEMs or original device manufacturers (ODMs), enable the functionality of AMD PSB in their manufacturing flow by permanently fusing policy into the silicon.

The OEM or ODM's final BIOS image contains the AMD public key and the OEM BIOS-signing public key (signed with the AMD private key). When a system powers on, the AMD Security Processor (ASP) starts executing the immutable on-chip Boot ROM. It authenticates and loads multi-stage ASP Boot Loaders from SPI/Low Pin Count Flash into its internal memory, which initializes the silicon and the system memory.

Once the system memory is initialized, the ASP Boot Loaders load and authenticate the OEM BIOS-signing public key, followed by authenticating the initial BIOS code. Once the verification is successful, ASP releases the x86 core to execute authenticated initial BIOS code. The BIOS can continue CoT for other components by means of SB. If PSB authentication fails, the system is forced to shut down.

AMD PSB supports revocation and rollback protection of BIOS images through the OEM BIOS-signing key revision ID and rollback protection.

C.2 Data Protection and Confidential Computing

C.2.1 Memory Isolation

C.2.1.1 AMD Secure Memory Encryption (SME)/Transparent Secure Memory Encryption (TSME)

AMD Secure Memory Encryption (SME) is a memory-encryption technology from AMD which helps protect data in DRAM by encrypting system memory content [44]. When enabled, memory content is encrypted via dedicated hardware in the on-die memory controllers. Each controller includes a high-performance AES engine that encrypts data when it is written to DRAM and decrypts it when read. The encryption of data is done with an encryption key in a mode that utilizes an additional physical address-based tweak to protect against ciphertext block move attacks.

The encryption key used by the AES engine with SME is randomly generated on each system reset and is not visible to any software running on the CPU cores. This key is managed entirely by the AMD Secure Processor that functions as a dedicated security subsystem integrated within the AMD System-on-Chip (SOC). The key is generated using the onboard NIST SP 800-90

This publication is available free of charge from: https://doi.org/10.6028/NIST.IR.8320

compliant hardware RNG and is stored in dedicated hardware registers where it is never exposed outside the SOC in the clear.

Two modes of memory encryption are supported for various use cases. The simplest mode is Transparent Secure Memory Encryption (TSME), which is a BIOS option and enables memory encryption automatically on all memory accesses. TSME works in the background and requires no software interaction. Another supported mode is the OS-managed Secure Memory Encryption (SME) mode in which individual pages of memory may be marked for encryption via CPU page tables. SME provides additional flexibility if only a subset of memory needs to be encrypted but does require appropriate software support.

Encrypted memory provides strong protection against cold boot, DRAM interface snooping, and similar types of attacks.

C.2.2 VM Isolation

C.2.2.1 AMD Secure Encrypted Virtualization (SEV)

The AMD Secure Encrypted Virtualization (SEV) feature is designed to isolate VMs from the hypervisor. When SEV is enabled, individual VMs are encrypted with an AES encryption key. When a component such as the hypervisor attempts to read memory inside a guest, it is only able to see the data in its encrypted form. This provides strong cryptographic isolation between the VMs, as well as between the VMs and the hypervisor.

To protect SEV-enabled guests, the SEV firmware assists in the enforcement of three main security properties: authenticity of the platform, attestation of a launched guest, and confidentiality of the guest's data.

Authenticating the platform prevents malicious software or a rogue device from masquerading as a legitimate platform. The authenticity of the platform is proven with its identity key. This key is signed by AMD to demonstrate that the platform is an authentic AMD platform with SEV capabilities.

Attestation of the guest launch proves to guest owners that their guests securely launched with SEV enabled. A signature of various components of the SEV-related guest state, including initial contents of memory, is provided by the firmware to the guest owner to verify that the guest is in the expected state. With this attestation, a guest owner can ensure that the hypervisor did not interfere with the initialization of SEV before transmitting confidential information to the guest.

Confidentiality of the guest is accomplished by encrypting memory with a memory encryption key that only the SEV firmware knows. The SEV management interface does not allow the memory encryption key or any other secret SEV state to be exported outside of the firmware without properly authenticating.

AMD SEV has two additional modes:

- SEV With Encrypted State (SEV-ES): This mode encrypts and protects VM registers from being read or modified by a malicious hypervisor or VM [45].

This publication is available free of charge from: https://doi.org/10.6028/NIST.IR.8320

- SEV with Secure Nested Paging (SEV-SNP): This mode adds strong memory integrity protection to help prevent malicious hypervisor-based attacks like data replay and memory remapping. [46]

This publication is available free of charge from: https://doi.org/10.6028/NIST.IR.8320

Appendix D—Arm Technology Examples

This section describes a number of Arm technology examples that map back to the key concepts described in the various sections of the document.

D.1 Platform Integrity Verification

In order to understand the Secure Boot environment, the Security states and Exception Levels implemented by TrustZone must first be understood. The following sections describe the technologies available for A-profile Arm processors.[2] [47] [48]

D.1.1 Arm TrustZone Trusted Execution Environment (TEE) for Armv8-A

D.1.1.1 The Normal (Non-Secure) World and Secure World

TrustZone [49] provides two execution environments built into the processor with system-wide hardware enforced isolation between them. There are two Security states: secure and non-secure. They map to the Secure world (SW) and the Normal world (NW, also sometimes referred to as the Non-Secure world), respectively. Each processor implements both worlds but can execute in only one world at any given time, independently of which world each of the other processors in a multi-processor implementation is executing. For example, core0 might be executing in the NW, while core1 is executing in the SW, core2 is executing in the NW, etc., all concurrently. The SW and NW concept extends beyond the processor to include memory, software, bus transactions, interrupts, and peripherals within an SoC.

The NW runs a Rich Execution Environment (REE), which typically includes a large number of applications, a complex OS (e.g., Linux), and often a hypervisor. The REE presents a broad attack surface. The SW provides a TEE, which runs a smaller and simpler software stack than the REE. The TEE may include several trusted services, a lightweight kernel, and, if the processor supports Secure Exception Level 2 (SEL2, explained in Appendix D.1.1.2), a simple hypervisor. The TEE has a much smaller attack surface and does not run arbitrary code, making it much less vulnerable to attack compared to the REE. Also, the SELs provide additional protection within the TEE.

As shown in Figure 9, there is a hardware-enforced isolation boundary between the NW and the SW. For A-class processors, the NW requests a secure service from the SW by issuing a Secure Monitor Call (SMC) to effect the transition from the NW to the SW and back via the Secure Monitor, which runs in the SW at the highest exception level, EL3. The Secure Monitor hands the information to the Secure Partition Manager (SPM) executing at SEL2, which invokes the secure service in a Trusted Application (TA) running in a Secure Partition (SP).

The SW and NW address spaces can be split into several regions. Each region is specified as secure or non-secure. The registers to control the address space partitioning are limited to SW access only, ensuring that only the SW software can partition memory. The SW and NW

[2] This description also applies to the recently announced Armv9-A Architecture.

This publication is available free of charge from: https://doi.org/10.6028/NIST.IR.8320

memory partitioning is not expected to change dynamically at runtime because it is only configured once during system boot,[3] which always takes place in the secure state.

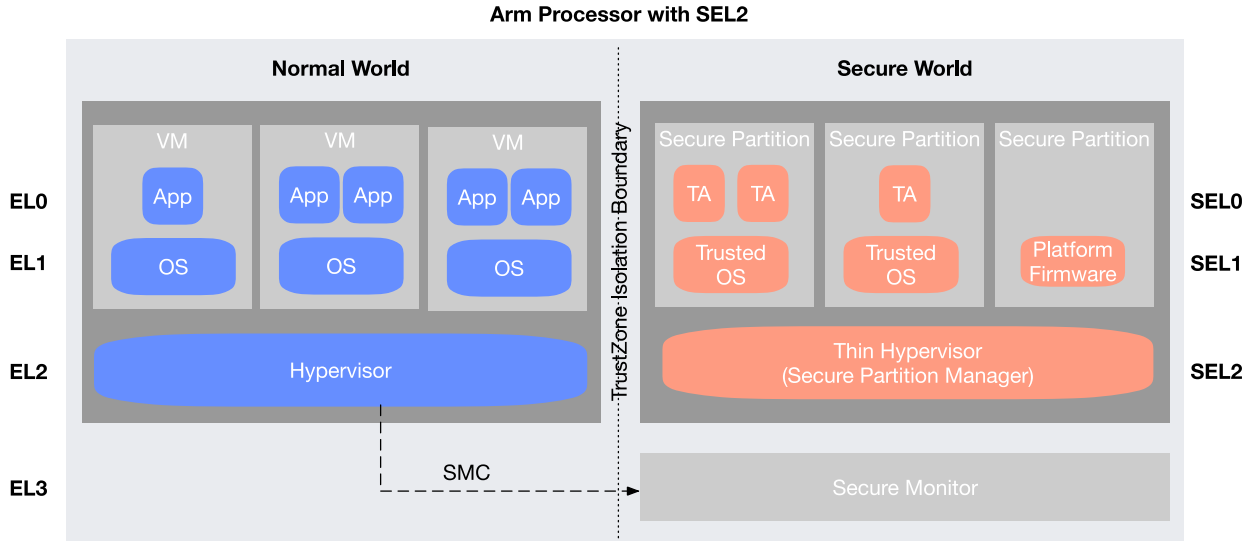


Figure 9: Arm Processor with TrustZone

The architecture provides two physical address spaces (PAS): secure (SW) and non-secure (NW). While in non-secure state, virtual addresses always translate to non-secure physical addresses. Software in non-secure state can only access non-secure resources. In secure state, firmware running at Exception Level 3 (EL3) and SEL2 can access both the secure and non-secure physical address spaces. SEL0 and SEL1 may access a non-secure physical address if it has been mapped to the corresponding page table entry by SEL2 for an SP. When executing in the SW, code is never fetched or executed from the NW address space.

Input/output (I/O) devices can be assigned to the SW or the NW. Memory-mapped devices follow the same access rules as described for memory accesses.

D.1.1.2 Exception Levels

In the NW, there are three Exception Levels (ELs):

- **EL0** is the EL where applications execute in the NW. They have visibility to their application address space created by the OS running at EL1. EL0 is the least privileged EL.
- **EL1** is the EL at which the OS executes. The OS manages the application address spaces that it creates at EL0 and owns all of the memory assigned to it. The OS may be a bare-metal OS, which runs on the hardware directly, or it may be located in a VM created by a hypervisor.

