

Withdrawn Draft

Warning Notice

The attached draft document has been withdrawn, and is provided solely for historical purposes. It has been superseded by the document identified below.

Withdrawal Date May 4, 2022

Original Release Date October 27, 2021

Superseding Document

Status Final

Series/Number NIST Interagency or Internal Report 8320

Title Hardware-Enabled Security: Enabling a Layered Approach to Platform Security for Cloud and Edge Computing Use Cases

Publication Date October 2021

DOI <https://doi.org/10.6028/NIST.IR.8320>

CSRC URL <https://csrc.nist.gov/publications/detail/nistir/8320>

Additional Information

2 **Hardware-Enabled Security:**

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

5
6 Michael Bartock
7 Murugiah Souppaya
8 Ryan Savino
9 Tim Knoll
10 Uttam Shetty
11 Mourad Cherfaoui
12 Raghu Yeluri
13 Akash Malhotra
14 Don Banks
15 Michael Jordan
16 Dimitrios Pendarakis
17 J. R. Rao
18 Peter Romness
19 Karen Scarfone
20

21
22 This publication is available free of charge from:
23 <https://doi.org/10.6028/NIST.IR.8320-draft2>
24

30
31
32
33

2nd Draft NISTIR 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*

Akash Malhotra
*AMD Product Security and Strategy Group
Austin, TX*

Peter Romness
*Cisco
McLean, VA*

Karen Scarfone
*Scarfone Cybersecurity
Clifton, VA*

This publication is available free of charge from:
<https://doi.org/10.6028/NIST.IR.8320-draft2>

34
35

October 2021



U.S. Department of Commerce
Gina Raimondo, Secretary

36
37
38
39
40
41
42
43

National Institute of Standards and Technology
*James K. Olthoff, Performing the Non-Exclusive Functions and Duties of the Under Secretary of Commerce for
Standards and Technology & Director, National Institute of Standards and Technology*

44 National Institute of Standards and Technology Interagency or Internal Report 8320
45 95 pages (October 2021)

46 This publication is available free of charge from:
47 <https://doi.org/10.6028/NIST.IR.8320-draft2>

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

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

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

61 **Public comment period: *October 27, 2021 through December 6, 2021***

62 National Institute of Standards and Technology
63 Attn: Applied Cybersecurity Division, Information Technology Laboratory
64 100 Bureau Drive (Mail Stop 2000) Gaithersburg, MD 20899-2000
65 Email: hwsec@nist.gov

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

67

Reports on Computer Systems Technology

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

76

Abstract

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

85

Keywords

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

89

Disclaimer

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

93

Acknowledgments

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

96 • From AMD: David Kaplan and Kathir Nadarajah

97 • From Arm: Yuval Elad, Nicholas Wood, Joanna Farley, Paul Howard, Dan Handley,
98 Stuart Yoder, Erik Jacobson, and Mark Knight

99 • From Cisco: Jeff Schutt, Scott Phuong, Charlie Hsu, Timothy Wilson-Johnston, and
100 Justin Salerno

101 • From IBM: Jonathan Bradbury, James Bottomley, Debapriya Chatterjee, Chris Engel,
102 Ken Goldman, Guerney Hunt, Kathryn Ignaszewski, Hani Jamjoom, Elaine Palmer,
103 Harmeet Singh, and Balaram Sinharoy

104 • From Intel Corporation: Ravi Sahita, Alex Eydelberg, Sugumar Govindarajan, Kapil
105 Sood, Jeanne Guillory, David Song, Scott Raynor, Scott Huang, Matthew Areno, Charlie
106 Stark, Subomi Laditan, Kamal Natesan, Haidong Xia, Jerry Wheeler, Dhinesh
107 Manoharan, and John Pennington

108 • From Red Hat: Luke Hinds and Mark Bohannon for contributing content related to
109 Keylime

110

Audience

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

116

Trademark Information

117 All registered trademarks or trademarks belong to their respective organizations.

118

Call for Patent Claims

119 This public review includes a call for information on essential patent claims (claims whose use
120 would be required for compliance with the guidance or requirements in this Information
121 Technology Laboratory (ITL) draft publication). Such guidance and/or requirements may be
122 directly stated in this ITL Publication or by reference to another publication. This call also
123 includes disclosure, where known, of the existence of pending U.S. or foreign patent applications
124 relating to this ITL draft publication and of any relevant unexpired U.S. or foreign patents.

125

126 ITL may require from the patent holder, or a party authorized to make assurances on its behalf,
127 in written or electronic form, either:

128

129 a) assurance in the form of a general disclaimer to the effect that such party does not hold
130 and does not currently intend holding any essential patent claim(s); or

131

132 b) assurance that a license to such essential patent claim(s) will be made available to
133 applicants desiring to utilize the license for the purpose of complying with the guidance
134 or requirements in this ITL draft publication either:

135

136 i. under reasonable terms and conditions that are demonstrably free of any unfair
137 discrimination; or

138 ii. without compensation and under reasonable terms and conditions that are
139 demonstrably free of any unfair discrimination.

140

141 Such assurance shall indicate that the patent holder (or third party authorized to make assurances
142 on its behalf) will include in any documents transferring ownership of patents subject to the
143 assurance, provisions sufficient to ensure that the commitments in the assurance are binding on
144 the transferee, and that the transferee will similarly include appropriate provisions in the event of
145 future transfers with the goal of binding each successor-in-interest.

146

147 The assurance shall also indicate that it is intended to be binding on successors-in-interest
148 regardless of whether such provisions are included in the relevant transfer documents.

149

150 Such statements should be addressed to: hwsec@nist.gov

151

152 **Table of Contents**

153 **1 Introduction 1**

154 **2 Hardware Platform Security Overview 3**

155 **3 Platform Integrity Verification 5**

156 3.1 Hardware Security Module (HSM) 5

157 3.2 The Chain of Trust (CoT) 6

158 3.3 Supply Chain Protection 7

159 **4 Software Runtime Protection Mechanisms 8**

160 4.1 Return Oriented Programming (ROP) and Call/Jump Oriented Programming

161 (COP/JOP) Attacks 8

162 4.2 Address Translation Attacks 8

163 4.3 Memory Safety Violations 9

164 4.4 Side-Channel Attacks 10

165 **5 Data Protection and Confidential Computing 12**

166 5.1 Memory Isolation 12

167 5.2 Application Isolation 13

168 5.3 VM Isolation 13

169 5.4 Cryptographic Acceleration 14

170 **6 Remote Attestation Services 15**

171 6.1 Platform Attestation 15

172 6.2 TEE Attestation 17

173 **7 Cloud Use Case Scenarios Leveraging Hardware-Enabled Security 19**

174 7.1 Visibility to Security Infrastructure 19

175 7.2 Workload Placement on Trusted Platforms 19

176 7.3 Asset Tagging and Trusted Location 21

177 7.4 Workload Confidentiality 22

178 7.5 Protecting Keys and Secrets 24

179 **8 Next Steps 26**

180 **References 27**

181 **List of Appendices**

182

183 **Appendix A— Vendor-Agnostic Technology Examples 35**

184 A.1 Platform Integrity Verification 35

185 A.1.1 UEFI Secure Boot (SB) 35

186 A.2 Keylime 36

187 **Appendix B— Intel Technology Examples..... 37**

188 B.1 Platform Integrity Verification 37

189 B.1.1 The Chain of Trust (CoT)..... 37

190 B.1.2 Supply Chain Protection 41

191 B.2 Software Runtime Protection Mechanisms 42

192 B.2.1 Return Oriented Programming (ROP) and Call/Jump Oriented

193 Programming (COP/JOP) Attacks 42

194 B.2.2 Address Translation Attacks 42

195 B.3 Data Protection and Confidential Computing 44

196 B.3.1 Memory Isolation 44

197 B.3.2 Application Isolation..... 45

198 B.3.3 VM Isolation..... 46

199 B.3.4 Cryptographic Acceleration 46

200 B.3.5 Technology Example Summary 47

201 B.4 Remote Attestation Services..... 48

202 B.4.1 Intel Security Libraries for the Data Center (ISecL-DC)..... 48

203 B.4.2 Technology Summary..... 48

204 **Appendix C— AMD Technology Examples..... 49**

205 C.1 Platform Integrity Verification 49

206 C.1.1 AMD Platform Secure Boot (AMD PSB) 49

207 C.2 Data Protection and Confidential Computing 49

208 C.2.1 Memory Isolation 49

209 C.2.2 VM Isolation..... 50

210 **Appendix D— Arm Technology Examples..... 52**

211 D.1 Platform Integrity Verification 52

212 D.1.1 Arm TrustZone TEE for Armv8-A..... 52

213 D.1.2 The Chain of Trust (CoT)..... 55

214 D.1.3 Platform Security Architecture (PSA) Functional APIs..... 57

215 D.1.4 Platform AbstRaction for SEcURITY (Parsec) 59

216 D.2 Software Runtime Protection Mechanisms 61

217 D.2.1 Return Oriented Programming (ROP) and Jump Oriented
218 Programming (JOP) Attacks..... 61

219 D.2.2 Memory Safety Violations..... 62

220 D.2.3 Side-Channel Attacks..... 64

221 D.3 Data Protection and Confidential Computing..... 66

222 D.3.1 Confidential Compute Architecture (CCA)..... 66

223 D.3.2 Cryptographic Acceleration..... 71

224 **Appendix E— Cisco Technology Examples..... 72**

225 E.1 Platform Integrity Verification..... 72

226 E.1.1 Cisco Platform Roots of Trust..... 72

227 E.1.2 The Chain of Trust (CoT)..... 73

228 E.2 Supply Chain Protection..... 73

229 E.3 Software Runtime Protections..... 73

230 E.4 Data Protection and Confidential Computing..... 74

231 E.5 Platform Attestation..... 74

232 E.6 Visibility to Security Infrastructure..... 74

233 E.7 Workload Placement on Trusted Platforms..... 74

234 **Appendix F— IBM Technology Examples..... 75**

235 F.1 Platform Integrity Verification..... 75

236 F.1.1 Hardware Security Module (HSM)..... 75

237 F.1.2 IBM Chain of Trust (CoT)..... 75

238 F.2 Software Runtime Protection Mechanisms..... 76

239 F.2.1 IBM ROP and COP/JOP Attack Defenses..... 76

240 F.3 Data Protection and Confidential Computing..... 76

241 F.3.1 IBM Memory Isolation Technology..... 76

242 F.3.2 IBM Application Isolation Technology..... 77

243 F.3.3 IBM VM Isolation Technology..... 77

244 F.3.4 IBM Cryptographic Acceleration Technology..... 78

245 F.4 Remote Attestation Services..... 78

246 F.4.1 IBM Platform Attestation Tooling..... 78

247 F.4.2 IBM Continuous Runtime Attestation..... 78

248 **Appendix G— Acronyms and Abbreviations..... 79**

249 **Appendix H— Glossary..... 85**

250
251
252
253
254
255
256
257
258
259
260
261
262
263

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 22

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

264 1 Introduction

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

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

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

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

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

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

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

2 Hardware Platform Security Overview

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

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

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

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

345 Without physical control over or visibility into platform configurations, conventional security
346 best practices and regulatory requirements become difficult or impossible to implement. With
347 new regulatory structures like the European General Data Protection Regulation (GDPR)
348 introducing high-stakes fines for noncompliance, having visibility and control over where data
349 may be accessed is more important than ever before. Top concerns among security professionals

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

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

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

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

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

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

389 **3 Platform Integrity Verification**

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

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

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

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

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

419 **3.1 Hardware Security Module (HSM)**

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

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

429 to store digest measurements of platform firmware and configuration during the boot process.
430 Each firmware module is measured by generating a digest, which is then extended to a TPM
431 platform configuration register (PCR). Multiple firmware modules can be extended to the same
432 PCR, and the TPM specification provides guidelines for which firmware measurements are
433 encompassed by each PCR [6].

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

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

448 **3.2 The Chain of Trust (CoT)**

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

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

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

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

- 471 • [UEFI Secure Boot \(SB\)](#)
- 472 • [Intel Trusted Execution Technology \(TXT\)](#)
- 473 • [Intel Boot Guard](#)
- 474 • [Intel Platform Firmware Resilience \(PFR\)](#)
- 475 • [Intel Technology Example Summary](#)
- 476 • [AMD Platform Secure Boot](#)
- 477 • [Arm TrustZone Trusted Execution Environment \(TEE\) for Armv8-A](#)
- 478 • [Arm Secure Boot and the Chain of Trust \(CoT\)](#)
- 479 • [Cisco Platform Roots of Trust](#)
- 480 • [IBM Chain of Trust \(CoT\)](#)

481 **3.3 Supply Chain Protection**

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

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

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

- 496 • [Intel Transparent Supply Chain \(TSC\)](#)
- 497 • [Intel PFR with Protection in Transit \(PIT\)](#)

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/projects/building-blocks/supply-chain-assurance>.

498 **4 Software Runtime Protection Mechanisms**

499 This section describes various software runtime attacks and protection mechanisms.

500 **4.1 Return Oriented Programming (ROP) and Call/Jump Oriented Programming** 501 **(COP/JOP) Attacks**

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

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

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

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

- 523 • [Intel Control-Flow Enforcement Technology \(CET\)](#)
- 524 • [Arm Pointer Authentication Code \(PAC\)](#)
- 525 • [Arm Branch Target Identification \(BTI\)](#)
- 526 • [IBM ROP and COP/JOP Attack Defenses](#)

527 **4.2 Address Translation Attacks**

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

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

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

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

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

- 555 • [Intel Hypervisor Managed Linear Address Translation \(HLAT\)](#)
- 556 • [Intel Supervisor Mode Execution Prevention \(SMEP\) and Supervisor Mode Access
557 Prevention \(SMAP\)](#)
- 558 • [AMD Supervisor Mode Execution Prevention \(SMEP\) and Supervisor Mode Access
559 Prevention \(SMAP\)](#)

560 4.3 Memory Safety Violations

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

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

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

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

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

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

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

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

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

- 604 • [Arm Privileged Access Never \(PAN\)](#)
- 605 • [Arm User \(EL0\) Execute Never \(UXN\) and Privileged \(EL1/EL2\) Execute Never \(PXN\)](#)
- 606 • [Arm Memory Tagging Extension \(MTE\)](#)
- 607 • [Arm Hardware Enforced Capability-based Architecture \(Morello and CHERI\)](#)

608 **4.4 Side-Channel Attacks**

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

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

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

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

- 628 • [Arm Mitigations Against Side-Channel Attacks](#)

629 **5 Data Protection and Confidential Computing**

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

639 While containers can provide a level of convenience, vulnerabilities in the kernel space and
640 shared layers can be susceptible to widespread exploitation, making security for the underlying
641 platform even more important. With the need for additional protection in the virtualized
642 workspace, an emphasis has been placed on encrypting data both at rest and while in use. *At-rest*
643 encryption provides protection for data on disk. This typically refers to an unmounted data store
644 and protects against threats such as the physical removal of a disk drive. Protecting and securing
645 cloud data while *in use*, also referred to as *confidential computing*, utilizes hardware-enabled
646 features to isolate and process encrypted data in memory so that the data is at less risk of
647 exposure and compromise from concurrent workloads or the underlying system and platform
648 [17]. This section describes technologies that can be leveraged for providing confidential
649 computing for cloud and edge.

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

659 **5.1 Memory Isolation**

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

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

- 669 • [Intel TME and Multi-Key Total Memory Encryption \(MKTME\)](#)
- 670 • [AMD Secure Memory Encryption \(SME\)/Transparent Memory Encryption \(TSME\)](#)
- 671 • [Arm Realm Memory Isolation and Protection](#)
- 672 • [Arm External Memory \(DRAM\) Encryption and Integrity with CCA](#)
- 673 • [IBM Memory Isolation Technology](#)

674 **5.2 Application Isolation**

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

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

- 684 • [Intel Software Guard Extensions \(SGX\)](#)
- 685 • [Arm Confidential Compute Architecture \(CCA\)](#)
- 686 • [Arm TrustZone Trusted Execution Environment \(TEE\) for Armv8-A](#)
- 687 • [Arm Realm Memory Isolation and Protection](#)
- 688 • [IBM Application Isolation Technology](#)

689 **5.3 VM Isolation**

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

697 VM isolation can be used to help protect workloads in multi-tenant environments like public and
698 hybrid clouds. Isolating entire VMs translates to protection against malicious insiders at the
699 cloud provider, or malware exposure and data leakage to other tenants with workloads running
700 on the same platform. Many modern cloud deployments use VMs as container worker nodes.
701 This provides a highly consistent and scalable way to deploy containers regardless of the
702 underlying physical platforms. With full VM isolation, the virtual workers hosting container
703 workloads can be effectively isolated without impacting the benefits of abstracting the container
704 from the underlying platform.