[3] Before CCA is introduced, TF-A does not define dynamic memory transfer between the two worlds. CCA does support this operation (see Appendix D.3).

This publication is available free of charge from: https://doi.org/10.6028/NIST.IR.8320

- **EL2** is the EL at which the hypervisor executes. It owns and manages the NW memory, and it manages the VMs. A VM is composed of an OS running at EL1 and the applications it creates that run at EL0. The VM may also have non-secure devices assigned to it.

Higher ELs (i.e., with a larger EL number) have the privilege to access registers that control lower ELs. In the general operation of the system, the privileged ELs will usually control their own configuration. However, more privileged ELs will sometimes access registers associated with lower ELs to, for example, read and write the register set as part of a save-and-restore operation during a context switch or power management operation. EL1 and EL0 share the same MMU configuration, and control is restricted to privileged code running at EL1.

The NW ELs, EL0 through EL2, are mirrored in the SW and serve similar purposes:

- **SEL0** is the SEL where TAs execute in the SW. They provide secure services to the NW (e.g., Platform Security Architecture [PSA] crypto services, Digital Rights Management, a firmware TPM, a secure interface to a shared hardware device such as a discrete TPM or an HSM). TAs have visibility to their application address space created by the Trusted Operating System (TOS), e.g., OP-TEE, at their inception. One SEL0 TA cannot access any other SEL0 TA memory unless the memory is specifically mapped as shared by the SEL1 TOS. SEL0 is the least privileged SEL in the SW.
- **SEL1** is the SEL at which the TOS executes, shown in an SP in Figure 9. The TOS manages the application address spaces and owns all of the memory assigned to the SP. Alternatively, vendor-provided platform firmware may execute in its own SP at SEL1; this is referred to as a bare-metal SP. Each SP represents a separate security domain within the SW and is essentially the SW equivalent of an NW VM.
- **SEL2** is the EL at which the SPM executes. It is a simple hypervisor and manages the SPs and their address spaces. It routes messages to and from the SPs. The separation of address spaces also allows moving manufacturer-provided platform firmware that, in the previous implementation without SEL2 ran at EL3, to an isolated SP running at SEL1 (shown in Figure 9).

EL3 is a special EL [49]. EL3 always executes in the SW. It manages the transition between worlds, and it runs the firmware that provides the Secure Monitor services, including:

- Initial boot (BL2) execution
- SMC intercept and dispatcher, which handles incoming SMCs and routes them
- Maintains the non-secure to secure isolation and memory
- Power State Coordination Interface – low-level power management
- System Control and Management Interface – OS-independent software interfaces used in system management
- Reliability, availability, and serviceability error delivery
- Software Delegated Exception Interface – provides a high-priority event delivery mechanism, which has higher priority than interrupts that target OSs and hypervisors

This publication is available free of charge from: https://doi.org/10.6028/NIST.IR.8320

Trusted Firmware provides firmware support for dealing with Arm System Intellectual Property (IP), like interconnects. Silicon Providers (SiPs) provide firmware support to handle custom or third-party IP. This includes SoC-specific power management.

See Appendix D.3.1 for the extensions added to this architecture by CCA.

D.1.1.3 Trusted Applications and Secure Services

TAs in SPs are intended to run only for short periods of time so as not to block the NW from executing for long on a given processor. The SW is not intended to run general-purpose applications, but rather secure services like those previously described for SEL1. For the most part, these secure services are expected to be global services offered to the NW apps and to live for the life of the boot. However, their use may be restricted to specific NW applications by implementing an authentication mechanism in the SW to ensure that the NW requester is authorized to use the service.

Scheduling of the SW is on a per-processor basis and is implemented via a secure interrupt handled by the Secure Monitor at EL3. Explicit calls for services such as Platform Management Communications Infrastructure, for example, are typically blocking on that processor; control will only be returned to the Non-Secure state when the requested operation is complete. However, these calls tend to be short and infrequent. SMC calls for secure services from a TA are scheduled by the host OS/hypervisor. Most secure service requests will be short. However, if the TA needs to run for an extended period, the SMC Calling Convention allows the TA to return control to the NW, where the host OS (e.g., Linux) can reschedule the SW by reissuing the SMC at a convenient time. The SW will pick up where it was. This cooperative scheduling approach allows the NW to control scheduling as needed.

D.1.1.4 Debugging, Tracing, and Profiling

Arm systems include extensive features to support debugging, tracing, and profiling. The SoC needs to be configured properly to ensure that these features cannot be used to compromise the security of the system. Different developers are trusted to debug different parts of the system during different stages of its security lifecycle. Rate signals to control use of the features in Secure state and Non-Secure state are available to address these requirements.

D.1.2 Arm Secure Boot and the Chain of Trust (CoT)

The boot firmware can only be trusted if all the software components that run in the boot flow are trusted. This is referred to as the Chain of Trust. Trusted Firmware-A[4] (TF-A) [12] implements boot firmware that loads, authenticates, and verifies each load of boot code before transferring control to it. Verification ensures the integrity of the boot firmware and critical data

[4] TF-A provides a reference implementation of secure world software. It is an open governance community project hosted by Linaro. Support for A-Profile Arm processors (Cortex and Neoverse) is currently available as open source at https://git.trustedfirmware.org/TF-A/trusted-firmware-a.git/. Functionality focuses on trusted boot and a small, trusted runtime (EL3 code).

This publication is available free of charge from: https://doi.org/10.6028/NIST.IR.8320

by detecting that it has not been corrupted or modified in any way and is authentic. In this way, the boot firmware establishes a complete cascading CoT.

On Arm, Secure Boot[5] takes place within the SW provided by the Arm Processor Element (PE) TrustZone implementation (see Appendix D.3.1.3). When a power-on/reset processor event occurs, the system begins to execute the boot ROM[6] code at EL3 on the boot core.[7] The immutable ROM provides the hardware RoT for the processor complex and is implicitly trusted.

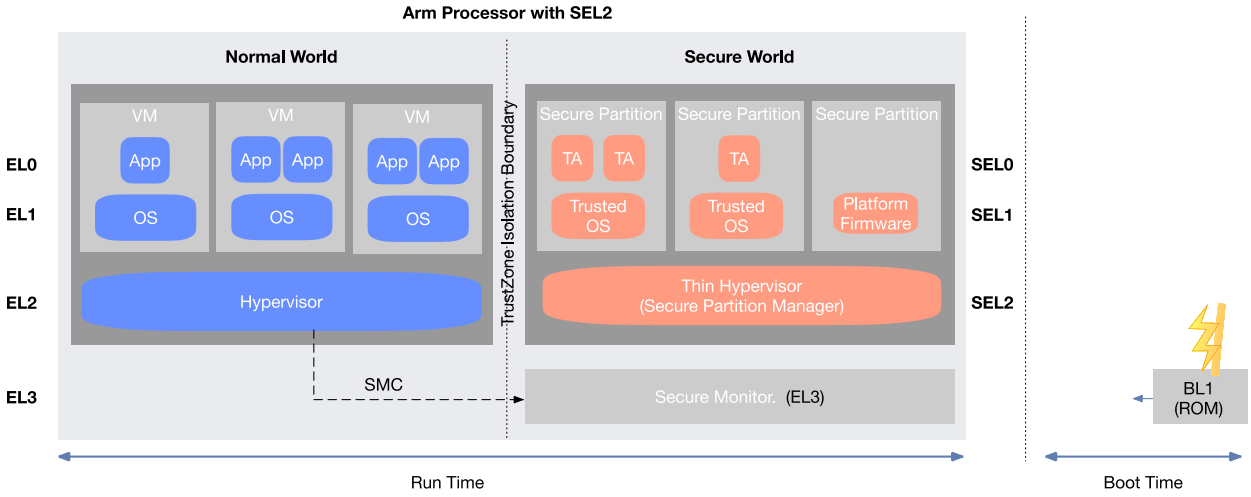


Figure 10: Boot-Time and Run-Time Firmware

During manufacture, the SiP and the OEM or ODM provide the ROM code (BL1 in Figure 10) and the first mutable load of boot code (BL2). The boot ROM code is typically small and simple. Its main function is to load the second-stage boot code (BL2), the first mutable boot code, from the non-volatile firmware storage device, typically flash, and verify it using the immutable root of trust public key provisioned at manufacture. The specific public key algorithm used for authentication is defined by the implementation. This process ensures first instruction integrity. If BL2 passes verification, BL1 transfers control to it. BL2 then performs some system initialization of the platform, like setting up the memory controller for off-chip DRAM, for example. BL2 then loads and cryptographically verifies all subsequent loads of boot, including the EL3 Boot and Runtime firmware (BL3-1); the SEL2, SEL1, and SEL0 firmware (BL3-2); and the first non-secure load of boot (BL3-3 – e.g., UEFI, U-Boot), which executes at EL2 when it receives control. BL2 uses Key and Content self-signed X.509 certificates to verify the BL3-x loads. BL3-3 is the end of the boot chain and represents the boot CoT.

---

5 In this section, references to "Secure Boot" include Verified Boot, where each boot code module is authenticated after loading it from the boot media, typically flash, to make sure that it has not been modified or corrupted, before transferring control to it, and Measured Boot, where each boot code module is measured using a hash function over the code and configuration data and is extended into a PCR. Verified boot and measured boot are complementary. UEFI Secure Boot, which continues the verification and measurement, is also included in the SRTM for Arm systems.

6 Boot ROMs are typically implemented as either mask ROM, or by embedded flash with hardware support to ensure that it cannot be altered once programmed. The design of the immutable first load of boot is not restricted to specific implementations. Only the architectural requirement that it is immutable must be met.

7 The other cores are held in reset until boot completes. This serialization avoids any security vulnerabilities that would be created due to concurrent execution of boot code on multiple processors.

This publication is available free of charge from: https://doi.org/10.6028/NIST.IR.8320

The boot process can optionally measure all the boot firmware up to and including BL3-3. BL3-3, when it executes, can also provide measurements for firmware it loads (e.g., runtime drivers). This is an optional feature. The process described above is changed so that the boot firmware is loaded, verified, then measured before transferring control. For Arm, the measurements are extended into PCRs: PCR0 (all signed firmware and data) and PCR1 (all critical configuration data – e.g., debug port settings). The PCR implementation is platform-defined and is hidden by a firmware API that provides only read and extend operations. The PCR implementation need only guarantee that the architectural behavior and security of measurement PCRs are adhered to. For example, the PCRs may be implemented on-chip in a protected memory location or in a firmware TPM implementation, or in an external device like a discrete TPM or a Secure Element. PCR0 and PCR1 are cleared to zero at reset.

The boot measurements can be used to attest to the firmware that was booted on the system. An attestation key, also known as an endorsement key (EK), cryptographically proves the device identity, and therefore trustworthiness, to external entities. A device attestation key can be used by many different attestation schemes (e.g., the Fast Identity Online [FIDO] Alliance, the TCG TPM, Platform Security Architecture [PSA], and CCA). A remote verifier can be used to compare the attested measurement against a list of known "good" measurements in order to decide whether the system is in a valid secure state. Policy can then be used to decide what action to take if the comparison fails.

A monotonically incremented counter is supported to prevent rollback of firmware, including configuration data, to a previous version since that previous version of firmware may have exploitable vulnerabilities. However, rollback for recovery purposes can be permitted if authorized.

D.1.3 Platform Security Architecture (PSA) Functional APIs

The PSA Functional APIs define the foundations from which security services are built, allowing devices to be secure-by-design. The three APIs (cryptography, storage, and attestation) provide a consistent developer experience for system software and application developers, enabling interoperability across different hardware implementations of the RoT.

D.1.3.1 The PSA Cryptographic API (Crypto API)

The PSA Crypto API provides a portable interface to cryptographic operations on a wide range of hardware. The interface provides access to the low-level primitives used in modern cryptography. It does not require that the user has access to the key material; instead, it uses opaque key identifiers. It defines an interface for cryptographic services, including cryptography primitives and a key storage functionality. The interface is designed to be both scalable and modular, allowing devices to only implement what they need.

Implementations can isolate the crypto processor from the calling application, and can isolate multiple calling applications, one from another. The implementation can be separated into a front end and a back end. In an isolated implementation, the back end is located in an isolated environment, which is protected from the front end. Various technologies can provide protection, for example:

This publication is available free of charge from: https://doi.org/10.6028/NIST.IR.8320

- OS process isolation
- Partition isolation, either with a VM or TEE environment like TrustZone
- Physically separate hardware devices

A low-level cryptographic interface is defined, where the caller explicitly chooses which algorithm and security parameters they want to use. All cryptographic functionality operates according to the algorithm specified by the caller. Generic higher-level interfaces, where the implementation chooses the best algorithm for a purpose, are not specified. However, higher-level libraries can be built on top of the PSA Crypto API.

Some intended use cases for the PSA Crypto API are:

- TLS
- Secure storage encryption
- Network credentials (e.g., X.509)
- Device pairing (e.g., Near Field Communication [NFC] token pairing or Bluetooth key agreement protocols)
- Verified boot (firmware integrity and authentication)
- Attestation (the primitives provided are suitable for attestation use cases)
- Factory provisioning (these APIs can be used to generate device unique identity keys for population at the factory)

Interfaces for the following types of symmetric cryptographic operation are provided:

- Message digests, commonly known as hash functions
- Message authentication codes (MACs)
- Symmetric ciphers
- Authenticated encryption with associated data (AEAD)

Both a pair of single-part functions (e.g., encrypt, decrypt) and a series of functions that permit multi-part operations are defined for each type of symmetric cryptographic operation (e.g., allocate, initialize, setup, update, and finish).

D.1.3.2 The PSA Storage API

The PSA Storage APIs provide key/value storage interfaces for use with device-protected storage. They describe the interface for the storage provided by the PSA RoT (the PSA Internal Trusted Storage [ITS] API), as well as an interface for external protected storage (the PSA Protected Storage [PS] API). Two use cases are covered: secure storage for device secret data (ITS), and protection for data-at-rest (PS). ITS is a more specialized API and is intended to provide data integrity and/or privacy. For example, a device identity key requires both confidentiality and integrity, whereas a public key is public data requiring integrity but not privacy. PS is the general-purpose API that will be used most often and is intended to protect

This publication is available free of charge from: https://doi.org/10.6028/NIST.IR.8320

larger data sets against physical attacks. It provides the ability to store data on external flash, with a promise of data-at-rest protection, including device-bound encryption, integrity, and replay protection. It is possible to select the appropriate protection level, e.g., encryption only, or integrity only, or all three, depending on the threat model of the device and the nature of its deployment.

Consistent APIs for accessing storage allow software to be written in a platform-independent manner, improving portability across PSA-supported platforms.

D.1.3.3 The PSA Attestation API

The PSA Attestation API is a standard interface provided by the PSA RoT, which is defined in the PSA Security Model. The API can be used either to directly sign data or to bootstrap trust in other attestation schemes. PSA provides a framework and the minimal generic security features allowing OEM and service providers to integrate various attestation schemes on top of the PSA RoT. The PSA RoT reports information (claims) that can be used to determine the exact implementation of the PSA RoT and its security state. If the PSA RoT loads other components, it also includes information about them. Other components outside of the PSA RoT can add information to the report by calling the provided API, which will include and sign the additional information. The PSA RoT signs attestation reports using the initial attestation key.

Each device instance contains a protected attestation key that can be used to prove that it is a particular certified implementation. The attestation identity can be verified in an attestation process and checked against certification information. At the end of the process the verifier can establish a secure connection to the attested endpoint and deliver credentials. The combination of a hardware entity and the software installed on that entity can be certified to conform to some published security level. A party may want to check the received list of claims against a database of known measurements for each component in order to decide which level of trust should be applied.

Initial attestation requires three services:

- Enrollment verification service, enforcing policy as part of service enrollment of the device
- Production verification service (OEM), providing the production state of an attestation identity
- Certification verification service (third party), verifying that all attested components are up to date, signed correctly, and certified to work together

The API must be provided by the implementation.

D.1.4 Platform AbstRaction for SECurity (Parsec)

Parsec [50] is the Platform AbstRaction for SECurity, an open-source initiative to provide a common API to secure services in a platform-agnostic way. It provides a micro-service that maps easy-to-consume security APIs, in the language of choice, to security primitives found in various different hardware implementations. It is part of the CNCF sandbox.

This publication is available free of charge from: https://doi.org/10.6028/NIST.IR.8320

Parsec aims to define a software standard for interacting with secure object storage and cryptography services, creating a common way to interface with functions that have traditionally been accessed by more specialized APIs. Parsec provides an ecosystem of libraries in a variety of programming languages. Each library is designed to be simple to consume. This ecosystem makes secure facilities available to developers across a broad range of use cases in infrastructure computing, edge computing, and IoT.

Computing platforms have evolved to offer a range of facilities for secure storage and secure operations. There are hardware-backed facilities such as the HSM and TPM, there are firmware services running in TEEs, and there are also cloud-based Parsec services. Security facilities may be provided purely in software, where they are protected by mechanisms provided in the OS. Parsec is built on PSA. The core component of Parsec is the Parsec security service. It is a background process that runs on the host platform and provides connectivity with the secure facilities of that host and exposes the wire protocol based on PSA Functional APIs.

The Parsec service listens on a suitable transport medium. The transport technology is one of Parsec's many pluggable components, and no single transport is mandated. Choice of transport is dependent on the OS and the deployment. On Linux-based systems where the client applications are running in containers (isolation with a shared kernel), the transport can be based on Unix sockets. Client applications make connections with the service by posting API requests to the transport endpoint. This is usually done via a client library that hides the details of both the wire protocol and the transport.

A single instance of the Parsec service executes on each physical host. In virtualized environments, the Parsec service may reside on a specially assigned guest, or potentially within the hypervisor. Another option is running Parsec on each individual guest, with back-end support running in a TEE/Secure Enclave. The Parsec service does not support remote client applications; each physical host or node must have its own instance of the service. However, it is possible for the service to initiate outbound remote calls of other services, such as cloud-hosted HSM services.

The Parsec service is also responsible for brokering access to the underlying security facilities amongst the multiple client applications in a multi-tenant environment. Parsec is able to support multi-tenant scenarios by making use of a client identity, which is a string or token that each client application passes to the Parsec service with every API call. Parsec does not mandate any particular format or semantics for these client identity strings. They can be passed opaquely to the service as octet sequences using the wire protocol. The only requirement on a client identity string is that it must be verifiable by the Parsec service in some way. The component that offers this verification is known as an identity provider.

The identity provider is not part of Parsec. It can be another component or service, residing either locally or remote. It could be a container orchestrator or other runtime management service, or even a function of the OS. It is expected that the Parsec service can establish trust with the identity provider via some suitable means. Parsec offers a pluggable mechanism for communicating with different identity providers.

This publication is available free of charge from: https://doi.org/10.6028/NIST.IR.8320

Examples of client identity management include the following:

- The client identity can simply be an OS entity such as the process identifier, user identifier, or group identifier of the client application's process. When Unix sockets are used for the wire protocol transport, Parsec can verify the identity by making system calls to perform a peer credential check [51]. In this case, the identity provider is effectively the OS kernel.
- The client identity can be a Secure Production Identity Framework for Everyone (SPIFFE) Verifiable Identity Document (SVID) [52]. In this case, the identity provider would be a SPIFFE runtime component such as a local SPIFFE Runtime Environment (SPIRE) service [53], with which the Parsec service communicates using the SPIFFE Workload API [54].

Other methods of verification are possible. However the client identity is verified, Parsec will use the resulting identity string as a namespace for all keys and other stored assets. The Parsec service will ensure that each client application is only given visibility and access over its own namespace.

Parsec meets the need for a new platform abstraction that offers a common palette of security primitives via a software interface that is both agnostic with respect to the underlying hardware capabilities, and also capable of supporting multiple client applications on the same host, whether those be within containers or VMs.

For more information, see also [55] [56] [57].

D.2 Software Runtime Protection Mechanisms

D.2.1 Return Oriented Programming (ROP) and Jump Oriented Programming (JOP) Attacks

D.2.1.1 Arm Pointer Authentication Code (PAC)

Attacks frequently attempt to subvert software control flow. PAC [48] is a feature introduced to block these types of attacks. It adds functionality that supports address authentication of the contents of a register before that register is used as the target of an indirect branch, or as a load. PAC is a strong defense against ROP attacks, which attempt to make a function return to the wrong location.