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

- 706 • [Intel Trust Domain Extensions \(TDX\)](#)
- 707 • [AMD Secure Encrypted Virtualization \(SEV\)](#)
- 708 • [Arm Confidential Compute Architecture \(CCA\)](#)
- 709 • [IBM VM Isolation Technology](#)

710 **5.4 Cryptographic Acceleration**

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

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

- 719 • [Intel Advanced Encryption Standard New Instructions \(AES-NI\)](#)
- 720 • [Intel QuickAssist Technology \(QAT\) with Intel Key Protection Technology \(KPT\)](#)
- 721 • [AMD Advanced Encryption Standard](#)
- 722 • [Arm Cryptographic Acceleration](#)
- 723 • [IBM Cryptographic Acceleration Technology](#)

6 Remote Attestation Services

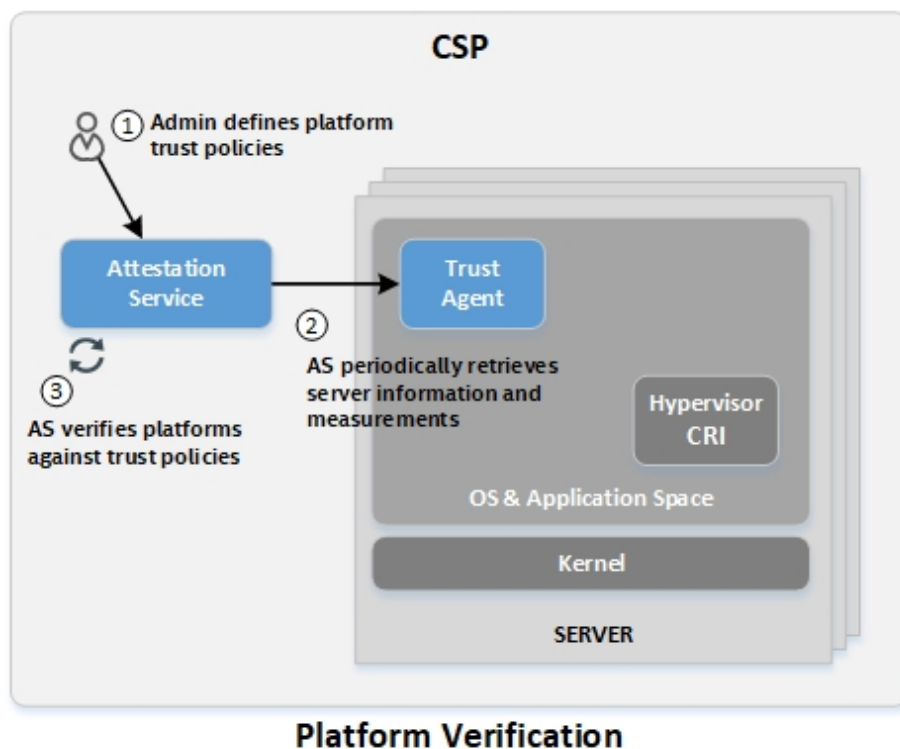
725 Measuring a server's firmware/configuration and extending these measurements to a hardware
726 interface can help keep track of which firmware is running on a platform. Some platform
727 integrity technologies can even perform local attestation and enforcement of firmware and
728 configuration on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

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

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

6.1 Platform Attestation

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



758
759

Figure 1: Notional Example of Remote Attestation Service

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

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

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

- 780 • [Intel Security Libraries for the Data Center \(ISecL-DC\)](#)
- 781 • [Remote Attestation Service - Project Veraison \(VERificAtIon of atteStatiON\)](#)
- 782 • [IBM Platform Attestation Tooling](#)

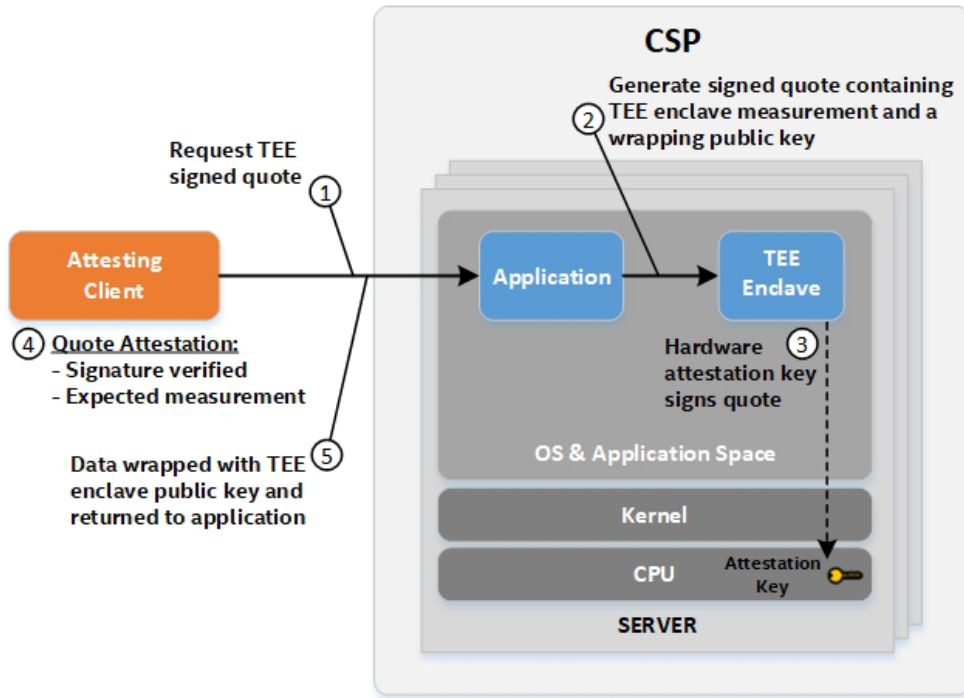
783 **6.2 TEE Attestation**

784 There are instances when the high assurance that the output of the processing in a TEE can be
785 trusted should be extended to an external attesting client. This is achieved thanks to a TEE
786 attestation flow. *TEE attestation* involves the generation of a verifiable cryptographic quote of
787 the enclave by the TEE. The quote is then sent to the attesting client, which can validate the
788 signature of the quote. If the signature is valid, the attesting client concludes that the remote code
789 is running in a genuine TEE enclave.

790 A quote usually contains the measurement of the TEE enclave, as well as data related to the
791 authenticity of the TEE and the compliant version of it. The measurement is a digest of the
792 content of the enclave (e.g., code and static data) and other information. The measurement
793 obtained at build time is typically known to the attesting client and is compared against a
794 measurement contained in the quote that is actively taken during runtime. This allows the
795 attesting client to determine that the remote code has not been tampered with. A quote may also
796 contain the enclave's developer signature and platform trusted computing base (TCB)
797 information. The authenticity and version of the TEE are verified against TEE provider
798 certificates that are accessible to the tenant or attesting client.

799 The quote may also contain the public key part of an enclave key pair or a secure hash of the
800 public/private key part if there is a limitation on the size of the quote. In the latter case, the
801 public key part must be communicated along with the quote. The public key allows the attesting
802 client to wrap secrets that it wants to send to the enclave. This capability allows the attesting
803 client to provision secrets directly to the TEE enclave without needing to trust any other software
804 running on the server.

805 Figure 2 shows an example TEE attestation flow.



806

807

Figure 2: Notional Example of TEE Attestation Flow

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

809

- [Arm Secure Boot and the Chain of Trust \(CoT\)](#)

810

- [IBM Continuous Runtime Attestation](#)

811 **7 Cloud Use Case Scenarios Leveraging Hardware-Enabled Security**

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

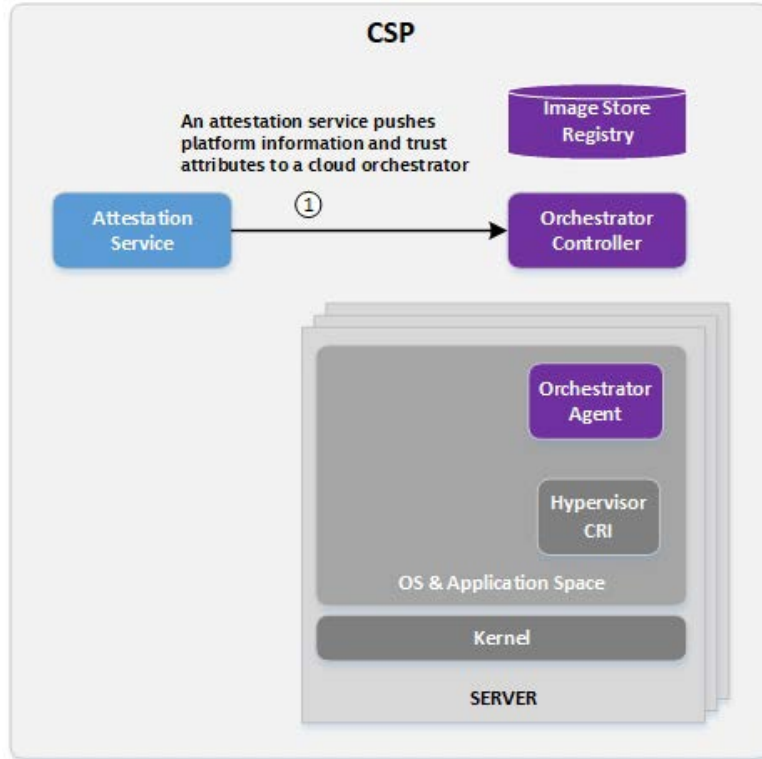
815 **7.1 Visibility to Security Infrastructure**

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

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

828 **7.2 Workload Placement on Trusted Platforms**

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

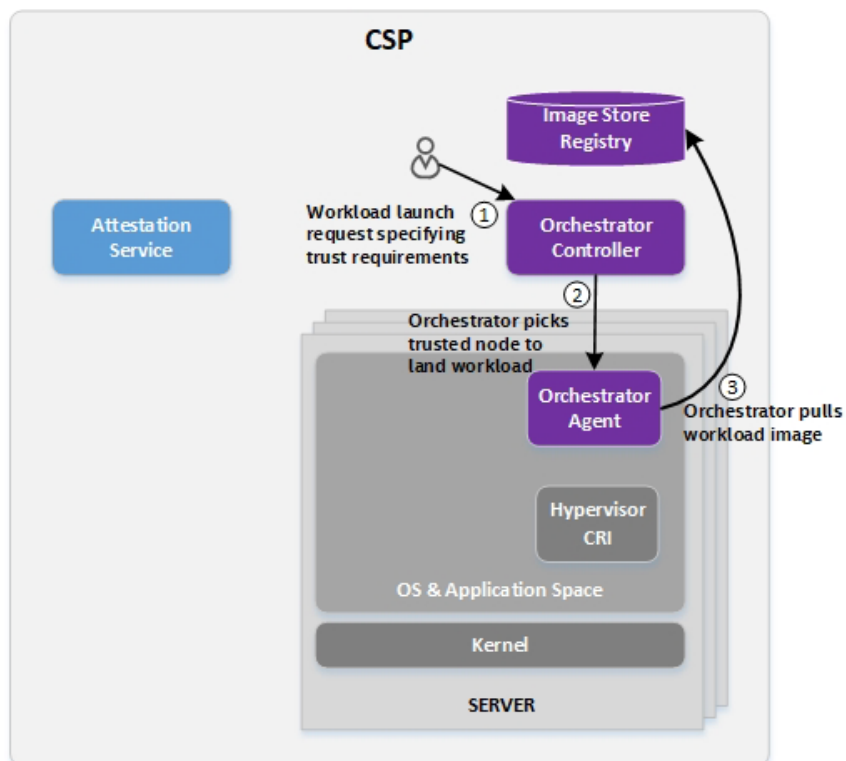


835

836

Figure 3: Notional Example of Orchestrator Platform Labeling

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



848

849

Figure 4: Notional Example of Orchestrator Scheduling

850 7.3 Asset Tagging and Trusted Location

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

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

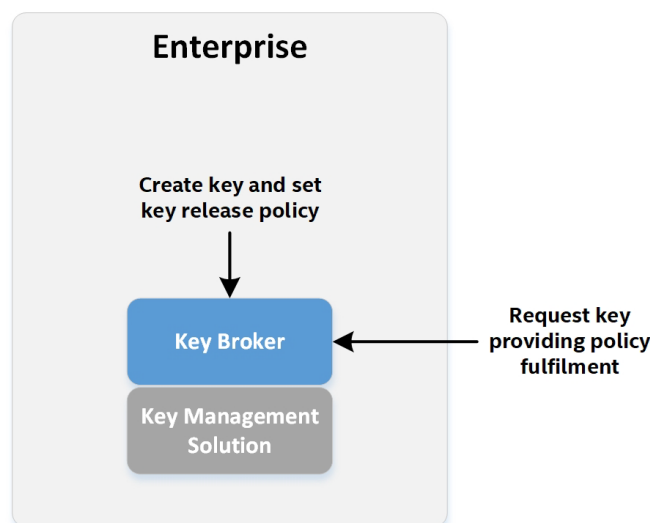
869 Outside of specific regulatory requirements, an organization may wish to segregate workloads by
 870 department. For example, human resources and finance information could be restricted to

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

879 7.4 Workload Confidentiality

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

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

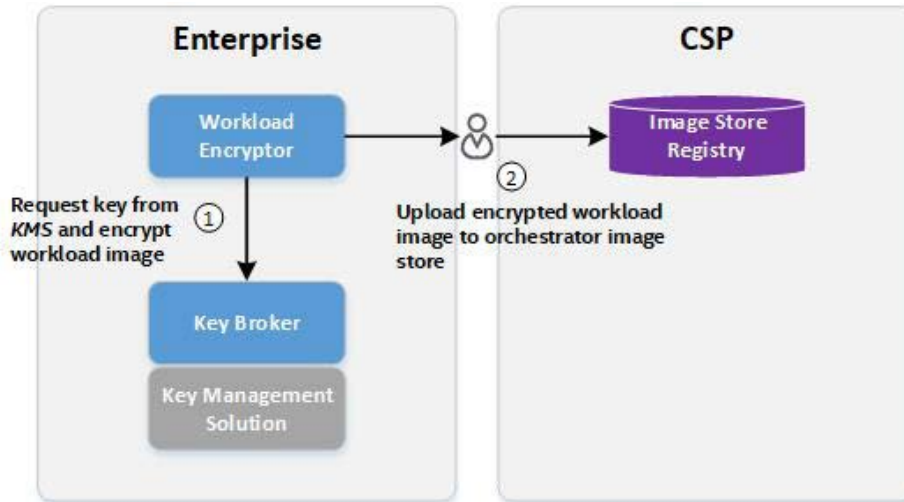


898

899

Figure 5: Notional Example of Key Brokerage

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

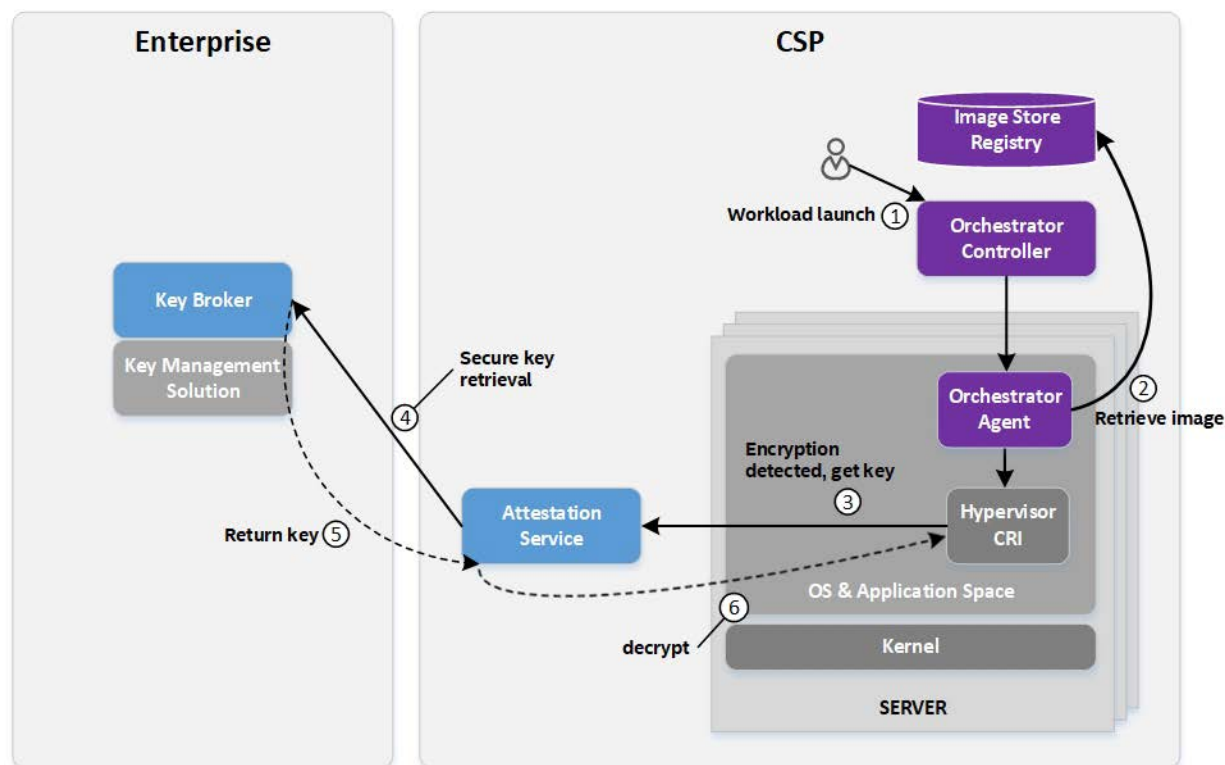


903

904

Figure 6: Notional Example of Workload Image Encryption

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



911

912

Figure 7: Notional Example of Workload Decryption

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

921 7.5 Protecting Keys and Secrets

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

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

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

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

8 Next Steps

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

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

953 NIST is also working on other publications on hardware-enabled security as part of the NCCoE
954 Trusted Cloud project. More information on the project and links to the other publications are
955 available at <https://www.nccoe.nist.gov/projects/building-blocks/trusted-cloud>.