With PAC, hardware ensures that the return is made to the correct location by preserving the original pointer value. The upper bits of a 64-bit pointer are used to store the PAC, which is a cryptographic signature on the pointer value and some additional specified context. Special instructions have been introduced to add a PAC to a pointer, verify an authenticated pointer's PAC, and restore the original pointer value. The authentication operation regenerates the PAC and compares it with the value that is stored in the pointer. If authentication succeeds, a pointer without the PAC is returned. If authentication fails, an invalid pointer is returned. An exception is raised if the pointer is subsequently used. This gives the system a way to make cryptographically strong guarantees about the likelihood that certain pointers have been tampered with by attackers.

This publication is available free of charge from: https://doi.org/10.6028/NIST.IR.8320

D.2.1.2 Arm Branch Target Identification (BTI)

Once an attacker has found a vulnerability to exploit, their next aim is to execute code to gain control of the machine they have accessed. Techniques used to modify the control flow include ROP and JOP attacks. These techniques find small sections (called gadgets) of vulnerable programs that can be chained together to achieve the attacker's goal.

Systems supporting Branch Target Identification (BTI) [48] can enforce that indirect branches only target code locations that start with one of the accepted BTI instructions. Pages can be marked as containing BTI instructions. Indirect branches can only branch to locations identified as having BTI instructions, which reduces the ability of an attacker to execute arbitrary code. The BTI feature works together with PAC to significantly reduce the number of gadgets available to an attacker.

D.2.2 Memory Safety Violations

D.2.2.1 Arm Privileged Access Never (PAN)

PAN helps to prevent an OS kernel or a hypervisor from being exploited to erroneously access memory allocated to a user-mode application (EL0). If PAN is enabled, any attempt by the kernel or hypervisor to access a page controlled by a user-mode attacker will be prevented. The access will result in a permission fault and will not result in the data or instruction being cached.

Sometimes the OS or hypervisor does need to access unprivileged regions, for example, to write to a buffer owned by an application. To support this, the instruction set provides unprivileged loads and stores that are not blocked by PAN (LDTR and STTR). They are checked against EL0 permission checking even when executed by the OS at EL1 or EL2. While application code needs to be executable in user space (EL0), it should never be executed with kernel permissions (EL1/EL2), so PAN controls these accesses.

D.2.2.2 Arm User (EL0) Execute Never (UXN) and Privileged (EL1/EL2) Execute Never (PXN)

User (EL0) Execute Never (UXN) and Privileged (EL1/EL2) Execute Never (PXN) provide protection against "stack smashing" attacks where malicious software attempts to write new opcodes into memory and then attempts to execute the newly modified memory. Typically this attack is targeted at stack memory. These execute permissions are stored in the page table entries. Any attempt to branch to an address within a marked page triggers a permission fault. The translation table attributes and write controls can block execution from any location that the malicious code could write to.

D.2.2.3 Arm Memory Tagging Extension (MTE)

Memory tagging enables developers to identify spatial and temporal memory safety violations in their programs (e.g., use-after-free, use-out-of-scope, use-before-initialization, bounds violations). MTE [48] [58] [59] is designed to quickly detect memory safety violations and provide robustness against attacks that are attempting to subvert code. MTE is a lightweight, probabilistic version of a lock and key system where one of a limited set of lock values can be

This publication is available free of charge from: https://doi.org/10.6028/NIST.IR.8320

associated with the memory locations forming part of an allocation, and the equivalent key is stored in unused high bits of addresses used as references to that allocation. On each use of a reference, the key is checked to make sure that it matches the lock before an access is made. If the key matches the lock, the memory access is permitted; otherwise, the invalid access can either be recorded for later reference (and execution is allowed to continue) or be faulted (and execution is halted). On freeing an allocation, the lock value associated with each location is changed to one of the other lock values so further uses of the reference have a reasonable probability of failure. Hard-to-catch memory safety errors can be detected and eliminated more easily, which aids reliability and improves product security.

MTE provides architectural support for run-time, always-on detection of various classes of memory errors and can aid with debugging to eliminate vulnerabilities before they can be exploited. MTE implements support for storage access and checking of the lock values in hardware. Software selects and sets the values on allocation and deallocation.

Instructions are added for use by general-purpose software to set Logical Address Tags, manipulate Logical Address Tags, store Allocation Tags to memory, and load Allocation Tags from memory. Additional instructions are added for use by system software and external debug agents to efficiently transfer Allocation Tags to and from memory. The extension is expected to be generally applicable to 64-bit software written in C and C++ that does not use the Logical Address Tag bits for other purposes. Use in mixed language environments, e.g., C/C++ code interacting with JIT) compiled or interpreted languages is also expected to benefit. Applicability to software in other languages will vary. Since MTE imposes no changes to standard C/C++ application binary interfaces (ABIs), incremental deployment across and within ELs is possible.

MTE and PAC can both be enabled at the same time.

D.2.2.4 Arm Hardware Enforced Capability-Based Architecture (Morello and CHERI)

Arm and the University of Cambridge are collaborating in the development of the Capability Hardware Enhanced RISC Instructions (CHERI) architecture that provides a capability approach to memory safety [60]. Arm has developed a prototype architecture, Morello [13] [14] [61], that adapts the hardware concepts of CHERI. Morello is a research program led by Arm in association with partners and funded by UK Research and Innovation (UKRI) as part of the UK government Digital Security by Design (DSbD) program. This new approach to cybersecurity requires a significant change in how the hardware architecture is designed, as well as how the software running on devices that support the CHERI capability architecture is programmed to take advantage of the new features.

The Morello architecture aims to improve the robustness and security of systems through two design goals: fine-grained memory protection leading to increased memory safety, and scalable compartmentalization. To achieve these goals, the Morello architecture introduces the principles defined by CHERI, including the principles of least privilege and intentional use. The Morello architecture is backwards compatible with, and complementary to, the existing Armv8-A architecture.

This publication is available free of charge from: https://doi.org/10.6028/NIST.IR.8320

The CHERI CPU architecture adds 128-bit "capabilities" plus a memory tag bit. The capability contains the address, bounds information, permission information, and an object type. The memory tag bit is metadata that distinguishes a capability from normal data and prevents "forging" of a capability. A capability can be used in place of a normal pointer in some or all situations. Simply replacing all pointers with capabilities gives scope for strong spatial memory protection. Loads and stores using capabilities as addresses are checked to be within the capability address range and matching the supplied permissions. Bounds cannot be arbitrarily increased, permissions cannot be relaxed, and the tag cannot be changed.

Three example use cases are:

- **Kernel access control**: Due to tagged memory and constrained manipulation of capability segment descriptors, a process cannot create a capability segment descriptor that describes memory for which it does not already possess a descriptor.
- **Sandboxing**: Capabilities can also be used for sandboxing a process into sub-address spaces, essentially allowing a process to have its own memory protection policy for program portions.
- **Pointer safety**: Capabilities can also be used for pointer safety using automated bounds checking.

The Morello System Development Platform (SDP) is a prototype demonstrator board that contains a Morello SoC. The SDP serves as the DSbD technology platform prototype for the Morello architecture. Capabilities are introduced to the Arm v8-A architecture profile as an extension of the Arm v8 AArch64 state, with the principles proposed in version 8 of the CHERI ISA [60], to provide hardware support for fine-grained protection and building blocks for secure, scalable compartmentalization.

A pre-silicon Fixed Virtual Platform (FVP) System Model of Morello is available now. It enables the development of software without the requirement for the prototype hardware. Arm FVP models use binary translation technology to deliver fast simulations of the Arm-based system. The Morello FVP provides a functionally accurate model of the Morello SoC IP implementation. The FVP enables industry and academic partners to test the new capability-based prototype architecture in real-world use cases. The Arm Development Studio: Morello Edition supports the Morello FVP, which includes software models of the Rainier cores. The Morello SDP board is closely based on the Arm® Neoverse™ N1 System Development Platform board. Specifications, models, and Morello demonstrator boards are available.

D.2.3 Arm Mitigations Against Side-Channel Attacks

It is possible for attackers to exploit undesirable side-effects of out-of-order execution and speculative execution in modern processors to breach the separation between OS and processes and between processes in order to steal data. Physical access to the system is often not needed in order to mount an attack. An attacker can potentially breach typical process and privilege separation by using specially crafted software to gather sensitive information from other software that is running on the same system.

This publication is available free of charge from: https://doi.org/10.6028/NIST.IR.8320

Arm has implemented a number of mitigations against side-channel attacks. At this time, four variant mechanisms have been identified, each potentially using the speculation of a processor to influence which cache entries have been allocated in a way to extract some information that would not otherwise be accessible to software. There are barrier instructions added that allow mitigation of Spectre and Meltdown variants 1 (CVE-2017-5753), 2 (CVE-2017-5715), 3 (CVE-2017-5754), and 3a (CVE-2018-3640), as well as memory disambiguation control to mitigate variant 4 (CVE-2018-3639). See [48] [62] [63] [64] for a more complete description.

D.2.3.1 Speculation Barriers

A new barrier, a Speculation Barrier, is added to the architecture to control memory speculation. Until the barrier completes, the execution of any instruction appearing later in the program order than the barrier cannot be performed speculatively, to the extent that such speculation can be observed through side-channels as a result of control flow speculation or data value speculation, but can be speculatively executed as a result of predicting that a potentially exception-generating instruction has not generated an exception. In particular, it cannot cause a speculative allocation into any caching structure where the allocation of that entry could be indicative of any data value present in memory or in the registers. The instruction can complete once it is known not to be speculative, and all data values generated by instructions appearing in program order before the Speculation Barrier have their predicted values confirmed.

D.2.3.2 Predictor Invalidates

For all execution prediction resources that predict addresses or register values, the architecture requires that the speculative execution at one hardware-defined context is separated in a hard-to-determine manner from the predictions trained in a different hardware-defined context. For the purpose of this definition, the hardware-defined context is determined by the following:

- The EL
- The Security state
- When executing at EL1, the Virtual Machine IDentifier (VMID)
- When executing at EL0, the Address Space IDentifier (ASID) and the VMID

D.2.3.3 Synchronization Barriers

The Arm architecture is a weakly-ordered memory architecture that supports out-of-order completion. *Memory barrier* is the general term applied to an instruction, or sequence of instructions, that forces synchronization events by a PE with respect to retiring Load/Store instructions. The memory barriers defined by the Armv8 architecture provide a range of functionality, including ordering and completion of load/store instructions, and context synchronization. The new instructions provide a synchronization barrier for instructions, data, trace, error, and profiling.

D.2.3.4 Arm Trusted Firmware (TF-A) and Linux

Arm has contributed updates to the open-source Trusted Firmware (TF-A) project [12] and has developed Linux kernel and Android patches that take advantage of those implemented updates.

This publication is available free of charge from: https://doi.org/10.6028/NIST.IR.8320

Patches have been upstreamed for both AArch64 and AArch32. Mitigations for Variant 1, 2, 3, and 4 are available for both AArch64 and AArch32. Trusted Firmware has implemented a new SMC call to support some of these mitigations. The software mitigations described in the Cache Speculation Side-channels paper [62] should be deployed where protection against malicious applications is required by the threat model. Arm introduced processor features in Armv8.5-A [47], which can be implemented from Armv8.0, that provide resilience to this type of attack.

D.2.3.5 Timing Insensitive DP Instructions

A new option can be set to ensure Data Independent Timing for most classes of the data processing instructions – i.e., the time taken for instructions is independent of the values of the data supplied in any of their registers. In addition, the response of these instructions to asynchronous exceptions does not vary based on the values supplied in any of their registers. This includes the following:

- All cryptographic instructions
- The set of instructions that use the General Purpose Register File
- The set of instructions that use the Single Instruction, Multiple Data (SIMD) & Floating Point Register File
- All loads and stores are timing-insensitive to the value being loaded or stored

This prevents an attacker from inferring anything about the data being processed by an instruction or a set of instructions.

D.3 Data Protection and Confidential Computing

D.3.1 Arm Confidential Compute Architecture (CCA)

Arm's Confidential Compute Architecture (CCA) [65] establishes a Protected Execution Environment architecture that provides mechanisms that can be used to construct an environment where privilege does not imply any right of access. CCA isolates the execution environments from each other irrespective of privilege level. This separation of policies has many in-depth consequences for architecture, trust relationships, and contracts.

Arm CCA protects data in use by performing computation within a hardware-backed and remotely verifiable secure environment. It shields code, data, and execution from observation and modification by other software and hardware agents. With CCA, the owner of a Protected Execution Environment does not need to trust other co-hosted software or privileged hardware agents, such as direct memory access masters. The mechanisms providing the Protected Execution Environment are directly measured and reported using attestation to determine their trustworthiness.

D.3.1.1 Arm Realms

CCA provides architecture support for dynamically created entities called Realms. A Realm [66] [67] contains both user (EL0) and kernel space (EL1) code and data. The higher-privileged entity that manages Realm resources is the NW hypervisor. Realm tenants do not need to trust either

This publication is available free of charge from: https://doi.org/10.6028/NIST.IR.8320

the hypervisor or existing SW code. Realms are protected from each other; a Realm does not need to trust other Realms.

The CCA RoT enforces authenticity of the CCA platform by attesting to the boot state and security state of a Realm, the authenticity of the Realm content through verification and measurement, and the confidentiality of the guest data through Physical Address Space (PAS) protection and encryption.

Realm world is a world separate from both the NW and the SW which already exist in TrustZone. Realm world is designed for the exclusive use of Realms. A Realm protects the information within it from other system entities. Higher-privileged software retains responsibility for allocating and managing the resources utilized by Realms but cannot access their contents. Higher-privileged software also retains responsibility for scheduling within its Realms, but cannot otherwise control or directly observe their execution flow.

Specifically, CCA provides:

- Additional memory access control, orthogonal to the existing controls enforced using translation tables
- Execution state protection
- Trustworthy measurement (attestation) of the initial state
- A guarantee that the system configuration (for example, whether external debug is permitted) does not change during the lifetime of the Realm (immutability)

Realm data remains confidential even after Realm destruction or system reset.

Realms are explicitly designed to be created and destroyed on demand. A Non-secure hypervisor can create a new Realm at any time, much like it can create a new VM at any time. The hypervisor can add pages to a Realm or remove pages from a Realm at any time, much in the same way it manages the pages of other VMs. This contrasts with TrustZone architecture, which is not designed to support dynamic creation of Protected Execution Environments.

Realms are designed to support complex memory management schemes in existing OSs and hypervisors with minimally invasive changes. As a result, an architected mechanism is required to control access to memory used for the implementation of Realms. Control of accessibility of memory from a given world must be fine-grained (at 4K page granularity) and dynamic. All physical memory in a CCA system is divided into Granules. Every Granule has a set of properties that defines the constraints under which the corresponding addresses can be accessed. Violation of these constraints results in a fault.

D.3.1.2 Arm Realm Management Extension (RME)

The Realm Manager is responsible for managing Realms. A Realm is a VM consisting of an OS kernel (running at EL1) and a set of applications (executing at EL0). In addition to the two security states supported by TrustZone (Secure state and Non-secure state), CCA introduces two additional Security states supported by the Realm Management Extension (RME): Realm state and Root state. With RME, EL3 moves out of Secure state and into its own security state – Root

This publication is available free of charge from: https://doi.org/10.6028/NIST.IR.8320

state. RME provides isolation of EL3 from all other security states. Realm, Non-secure, and Secure states need to trust EL3. Secure and Realm state can be mutually distrusting because Secure state is hardware-isolated from both Non-secure state and Realm states, while Realm state is hardware-isolated from both the Non-secure and Secure states. All other security states contain EL2, EL1, and EL0 in both the Realm world and the NW. Figure 11 shows the addition of Realm world to the NW and the SW.

The Monitor security domain executes runtime firmware that manages security state switching and the assignment of resources among security states. Backwards compatibility is maintained with existing TrustZone use cases by retaining the Secure state. However, these existing use cases can take advantage of new RME features such as dynamic memory assignment.

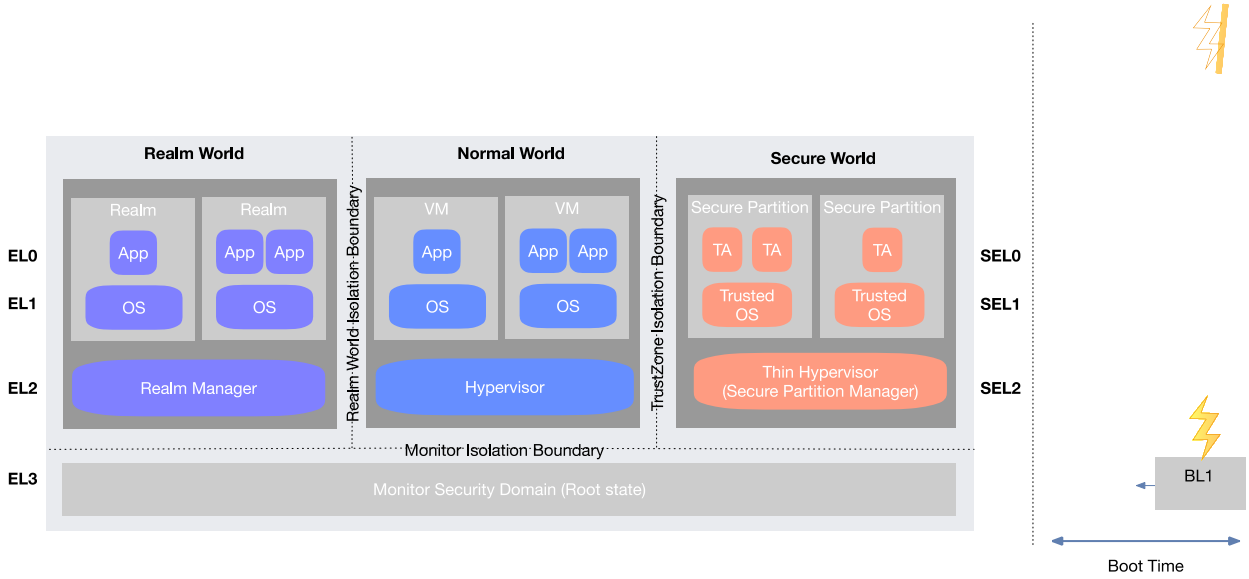


Figure 11: Root World (Monitor), Realm World, and Isolation Boundaries

CCA attestation allows a user of a service provided by a Realm – a reliant party – to determine the trustworthiness of the Realm and of the implementation of the CCA. The reliant party may be local or remote.

Dynamic TrustZone uses RME to provide an architected mechanism to assign pages of memory between the Non-Secure and Secure address spaces, at run time. Part of CCA, the RME in Armv9-A enables pages of memory to be dynamically transitioned from the NW to the SW and back again. Building on RME, CCA provides the following additional features that enhance a dynamic TrustZone solution:

- Firmware partitioning and isolation of the EL3 monitor, used to provide a stronger RoT and attestation services
- Encryption of all data in Secure assigned DRAM through the Memory Protection Engine

This publication is available free of charge from: https://doi.org/10.6028/NIST.IR.8320

D.3.1.3 Arm Realm Memory Isolation and Protection

As shown in Figure 11, TrustZone provides two security states each associated with its own PAS: the Secure PAS and the Non-secure PAS. RME extends this with two additional PASes: the Realm PAS and the Root PAS. They provide isolation guarantees for data belonging to each PAS. Each PAS has its own address translation regime. Device access to memory is subject to RME PAS isolation guarantees.

D.3.1.4 Arm External Memory (DRAM) Encryption and Integrity with CCA

Many Realm and CCA assets are held in external memory. Memory encryption with CCA is designed to provide additional isolation, and privacy among the Realm, Root, and SWs and within the Realm PAS. With CCA, memory is uniquely encrypted per world and location using a separate encryption context for each PAS (Realm, Root, Secure) using a different address tweak for each encryption data block to provide spatial isolation. Memory encryption is randomly re-seeded at boot time to make all existing encrypted memory contents undecipherable following system reset.

Data is encrypted by hardware before being written to external memory or to any shared cache that resides past the point of physical alias.

CCA protects against a number of different types of memory attacks, including:

- **Direct memory access**: An unauthorized agent on the same system attempts to directly access the contents of memory allocated to a different world, or to a Realm.
- **Probing**: An attacker attempts to physically access the contents of memory allocated to a world or a Realm—for example, using hardware probes or recording devices such as Non-Volatile Dual In-Line Memory Modules (NVDIMMs).
- **Leakage**: An attacker gains access to the contents of memory allocated to a world or a Realm, for example through misconfigurations or errors in the implementation of the CCA platform.