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/projects/building-blocks/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 (2020) *Cyber Supply Chain Risk Management*. Available at <https://csrc.nist.gov/Projects/cyber-supply-chain-risk-management>

- [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 (2021) *Trusted Firmware-A Project*. Available at <https://www.trustedfirmware.org/projects/tf-a/>
- [13] Arm (2021) *Morello*. Available at <https://developer.arm.com/architectures/cpu-architecture/a-profile/morello>
- [14] Arm (2021) *Arm Morello System Development Platform Preliminary 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 (2020) *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 (2020) *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 (2021) *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 (2018) *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://www.apress.com/gp/book/9781430261483>
- [24] Linux Kernel Organization (2020) *Intel® TXT Overview*. Available at https://www.kernel.org/doc/Documentation/intel_txt.txt
- [25] Intel Corporation (2020) *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). Available at <https://software.intel.com/content/www/us/en/develop/blogs/intel-release-new-technology-specifications-protect-rop-attacks.html>
- [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 <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 (2020) *Strengthen Enclave Trust with Attestation*. Available at <https://software.intel.com/en-us/sgx/attestation-services>
- [38] European Telecommunications Standards Institute (2020) *Network Functions Virtualisation (NFV)*. Available at <http://www.etsi.org/technologies-clusters/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 (2012) Intel Data Protection Technology with AES-NI and Secure Key. 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 (2020) *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). Available at <https://www.aspsys.com/images/solutions/hpc-processors/intel-xeon/Intel-Key-Protection-Technology.pdf>
- [43] Intel Corporation (2020) *Intel® Security Libraries for Data Center (Intel® SecL-DC)*. Available at <https://01.org/intel-secl>
- [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 <https://www.amd.com/system/files/TechDocs/SEV-SNP-strengthening-vm-isolation-with-integrity-protection-and-more.pdf>
- [47] Arm (2021) *Arm Architecture Reference Manual Armv8, for A-profile architecture*. Available at <https://developer.arm.com/documentation/ddi0487/latest>
- [48] Arm (2021) *Trusted Base System Architecture for Armv8-A, 3rd Edition*. Available at <https://developer.arm.com/documentation/den0021/f/?lang=en>
- [49] Arm (2021) *TrustZone for AArch64*. Available at <https://developer.arm.com/documentation/102418/0100/What-is-TrustZone-?lang=en>
- [50] Arm (2021) *Parsec*. Available at <https://developer.arm.com/solutions/infrastructure/developer-resources/security/parsec>
- [51] Parsec Project (2019) *Authenticators*. Available at https://parallaxsecond.github.io/parsec-book/parsec_service/authenticators.html
- [52] SPIFFE (2021) *SPIFFE Verifiable Identity Document (SVID)*. Available at <https://spiffe.io/docs/latest/spiffe-about/spiffe-concepts/#spiffe-verifiable-identity-document-svid>
- [53] SPIFFE (2021) *SPIRE Concepts*. Available at <https://spiffe.io/docs/latest/spire-about/spire-concepts/>
- [54] SPIFFE (2021) *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-acv>
- [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, Tsyklevich 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 Laboratory, Technical Report Number 951. Available at <https://www.cl.cam.ac.uk/techreports/UCAM-CL-TR-951.pdf>
- [61] Arm (2021) *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/support/arm-security-updates/speculative-processor-vulnerability/downloads>
- [63] Arm (2020) *Straight-line Speculation, Version 1.0*. Available at <https://developer.arm.com/support/arm-security-updates/speculative-processor-vulnerability/downloads>
- [64] Arm (2021) *Arm v8.5-A/v9 CPU updates, Version 1.5*. Available at <https://developer.arm.com/support/arm-security-updates/speculative-processor-vulnerability/downloads>
- [65] Arm (2021) *Introducing Arm Confidential Compute Architecture*. 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] Mandyam G, Lundblade L, Ballesteros M, O’Donoghue J (2021) *The Entity Attestation Token (EAT)*. (Internet Engineering Task Force, RATS Working Group), IETF Internet-Draft draft-ietf-rats-eat-10. <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 (2021) *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 <https://community.ibm.com/community/user/power/blogs/chris-engel1/2020/06/17/power9-introduces-secure-boot-to-powervm>
- [75] Heller D and 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 (2021) *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 and 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 and 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
- [87] IBM (2021) *CEX7S / 4769 Overview: IBM Systems cryptographic hardware products.* Available at <https://www.ibm.com/security/cryptocards/pciecc4/overview>
- [88] IBM (2015) *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

957 Appendix A—Vendor-Agnostic Technology Examples

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

960 A.1 Platform Integrity Verification

961 A.1.1 UEFI Secure Boot (SB)

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

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

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

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

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

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

998 On systems with a TPM chip enabled and supported by the system firmware, shim will extend
999 various PCRs with the digests of the targets it is loading [21]. Certificate hashes are also
1000 extended to the TPM, including system, vendor, MOK, and shim denylisted and allowlisted
1001 certificate digests.

1002 **A.2 Keylime**

1003 “[Keylime](#) is an open source project hosted by the [Cloud Native Computing Foundation \(CNCF\)](#),
1004 a vendor-neutral forum with more than 145 user organizations using cloud native technologies to
1005 build their products and services around many of the open source projects, including, for
1006 example, Kubernetes. Keylime provides a highly scalable remote boot attestation and runtime
1007 integrity measurement solution. Keylime enables users to monitor remote nodes using a
1008 hardware based cryptographic root of trust.

1009 Keylime was originally born out of the security research team in MIT's Lincoln Laboratory.

1010 Keylime provides an end-to-end solution for bootstrapping hardware-rooted cryptographic trust
1011 for remote machines, the provisioning of encrypted payloads, and run-time system integrity
1012 monitoring. It also provides a flexible framework for the remote attestation using a TPM-based
1013 hardware root of trust. Users can create their own customized actions that will trigger when a
1014 machine fails its attested measurements.

1015 Keylime's goal is to make TPM technology easily accessible to developers and users alike,
1016 without the need for a deep understanding of the lower levels of a TPM's operations. Amongst
1017 many scenarios, it is well suited to tenants who need to remotely attest machines not under their
1018 own full control (such as a consumer of hybrid cloud or a remote edge/IoT device in an insecure
1019 physical tamper-prone location).

1020 Keylime can be used to monitor an entire fleet of OpenShift worker nodes and take immediate
1021 action if any node is compromised. It measures trusted boot of machines and run-time integrity
1022 using the Linux Kernels Integrity Measurement Architecture.”

1023 Appendix B—Intel Technology Examples

1024 This section describes a number of Intel technology examples that map back to the key concepts
1025 described in the various sections of the document.

1026 B.1 Platform Integrity Verification

1027 B.1.1 The Chain of Trust (CoT)

1028 B.1.1.1 Intel Trusted Execution Technology (TXT)

1029 Intel Trusted Execution Technology (TXT) in conjunction with a TPM provides a hardware RoT
1030 available on Intel server and client platforms that enables “security capabilities such as measured
1031 launch and protected execution” [22]. TXT utilizes *authenticated code modules (ACMs)* that
1032 measure various pieces of the CoT during boot time and extend them to the platform TPM
1033 [2][22]. TXT’s ACMs are chipset-specific signed binaries that are called to perform functions
1034 required to enable the TXT environment. An ACM is loaded into and executed from within the
1035 CPU cache in an area referred to as the *authenticated code RAM (AC RAM)*. CPU microcode,
1036 which acts as the *core root of trust for measurement (CRTM)*, authenticates the ACM by
1037 verifying its included digital signature against a manufacturer public key with its digest hard-
1038 coded within the chipset. The ACM code, loaded into protected memory inside the processor,
1039 performs various tests and verifications of chipset and processor configurations.

1040 The ACMs needed to initialize the TXT environment are the BIOS and the Secure Initialization
1041 (SINIT) ACMs. Both are typically provided within the platform BIOS image. The Secure
1042 Initialization Authenticated Code Module (SINIT ACM) can be provisioned on disk as well
1043 [2][23]. The BIOS ACM is responsible for measuring the BIOS firmware to the TPM and
1044 performs additional BIOS-based security operations. The latest version of TXT converged with
1045 Intel Boot Guard Technology labels this ACM as the Startup ACM to differentiate it from the
1046 legacy BIOS ACM. The SINIT ACM is used to measure the system software or OS to the TPM,
1047 and it “initializes the platform so the OS can enter the secure mode of operation” [23].

1048 When the BIOS startup procedures have completed, control is transitioned to the OS loader. In a
1049 TXT-enabled system, the OS loader is instructed to load a special module called Trusted Boot
1050 before loading the first kernel module [23]. Trusted Boot (tboot) is an open-source, pre-
1051 kernel/virtual machine manager (VMM) module that integrates with TXT to perform a measured
1052 launch of an OS kernel/VMM. The tboot design typically has two parts: a preamble and the
1053 trusted core. The tboot preamble is most commonly executed by the OS loader but can be loaded
1054 at OS runtime. The tboot preamble is responsible for preparing SINIT input parameters and is
1055 untrusted by default. It executes the processor instruction that passes control to the CPU
1056 microcode. The microcode loads the SINIT into AC RAM, authenticates it, measures SINIT to
1057 the TPM, and passes control to it. SINIT verifies the platform configuration and enforces any
1058 present Launch Control Policies, measuring them and tboot trusted core to the TPM. The tboot
1059 trusted core takes control and continues the CoT, measuring the OS kernel and additional
1060 modules (like initrd) before passing control to the OS [24].

1061 Intel TXT includes a policy engine feature that provides the capability to specify known good
1062 platform configurations. These *Launch Control Policies (LCPs)* dictate which system software is

1063 permitted to perform a secure launch. LCPs can enforce specific platform configurations and
1064 tboot trusted core versions required to launch a system environment [23].

1065 **B.1.1.2 Intel Boot Guard**

1066 Intel Boot Guard provides a hardware RoT for authenticating the BIOS. An original equipment
1067 manufacturer (OEM) enables Boot Guard authentication on the server manufacturing line by
1068 permanently fusing a policy and OEM-owned public key into the silicon. When an Intel
1069 processor identifies that Boot Guard has been enabled on the platform, it authenticates and
1070 launches an ACM. The ACM loads the initial BIOS or Initial Boot Block (IBB) into the
1071 processor cache, authenticates it using the fused OEM public key, and measures it into the TPM.

1072 If the IBB authenticates properly, it verifies the remaining BIOS firmware, loads it into memory,
1073 and transfers execution control. The IBB is restricted to this limited functionality, which allows it
1074 to have a small enough size to fit in the on-die cache memory of Intel silicon. If the Boot Guard
1075 authentication fails, the system is forced to shut down. When the Boot Guard execution
1076 completes, the CoT can continue for other components by means of SB. TXT can be used in
1077 conjunction with these technologies to provide a dynamic trusted launch of the OS kernel and
1078 software.

1079 Because Boot Guard is rooted in permanent silicon fuses and authenticates the initial BIOS from
1080 the processor cache, it provides resistance from certain classes of physical attacks. Boot Guard
1081 also uses fuses to provide permanent revocation of compromised ACMs, BIOS images, and input
1082 polices.

1083 **B.1.1.3 Intel Platform Firmware Resilience (PFR)**

1084 Intel Platform Firmware Resilience (PFR) technology is a platform-level solution that creates an
1085 open platform RoT based on a programmable logic device. It is designed to provide firmware
1086 resiliency (in accordance with NIST SP 800-193 [4]) and comprehensive protection for various
1087 platform firmware components, including BIOS, Server Platform Services Firmware (SPS FW),
1088 and BMCs. PFR provides the platform owner with a minimal trusted compute base (TCB) under
1089 full platform-owner control. This TCB provides cryptographic authentication and automatic
1090 recovery of platform firmware to help guarantee correct platform operation and to return to a
1091 known good state in case of a malicious attack or an operator error such as a failed update.

1092 NIST SP 800-193 [4] outlines three guiding principles to support the resiliency of platforms
1093 against potentially destructive attacks:

- 1094 • **Protection:** Mechanisms for ensuring that platform firmware code and critical data
1095 remain in a state of integrity and are protected from corruption, such as the process for
1096 ensuring the authenticity and integrity of firmware updates
- 1097 • **Detection:** Mechanisms for detecting when platform firmware code and critical data have
1098 been corrupted
- 1099 • **Recovery:** Mechanisms for restoring platform firmware code and critical data to a state
1100 of integrity in the event that any such firmware code or critical data are detected to have

1101 been corrupted or when forced to recover through an authorized mechanism. Recovery is
1102 limited to the ability to recover firmware code and critical data.

1103 In addition, NIST SP 800-193 [4] provides guidance on meeting those requirements via three
1104 main functions of a Platform Root of Trust:

- 1105 • **Root of Trust for Update**, which is responsible for authenticating firmware updates and
1106 critical data changes to support platform protection capabilities; this includes signature
1107 verification of firmware updates as well as rollback protections during update.
- 1108 • **Root of Trust for Detection**, which is responsible for firmware and critical data
1109 corruption detection capabilities.
- 1110 • **Root of Trust for Recovery**, which is responsible for recovery of firmware and critical
1111 data when corruption is detected or when instructed by an administrator.

1112 PFR is designed to support NIST guidelines and create a resilient platform that is able to self-
1113 recover upon detection of attack or firmware corruption. This includes verification of all
1114 platform firmware and configuration at platform power-on time, active protection of platform
1115 non-volatile memory at runtime, and active protection of the Serial Peripheral Interface (SPI
1116 flash) and system management bus (SMBus). PFR functionality also incorporates monitoring the
1117 platform component's boot progress and providing automatic firmware recovery to a known
1118 good state upon detection of firmware or configuration corruption. PFR achieves this goal by
1119 utilizing a Field-Programmable Gate Array (FPGA) to establish an RoT.

1120 PFR technology defines a special pre-boot mode (T-1) where only the PFR FPGA is active. Intel
1121 Xeon processors and other devices that could potentially interfere with the boot process, such as
1122 the Platform Controller Hub (PCH)/Manageability Engine (ME) and BMC, are not powered.
1123 Boot critical firmware, like the BIOS, ME, and BMC, are cryptographically verified during T-1
1124 mode. In case of corruption, a recovery event is triggered, and the corrupted firmware in the
1125 active regions of the SPI flash is erased and restored with a known-good recovery copy. Once
1126 successful, the system proceeds to boot in a normal mode, leveraging Boot Guard for static RoT
1127 coverage.

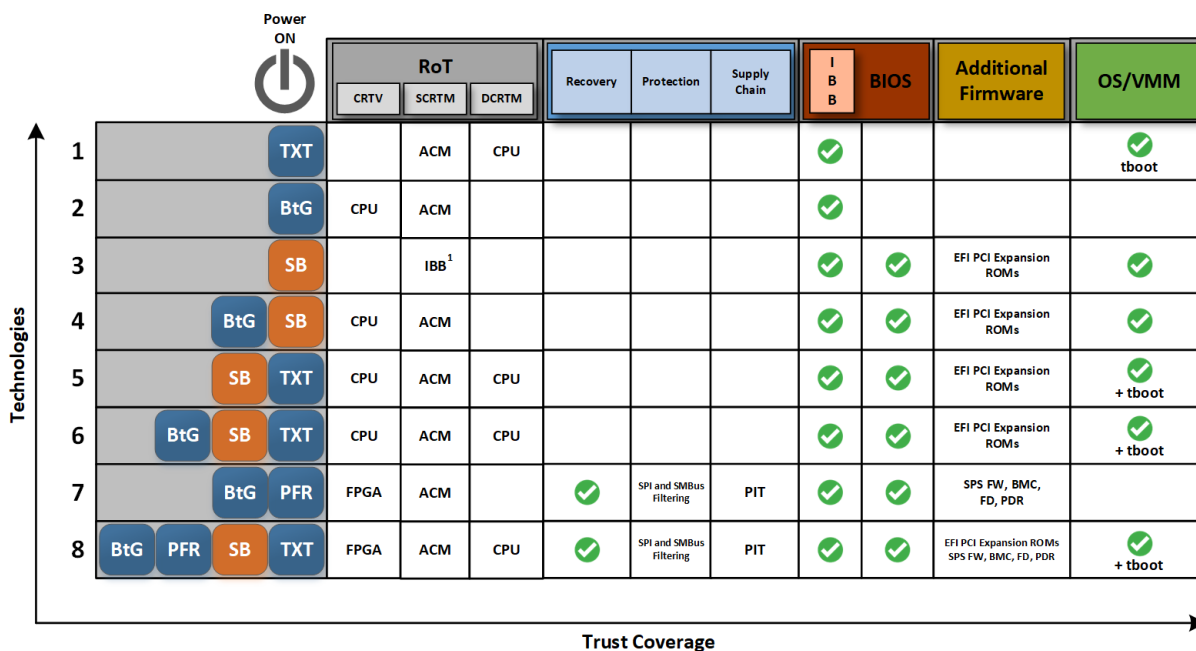
1128 The PFR FPGA RoT leverages a key hierarchy to authenticate data structures residing in SPI
1129 flash. The key hierarchy is based on a provisioned Root Key (RK) stored in the NVRAM of the
1130 FPGA RoT and a Code Signing Key (CSK) structure, which is endorsed by the RK, stored in the
1131 SPI flash, and used for the signing of lower-level data structures. The PFR FPGA uses this CSK
1132 to verify the digital signature of the Platform Firmware Manifest, which describes the expected
1133 measurements of the platform firmware. The PFR FPGA RoT verifies those measurements
1134 before allowing the system to boot. When a recovery is needed, either because measurements do
1135 not match the expected value or because a hang is detected during system bootup, the PFR
1136 FPGA RoT uses a recovery image to recover the firmware. The recovery image and any update
1137 images are stored in a compressed capsule format and verified using a digital signature.

1138 Each platform firmware storage is divided into three major sections: Active, Recovery, and
1139 Staging. The Recovery regions, as well as the static parts of the Active regions, are write-
1140 protected from other platform components by the PFR FPGA RoT. The Staging region is open to
1141 the other platform components for writing in order to provide an area to place digitally signed

1142 and compressed update capsules, which are then verified by the PFR FPGA RoT before being
 1143 committed to the Active or Recovery regions. The Recovery copy can be updated in T-1 mode
 1144 once the PFR FPGA has verified the digital signature of the update capsule and confirmed that
 1145 the recovery image candidate is bootable.

1146 **B.1.1.4 Technology Example Summary**

1147 There are several technologies that provide different levels of platform integrity and trust.
 1148 Individual technologies do not provide a complete CoT. When used in combination, they can
 1149 provide comprehensive coverage all the way up to the OS and VMM layer. Figure 8 outlines the
 1150 firmware and software coverage of each existing CoT technology example.



1 IBB is meant to describe the portion of BIOS which performs the first measurement

1151

1152

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

1153 Figure 8 identifies the components of each technology that make up the RoT in their own
 1154 respective chains and also shows a rough outline of the firmware and software coverage of each
 1155 technology.

1156 Because many technologies are available, it can be difficult to decide on the correct combination
 1157 for deployment. Figure 8 illustrates the possible combinations of technologies that extend
 1158 measurements to a TPM for platform integrity attestation. Note that each combination includes at
 1159 least one hardware technology to ensure an RoT implementation. A complementary option for
 1160 extending the CoT up through the OS can also be provided. Including only the hardware
 1161 technologies would break the CoT by supplying integrity measurements for only pre-OS
 1162 firmware. Using only SB will use firmware as the RoT that does not have hardware security
 1163 protections and is much more susceptible to attack. By enabling both parts, the CoT can be
 1164 extended from a hardware RoT into the OS and beyond.