D.3.1.5 Arm CCA Firmware Boot

CCA root world firmware is booted before the firmware described for the SW in Appendix D.3.1.4. It starts with immutable initial boot code that may be in an on-chip ROM or locked on-chip storage. On a secured system, the immutable initial boot code is inherently trusted and is not verified or measured. It is considered a part of the CCA hardware security domain and identified by the CCA platform attestation ID. Any updatable CCA firmware, including Realm management security domain firmware, is both verified and measured and is reported in a CCA platform attestation.

Realms are measured by Realm management security domain firmware at Realm creation and reported in a Realm attestation bound to the current CCA platform attestation.

The SW boot chain may be started by Monitor firmware. The NW boot chain is started by the SW when it instantiates the boot loader (e.g., UEFI) in the NW. They continue independently from CCA.

This publication is available free of charge from: https://doi.org/10.6028/NIST.IR.8320

The CCA Hardware Enforced Security (HES) is hosted on a trusted subsystem and implements core CCA security services such as:

- The CCA platform boot state
- CCA platform attestation services
- CCA Root parameters
- The CCA system security lifecycle
- CCA security provisioning
- Collating the CCA hardware boot state

CCA firmware includes:

- On the HES device:
  - HES firmware
  - Trusted subsystem firmware for the HES host
- On the Application Processor (PE) host:
  - Application PE firmware for the Monitor security domain and the Realm Management security domain
  - Trusted subsystem firmware for all other trusted subsystems within the CCA system security domain

D.3.1.6 Arm RME and Debug, Trace, Profiling, and Performance Monitoring Protection

Arm systems include extensive features to support debugging, tracing, and profiling. The RME provides controls that can limit which parts of the system can be debugged. Signals to enable different debug, trace, and profiling features are provided that help to manage the features. External debug signals are typically connected to fuses or an authentication module where debug for each state can be enabled or disabled, one for each of the four signals. These are usually managed and disabled depending on the current lifecycle state of the device. External access to Performance Monitoring Units is regulated by the RME.

Hardware registers control when self-hosted Debug Trace, Profiling, and Performance Monitoring are context-switched when switching between security states or Realms.

D.3.1.7 Remote Attestation Service - Project Veraison (VERificAtIon of atteStatiON)

This open software initiative is creating software that can be used to build device attestation verification services that can support many architectures. To support the Arm CCA, contributions to the Veraison project will develop plug-ins that implement the Arm CCA attestation model. Veraison supports verification (has reference implementations for Entity Attestation Token [68], EAT PSA Profile, and Device Identifier Composition Engine [69]) and provisioning (an API to allow provisioning of Reference Values/Endorsements from supply

This publication is available free of charge from: https://doi.org/10.6028/NIST.IR.8320

chain, object revocation, multi-tenancy data separation, as well as an audit trail). It is intended as a reference deployment that can be either self-hosted or platform-as-a-service hosted.

D.3.2 Arm Cryptographic Acceleration

D.3.2.1 Cryptography Extensions

The ARMv8A Cryptography Extensions [47] have added 32 new Advanced SIMD instructions that operate on the vector register file. They can be used to accelerate the cryptographic algorithms listed here:

- Secure Hash Algorithm 1 (SHA1), SHA256, and AES: these instructions are added to both the A32 and A64 instruction sets
- SHA512, SHA3, SM3, and SM4: these instructions are added to the A64 instruction set only

They provide three to ten times better software encryption performance. They are useful for small granule decryption and encryption that is too small to efficiently offload to an external hardware accelerator.

D.3.2.2 True Random Number Generation (TRNG)

A True Random Number Generator (TRNG) provides entropy in the form of random numbers from the sampled output of an unpredictable physical process rather than by means of an algorithm (a Deterministic Random Bit Generator [DRBG]). The main application for truly random numbers is in cryptography, where they are used to generate random cryptographic keys (e.g., Elliptic Curve Digital Signature Algorithm [ECDSA] key pairs, seed, and nonce; symmetric MAC keys) to store and transmit data securely (e.g., encryption protocols such as TLS). Keys generated using a TRNG are unpredictable and therefore are highly resistant to guessing attacks based on understanding an algorithmic implementation.

Two new random number instructions, RNDR and RNDRRS, have been added [47]. They return a 64-bit random number into a general-purpose register. A read to the RNDRRS register will cause a reseeding of the random number before the new random number is generated and returned.

The DRBG produces random numbers from a cryptographically secure algorithm and is seeded from the TRNG. The TRNG conforms to several standards, including NIST SP 800-90B, NIST SP 800-22, FIPS 140-2, and British Standards Institution (BSI) AIS-31. The DRBG algorithm conforms to the NIST SP 800-90A Rev 1 standard. The entire random number generation conforms to the NIST SP 800-90C standard.

This publication is available free of charge from: https://doi.org/10.6028/NIST.IR.8320

Appendix E—Cisco Technology Examples

This section describes a number of Cisco technology examples that map back to the key concepts described in the various sections of the document. Examples provided are focused on Cisco's Unified Computing System (UCS), an open server that includes x86 processors from other vendors listed in the appendices and enables users to overlay additional technologies called out in this document.

E.1 Platform Integrity Verification

E.1.1 Cisco Platform Roots of Trust

To ensure the highest possible degree of integrity, Cisco computing platforms use a defense-in-depth methodology to design a platform that prevents both common and sophisticated attacks by employing technology rooted in various hardware components. Commonly referred to as hardware security modules in this report, there are two Platform Roots of Trust (PRoT) on Cisco UCS products. The Cisco Integrated Management Controller is the PRoT on Cisco Servers, and Cisco's Chassis Management Controller (CMC) is the PRoT for Cisco IO Module, Cisco Intelligent Fabric Manager, and the chassis mainframe.

Both PRoTs on Cisco computing platforms implement hardware-anchored RoT for firmware integrity and authenticity. They are anchored in immutable memory (e.g., ROM), in an embedded SoC or anchored to another hardware chip.

Cisco extends platform integrity to include platform authenticity, rooted in either Cisco's Trust Anchor Module such as the discrete Anti-Counterfeit Technology 2 or an exclusive TPM for platform-level-only usage. This proprietary, tamper-resistant chip is found in many Cisco products and features nonvolatile secure storage, Secure Unique Device Identifier, and cryptographic services, including random number generation, secure storage, key management, and cryptographic services to the running OS and applications. Platform authenticity ensures that the platform is not counterfeit and enables proper manufacturing provisioning of critical security parameters.

Cisco's platform integrity capabilities also deter certain physical attacks. For critical signals, Cisco server Printed Circuit Boards (PCBs) are fabricated such that these signals are routed in intermediate PCB layers instead of being accessible on the top or bottom layers of the board where easy probing and modification to electrical integrity can occur. Additionally, Cisco uses a pin-side-down packaging design (e.g., ball grid array) for solder-down components to prevent easy electrical manipulation of these pins.

Because servers typically contain interchangeable components, sub-assemblies, parts, or devices, Cisco computing platforms provide authentication for Security Protocol and Data Model (SPDM) enabled devices (e.g., a storage controller) or other Cisco Field Replacement Units (FRU) (e.g., Cisco's Virtual Interface Card). SPDM is the standard for securing communications with and authenticating devices within a platform.

This publication is available free of charge from: https://doi.org/10.6028/NIST.IR.8320

E.1.2 Cisco Chain of Trust (CoT)

Cisco's holistic approach to platform integrity focuses on protecting the platform as it operates. Firmware integrity and authenticity (commonly known as secure boot or a CoT) start at a hardware-anchored RoT. Firmware integrity is crucial because firmware either controls a particular device (e.g., a network controller) or various devices and operations on the platform, such as in Cisco's Integrated Management Controller and CMC.

Cisco's PRoT establishes a chain-of-trust from hardware to firmware to software integrity by ensuring that the initial boot of the host CPUs have additional protections. Cisco servers' PRoTs do an additional verification of BIOS code and check the integrity of the PCB by matching expected values to actual values. This provides extra protection against specific single component swap attacks. All of Cisco's platform components implement secure boot and a CoT. More information can be found here.

E.2 Cisco Supply Chain Protection

Cisco's comprehensive program to manage supply chain risks and provide protection to customers is highlighted in *Best Practices In Cyber Supply Chain Risk Management*. This supply chain protection program includes processes for Cisco to validate the authenticity and integrity of platform hardware when installing Cisco critical security parameters on Cisco PRoTs. These security parameters are rooted in the PRoT, establishing a secure identity from which the CoT can be authenticated.

E.3 Cisco Software Runtime Protections

Modern platforms contain many intelligent devices that run some form of code. While this code is not considered hardware, it plays a major role in affecting the overall integrity of a platform and is critical to ensuring the platform performs at the highest level of security posture. Whether it is FPGA code where configuration is passed into a voltage regulator or complex software that runs on the PRoT, Cisco employs additional integrity checks to ensure the code or critical security parameters are not maliciously modified.

Cisco's PRoT firmware architecture utilizes many hardware-level security features appropriate for the constraints and the security requirements of the design. For example, Cisco ensures code pages and data pages are isolated so code cannot run from data pages. Application pages are randomized through address space layout randomization (ASLR). As described above, firmware employs hardware anchors for securing secret or sensitive data at rest, in transit, and in operation. On some platforms, ROM code is utilized to ensure firmware authenticity throughout the boot process. For critical applications and products, Cisco utilizes a third-party agency or an internal security agency whose specialty is analyzing source code for security-related issues with an attacker's mindset. Cisco firmware also seeks FIPS certification and Common Criteria certification to further ensure a certain level of security posture for the user.

Once a component fully boots and its firmware is operational, runtime firmware integrity is another important aspect of overall platform integrity and supply chain protection. Firmware components and their contents are controlled and vetted. Open source or purchased software components are analyzed internally before integrating them. The analysis includes looking for

This publication is available free of charge from: https://doi.org/10.6028/NIST.IR.8320

system() calls common in C or equivalent languages to ensure there aren't malicious command-line injections, ensuring the code provides input sanitization, verifying the code passes static analysis, and resolving any errors and warnings from static analysis appropriately. Internally, Cisco ensures that build infrastructure and development servers meet stringent hardening requirements and safeguard the flow of software from the very early stages of development to an official release for users, a process that prevents malicious code injections in the software supply chain. Employed methods include additional integrity checks for code at rest. This process is constantly reviewed and improved. Cisco's PRoT for certain platforms and FRUs expose firmware measurements to end users to verify the state of that component. Additionally, Cisco's PRoTs have internal processes that self-monitor critical resources or critical security parameters to ensure that they have not been tampered with.

E.4 Cisco Data Protection and Confidential Computing

Another aspect of firmware integrity at runtime is data consumption and data processing. Cisco's PRoT utilizes hardware-anchored secure storage to provide data at rest protection. These protections are enabled through hardware cryptographic trust anchors, similar to Trusted Execution Environments (TEE), that securely store secrets such as encryption keys and protect against certain side channel attacks such as physical attacks using radio frequency (RF) techniques. Data processing over the network is secured using standard security protocols such as TLS with identities rooted back to a Cisco Trust Anchor Module.

E.5 Cisco Platform Attestation

A Cisco PRoT continuously monitors the components of the system to detect degraded integrity. It can alert or take corrective action, up to and including lockdown mode, which prevents usage until the issue is remediated. A Cisco PRoT offers alerts or notification through numerous protocols: Simple Network Management Protocol, Simple Mail Transfer Protocol, syslog forwarding, Redfish, and Cisco's xAPI. The PRoT monitors other components on the platform and, depending on their status and their role, it may take corrective actions or alert the user. If the integrity degradation is severe enough, it goes into lockdown mode and prevents continued usage until the issue is remediated.

E.6 Cisco Visibility to Security Infrastructure

The Cisco UCS platform provides cryptographically verifiable reports of platform integrity and security details, including BIOS measurements for the PRoT.

E.7 Cisco Workload Placement on Trusted Platforms

Cisco Intersight is a cloud-based or on-premises management solution that checks the UCS platform hardware for authenticity via the Cisco PRoT before allowing it to be claimed or managed by the Intersight management solution.

This publication is available free of charge from: https://doi.org/10.6028/NIST.IR.8320

Appendix F—IBM Technology Examples

This section describes a number of IBM technology examples that map back to the key concepts described in previous sections of this document.

F.1 Platform Integrity Verification

F.1.1 Hardware Security Module (HSM)

HSMs include cryptographic co-processors that additionally provide very strong tamper-detection, prevention, and response capabilities against physical attacks, such as the IBM 4758 and its successors. IBM HSMs are available as features of IBM Z, LinuxONE, and IBM POWER Systems [70]. HSMs offer additional layers of protection in cloud deployment use cases [71].

F.1.2 IBM Chain of Trust (CoT)

IBM Z and IBM POWER Systems are equipped with a hardware RoT to enable key security capabilities, including trusted boot and enhanced data confidentiality.

TPM is a special type of HSM. POWER9 servers include a TPM 2.0. The TPM 2.0 specification can be found at the TCG webpage [72][73].

POWER 9 Enterprise Systems implement hardware and firmware enhancements to make them even more secure for hybrid cloud deployments. One enhancement is a Secure Initial Program Load (IPL) Process or Secure Boot that only allows platform manufacturer-signed Hostboot and POWER Hypervisor related firmware, up through and including Partition Firmware, to run on the system. Another enhancement is a framework to support remote attestation of the system firmware stack through a hardware TPM.

Secure Boot implements a processor-based chain of trust based in the POWER9 processor hardware and enabled by the POWER9 firmware stack. Secure Boot provides for a trusted firmware base to enhance confidentiality and integrity of customer data in a virtualized environment. POWER9 Trusted Boot provides for measurements of system configuration and initial program load (IPL) path code, which can be used later as proof to a third party via attestation of the system's initial IPL path configuration. In order to create a CRTM, a Secure Boot flow is used, which adds cryptographic checks to each phase of the IPL process until communications with the TPM are established. This flow aims to assert the integrity of all firmware that is to be executed on the core processors, thereby preventing any unauthorized or maliciously modified firmware from running. A firmware component verification failure will prevent the IPL from completing if the component is deemed critical for system functionality. If the component is not a core critical function, the failed image will not be executed, the IPL will be allowed to complete, and appropriate notifications will be presented.

Details of IBM POWER Systems' secure and trusted boot implementation can be found in [74][75]. Details of the IBM Z and LinuxONE secure boot implementation can be found in [76].

This publication is available free of charge from: https://doi.org/10.6028/NIST.IR.8320

F.2 Software Runtime Protection Mechanisms

F.2.1 IBM ROP and COP/JOP Attack Defenses

The POWER platform added four instructions (hashst, hashchk, hashstp, hashchkp) to handle ROP in the Power ISA 3.1B starting in the Power10 processor.

F.3 Data Protection and Confidential Computing

In current computing environments, applications rely on system software for providing services, such as managing access to the computing system's resources. System software (at the minimum) includes an OS but may also include a hypervisor. Conventionally, system software must be trusted because it has complete control over applications and their data. The OS or the hypervisor can access or modify the data of any application, or potentially tamper with any security features implemented by the application without being detected. Consequently, the underlying software must be part of the Trusted Computing Base (TCB).

In shared environments, customers are forced to trust that the entities that develop, configure, deploy, and control the system software are not malicious. Customers must also trust that the systems software is invulnerable to attacks that escalate privilege and compromise the confidentiality and integrity of the customers' applications. This broad trust requirement is often difficult to justify and poses a significant risk, especially for customers who adopt public cloud services. IBM has led efforts across the industry to address such concerns by reducing the size of the TCB and introducing technologies that make customer data inaccessible to system and cloud administrators.

F.3.1 IBM Memory Isolation Technology

While confidential computing (see VM isolation below for examples) and encrypting content in memory are effective security technologies for achieving memory isolation, using these technologies in combination with other isolation technologies can provide a robust multi-layered protection scheme.

One example of such an isolation technology is IBM Z Processor Resource/System Manager (PR/SM). PR/SM is a type 1 hypervisor integrated with all IBM Z models that transforms physical resources into virtual resources so that many logical partitions (LPARs) can share the same physical resources. It provides the ability to divide physical system resources (dedicated or shared) into isolated logical partitions. Each logical partition operates like an independent system running its own operating environment. PR/SM enables each logical partition to have dedicated or shared processors and I/O, and dedicated memory (which you can dynamically reconfigure as needed). PR/SM provides the security administrator the ability to define a completely secure system configuration. When the system is defined in such a manner, total separation of the logical partitions is achieved, thereby preventing a partition from gaining any knowledge of another partition's operation. This isolation technology is evaluated at Common Criteria Evaluation Assurance Level (EAL) 5+ [77].

This publication is available free of charge from: https://doi.org/10.6028/NIST.IR.8320

F.3.2 IBM Application Isolation Technology

IBM Hyper Protect Virtual Servers are a new technology based on IBM Secure Service Containers to protect workloads on IBM Z and LinuxONE throughout the application lifecycle. The IBM Secure Service Container (SSC) is a container runtime technology enabling you to quickly and securely deploy software appliances on servers. An appliance is an integration of OS, middleware, and software components that work autonomously and provide core services and infrastructures that focus on consumability and security.

The goal of SSC technology is to provide a run-time environment to workloads, including access to network, storage, and crypto adapters. The primary goal is to protect any data created by the workload and only give a specific workload instance access to its data. That means even a second workload of the same type never gets access to any data created by another instance. SSC only runs inside of an SSC-type logical partition. Firmware disables memory access for SSC-type logical partitions, and a special bootload/bootchain ensures that only signed SSC-based appliances can be started in such a partition. There is no SSH access into a hosting appliance, and activities can only be triggered through well-defined and tested APIs, always with the claim in mind, that only a workload has access to its data.

Details of IBM Hyper Protect Virtual Servers can be found here [78].

F.3.3 IBM VM Isolation Technology

Both IBM POWER Protected Execution Facility (PEF) and IBM Secure Execution for Linux (IBM Z and LinuxONE) are examples of TEEs providing VM/memory isolation. IBM POWER PEF enables support for secure virtual machines (SVMs) on IBM Power Systems. PEF protects SVMs from other software while the SVMs are at rest, in transit, and while running. SVMs are supported by a new mode in the IBM Power Architecture called Ultravisor mode that has higher privilege than the hypervisor mode. An SVM can run only on systems that support PEF and are verified by the customer who created the SVM. Each system that supports PEF has a public/private key pair where the private key is known only to the system (not exposed to the owner of the system). More details about PEF can be found at [78]. Instructions for how to set up the PEF-enabled software stack can be found at [79]. The PEF/Ultravisor code is available as open source at [80].

IBM Secure Execution for Linux is a hardware-based security technology that is built into the IBM z15T and LinuxONE III generation systems. It is designed to provide scalable isolation for individual workloads to help protect them from not only external attacks but also insider threats. Secure Execution provides isolation between a kernel-based VM (KVM) hypervisor host and guests in virtual environments. This level of vertical isolation is designed to remove the ability for administrators to have total visibility into the sensitive workloads being hosted on VMs and individual containers. Secure Execution provides hardened access restrictions to protect intellectual property and proprietary secrets while allowing administrators to manage and deploy workloads as black boxes and continue normal job functions. Secure Execution also helps enterprises provide isolation between individual multi-tenant workloads running on a shared logical partition. Details of IBM Secure Execution for Linux can be found in [81].

This publication is available free of charge from: https://doi.org/10.6028/NIST.IR.8320

IBM POWER PEF and IBM Secure Execution for Linux leveraged several key innovations led by IBM Research towards trusted execution such as SecureBlue [82] and SecureBlue++ [83][84], which laid the foundations of secure application isolation.

F.3.4 IBM Cryptographic Acceleration Technology

IBM Z, LinuxONE, and IBM Power offer integrated PCIe-attached HSMs. Examples include IBM 4767 offered as IBM Z and LinuxONE feature CryptoExpress5s, IBM Power Systems features EJ32 and EJ33 and IBM 4769 offered as IBM Z and LinuxONE feature CryptoExpress7s and soon to be offered in IBM Power Systems [85][86].

On IBM Z and LinuxONE, the CryptoExpress features provide the flexibility to support different types of workloads and may be configured as a cryptographic accelerator, an IBM Common Cryptographic Architecture cryptographic coprocessor, or an IBM Enterprise Public Key Cryptography Standards (PKCS) #11 (EP11) cryptographic coprocessor [87].