1165 These combinations will help ensure that appropriate measurements are extended to a TPM for
 1166 integrity attestation and can prevent a server from booting if specific security modules are
 1167 compromised. The attestation mechanisms provided by these technologies give cryptographic
 1168 proof of the integrity of measured components, which can be used to provide visibility into
 1169 platform security configurations and prove integrity. Note the combination of UEFI SB with
 1170 TXT in Figure 8. This combination provides the UEFI SB signature verification capability on top
 1171 of the tboot integrity measurement in the OS/VMM layer.

1172 In addition to attestation, PFR provides both additional verification of platform firmware and
 1173 adds automatic recovery of compromised firmware to known good versions. PFR works with any
 1174 combination of CoT technologies, providing a defense and resilience against firmware attack
 1175 vectors. Combining a hardware-based firmware resilience technology like PFR with a hardware-
 1176 based CoT configuration is part of a layered security strategy.

1177 **B.1.2 Supply Chain Protection**

1178 **B.1.2.1 Intel Transparent Supply Chain (TSC)**

1179 “Intel Transparent Supply Chain (TSC) is a set of policies and procedures implemented at ODM
 1180 factories that enable end-users to validate where and when every component of a platform was
 1181 manufactured” [25]. “Intel TSC tools allow platform manufacturers to bind platform information
 1182 and measurements using [a TPM]. This allows customers to gain traceability and accountability
 1183 for platforms with component-level reporting” [26].

1184 Intel TSC provides the following key features [25]:

- 1185 • Digitally signed statement of conformance for every platform
- 1186 • Platform certificates linked to a discrete TPM, providing system-level traceability
- 1187 • Component-level traceability via a direct platform data file that contains integrated
 1188 components, including a processor, storage, memory, and add-in cards
- 1189 • Auto Verify tool that compares the snapshot of the direct platform data taken during
 1190 manufacturing with a snapshot of the platform components taken at first boot
- 1191 • Firmware load verification
- 1192 • Conformity with Defense Federal Acquisition Regulation Supplement (DFARS)
 1193 246.870-2 [27]

1194 **B.1.2.2 PFR with Protection in Transit (PIT)**

1195 In addition to the platform protection, detection, and recovery features, PFR also offers
 1196 protection in transit (PIT) or supply chain protection. Platform lockdown requires that a
 1197 password be present in the PFR FPGA as well as a radio frequency (RF) component. The
 1198 password is removed before platform shipment and must be replaced before the platform will be
 1199 allowed to power up. With platform firmware sealing, the PFR FPGA computes hashes of
 1200 platform firmware in the PCH and BMC attached flash devices, including static and dynamic
 1201 regions, and stores them in an NVRAM space before shipment. Upon delivery, the PFR FPGA

1202 will recompute the hashes and report any mismatches to ensure that the firmware has not been
1203 tampered with during system transit.

1204 **B.2 Software Runtime Protection Mechanisms**

1205 **B.2.1 Return Oriented Programming (ROP) and Call/Jump Oriented Programming** 1206 **(COP/JOP) Attacks**

1207 **B.2.1.1 Intel Control-Flow Enforcement Technology (Intel CET)**

1208 Intel CET is an instruction set extension to implement control flow integrity and defend against
1209 ROP and COP/JOP style subversion attacks. ROP and similarly COP/JOP have been the
1210 prevalent attack methodology for stealth exploit writers targeting vulnerabilities in programs.
1211 [28]

1212 Intel CET prevents this class of exploits by providing the following capabilities:

- 1213 • Shadow stack – return address protection to defend against ROP
- 1214 • Indirect branch tracking – free branch protection to defend against COP/JOP

1215 “CET introduces a shadow stack system to detect and thwart the stack manipulation required by
1216 ROP” [29]. This second stack is used exclusively for control transfer operations and is designed
1217 to be protected from application code memory accesses while keeping track of CPU stored
1218 copies of the return addresses [30]. “When CET is enabled, a CALL instruction pushes the return
1219 address into a shadow stack in addition to its normal behavior of pushing return address into the
1220 normal stack (no changes to traditional stack operation). The return instructions (e.g. RET) pops
1221 return address from both shadow and traditional stacks, and only transfers control to the popped
1222 address if return addresses from both stacks match. [...] The page table protections for shadow
1223 stack are also designed to protect the integrity of the shadow stack by preventing unintended or
1224 malicious switching of shadow stack and/or overflow and underflow of shadow stack.” [31]

1225 “CET also adds an indirect branch tracking capability to provide software the ability to restrict
1226 COP/JOP attacks.” [30] This ENDBRANCH instruction is a new addition to Intel Instruction Set
1227 Architecture (ISA). It marks legal targets for an indirect branch or jump, forcing the CPU to
1228 generate an exception for unintended or malicious operations [31].

1229 “Intel has been actively collaborating with Microsoft and other industry partners to address
1230 control-flow hijacking by using Intel’s CET technology to augment the previous software-only
1231 control-flow integrity solutions. Intel’s CET, when used properly by software, is a big step in
1232 helping to prevent exploits from hijacking the control-flow transfer instructions.” [31] A security
1233 analysis of Intel CET is published in [32].

1234 **B.2.2 Address Translation Attacks**

1235 **B.2.2.1 Intel Hypervisor Managed Linear Address Translation (HLAT)**

1236 Hypervisor managed linear address translation (HLAT) is a capability to enable Intel
1237 Virtualization Technology (Intel VT-x) based security monitors to enforce runtime protection

1238 and integrity assertions on OS-managed page tables. This helps protect kernel assets, as well as
1239 in-band security agents and agent-monitored assets from OS page-table attacks.

1240 “[HLAT] is intended to be used by a Hypervisor/Virtual Machine Monitor (VMM) to enforce
1241 guest linear translation (to guest physical mappings). When combined with the existing Extended
1242 Page Table (EPT) capability, HLAT enables the VMM to ensure the integrity of combined guest
1243 linear translation (mappings and permissions) cached by the processor TLB, via a reduced
1244 software TCB managed by the VMM.” [33] In this fashion, the VMM-enforced guest
1245 translations are more protected from alterations by untrusted system software adversaries. [33]

1246 “This feature is intended to augment the security functionality for a type of Virtual Machine
1247 Monitor (VMM) that may use legacy EPT read/write/execute (XWR) permission bits (bits 2:0 of
1248 the EPTE) as well as the new user-execute (XU) access bit (bit 10 of the EPTE) to ensure the
1249 integrity of code/data resident in guest physical memory assigned to the guest OS. EPT
1250 permissions are also used in these VMMs to isolate memory; for example, to host a Secure
1251 Kernel (SK) that can manage security properties for the general purpose kernel (GPK). For such
1252 usages, it is important that the VMM ensure that the guest linear address mappings which are
1253 used by the General Purpose Kernel to refer to the EPT monitored guest physical pages are
1254 access-controlled as well.” [33]

1255 “VMMs could enforce the integrity of these specific guest linear to guest physical mappings
1256 (paging structures) by using legacy EPT permissions to mark the guest physical memory
1257 containing the relevant guest paging structures as read-only. The intent of marking these guest
1258 paging structures as read-only is to ensure an invalid mapping is not created by guest software.
1259 However, such page-table edit control techniques are known to cause very high overheads due to
1260 the requirement that the VMM must monitor all paging contexts created by the (Guest) operating
1261 system. HLAT enables a VMM to enforce the integrity of guest linear mappings without this
1262 high overhead.” [33]

1263 HLAT utilizes a processor mechanism that implements an alternate Intel Itanium architecture
1264 (IA) paging structure managed in guest physical memory by a Secure Kernel. This paging
1265 structure contains guest linear to guest physical translations that the VMM/Secure Kernel wants
1266 to enforce.

1267 Additionally, to accommodate legacy page-table monitoring approaches, HLAT defines two new
1268 EPT control bits in EPT leaf entries. A “Paging-Write” control bit specifies which guest physical
1269 pages hold HLAT or legacy IA paging structures. This allows the processor to use the Paging-
1270 Write as permission to perform A/D bit writes, instead of the software W permission in the
1271 EPTE. A “Verify Paging-Write” control bit specifies which guest physical pages should only be
1272 referenced via translation (guest) paging structures marked as Paging-writable under EPT [33].

1273 **B.2.2.2 Intel Supervisor Mode Execution Prevention (SMEP) and Supervisor Mode** 1274 **Access Prevention (SMAP)**

1275 Supervisor Mode Execution Prevention (SMEP) and Supervisor Mode Access Prevention
1276 (SMAP) are opt-in capabilities that can be used by systems software (such as the kernel) to
1277 harden the privilege separation between user-mode and kernel-mode. These capabilities further

1278 enforce the user/supervisor properties specified via address translation mechanisms by mitigating
1279 malicious code execution or malicious use of data setup by processes executing in user-mode.

1280 Intel OS Guard, also known as SMEP, helps prevent execution out of untrusted application
1281 memory while operating at a more privileged (supervisor) level. “[When] enabled, the operating
1282 system will not be allowed to directly execute application code, even speculatively. This makes
1283 branch target injection attacks on the OS substantially more difficult by forcing the attacker to
1284 find gadgets within the OS code. It is also more difficult for an application to train OS code to
1285 jump to an OS gadget. All major operating systems enable SMEP support by default.” [34]

1286 SMAP is a security feature that helps prevent unauthorized kernel consumption of data
1287 accessible to user space [35]. An enabling SMAP bit in the CR4 control register will cause a
1288 page fault to be triggered when there is any attempt to access user-space memory while running
1289 in a privileged mode. When access to user space memory is needed by the kernel, a separate AC
1290 flag is toggled to allow the required access [36]. “Two new instructions (STAC and CLAC) are
1291 provided to manipulate that flag relatively quickly.” When the AC flag is set in protection mode
1292 under normal operating circumstances, SMAP blocks a whole class of exploits where the kernel
1293 is fooled into reading from (or writing to) user-space memory by mistake. SMAP also allows for
1294 the early discovery of kernel bugs where developers dereference user space pointers directly
1295 from the kernel [36].

1296 **B.3 Data Protection and Confidential Computing**

1297 **B.3.1 Memory Isolation**

1298 **B.3.1.1 Intel TME and Intel Multi-Key TME (Intel MKTME)**

1299 Intel Total Memory Encryption (Intel TME) provides the capability to encrypt the entire physical
1300 memory of a system. This capability is typically enabled in the very early stages of the boot
1301 process with a small change to the BIOS. Once this change is configured and locked, all data on
1302 the external memory buses of a CPU and any additional DIMMs will be encrypted using 128-bit
1303 keys utilizing the NIST standard AES-XTS algorithm. The encryption key used for Intel TME
1304 uses a hardware RNG implemented in the Intel CPU, and the keys are not accessible by software
1305 or by using external interfaces to the CPU. The architecture is flexible and will support
1306 additional memory protection schemes in the future. Intel TME is intended to support
1307 unmodified existing system and application software. The overall performance impact of TME is
1308 likely to be relatively small and highly dependent on workload.

1309 Intel Multi-Key Total Memory Encryption (Intel MKTME) builds on Intel TME and adds
1310 support for multiple encryption keys. The CPU implementation supports a fixed number of
1311 encryption keys, and software can configure a CPU to use a subset of available keys. Software
1312 manages the use of keys and can use each of the available keys for encrypting any page of the
1313 memory. Thus, Intel MKTME allows page granular encryption of memory. By default, Intel
1314 MKTME uses the Intel TME encryption key unless explicitly specified by software.

1315 In addition to supporting a CPU-generated ephemeral key (not accessible by software or by using
1316 external interfaces to a CPU), Intel MKTME also supports software-provided keys. Software-
1317 provided keys are particularly useful when used with nonvolatile memory, when combined with

1318 attestation mechanisms or used with key provisioning services. An OS may be enabled to take
1319 additional advantage of the Intel MKTME capability, both in native and virtualized
1320 environments. When properly enabled, Intel MKTME is available to each guest OS in a
1321 virtualized environment, and the guest OS can take advantage of Intel MKTME in the same ways
1322 as a native OS.

1323 **B.3.2 Application Isolation**

1324 **B.3.2.1 Intel Software Guard Extensions (SGX)**

1325 Intel Software Guard Extensions (SGX) is a set of instructions that increases the security of
1326 application code and data. Developers can partition security-sensitive code and data into an *SGX*
1327 *enclave*, which is executed in a CPU protected region. The developer creates and runs SGX
1328 enclaves on server platforms where only the CPU is trusted to provide attestations and protected
1329 execution environments for enclave code and data. SGX also provides an enclave remote
1330 attestation mechanism. This mechanism allows a remote provider to verify the following [37]:

- 1331 1. The enclave is running on a real Intel processor inside an SGX enclave.
- 1332 2. The platform is running at the latest security level (also referred to as the *TCB version*).
- 1333 3. The enclave's identity is as claimed.
- 1334 4. The enclave has not been tampered with.

1335 Once all of this is verified, the remote attester can then provision secrets into the enclave. SGX
1336 enclave usage is reserved for Ring-3 applications and cannot be used by an OS or BIOS
1337 driver/module.

1338 SGX removes the privileged software (e.g., OS, VMM, System Management Mode devices) and
1339 unprivileged software (e.g., Ring-3 applications, VMs, containers) from the trust boundary of the
1340 code running inside the enclave, enhancing security of sensitive application code and data. An
1341 SGX enclave trusts the CPU for execution and memory protections. SGX encrypts memory to
1342 protect against memory bus snooping and cold boot attacks for enclave code and data in host
1343 Dynamic Random Access Memory (DRAM). SGX includes ISA instructions that can be used to
1344 handle Enclave Page Cache page management for creating and initializing enclaves.

1345 SGX relies on the system UEFI BIOS and OS for initial provisioning, resource allocation, and
1346 management. However, once an SGX enclave starts execution, it is running in a
1347 cryptographically isolated environment separate from the OS and BIOS.

1348 SGX can allow any application (whole or part of) to run inside an enclave and puts application
1349 developers in control of their own application security. However, it is recommended that
1350 developers keep the SGX code base small, validate the entire system (including software side
1351 channel resistance), and follow other secure software development guidelines.

1352 SGX enclaves can be used for applications ranging from protecting private keys and managing
1353 security credentials to providing security services. In addition, industry security standards, like
1354 European Telecommunications Standards Institute (ETSI) Network Functions Virtualization
1355 (NFV) Security (ETSI NFV SEC) [38], have defined and published requirements for Hardware

1356 Mediated Execution Enclaves (HMEEs) for the purposes of NFV, 5G, and edge security. SGX is
1357 an HMEE.

1358 **B.3.3 VM Isolation**

1359 **B.3.3.1 Intel Trust Domain Extensions (Intel TDX)**

1360 Intel Trust Domain Extensions (Intel TDX) introduces new architectural elements to deploy
1361 hardware-isolated VMs called trust domains (TDs). Intel TDX is designed to isolate VMs from
1362 the VMM/hypervisor and any non-TD software on the platform to protect TDs from a broad
1363 range of software. TDX is built using a combination of Virtual Machine Extensions, (VMX) ISA
1364 extensions, MKTME technology, and a CPU-attested software module called the TDX-SEAM
1365 module. TDX isolates VMs from many hardware threats and most software-based threats,
1366 including from the VMM and other CSP software. TDX helps give the cloud tenant control of
1367 their own data security and intellectual property protection. TDX does this while maintaining the
1368 CSP role of managing resources and cloud platform integrity.

1369 The TDX solution provides the following capabilities to TDs to address the security challenges:

- 1370 • Memory and CPU state confidentiality and integrity to help keep the sensitive IP and
1371 workload data secure from most software-based attacks and many hardware-based
1372 attacks. The workload now has a tool that supports excluding the firmware, software,
1373 devices, and operators of the cloud platform from the TCB. The workloads can use this
1374 tool to foster more secure access to CPU instructions and other CPU features. The
1375 workload can have this ability irrespective of the cloud infrastructure used to deploy the
1376 workload.
- 1377 • Remote attestation enables a relying party (either the owner of the workload or a user of
1378 the services provided by the workload) to establish that the workload is running on a
1379 TDX-enabled platform located within a TD prior to providing that workload data.
1380 Remote attestation aims to allow the owners and consumers of the service to digitally
1381 determine the version of the TCB they are relying on to help secure their data. The VMM
1382 remains the platform resource manager, and TDs should not cause denial of service to the
1383 VMM. Defending TDs against denial of service by the VMM is not a goal.

1384 TDX also augments defense of the TD against limited forms of attacks that use physical access
1385 to the platform memory, such as offline, DRAM analysis (example: cold-boot attacks), and
1386 active attacks of DRAM interfaces, including capturing, modifying, relocating, splicing, and
1387 aliasing memory contents [39]. The VMM continues to be the resource manager, and TDs do not
1388 have privileges to deny service to the VMM.

1389 **B.3.4 Cryptographic Acceleration**

1390 **B.3.4.1 Intel Advanced Encryption Standard New Instructions (Intel AES-NI)**

1391 Intel AES New Instructions (Intel AES-NI) is an encryption instruction set that improves
1392 hardware performance of the Advanced Encryption Standard (AES) algorithm and accelerates
1393 data encryption. Intel AES-NI consists of seven new instructions that accelerate encryption and
1394 decryption and improve key generation and matrix manipulation, all while aiding in carry-less

1395 multiplication. This minimizes application performance concerns inherent in conventional
1396 cryptographic processing and helps provide enhanced security by addressing side channel attacks
1397 on AES associated with conventional software methods of table lookups [40].

1398 AES is the most widely used standard for protecting network traffic, personal data, and corporate
1399 IT infrastructures. By implementing certain intensive sub-steps of the AES algorithm into the
1400 hardware, Intel AES-NI strengthens and accelerates execution of the AES application [40].