IBM Z and LinuxONE also provide the CP Assist for Cryptographic Functions (CPACF) for high-performance on-core cryptographic acceleration for symmetric and asymmetric cryptographic operations. In conjunction with the IBM CryptoExpress adapter, keys that reside in the HSM may be exported in an encrypted fashion and used by CPACF so that the key is never in the clear in hypervisor, OS, or application memory [88].

F.4 Remote Attestation Services

F.4.1 IBM Platform Attestation Tooling

The IBM TPM Attestation Client Server Framework from IBM Research is open-source tooling to perform platform attestation [89].

F.4.2 IBM Continuous Runtime Attestation

Continuous monitoring of an application agent is enabled by extending measurements throughout its runtime, not just at startup. For example, when enabled, the Integrity Measurement Architecture (IMA) in the Linux kernel will continuously extend runtime measurements [90]. These measurements can be attested periodically to a verification service, which not only checks for unexpected changes to the application agent but also monitors its dynamic behavior [91].

This publication is available free of charge from: https://doi.org/10.6028/NIST.IR.8320

Appendix G—Acronyms and Abbreviations

Selected acronyms and abbreviations used in this paper are defined below.

ABI	Application Binary Interface
AC RAM	Authenticated Code Random Access Memory
ACM	Authenticated Code Module
AEAD	Authenticated Encryption with Associated Data
AES	Advanced Encryption Standard
AMD PSB	AMD Platform Secure Boot
API	Application Programming Interface
AS	Attestation Service
ASID	Address Space IDentifier
ASLR	Address Space Layout Randomization
ASP	AMD Security Processor
BIOS	Basic Input/Output System
BMC	Board Management Controller
BSI	British Standards Institution
BTI	Branch Target Identification
CA	Certificate Authority
CCA	(Arm) Confidential Compute Architecture
CHERI	Capability Hardware Enhanced RISC Instructions
CMC	(Cisco) Chassis Management Controller
CNCF	Cloud Native Computing Foundation
COP	Call Oriented Programming
CoT	Chain of Trust
CPACF	CP Assist for Cryptographic Functions
CPU	Central Processing Unit
CRD	Custom Resource Definition
CRI	Container Runtime Interface
CRTM	Core Root of Trust for Measurement
CRTV	Core Root of Trust for Verification
CSK	Code Signing Key
CSP	Cloud Service Provider
DCRTM	Dynamic Core Root of Trust for Measurement
DFARS	Defense Federal Acquisition Regulation Supplement
DICE	Device Identifier Composition Engine

This publication is available free of charge from: https://doi.org/10.6028/NIST.IR.8320

DIMM	Dual In-Line Memory Module
DRAM	Dynamic Random-Access Memory
DRBG	Deterministic Random Bit Generator
DSbD	Digital Security by Design
EAL	Evaluation Assurance Level
EAT	Entity Attestation Token
ECDSA	Elliptic Curve Digital Signature Algorithm
EL	Exception Level
EPT	Extended Page Table
ETSI	European Telecommunications Standards Institute
ETSI NFV SEC	European Telecommunications Standards Institute Network Functions Virtualization Security
FIDO	Fast Identity Online (Alliance)
FIPS	Federal Information Processing Standard
FOIA	Freedom of Information Act
FPGA	Field Programmable Gate Array
FRU	Field Replacement Unit
FVP	Fixed Virtual Platform
GDPR	General Data Protection Regulation
HES	Hardware Enforced Security
HIPAA	Health Insurance Portability and Accountability Act
HLAT	Hypervisor Managed Linear Address Translation
HMEE	Hardware Mediated Execution Enclave
HSM	Hardware Security Module
I/O	Input/Output
IA	Intel Itanium Architecture
IBB	Initial Boot Block
IMA	Integrity Measurement Architecture
Intel AES-NI	Intel Advanced Encryption Standard New Instructions
Intel CET	Intel Control-Flow Enforcement Technology
Intel MKTME	Intel Multi-Key Total Memory Encryption
Intel TDX	Intel Trust Domain Extensions
Intel TME	Intel Total Memory Encryption
Intel TSC	Intel Transparent Supply Chain
Intel VT-x	Intel Virtualization Technology
IoT	Internet of Things

This publication is available free of charge from: https://doi.org/10.6028/NIST.IR.8320

IPL	Initial Program Load
IPsec	Internet Protocol Security
IR	NIST Interagency or Internal Report
ISA	Instruction Set Architecture
ISecL-DC	Intel Security Libraries for the Data Center
IT	Information Technology
ITL	Information Technology Laboratory
ITS	Internal Trusted Storage (API)
JIT	Just-in-Time
JOP	Jump Oriented Programming
KEK	Key Exchange Key
KMIP	Key Management Interoperability Protocol
KMS	Key Management Service
KPT	Key Protection Technology
KVM	Kernel-Based Virtual Machine
LCP	Launch Control Policy
LPAR	Logical Partition
MAC	Message Authentication Code
ME	Manageability Engine
MMU	Memory Management Unit
MOK	Machine Owner Key
MTE	Memory Tagging Extension
NCCoE	National Cybersecurity Center of Excellence
NFC	Near Field Communication
NFV	Network Functions Virtualization
NIST	National Institute of Standards and Technology
NVDIMM	Non-Volatile Dual In-Line Memory Module
NVRAM	Non-Volatile Random-Access Memory
NW	Normal World, Non-Secure World
ODM	Original Device Manufacturer
OEM	Original Equipment Manufacturer
OS	Operating System
PAC	Pointer Authentication Code
PAN	Privileged Access Never
Parsec	Platform AbstRaction for SECurity
PAS	Physical Address Space

This publication is available free of charge from: https://doi.org/10.6028/NIST.IR.8320

PCB	Printed Circuit Board
PCH	Platform Controller Hub
PCIe	Peripheral Component Interconnect Express
PCR	Platform Configuration Register
PE	Processor Element
PEF	(IBM POWER) Protected Execution Facility
PFR	Platform Firmware Resilience
PIT	Protection in Transit
PK	Platform Key
PKCS	Public Key Cryptography Standards
PR/SM	(IBM Z) Processor Resource/System Manager
PRoT	Platform Root of Trust
PS	Protected Storage (API)
PSA	Platform Security Architecture
PXN	Privileged Execute Never
QAT	QuickAssist Technology
RAM	Random Access Memory
REE	Rich Execution Environment
RF	Radio Frequency
RK	Root Key
RME	Realm Management Extension
RNG	Random Number Generator
ROM	Read-Only Memory
ROP	Return Oriented Programming
RoT	Root of Trust
RTU	Root of Trust for Update
RW	Read/Write
RWX	Read/Write/Execute
SB	UEFI Secure Boot
SCRTM	Static Core Root of Trust for Measurement
SDEI	Software Delegated Exception Interface
SDP	(Morello) System Development Platform
SEL	Secure Exception Level
SEV	Secured Encrypted Virtualization
SEV-ES	Secured Encrypted Virtualization with Encrypted State
SEV-SNP	Secured Encrypted Virtualization with Secured Nested Paging

This publication is available free of charge from: https://doi.org/10.6028/NIST.IR.8320

SGX	Software Guard Extensions
SHA	Secure Hash Algorithm
SIMD	Single Instruction, Multiple Data
SINIT ACM	Secure Initialization Authenticated Code Module
SiP	Silicon Provider
SK	Secure Kernel
SMAP	Supervisor Mode Access Prevention
SMBus	System Management Bus
SMC	Secure Monitor Call
SME	Secure Memory Encryption
SMEP	Supervisor Mode Execution Prevention
SMM	System Management Mode
SoC	System-on-Chip
SP	Special Publication, Secure Partition
SPDM	Security Protocol and Data Model
SPI	Serial Peripheral Interface
SPIFFE	Secure Production Identity Framework for Everyone
SPIRE	SPIFFE Runtime Environment
SPM	Secure Partition Manager
SPS FW	Server Platform Services Firmware
SSC	(IBM) Secure Service Container
SVID	SPIFFE Verifiable Identity Document
SVM	Secure Virtual Machine
SW	Secure World
TA	Trusted Application
TCB	Trusted Compute Base, Trusted Computing Base
TCG	Trusted Computing Group
TD	Trust Domain
TEE	Trusted Execution Environment
TF-A	Trusted Firmware-A
TLS	Transport Layer Security
TOS	Trusted Operating System
TPM	Trusted Platform Module
TRNG	True Random Number Generator
TSME	Transparent Memory Encryption
TXT	Trusted Execution Technology

This publication is available free of charge from: https://doi.org/10.6028/NIST.IR.8320

UCS	(Cisco) Unified Computing System
UEFI	Unified Extensible Firmware Interface
UKRI	UK Research and Innovation
USB	Universal Serial Bus
UXN	User Execute Never
Veraison	VERificAtIon of atteStatiON
VM	Virtual Machine
VMID	Virtual Machine IDentifier
VMM	Virtual Machine Manager, Virtual Machine Monitor
VMX	Virtual Machine Extensions
XTS	xor-encrypt-xor (XEX) Based Tweaked-Codebook Mode with Ciphertext Stealing

This publication is available free of charge from: https://doi.org/10.6028/NIST.IR.8320

Appendix H—Glossary

Asset Tag	Simple key value attributes that are associated with a platform (e.g., location, company name, division, or department).
Chain of Trust (CoT)	A method for maintaining valid trust boundaries by applying a principle of transitive trust, where each software module in a system boot process is required to measure the next module before transitioning control.
Confidential Computing	Hardware-enabled features that isolate and process encrypted data in memory so that the data is at less risk of exposure and compromise from concurrent workloads or the underlying system and platform.
Cryptographic Accelerator	A specialized separate coprocessor chip from the main processing unit where cryptographic tasks are offloaded to for performance benefits.
Hardware-Enabled Security	Security with its basis in the hardware platform.
Platform Trust	An assurance in the integrity of the underlying platform configuration, including hardware, firmware, and software.
Root of Trust (RoT)	A starting point that is implicitly trusted.
Shadow Stack	A parallel hardware stack that applications can utilize to store a copy of return addresses that are checked against the normal program stack on return operations.
Trusted Execution Environment (TEE)	An area or enclave protected by a system processor.

This publication is available free of charge from: https://doi.org/10.6028/NIST.IR.8320