1401 **B.3.4.2 Intel QuickAssist Technology (QAT) with Intel Key Protection Technology** 1402 **(KPT)**

1403 Intel QuickAssist Technology (QAT) is a high-performance hardware accelerator for performing
1404 cryptographic, security, and compression operations. Applications like VMs, containers, and
1405 Function as a Service call Intel QAT using industry-standard OpenSSL, TLS, and Internet
1406 Protocol Security (IPsec) interfaces to offload symmetric and asymmetric cryptographic
1407 operations. Cloud, multi-tenancy, NFV, edge, and 5G infrastructures and applications are best
1408 suited for QAT for all types of workloads, including software-defined networks, content delivery
1409 networks, media, and storage [41].

1410 Intel Key Protection Technology (KPT) helps enable customers to secure their keys to be used
1411 with QAT through a bring-your-own-key paradigm. KPT allows customers to deliver their own
1412 cryptographic keys to the QAT device in the target platform where their workload is running.
1413 KPT-protected keys are never in the clear in host DRAM or in transit. The customers encrypt
1414 their workload key (e.g., RSA private key for Nginx) using KPT inside their HSMs. This
1415 encrypted workload key is delivered to the target QAT platform, where it is decrypted
1416 immediately prior to use. KPT provides key protection at rest, in transit, and while in use [42].

1417 **B.3.5 Technology Example Summary**

1418 Cloud infrastructure creates improvements in the efficiency, agility, and scalability of data center
1419 workloads by abstracting hardware from the application layer. This introduces new security
1420 concerns as workloads become multi-tenant, attack surfaces become shared, and infrastructure
1421 administrators from the cloud operator gain access to underlying platforms. Isolation techniques
1422 provide answers to these concerns by adding protection to VMs, applications, and data during
1423 execution, and they represent a crucial layer of a layered security approach for data center
1424 security architecture.

1425 Various isolation techniques exist and can be leveraged for different security needs. Full memory
1426 isolation defends a platform against physical memory extraction techniques, while the same
1427 technology extended with multiple keys allows individual VMs or platform tenants to have
1428 uniquely encrypted memory. Future generations of these technologies will allow full memory
1429 isolation of VMs, protecting them against malicious infrastructure insiders, multi-tenant
1430 malware, and more. Application isolation techniques allow individual applications to create
1431 isolated enclaves that require implicit trust in the platform CPU and nothing else and that have
1432 the ability to provide proof of the enclave to other applications before data is sent.

1433 **B.4 Remote Attestation Services**

1434 **B.4.1 Intel Security Libraries for the Data Center (ISecL-DC)**

1435 ISecL-DC is an open-source remote attestation implementation of a set of building blocks that
1436 utilize Intel Security features to discover, attest, and enable critical foundation security and
1437 confidential computing use-cases. This middleware technology provides a consistent set of
1438 application programming interfaces (APIs) for easy integration with cloud management software
1439 and security monitoring and enforcement tools. ISecL-DC applies the remote attestation
1440 fundamentals described in this section and standard specifications to maintain a platform data
1441 collection service and an efficient verification engine to perform comprehensive trust
1442 evaluations. These trust evaluations can be used to govern different trust and security policies
1443 applied to any given workload, as referenced in the workload scheduling use case in Section 7.2.
1444 In future generations, the product will be extended to include TEE attestation to provide
1445 assurance and validity of the TEE to enable confidential computing [43].

1446 **B.4.2 Technology Summary**

1447 Platform attestation provides auditable foundational reports for server firmware and software
1448 integrity and can be extended to include the location of other asset tag information stored in a
1449 TPM, as well as integrity verification for applications installed on the server. These reports
1450 provide visibility into platform security configurations and can be used to control access to data
1451 and workloads. Platform attestation is performed on a per-server basis and typically consumed
1452 by cloud orchestration or a wide variety of infrastructure management platforms.

1453 TEE attestation provides a mechanism by which a user or application can validate that a genuine
1454 TEE enclave with an acceptable TCB is actually being used before releasing secrets or code to
1455 the TEE. Formation of a TEE enclave is performed at the application level, and TEE attestations
1456 are typically consumed by a user or application requiring evidence of enclave security before
1457 passing secrets.

1458 These different attestation techniques serve complementary purposes in a cloud deployment in
1459 the data center or at the edge computing facility.

1460 Appendix C—AMD Technology Examples

1461 This section describes a number of AMD technology examples that map back to the key
1462 concepts described in the various sections of the document.

1463 C.1 Platform Integrity Verification**1464 C.1.1 AMD Platform Secure Boot (AMD PSB)**

1465 AMD Platform Secure Boot (AMD PSB) provides a hardware RoT to authenticate the initial
1466 Platform BIOS code during the boot process of the server. Manufacturers of server systems, like
1467 OEMs or original device manufacturers (ODMs), enable the functionality of AMD PSB in their
1468 manufacturing flow by permanently fusing policy into the silicon.

1469 The OEM or ODM's final BIOS image contains the AMD public key and the OEM BIOS-
1470 signing public key (signed with the AMD private key). When a system powers on, the AMD
1471 Security Processor (ASP) starts executing the immutable on-chip Boot ROM. It authenticates
1472 and loads multi-stage ASP Boot Loaders from SPI/Low Pin Count Flash into its internal
1473 memory, which initializes the silicon and the system memory.

1474 Once the system memory is initialized, the ASP Boot Loaders load and authenticate the OEM
1475 BIOS-signing public key, followed by authenticating the initial BIOS code. Once the verification
1476 is successful, ASP releases the x86 core to execute authenticated initial BIOS code. The BIOS
1477 can continue CoT for other components by means of SB. If PSB authentication fails, the system
1478 is forced to shut down.

1479 AMD PSB supports revocation and rollback protection of BIOS images through the OEM BIOS-
1480 signing key revision ID and rollback protection.

1481 C.2 Data Protection and Confidential Computing**1482 C.2.1 Memory Isolation****1483 C.2.1.1 AMD Secure Memory Encryption (SME)/Transparent Secure Memory
1484 Encryption (TSME)**

1485 AMD Secure Memory Encryption (SME) is a memory encryption technology from AMD which
1486 helps protect data in DRAM by encrypting system memory content [44]. When enabled, memory
1487 content is encrypted via dedicated hardware in the on-die memory controllers. Each controller
1488 includes a high-performance AES engine that encrypts data when it is written to DRAM and
1489 decrypts it when read. The encryption of data is done with an encryption key in a mode that
1490 utilizes an additional physical address-based tweak to protect against ciphertext block move
1491 attacks.

1492 The encryption key used by the AES engine with SME is randomly generated on each system
1493 reset and is not visible to any software running on the CPU cores. This key is managed entirely
1494 by the AMD Secure Processor that functions as a dedicated security subsystem integrated within
1495 the AMD System-on-Chip (SOC). The key is generated using the onboard NIST SP 800-90

1496 compliant hardware RNG and is stored in dedicated hardware registers where it is never exposed
1497 outside the SOC in the clear.

1498 Two modes of memory encryption are supported for various use cases. The simplest mode is
1499 Transparent Secure Memory Encryption (TSME), which is a BIOS option and enables memory
1500 encryption automatically on all memory accesses. TSME works in the background and requires
1501 no software interaction. Another supported mode is the OS-managed Secure Memory Encryption
1502 (SME) mode in which individual pages of memory may be marked for encryption via CPU page
1503 tables. SME provides additional flexibility if only a subset of memory needs to be encrypted but
1504 does require appropriate software support.

1505 Encrypted memory provides strong protection against cold boot, DRAM interface snooping, and
1506 similar types of attacks.

1507 **C.2.2 VM Isolation**

1508 **C.2.2.1 AMD Secure Encrypted Virtualization (SEV)**

1509 The AMD Secure Encrypted Virtualization (SEV) feature is designed to isolate VMs from the
1510 hypervisor. When SEV is enabled, individual VMs are encrypted with an AES encryption key.
1511 When a component such as the hypervisor attempts to read memory inside a guest, it is only able
1512 to see the data in its encrypted form. This provides strong cryptographic isolation between the
1513 VMs, as well as between the VMs and the hypervisor.

1514 To protect SEV-enabled guests, the SEV firmware assists in the enforcement of three main
1515 security properties: authenticity of the platform, attestation of a launched guest, and
1516 confidentiality of the guest's data.

1517 Authenticating the platform prevents malicious software or a rogue device from masquerading as
1518 a legitimate platform. The authenticity of the platform is proven with its identity key. This key is
1519 signed by AMD to demonstrate that the platform is an authentic AMD platform with SEV
1520 capabilities.

1521 Attestation of the guest launch proves to guest owners that their guests securely launched with
1522 SEV enabled. A signature of various components of the SEV-related guest state, including initial
1523 contents of memory, is provided by the firmware to the guest owner to verify that the guest is in
1524 the expected state. With this attestation, a guest owner can ensure that the hypervisor did not
1525 interfere with the initialization of SEV before transmitting confidential information to the guest.

1526 Confidentiality of the guest is accomplished by encrypting memory with a memory encryption
1527 key that only the SEV firmware knows. The SEV management interface does not allow the
1528 memory encryption key or any other secret SEV state to be exported outside of the firmware
1529 without properly authenticating.

1530 AMD SEV has two additional modes:

- 1531 • SEV With Encrypted State (SEV-ES): This mode encrypts and protects VM registers
1532 from being read or modified by a malicious hypervisor or VM [45].

- 1533 • SEV with Secure Nested Paging (SEV-SNP): This mode adds strong memory integrity
1534 protection to help prevent malicious hypervisor-based attacks like data replay and
1535 memory remapping. [46]

1536 Appendix D—Arm Technology Examples

1537 This section describes a number of Arm technology examples that map back to the key concepts
1538 described in the various sections of the document.

1539 D.1 Platform Integrity Verification

1540 In order to understand the Secure Boot environment, the Security states and Exception Levels
1541 implemented by TrustZone must first be understood. The following sections describe the
1542 technologies available for A-profile Arm processors.² [47] [48]

1543 D.1.1 Arm TrustZone TEE for Armv8-A**1544 D.1.1.1 The Normal (Non-Secure) World and Secure World**

1545 TrustZone [49] provides two execution environments built into the processor with system-wide
1546 hardware enforced isolation between them. There are two Security states: secure and non-secure.
1547 They map to the Secure world (SW) and the Normal world (NW, also sometimes referred to as
1548 the Non-Secure world), respectively. Each processor implements both worlds but can execute in
1549 only one world at any given time, independently of which world each of the other processors in a
1550 multi-processor implementation is executing. For example, core0 might be executing in the NW,
1551 while core1 is executing in the SW, core2 is executing in the NW, etc., all concurrently. The SW
1552 and NW concept extends beyond the processor to include memory, software, bus transactions,
1553 interrupts, and peripherals within an SoC.

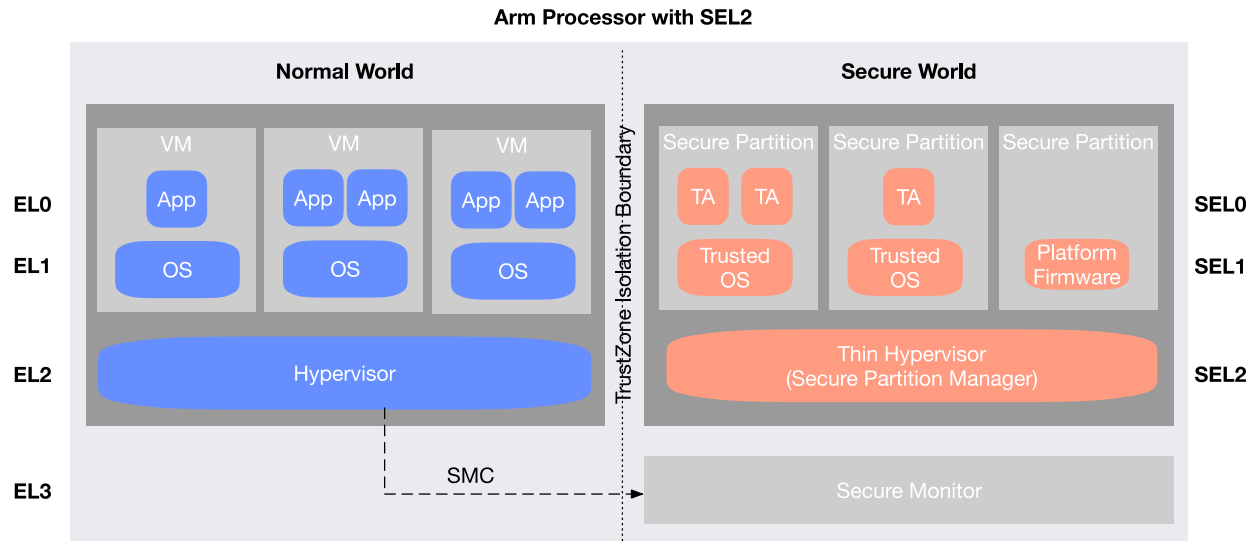
1554 The NW runs a Rich Execution Environment (REE), which typically includes a large number of
1555 applications, a complex OS (e.g., Linux), and often a hypervisor. The REE presents a broad
1556 attack surface. The SW provides a TEE, which runs a smaller and simpler software stack than the
1557 REE. The TEE may include several trusted services, a lightweight kernel, and, if the processor
1558 supports Secure Exception Level 2 (SEL2, explained in Appendix D.1.1.2), a simple hypervisor.
1559 The TEE has a much smaller attack surface and does not run arbitrary code, making it much less
1560 vulnerable to attack compared to the REE. Also, the SELs provide additional protection within
1561 the TEE.

1562 As shown in Figure 9, there is a hardware-enforced isolation boundary between the NW and the
1563 SW. For A-class processors, the NW requests a secure service from the SW by issuing a Secure
1564 Monitor Call (SMC) to effect the transition from the NW to the SW and back via the Secure
1565 Monitor, which runs in the SW at the highest exception level, EL3. The Secure Monitor hands
1566 the information to the Secure Partition Manager (SPM) executing at SEL2, which invokes the
1567 secure service in a Trusted Application (TA) running in a Secure Partition (SP).

1568 The SW and NW address spaces can be split into several regions. Each region is specified as
1569 secure or non-secure. The registers to control the address space partitioning are limited to SW
1570 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.

1571 memory partitioning is not expected to change dynamically at runtime because it is only
1572 configured once during system boot,³ which always takes place in the secure state.



1573

1574

Figure 9: Arm Processor with TrustZone

1575 The architecture provides two physical address spaces (PAS): secure (SW) and non-secure
1576 (NW). While in non-secure state, virtual addresses always translate to non-secure physical
1577 addresses. Software in non-secure state can only access non-secure resources. In secure state,
1578 firmware running at Exception Level 3 (EL3) and SEL2 can access both the secure and non-
1579 secure physical address spaces. SEL0 and SEL1 may access a non-secure physical address if it
1580 has been mapped to the corresponding page table entry by SEL2 for an SP. When executing in
1581 the SW, code is never fetched or executed from the NW address space.

1582 Input/output (I/O) devices can be assigned to the SW or the NW. Memory-mapped devices
1583 follow the same access rules as described for memory accesses.

1584 **D.1.1.2 Exception Levels**

1585 In the NW, there are three Exception Levels (ELs):

- 1586 • **EL0** is the EL where applications execute in the NW. They have visibility to their
1587 application address space created by the OS running at EL1. EL0 is the least privileged
1588 EL.
- 1589 • **EL1** is the EL at which the OS executes. The OS manages the application address spaces
1590 that it creates at EL0 and owns all of the memory assigned to it. The OS may be a bare-
1591 metal OS, which runs on the hardware directly, or it may be located in a VM created by a
1592 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).

1593 • **EL2** is the EL at which the hypervisor executes. It owns and manages the NW memory,
1594 and it manages the VMs. A VM is composed of an OS running at EL1 and the
1595 applications it creates that run at EL0. The VM may also have non-secure devices
1596 assigned to it.

1597 Higher ELs (i.e., with a larger EL number) have the privilege to access registers that control
1598 lower ELs. In the general operation of the system, the privileged ELs will usually control their
1599 own configuration. However, more privileged ELs will sometimes access registers associated
1600 with lower ELs to, for example, read and write the register set as part of a save-and-restore
1601 operation during a context switch or power management operation. EL1 and EL0 share the same
1602 MMU configuration, and control is restricted to privileged code running at EL1.

1603 The NW ELs, EL0 through EL2, are mirrored in the SW and serve similar purposes:

1604 • **SEL0** is the SEL where TAs execute in the SW. They provide secure services to the NW
1605 (e.g., Platform Security Architecture [PSA] crypto services, Digital Rights Management,
1606 a firmware TPM, a secure interface to a shared hardware device such as a discrete TPM
1607 or an HSM). TAs have visibility to their application address space created by the Trusted
1608 Operating System (TOS), e.g., OP-TEE, at their inception. One SEL0 TA cannot access
1609 any other SEL0 TA memory unless the memory is specifically mapped as shared by the
1610 SEL1 TOS. SEL0 is the least privileged SEL in the SW.

1611 • **SEL1** is the SEL at which the TOS executes, shown in an SP in Figure 9. The TOS
1612 manages the application address spaces and owns all of the memory assigned to the SP.
1613 Alternatively, vendor-provided platform firmware may execute in its own SP at SEL1;
1614 this is referred to as a bare-metal SP. Each SP represents a separate security domain
1615 within the SW and is essentially the SW equivalent of an NW VM.

1616 • **SEL2** is the EL at which the SPM executes. It is a simple hypervisor and manages the
1617 SPs and their address spaces. It routes messages to and from the SPs. The separation of
1618 address spaces also allows moving manufacturer-provided platform firmware that, in the
1619 previous implementation without SEL2 ran at EL3, to an isolated SP running at SEL1
1620 (shown in Figure 9).

1621 **EL3** is a special EL. [49] EL3 always executes in the SW. It manages the transition between
1622 worlds, and it runs the firmware that provides the Secure Monitor services, including:

- 1623 • Initial boot (BL2) execution
- 1624 • SMC intercept and dispatcher, which handles incoming SMCs and routes them
- 1625 • Maintains the non-secure to secure isolation and memory
- 1626 • Power State Coordination Interface – low-level power management
- 1627 • System Control and Management Interface – OS-independent software interfaces used in
1628 system management
- 1629 • Reliability, availability, and serviceability error delivery
- 1630 • Software Delegated Exception Interface – provides a high-priority event delivery
1631 mechanism, which has higher priority than interrupts that target OSs and hypervisors

1632 Trusted Firmware provides firmware support for dealing with Arm System Intellectual Property
1633 (IP), like interconnects. Silicon Providers (SiPs) provide firmware support to handle custom or
1634 third-party IP. This includes SoC-specific power management.

1635 See Appendix D.3.1 for the extensions added to this architecture by CCA.

1636 **D.1.1.3 Trusted Applications and Secure Services**

1637 TAs in SPs are intended to run only for short periods of time so as not to block the NW from
1638 executing for long on a given processor. The SW is not intended to run general-purpose
1639 applications, but rather secure services like those previously described for SEL1. For the most
1640 part, these secure services are expected to be global services offered to the NW apps and to live
1641 for the life of the boot. However, their use may be restricted to specific NW applications by
1642 implementing an authentication mechanism in the SW to ensure that the NW requester is
1643 authorized to use the service.

1644 Scheduling of the SW is on a per-processor basis and is implemented via a secure interrupt
1645 handled by the Secure Monitor at EL3. Explicit calls for services such as Platform Management
1646 Communications Infrastructure, for example, are typically blocking on that processor; control
1647 will only be returned to the Non-Secure state when the requested operation is complete.
1648 However, these calls tend to be short and infrequent. SMC calls for secure services from a TA
1649 are scheduled by the host OS/hypervisor. Most secure service requests will be short. However, if
1650 the TA needs to run for an extended period, the SMC Calling Convention allows the TA to return
1651 control to the NW, where the host OS (e.g., Linux) can re-schedule the SW by reissuing the
1652 SMC at a convenient time. The SW will pick up where it was. This cooperative scheduling
1653 approach allows the NW to control scheduling as needed.

1654 **D.1.1.4 Debugging, Tracing, and Profiling**

1655 Arm systems include extensive features to support debugging, tracing, and profiling. The SoC
1656 needs to be configured properly to ensure that these features cannot be used to compromise the
1657 security of the system. Different developers are trusted to debug different parts of the system
1658 during different stages of its security lifecycle. Rate signals to control use of the features in
1659 Secure state and Non-Secure state are available to address these requirements.

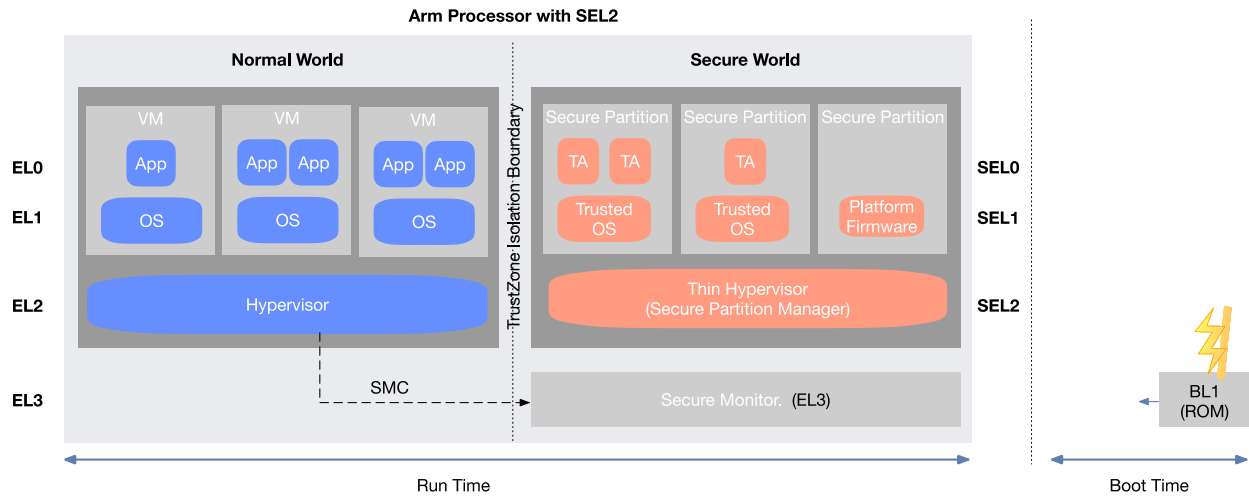
1660 **D.1.2 The Chain of Trust (CoT)**

1661 The boot firmware can only be trusted if all the software components that run in the boot flow
1662 are trusted. This is referred to as the Chain of Trust. Trusted Firmware-A⁴ (TF-A) [12]
1663 implements boot firmware that loads, authenticates, and verifies each load of boot code before
1664 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).

1665 by detecting that it has not been corrupted or modified in any way and is authentic. In this way,
1666 the boot firmware establishes a complete cascading CoT.

1667 On Arm, Secure Boot⁵ takes place within the SW provided by the Arm Processor Element (PE)
1668 TrustZone implementation (see Appendix D.3.1.3). When a power-on/reset processor event
1669 occurs, the system begins to execute the boot ROM⁶ code at EL3 on the boot core.⁷ The
1670 immutable ROM provides the hardware RoT for the processor complex and is implicitly trusted.



1671

1672

Figure 10: Boot-Time and Run-Time Firmware

1673 During manufacture, the SiP and the OEM or ODM provide the ROM code (BL1 in Figure 10)
1674 and the first mutable load of boot code (BL2). The boot ROM code is typically small and simple.
1675 Its main function is to load the second-stage boot code (BL2), the first mutable boot code, from
1676 the non-volatile firmware storage device, typically flash, and verify it using the immutable root
1677 of trust public key provisioned at manufacture. The specific public key algorithm used for
1678 authentication is defined by the implementation. This process ensures first instruction integrity.
1679 If BL2 passes verification, BL1 transfers control to it. BL2 then performs some system
1680 initialization of the platform, like setting up the memory controller for off-chip DRAM, for
1681 example. BL2 then loads and cryptographically verifies all subsequent loads of boot, including
1682 the EL3 Boot and Runtime firmware (BL3-1); the SEL2, SEL1, and SEL0 firmware (BL3-2);
1683 and the first non-secure load of boot (BL3-3 – e.g., UEFI, U-Boot), which executes at EL2 when
1684 it receives control. BL2 uses Key and Content self-signed X.509 certificates to verify the BL3-x
1685 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.

1686 The boot process can optionally measure all the boot firmware up to and including BL3-3. BL3-
1687 3, when it executes, can also provide measurements for firmware it loads (e.g., runtime drivers).
1688 This is an optional feature. The process described above is changed so that the boot firmware is
1689 loaded, verified, then measured before transferring control. For Arm, the measurements are
1690 extended into PCRs: PCR0 (all signed firmware and data) and PCR1 (all critical configuration
1691 data – e.g., debug port settings). The PCR implementation is platform-defined and is hidden by a
1692 firmware API that provides only read and extend operations. The PCR implementation need only
1693 guarantee that the architectural behavior and security of measurement PCRs are adhered to. For
1694 example, the PCRs may be implemented on-chip in a protected memory location or in a
1695 firmware TPM implementation, or in an external device like a discrete TPM or a Secure
1696 Element. PCR0 and PCR1 are cleared to zero at reset.

1697 The boot measurements can be used to attest to the firmware that was booted on the system. An
1698 attestation key, also known as an endorsement key (EK), cryptographically proves the device
1699 identity, and therefore trustworthiness, to external entities. A device attestation key can be used
1700 by many different attestation schemes (e.g., the Fast Identity Online [FIDO] Alliance, the
1701 Trusted Computing Group [TCG] TPM, Platform Security Architecture [PSA], and CCA). A
1702 remote verifier can be used to compare the attested measurement against a list of known “good”
1703 measurements in order to decide whether the system is in a valid secure state. Policy can then be
1704 used to decide what action to take if the comparison fails.

1705 A monotonically incremented counter is supported to prevent rollback of firmware, including
1706 configuration data, to a previous version since that previous version of firmware may have
1707 exploitable vulnerabilities. However, rollback for recovery purposes can be permitted if
1708 authorized.

1709 **D.1.3 Platform Security Architecture (PSA) Functional APIs**

1710 The PSA Functional APIs define the foundations from which security services are built, allowing
1711 devices to be secure-by-design. The three APIs (cryptography, storage, and attestation) provide a
1712 consistent developer experience for system software and application developers, enabling
1713 interoperability across different hardware implementations of the RoT.

1714 **D.1.3.1 The PSA Cryptographic API (Crypto API)**

1715 The PSA Crypto API provides a portable interface to cryptographic operations on a wide range
1716 of hardware. The interface provides access to the low-level primitives used in modern
1717 cryptography. It does not require that the user has access to the key material; instead, it uses
1718 opaque key identifiers. It defines an interface for cryptographic services, including cryptography
1719 primitives and a key storage functionality. The interface is designed to be both scalable and
1720 modular, allowing devices to only implement what they need.

1721 Implementations can isolate the crypto processor from the calling application, and can isolate
1722 multiple calling applications, one from another. The implementation can be separated into a front
1723 end and a back end. In an isolated implementation, the back end is located in an isolated
1724 environment, which is protected from the front end. Various technologies can provide protection,
1725 for example:

- 1726 • OS process isolation
 - 1727 • Partition isolation, either with a VM or TEE environment like TrustZone
 - 1728 • Physically separate hardware devices
- 1729 A low-level cryptographic interface is defined, where the caller explicitly chooses which
1730 algorithm and security parameters they want to use. All cryptographic functionality operates
1731 according to the algorithm specified by the caller. Generic higher-level interfaces, where the
1732 implementation chooses the best algorithm for a purpose, are not specified. However, higher-
1733 level libraries can be built on top of the PSA Crypto API.
- 1734 Some intended use cases for the PSA Crypto API are:
- 1735 • TLS
 - 1736 • Secure storage encryption
 - 1737 • Network credentials (e.g., X.509)
 - 1738 • Device pairing (e.g., Near Field Communication [NFC] token pairing or Bluetooth key
1739 agreement protocols)
 - 1740 • Verified boot (firmware integrity and authentication)
 - 1741 • Attestation (the primitives provided are suitable for attestation use cases)
 - 1742 • Factory provisioning (these APIs can be used to generate device unique identity keys for
1743 population at the factory)
- 1744 Interfaces for the following types of symmetric cryptographic operation are provided:
- 1745 • Message digests, commonly known as hash functions
 - 1746 • Message authentication codes (MACs)
 - 1747 • Symmetric ciphers
 - 1748 • Authenticated encryption with associated data (AEAD)
- 1749 Both a pair of single-part functions (e.g., encrypt, decrypt) and a series of functions that permit
1750 multi-part operations are defined for each type of symmetric cryptographic operation (e.g.,
1751 allocate, initialize, setup, update, and finish).

1752 **D.1.3.2 The PSA Storage API**

1753 The PSA Storage APIs provide key/value storage interfaces for use with device-protected
1754 storage. They describe the interface for the storage provided by the PSA RoT (the PSA Internal
1755 Trusted Storage [ITS] API), as well as an interface for external protected storage (the PSA
1756 Protected Storage [PS] API). Two use cases are covered: secure storage for device secret data
1757 (ITS), and protection for data-at-rest (PS). ITS is a more specialized API and is intended to
1758 provide data integrity and/or privacy. For example, a device identity key requires both
1759 confidentiality and integrity, whereas a public key is public data but requires integrity but not
1760 privacy. PS is the general-purpose API that will be used most often and is intended to protect

1761 larger data sets against physical attacks. It provides the ability to store data on external flash,
1762 with a promise of data-at-rest protection, including device-bound encryption, integrity, and
1763 replay protection. It is possible to select the appropriate protection level, e.g., encryption only, or
1764 integrity only, or all three, depending on the threat model of the device and the nature of its
1765 deployment.

1766 Consistent APIs for accessing storage allow software to be written in a platform-independent
1767 manner, improving portability across PSA-supported platforms.

1768 **D.1.3.3 The PSA Attestation API**

1769 The PSA Attestation API is a standard interface provided by the PSA RoT, which is defined in
1770 the PSA Security Model. The API can be used either to directly sign data or to bootstrap trust in
1771 other attestation schemes. PSA provides a framework and the minimal generic security features
1772 allowing OEM and service providers to integrate various attestation schemes on top of the PSA
1773 RoT. The PSA RoT reports information (claims) that can be used to determine the exact
1774 implementation of the PSA RoT and its security state. If the PSA RoT loads other components, it
1775 also includes information about them. Other components outside of the PSA RoT can add
1776 information to the report by calling the provided API, which will include and sign the additional
1777 information. The PSA RoT signs attestation reports using the initial attestation key.

1778 Each device instance contains a protected attestation key that can be used to prove that it is a
1779 particular certified implementation. The attestation identity can be verified in an attestation
1780 process and checked against certification information. At the end of the process the verifier can
1781 establish a secure connection to the attested endpoint and deliver credentials. The combination of
1782 a hardware entity and the software installed on that entity can be certified to conform to some
1783 published security level. A party may want to check the received list of claims against a database
1784 of known measurements for each component in order to decide which level of trust should be
1785 applied.

1786 Initial attestation requires three services:

- 1787 • Enrollment verification service, enforcing policy as part of service enrollment of the
1788 device
- 1789 • Production verification service (OEM), providing the production state of an attestation
1790 identity
- 1791 • Certification verification service (third party), verifying that all attested components are
1792 up to date, signed correctly, and certified to work together

1793 The API must be provided by the implementation.

1794 **D.1.4 Platform AbstRaction for SECurity (Parsec)**

1795 Parsec [50] is the Platform AbstRaction for SECurity, an open-source initiative to provide a
1796 common API to secure services in a platform-agnostic way. It provides a micro-service that
1797 maps easy-to-consume security APIs, in the language of choice, to security primitives found in
1798 various different hardware implementations. It is part of the CNCF sandbox.

1799 Parsec aims to define a software standard for interacting with secure object storage and
1800 cryptography services, creating a common way to interface with functions that have traditionally
1801 been accessed by more specialized APIs. Parsec provides an ecosystem of libraries in a variety of
1802 programming languages. Each library is designed to be simple to consume. This ecosystem
1803 makes available secure facilities to developers across a broad range of use cases in infrastructure
1804 computing, edge computing, and IoT.

1805 Computing platforms have evolved to offer a range of facilities for secure storage and secure
1806 operations. There are hardware-backed facilities such as the HSM and TPM, there are firmware
1807 services running in TEEs, and there are also cloud-based Parsec services. Security facilities may
1808 be provided purely in software, where they are protected by mechanisms provided in the OS.
1809 Parsec is built on PSA. The core component of Parsec is the Parsec security service. It is a
1810 background process that runs on the host platform and provides connectivity with the secure
1811 facilities of that host and exposes the wire protocol based on PSA Functional APIs.

1812 The Parsec service listens on a suitable transport medium. The transport technology is one of
1813 Parsec's many pluggable components, and no single transport is mandated. Choice of transport is
1814 dependent on the OS and the deployment. On Linux-based systems where the client applications
1815 are running in containers (isolation with a shared kernel), the transport can be based on Unix
1816 sockets. Client applications make connections with the service by posting API requests to the
1817 transport endpoint. This is usually done via a client library that hides the details of both the wire
1818 protocol and the transport.

1819 A single instance of the Parsec service executes on each physical host. In virtualized
1820 environments, the Parsec service may reside on a specially assigned guest, or potentially within
1821 the hypervisor. Another option is running Parsec on each individual guest, with back-end support
1822 running in a TEE/Secure Enclave. The Parsec service does not support remote client
1823 applications; each physical host or node must have its own instance of the service. However, it is
1824 possible for the service to initiate outbound remote calls of other services, such as cloud-hosted
1825 HSM services.

1826 The Parsec service is also responsible for brokering access to the underlying security facilities
1827 amongst the multiple client applications in a multi-tenant environment. Parsec is able to support
1828 multi-tenant scenarios by making use of a client identity, which is a string or token that each
1829 client application passes to the Parsec service with every API call. Parsec does not mandate any
1830 particular format or semantics for these client identity strings. They can be passed opaquely to
1831 the service as octet sequences using the wire protocol. The only requirement on a client identity
1832 string is that it must be verifiable by the Parsec service in some way. The component that offers
1833 this verification is known as an identity provider.

1834 The identity provider is not part of Parsec. It can be another component or service, residing either
1835 locally or remote. It could be a container orchestrator or other runtime management service, or
1836 even a function of the OS. It is expected that the Parsec service can establish trust with the
1837 identity provider via some suitable means. Parsec offers a pluggable mechanism for
1838 communicating with different identity providers.

1839 Examples of client identity management include the following:

- 1840 • The client identity can simply be an OS entity such as the process identifier, user
1841 identifier, or group identifier of the client application's process. When Unix sockets are
1842 used for the wire protocol transport, Parsec can verify the identity by making system calls
1843 to perform a peer credential check [51]. In this case, the identity provider is effectively
1844 the OS kernel.
- 1845 • The client identity can be a Secure Production Identity Framework for Everyone
1846 (SPIFFE) Verifiable Identity Document (SVID) [52]. In this case, the identity provider
1847 would be a SPIFFE runtime component such as a local SPIFFE Runtime Environment
1848 (SPIRE) service [53], with which the Parsec service communicates using the SPIFFE
1849 Workload API [54].

1850 Other methods of verification are possible. However the client identity is verified, Parsec will
1851 use the resulting identity string as a namespace for all keys and other stored assets. The Parsec
1852 service will ensure that each client application is only given visibility and access over its own
1853 namespace.

1854 Parsec meets the need for a new platform abstraction that offers a common palette of security
1855 primitives via a software interface that is both agnostic with respect to the underlying hardware
1856 capabilities, and also capable of supporting multiple client applications on the same host,
1857 whether those be within containers or VMs.

1858 For more information, see also [55] [56] [57].

1859 **D.2 Software Runtime Protection Mechanisms**

1860 **D.2.1 Return Oriented Programming (ROP) and Jump Oriented Programming (JOP)** 1861 **Attacks**

1862 **D.2.1.1 Pointer Authentication Code (PAC)**

1863 Attacks frequently attempt to subvert software control flow. PAC [48] is a feature introduced to
1864 block these types of attacks. It adds functionality that supports address authentication of the
1865 contents of a register before that register is used as the target of an indirect branch, or as a load.
1866 PAC is a strong defense against ROP attacks, which attempt to make a function return to the
1867 wrong location.

1868 With PAC, hardware ensures that the return is made to the correct location by preserving the
1869 original pointer value. The upper bits of a 64-bit pointer are used to store the PAC, which is a
1870 cryptographic signature on the pointer value and some additional specified context. Special
1871 instructions have been introduced to add a PAC to a pointer, verify an authenticated pointer's
1872 PAC, and restore the original pointer value. The authentication operation regenerates the PAC
1873 and compares it with the value that is stored in the pointer. If authentication succeeds, a pointer
1874 without the PAC is returned. If authentication fails, an invalid pointer is returned. An exception
1875 is raised if the pointer is subsequently used. This gives the system a way to make
1876 cryptographically strong guarantees about the likelihood that certain pointers have been
1877 tampered with by attackers.

1878 D.2.1.2 Branch Target Identification (BTI)

1879 Once an attacker has found a vulnerability to exploit, their next aim is to execute code to gain
1880 control of the machine they have accessed. Techniques used to modify the control flow include
1881 ROP and JOP attacks. These techniques find small sections (called gadgets) of vulnerable
1882 programs that can be chained together to achieve the attacker's goal.

1883 Systems supporting Branch Target Identification (BTI) [48] can enforce that indirect branches
1884 only target code locations that start with one of the accepted BTI instructions. Pages can be
1885 marked as containing BTI instructions. Indirect branches can only branch to locations identified
1886 as having BTI instructions, which reduces the ability of an attacker to execute arbitrary code.
1887 The BTI feature works together with PAC to significantly reduce the number of gadgets
1888 available to an attacker.

1889 D.2.2 Memory Safety Violations**1890 D.2.2.1 Privileged Access Never (PAN)**

1891 PAN helps to prevent an OS kernel or a hypervisor from being exploited to erroneously access
1892 memory allocated to a user-mode application (EL0). If PAN is enabled, any attempt by the
1893 kernel or hypervisor to access a page controlled by a user-mode attacker, will be prevented. The
1894 access will result in a permission fault and will not result in the data or instruction being cached.

1895 Sometimes the OS or hypervisor does need to access unprivileged regions, for example, to write
1896 to a buffer owned by an application. To support this, the instruction set provides unprivileged
1897 loads and stores that are not blocked by PAN (LDTR and STTR). They are checked against EL0
1898 permission checking even when executed by the OS at EL1 or EL2. While application code
1899 needs to be executable in user space (EL0), it should never be executed with kernel permissions
1900 (EL1/EL2), so PAN controls these accesses.

**1901 D.2.2.2 User (EL0) Execute Never (UXN) and Privileged (EL1/EL2) Execute Never
1902 (PXN)**

1903 User (EL0) Execute Never (UXN) and Privileged (EL1/EL2) Execute Never (PXN) provide
1904 protection against "stack smashing" attacks where malicious software attempts to write new
1905 opcodes into memory and then attempts to execute the newly modified memory. Typically this
1906 attack is targeted at stack memory. These execute permissions are stored in the page table
1907 entries. Any attempt to branch to an address within a marked page triggers a permission fault.
1908 The translation table attributes and write controls can block execution from any location that the
1909 malicious code could write to.

1910 D.2.2.3 Memory Tagging Extension (MTE)

1911 Memory tagging enables developers to identify spatial and temporal memory safety violations in
1912 their programs (e.g., use-after-free, use-out-of-scope, use-before-initialization, bounds
1913 violations). MTE [48] [58] [59] is designed to quickly detect memory safety violations and
1914 provide robustness against attacks that are attempting to subvert code. MTE is a lightweight,
1915 probabilistic version of a lock and key system where one of a limited set of lock values can be

1916 associated with the memory locations forming part of an allocation, and the equivalent key is
1917 stored in unused high bits of addresses used as references to that allocation. On each use of a
1918 reference, the key is checked to make sure that it matches the lock before an access is made. If
1919 the key matches the lock, the memory access is permitted; otherwise, the invalid access can
1920 either be recorded for later reference (and execution is allowed to continue) or be faulted (and
1921 execution is halted). On freeing an allocation, the lock value associated with each location is
1922 changed to one of the other lock values so further uses of the reference have a reasonable
1923 probability of failure. Hard-to-catch memory safety errors can be detected and eliminated more
1924 easily, which aids reliability and improves product security.

1925 MTE provides architectural support for run-time, always-on detection of various classes of
1926 memory errors and can aid with debugging to eliminate vulnerabilities before they can be
1927 exploited. MTE implements support for storage access and checking of the lock values in
1928 hardware. Software selects and sets the values on allocation and deallocation.

1929 Instructions are added for use by general-purpose software to set Logical Address Tags,
1930 manipulate Logical Address Tags, store Allocation Tags to memory, and load Allocation Tags
1931 from memory. Additional instructions are added for use by system software and external debug
1932 agents to efficiently transfer Allocation Tags to and from memory. The extension is expected to
1933 be generally applicable to 64-bit software written in C and C++ that does not use the Logical
1934 Address Tag bits for other purposes. Use in mixed language environments, e.g., C/C++ code
1935 interacting with JIT) compiled or interpreted languages is also expected to benefit. Applicability
1936 to software in other languages will vary. Since MTE imposes no changes to standard C/C++
1937 application binary interfaces (ABIs), incremental deployment across and within ELs is possible.

1938 Both MTE and PAC can be enabled at the same time.

1939 **D.2.2.4 Hardware Enforced Capability-Based Architecture (Morello and CHERI)**

1940 Arm and the University of Cambridge are collaborating in the development of the Capability
1941 Hardware Enhanced RISC Instructions (CHERI) architecture that provides a capability approach
1942 to memory safety [60]. Arm has developed a prototype architecture, Morello [13] [14] [61], that
1943 adapts the hardware concepts of CHERI. Morello is a research program led by Arm in
1944 association with partners and funded by UK Research and Innovation (UKRI) as part of the UK
1945 government Digital Security by Design (DSbD) program. This new approach to cybersecurity
1946 requires a significant change in how the hardware architecture is designed, as well as how the
1947 software running on devices that support the CHERI capability architecture is programmed to
1948 take advantage of the new features.

1949 The Morello architecture aims to improve the robustness and security of systems through two
1950 design goals: fine-grained memory protection leading to increased memory safety, and scalable
1951 compartmentalization. To achieve these goals, the Morello architecture introduces the principles
1952 defined by CHERI, including the principles of least privilege and intentional use. The Morello
1953 architecture is backwards compatible with, and complementary to, the existing Armv8-A
1954 architecture.

1955 The CHERI CPU architecture adds 128-bit “capabilities” plus a memory tag bit. The capability
1956 contains the address, bounds information, permission information, and an object type. The

1957 memory tag bit is metadata that distinguishes a capability from normal data and prevents
 1958 “forging” of a capability. A capability can be used in place of a normal pointer in some or all
 1959 situations. Simply replacing all pointers with capabilities gives scope for strong spatial memory
 1960 protection. Loads and stores using capabilities as addresses are checked to be within the
 1961 capability address range and matching the supplied permissions. Bounds cannot be arbitrarily
 1962 increased, permissions cannot be relaxed, and the tag cannot be changed.

1963 Three example use cases are:

- 1964 • **Kernel access control:** Due to tagged memory and constrained manipulation of
 1965 capability segment descriptors, a process cannot create a capability segment descriptor
 1966 that describes memory for which it does not already possess a descriptor.
- 1967 • **Sandboxing:** Capabilities can also be used for sandboxing a process into sub-address
 1968 spaces, essentially allowing a process to have its own memory protection policy for
 1969 program portions.
- 1970 • **Pointer safety:** Capabilities can also be used for pointer safety using automated bounds
 1971 checking.

1972 The Morello System Development Platform (SDP) is a prototype demonstrator board that
 1973 contains a Morello SoC. The SDP serves as the DSbD technology platform prototype for the
 1974 Morello architecture. Capabilities are introduced to the Arm v8-A architecture profile as an
 1975 extension of the Arm v8 AArch64 state, with the principles proposed in version 8 of the CHERI
 1976 ISA [60], to provide hardware support for fine-grained protection and building blocks for secure,
 1977 scalable compartmentalization.

1978 A pre-silicon Fixed Virtual Platform (FVP) System Model of Morello is available now. It
 1979 enables the development of software without the requirement for the prototype hardware. Arm
 1980 FVP models use binary translation technology to deliver fast simulations of the Arm-based
 1981 system. The Morello FVP provides a functionally accurate model of the Morello SoC IP
 1982 implementation. The FVP enables industry and academic partners to test the new capability-
 1983 based prototype architecture in real-world use cases. The Arm Development Studio: Morello
 1984 Edition supports the Morello FVP, which includes software models of the Rainier cores. The
 1985 Morello SDP board is closely based on the Arm® Neoverse™ N1 System Development Platform
 1986 board. Specifications and models are available today; Morello demonstrator boards are targeted
 1987 for release in late 2021.

1988 **D.2.3 Side-Channel Attacks**

1989 It is possible for attackers to exploit undesirable side-effects of out-of-order execution and
 1990 speculative execution in modern processors to breach the separation between OS and processes
 1991 and between processes in order to steal data. Physical access to the system is often not needed in
 1992 order to mount an attack. An attacker can potentially breach typical process and privilege
 1993 separation by using specially crafted software to gather sensitive information from other software
 1994 that is running on the same system.

1995 Arm has implemented a number of mitigations against side-channel attacks. At this time, four
 1996 variant mechanisms have been identified, each potentially using the speculation of a processor to

1997 influence which cache entries have been allocated in a way to extract some information which
 1998 would not otherwise be accessible to software. There are barrier instructions added that allow
 1999 mitigation of Spectre and Meltdown variants 1 (CVE-2017-5753), 2 (CVE-2017-5715), 3 (CVE-
 2000 2017-5754), and 3a (CVE-2018-3640), as well as memory disambiguation control to mitigate
 2001 variant 4 (CVE-2018-3639). See [48] [62] [63] [64] for a more complete description.

2002 **D.2.3.1 Speculation Barriers**

2003 A new barrier, a Speculation Barrier, is added to the architecture to control memory speculation.
 2004 The semantics are that until the barrier completes, the execution of any instruction appearing
 2005 later in the program order than the barrier cannot be performed speculatively, to the extent that
 2006 such speculation can be observed through side-channels as a result of control flow speculation or
 2007 data value speculation, but can be speculatively executed as a result of predicting that a
 2008 potentially exception-generating instruction has not generated an exception. In particular, it
 2009 cannot cause a speculative allocation into any caching structure where the allocation of that entry
 2010 could be indicative of any data value present in memory or in the registers. The instruction can
 2011 complete once it is known not to be speculative, and all data values generated by instructions
 2012 appearing in program order before the Speculation Barrier have their predicted values confirmed.

2013 **D.2.3.2 Predictor Invalidates**

2014 For all execution prediction resources that predict addresses or register values, the architecture
 2015 requires that the speculative execution at one hardware-defined context is separated in a hard-to-
 2016 determine manner from the predictions trained in a different hardware-defined context. For the
 2017 purpose of this definition, the hardware-defined context is determined by the following:

- 2018 • The EL
- 2019 • The Security state
- 2020 • When executing at EL1, the Virtual Machine Identifier (VMID)
- 2021 • When executing at EL0, the Address Space Identifier (ASID) and the VMID

2022 **D.2.3.3 Synchronization Barriers**

2023 The Arm architecture is a weakly-ordered memory architecture that supports out-of-order
 2024 completion. *Memory barrier* is the general term applied to an instruction, or sequence of
 2025 instructions, that forces synchronization events by a PE with respect to retiring Load/Store
 2026 instructions. The memory barriers defined by the Armv8 architecture provide a range of
 2027 functionality, including ordering and completion of load/store instructions, and context
 2028 synchronization. The new instructions provide a synchronization barrier for instructions, data,
 2029 trace, error, and profiling.

2030 **D.2.3.4 Arm Trusted Firmware (TF-A) and Linux**

2031 Arm has contributed updates to the open-source Trusted Firmware (TF-A) project [12] and has
 2032 developed Linux kernel and Android patches that take advantage of those implemented updates.
 2033 Patches have been upstreamed for both AArch64 and AArch32. Mitigations for Variant 1, 2, 3,
 2034 and 4 are available for both AArch64 and AArch32. Trusted Firmware has implemented a new

2035 SMC call to support some of these mitigations. The software mitigations described in the Cache
2036 Speculation Side-channels paper [62] should be deployed where protection against malicious
2037 applications is required by the threat model. Arm introduced processor features in Armv8.5-A
2038 [47], which can be implemented from Armv8.0, that provide resilience to this type of attack.

2039 **D.2.3.5 Timing Insensitive DP Instructions**

2040 A new option can be set to ensure Data Independent Timing for most classes of the data
2041 processing instructions – i.e., the time taken for an instruction is independent of the values of the
2042 data supplied in any of their registers. In addition, the response of these instructions to
2043 asynchronous exceptions does not vary based on the values supplied in any of their registers.
2044 This includes the following:

- 2045 • All cryptographic instructions
- 2046 • The set of instructions that use the General Purpose Register File
- 2047 • The set of instructions that use the Single Instruction, Multiple Data (SIMD) & Floating
2048 Point Register File
- 2049 • All loads and stores are timing-insensitive to the value being loaded or stored

2050 This prevents an attacker from inferring anything about the data being processed by an
2051 instruction or a set of instructions.

2052 **D.3 Data Protection and Confidential Computing**

2053 **D.3.1 Confidential Compute Architecture (CCA)**

2054 Arm's Confidential Compute Architecture (CCA) [65] establishes a Protected Execution
2055 Environment architecture that provides mechanisms that can be used to construct an environment
2056 where privilege does not imply any right of access. CCA isolates the execution environments
2057 from each other irrespective of privilege level. This separation of policies has many in-depth
2058 consequences for architecture, trust relationships, and contracts.

2059 Arm CCA protects data in use by performing computation within a hardware-backed and
2060 remotely verifiable secure environment. It shields code, data, and execution from observation
2061 and modification by other software and hardware agents. With CCA, the owner of a Protected
2062 Execution Environment does not need to trust other co-hosted software or privileged hardware
2063 agents, such as direct memory access masters. The mechanisms providing the Protected
2064 Execution Environment are directly measured and reported using attestation to determine their
2065 trustworthiness.

2066 **D.3.1.1 Realms**

2067 CCA provides architecture support for dynamically created entities called Realms. A Realm [66]
2068 [67] contains both user (EL0) and kernel space (EL1) code and data. The higher-privileged entity
2069 that manages Realm resources is the NW hypervisor. Realm tenants do not need to trust either
2070 the hypervisor or existing SW code. Realms are protected from each other; a Realm does not
2071 need to trust other Realms.

2072 The CCA RoT enforces authenticity of the CCA platform by attesting to the boot state and
2073 security state of a Realm, the authenticity of the Realm content through verification and
2074 measurement, and the confidentiality of the guest data through Physical Address Space (PAS)
2075 protection and encryption.

2076 Realm world is a world separate from both the NW and the SW which already exist in
2077 TrustZone. Realm world is designed for the exclusive use of Realms. A Realm protects the
2078 information within it from other system entities. Higher-privileged software retains responsibility
2079 for allocating and managing the resources utilized by Realms but cannot access their contents.
2080 Higher-privileged software also retains responsibility for scheduling within its Realms, but
2081 cannot otherwise control or directly observe their execution flow.

2082 Specifically, CCA provides:

- 2083 • Additional memory access control, orthogonal to the existing controls enforced using
2084 translation tables
- 2085 • Execution state protection
- 2086 • Trustworthy measurement (attestation) of the initial state
- 2087 • A guarantee that the system configuration (for example, whether external debug is
2088 permitted) does not change during the lifetime of the Realm (immutability)

2089 Realm data remains confidential even after Realm destruction or system reset.

2090 Realms are explicitly designed to be created and destroyed on demand. A Non-secure hypervisor
2091 can create a new Realm at any time, much like it can create a new VM at any time. The
2092 hypervisor can add pages to a Realm or remove pages from a Realm at any time, much in the
2093 same way it manages the pages of other VMs. This contrasts with TrustZone architecture, which
2094 is not designed to support dynamic creation of Protected Execution Environments.

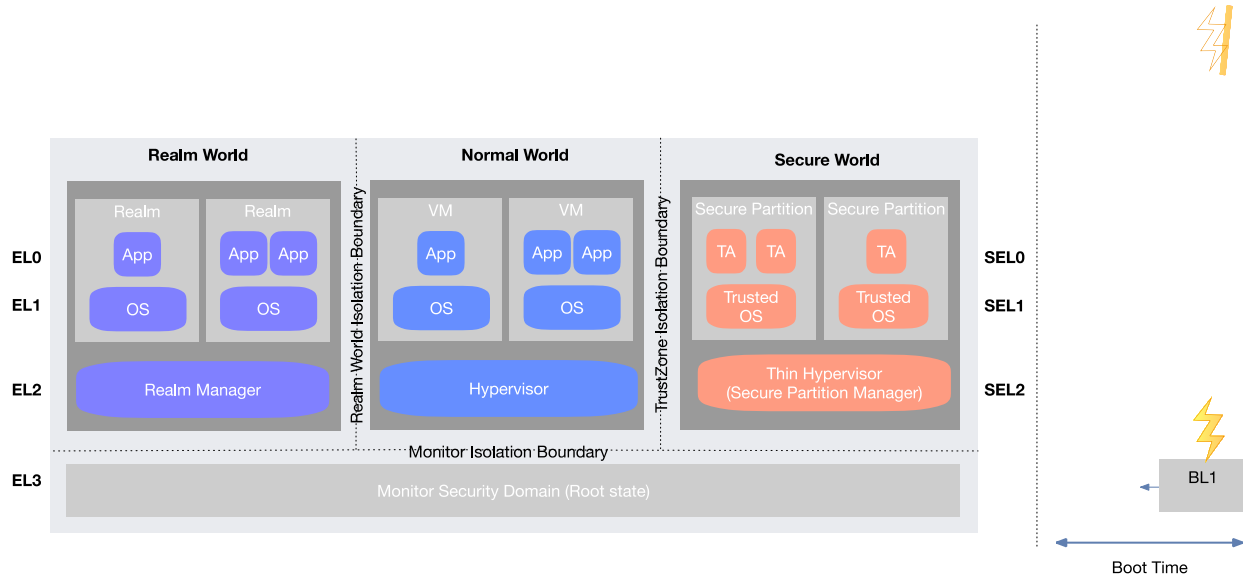
2095 Realms are designed to support complex memory management schemes in existing OSes and
2096 hypervisors with minimally invasive changes. As a result, an architected mechanism is required
2097 to control access to memory used for the implementation of Realms. Control of accessibility of
2098 memory from a given world must be fine-grained (at 4K page granularity) and dynamic. All
2099 physical memory in a CCA system is divided into Granules. Every Granule has a set of
2100 properties which defines the constraints under which the corresponding addresses can be
2101 accessed. Violation of these constraints results in a fault.

2102 **D.3.1.2 Realm Management Extension (RME)**

2103 The Realm Manager is responsible for managing Realms. A Realm is a VM consisting of an OS
2104 kernel (running at EL1) and a set of applications (executing at EL0). In addition to the two
2105 security states supported by TrustZone (Secure state and Non-secure state), CCA introduces two
2106 additional Security states supported by the Realm Management Extension (RME): Realm state
2107 and Root state. With RME, EL3 moves out of Secure state and into its own security state – Root
2108 state. RME provides isolation of EL3 from all other security states. Realm, Non-secure, and
2109 Secure states need to trust EL3. Secure and Realm state can be mutually distrusting because
2110 Secure state is hardware-isolated from both Non-secure state and Realm states, while Realm state

2111 is hardware-isolated from both the Non-secure and Secure states. All other security states contain
 2112 EL2, EL1, and EL0 in both the Realm world and the NW. Figure 11 shows the addition of Realm
 2113 world to the NW and the SW.

2114 The Monitor security domain executes runtime firmware that manages security state switching
 2115 and the assignment of resources among security states. Backwards compatibility is maintained
 2116 with existing TrustZone use cases by retaining the Secure state. However, these existing use
 2117 cases can take advantage of new RME features such as dynamic memory assignment.



2118
 2119 **Figure 11: Root World (Monitor), Realm World, and Isolation Boundaries**

2120 CCA attestation allows a user of a service provided by a Realm – a reliant party – to determine
 2121 the trustworthiness of the Realm and of the implementation of the CCA. The reliant party may be
 2122 local or remote.

2123 Dynamic TrustZone uses RME to provide an architected mechanism to assign pages of memory
 2124 between the Non-Secure and Secure address spaces, at run time. Part of CCA, the RME in
 2125 Armv9-A enables pages of memory to be dynamically transitioned from the NW to the SW and
 2126 back again. Building on RME, CCA provides the following additional features that enhance a
 2127 dynamic TrustZone solution:

- 2128 • Firmware partitioning and isolation of the EL3 monitor, used to provide a stronger RoT
 2129 and attestation services
- 2130 • Encryption of all data in Secure assigned DRAM through the Memory Protection Engine

2131 **D.3.1.3 Realm Memory Isolation and Protection**

2132 As shown in Figure 11, TrustZone provides two security states each associated with its own
 2133 PAS: the Secure PAS and the Non-secure PAS. RME extends this with two additional PASes:
 2134 the Realm PAS and the Root PAS. They provide isolation guarantees for data belonging to each

2135 PAS. Each PAS has its own address translation regime. Device access to memory is subject to
2136 RME PAS isolation guarantees.

2137 **D.3.1.4 External Memory (DRAM) Encryption and Integrity with CCA**

2138 Many Realm and CCA assets are held in external memory. Memory encryption with CCA is
2139 designed to provide additional isolation, and privacy among the Realm, Root, and SWs and
2140 within the Realm PAS. With CCA, memory is uniquely encrypted per world and location using a
2141 separate encryption context for each PAS (Realm, Root, Secure) using a different address tweak
2142 for each encryption data block to provide spatial isolation. Memory encryption is randomly re-
2143 seeded at boot time to make all existing encrypted memory contents undecipherable following
2144 system reset.

2145 Data is encrypted by hardware before being written to external memory or to any shared cache
2146 that resides past the point of physical alias.

2147 CCA protects against a number of different types of memory attacks, including:

- 2148 • **Direct memory access:** An unauthorized agent on the same system attempts to directly
2149 access the contents of memory allocated to a different world, or to a Realm.
- 2150 • **Probing:** An attacker attempts to physically access the contents of memory allocated to a
2151 world or a Realm—for example, using hardware probes or recording devices such as
2152 Non-Volatile Dual In-Line Memory Modules (NVDIMMs).
- 2153 • **Leakage:** An attacker gains access to the contents of memory allocated to a world or a
2154 Realm, for example through misconfigurations or errors in the implementation of the
2155 CCA platform.

2156 **D.3.1.5 CCA Firmware Boot**

2157 CCA root world firmware is booted before the firmware described for the SW in Appendix
2158 D.3.1.4. It starts with immutable initial boot code that may be in an on-chip ROM or locked on-
2159 chip storage. On a secured system, the immutable initial boot code is inherently trusted and is not
2160 verified or measured. It is considered a part of the CCA hardware security domain and identified
2161 by the CCA platform attestation ID. Any updatable CCA firmware, including Realm
2162 management security domain firmware, is both verified and measured and is reported in a CCA
2163 platform attestation.

2164 Realms are measured by Realm management security domain firmware at Realm creation and
2165 reported in a Realm attestation bound to the current CCA platform attestation.

2166 The SW boot chain may be started by Monitor firmware. The NW boot chain is started by the
2167 SW when it instantiates the boot loader (e.g., UEFI) in the NW. They continue independently
2168 from CCA.

2169 The CCA Hardware Enforced Security (HES) is hosted on a trusted subsystem and implements
2170 core CCA security services such as:

- 2171 • The CCA platform boot state
- 2172 • CCA platform attestation services
- 2173 • CCA Root parameters
- 2174 • The CCA system security lifecycle
- 2175 • CCA security provisioning
- 2176 • Collating the CCA hardware boot state

2177 CCA firmware includes:

- 2178 • On the HES device:
 - 2179 ○ HES firmware
 - 2180 ○ Trusted subsystem firmware for the HES host
- 2181 • On the Application Processor (PE) host:
 - 2182 ○ Application PE firmware for the Monitor security domain and the Realm
2183 Management security domain
 - 2184 ○ Trusted subsystem firmware for all other trusted subsystems within the CCA
2185 system security domain

2186 **D.3.1.6 RME and Debug, Trace, Profiling, and Performance Monitoring Protection**

2187 Arm systems include extensive features to support debugging, tracing, and profiling. The RME
2188 provides controls that can limit which parts of the system can be debugged. Signals to enable
2189 different debug, trace, and profiling features are provided that help to manage the features.
2190 External debug signals are typically connected to fuses or an authentication module where debug
2191 for each state can be enabled or disabled, one for each of the four signals. These are usually
2192 managed and disabled depending on the current lifecycle state of the device. External access to
2193 Performance Monitoring Units is regulated by the RME.

2194 Hardware registers control when self-hosted Debug Trace, Profiling, and Performance
2195 Monitoring are context-switched when switching between security states or Realms.

2196 **D.3.1.7 Remote Attestation Service - Project Veraison (VERificAtlon of 2197 atteStatiON)**

2198 This open software initiative is creating software that can be used to build device attestation
2199 verification services that can support many architectures. To support the Arm CCA,
2200 contributions to the Veraison project will develop plug-ins that implement the Arm CCA
2201 attestation model. Veraison supports verification (has reference implementations for Entity
2202 Attestation Token [68], EAT PSA Profile, and Device Identifier Composition Engine [69]) and
2203 provisioning (an API to allow provisioning of Reference Values/Endorsements from supply

2204 chain, object revocation, multi-tenancy data separation, as well as an audit trail). It is intended as
2205 a reference deployment that can be either self-hosted or platform-as-a-service hosted.

2206 **D.3.2 Cryptographic Acceleration**

2207 **D.3.2.1 Cryptography Extensions**

2208 The ARMv8A Cryptography Extensions [47] have added 32 new Advanced SIMD instructions
2209 that operate on the vector register file. They can be used to accelerate the cryptographic
2210 algorithms listed here:

- 2211 • Secure Hash Algorithm 1 (SHA1), SHA256, and AES: these instructions are added to
2212 both the A32 and A64 instruction sets
- 2213 • SHA512, SHA3, SM3, and SM4: these instructions are added to the A64 instruction set
2214 only

2215 They provide three to ten times better software encryption performance. They are useful for
2216 small granule decryption and encryption that is too small to efficiently offload to an external
2217 hardware accelerator.

2218 **D.3.2.2 True Random Number Generation (TRNG)**

2219 A True Random Number Generator (TRNG) provides entropy in the form of random numbers
2220 from the sampled output of an unpredictable physical process rather than by means of an
2221 algorithm (a Deterministic Random Bit Generator [DRBG]). The main application for truly
2222 random numbers is in cryptography, where they are used to generate random cryptographic keys
2223 (e.g., Elliptic Curve Digital Signature Algorithm [ECDSA] key pairs, seed, and nonce;
2224 symmetric MAC keys) to store and transmit data securely (e.g., encryption protocols such as
2225 TLS). Keys generated using a TRNG are unpredictable and therefore are highly resistant to
2226 guessing attacks based on understanding an algorithmic implementation.

2227 Two new random number instructions, RNDR and RNDRRS, have been added [47]. They return
2228 a 64-bit random number into a general-purpose register. A read to the RNDRRS register will
2229 cause a reseeding of the random number before the new random number is generated and
2230 returned.

2231 The DRBG produces random numbers from a cryptographically secure algorithm and is seeded
2232 from the TRNG. The TRNG conforms to several standards, including NIST SP 800-90B, NIST
2233 SP 800-22, FIPS 140-2, and British Standards Institution (BSI) AIS-31. The DRBG algorithm
2234 conforms to the NIST SP 800-90A Rev 1 standard. The entire random number generation
2235 conforms to the NIST SP 800-90C standard.

2236 Appendix E—Cisco Technology Examples

2237 This section describes a number of Cisco technology examples that map back to the key concepts
2238 described in the various sections of the document. Examples provided are focused on Cisco's
2239 Unified Computing System (UCS), an open server that includes x86 processors from other
2240 vendors listed in the appendices and enables users to overlay additional technologies called out
2241 in this document.

2242 E.1 Platform Integrity Verification**2243 E.1.1 Cisco Platform Roots of Trust**

2244 To ensure the highest possible degree of integrity, Cisco computing platforms use a defense-in-
2245 depth methodology to design a platform that prevents both common and sophisticated attacks by
2246 employing technology rooted in various hardware components. Commonly referred to as
2247 Hardware Security Modules in this report, there are two Platform Roots of Trust (PRoT) on
2248 Cisco UCS products. The Cisco Integrated Management Controller is the PRoT on Cisco
2249 Servers, and Cisco's Chassis Management Controller (CMC) is the PRoT for Cisco IO Module,
2250 Cisco Intelligent Fabric Manager, and the chassis mainframe.

2251 Both PRoTs on Cisco computing platforms implement hardware-anchored RoT for firmware
2252 integrity and authenticity. They are anchored in immutable memory (e.g., ROM), in an
2253 embedded SoC or anchored to another hardware chip.

2254 Cisco extends platform integrity to include platform authenticity, rooted in either Cisco's Trust
2255 Anchor Module such as the discrete Anti-Counterfeit Technology 2 or an exclusive TPM for
2256 platform-level-only usage. This proprietary, tamper-resistant chip is found in many Cisco
2257 products and features nonvolatile secure storage, Secure Unique Device Identifier, and
2258 cryptographic services, including random number generation, secure storage, key management,
2259 and cryptographic services to the running OS and applications. Platform authenticity ensures that
2260 the platform is not counterfeit and enables proper manufacturing provisioning of critical security
2261 parameters.

2262 Cisco's platform integrity capabilities also deter certain physical attacks. For critical signals,
2263 Cisco server Printed Circuit Boards (PCBs) are fabricated such that these signals are routed in
2264 intermediate PCB layers instead of being accessible on the top or bottom layers of the board
2265 where easy probing and modification to electrical integrity can occur. Additionally, Cisco uses a
2266 pin-side-down packaging design (e.g., ball grid array) for solder-down components to prevent
2267 easy electrical manipulation of these pins.

2268 Because servers typically contain interchangeable components, sub-assemblies, parts, or devices,
2269 Cisco computing platforms provide authentication for Security Protocol and Data Model
2270 (SPDM) enabled devices (e.g., a storage controller) or other Cisco Field Replacement Units
2271 (FRU) (e.g., Cisco's Virtual Interface Card). SPDM is the standard for securing communications
2272 with and authenticating devices within a platform.

2273 **E.1.2 The Chain of Trust (CoT)**

2274 Cisco's holistic approach to platform integrity focuses on protecting the platform as it operates.
2275 Firmware integrity and authenticity (commonly known as secure boot or a CoT) start at a
2276 hardware-anchored RoT. Firmware integrity is crucial because firmware either controls a
2277 particular device (e.g., a network controller) or various devices and operations on the platform,
2278 such as in Cisco's Integrated Management Controller and CMC.

2279 Cisco's PProT establishes a chain-of-trust from hardware to firmware to software integrity by
2280 ensuring that the initial boot of the host CPUs have additional protections. Cisco servers' PProT
2281 do an additional verification of BIOS code and check the integrity of the PCB by matching
2282 expected values to actual values. This provides extra protection against specific single
2283 component swap attacks. All of Cisco's platform components implement secure boot and a CoT.
2284 More information can be found [here](#).

2285 **E.2 Supply Chain Protection**

2286 Cisco's comprehensive program to manage supply chain risks and provide protection to
2287 customers is highlighted in [Best Practices In Cyber Supply Chain Risk Management](#). This supply
2288 chain protection program includes processes for Cisco to validate the authenticity and integrity
2289 of platform hardware when installing Cisco critical security parameters on Cisco PProTs. These
2290 security parameters are rooted in the PProT, establishing a secure identity from which the CoT
2291 can be authenticated.

2292 **E.3 Software Runtime Protections**

2293 Modern platforms contain many intelligent devices that run some form of code. While this code
2294 is not considered hardware, it plays a major role in affecting the overall integrity of a platform
2295 and is critical to ensuring the platform performs at the highest level of security posture. Whether
2296 it is FPGA code where configuration is passed into a voltage regulator or complex software that
2297 runs on the PProT, Cisco employs additional integrity checks to ensure the code or critical
2298 security parameters are not maliciously modified.

2299 Cisco's PProT firmware architecture utilizes many hardware-level security features appropriate
2300 for the constraints and the security requirements of the design. For example, Cisco ensures code
2301 pages and data pages are isolated so code cannot run from data pages. Application pages are
2302 randomized through address space layout randomization (ASLR). As described above, firmware
2303 employs hardware anchors for securing secret or sensitive data at rest, in transit, and in
2304 operation. On some platforms, ROM code is utilized to ensure firmware authenticity throughout
2305 the boot process. For critical applications and products, Cisco utilizes a third-party agency or an
2306 internal security agency whose specialty is analyzing source code for security-related issues with
2307 an attacker's mindset. Cisco firmware also seeks FIPS certification and Common Criteria
2308 certification to further ensure a certain level of security posture for the user.

2309 Once a component fully boots and its firmware is operational, runtime firmware integrity is
2310 another important aspect of overall platform integrity and supply chain protection. Firmware
2311 components and their contents are controlled and vetted. Open source or purchased software
2312 components are analyzed internally before integrating them. The analysis includes looking for

2313 system() calls common in C or equivalent languages to ensure there aren't malicious command-
2314 line injections, ensuring the code provides input sanitization, verifying the code passes static
2315 analysis, and resolving any errors and warnings from static analysis appropriately. Internally,
2316 Cisco ensures that build infrastructure and development servers meet stringent hardening
2317 requirements and safeguard the flow of software from the very early stages of development to an
2318 official release for users, a process that prevents malicious code injections in the software supply
2319 chain. Employed methods include additional integrity checks for code at rest. This process is
2320 constantly reviewed and improved. Cisco's PProT for certain platforms and FRUs expose
2321 firmware measurements to end users to verify the state of that component. Additionally, Cisco's
2322 PProTs have internal processes that self-monitor critical resources or critical security parameters
2323 to ensure that they have not been tampered with.

2324 **E.4 Data Protection and Confidential Computing**

2325 Another aspect of firmware integrity at runtime is data consumption and data processing. Cisco's
2326 PProT utilizes hardware-anchored secure storage to provide data at rest protection. These
2327 protections are enabled through hardware cryptographic trust anchors, similar to Trusted
2328 Execution Environments (TEE), that securely store secrets such as encryption keys and protect
2329 against certain side channel attacks such as physical attacks using radio frequency (RF)
2330 techniques. Data processing over the network is secured using standard security protocols such
2331 as TLS with identities rooted back to a Cisco Trust Anchor Module.

2332 **E.5 Platform Attestation**

2333 A Cisco PProT continuously monitors the components of the system to detect degraded integrity.
2334 It can alert or take corrective action, up to and including lockdown mode, which prevents usage
2335 until the issue is remediated. A Cisco PProT offers alerts or notification through numerous
2336 protocols: Simple Network Management Protocol, Simple Mail Transfer Protocol, syslog
2337 forwarding, Redfish, and Cisco's xAPI. The PProT monitors other components on the platform
2338 and, depending on their status and their role, it may take corrective actions or alert the user. If the
2339 integrity degradation is severe enough, it goes into lockdown mode and prevents continued usage
2340 until the issue is remediated.

2341 **E.6 Visibility to Security Infrastructure**

2342 The Cisco UCS platform provides cryptographically verifiable reports of platform integrity and
2343 security details, including BIOS measurements for the PProT.

2344 **E.7 Workload Placement on Trusted Platforms**

2345 Cisco Intersight is a cloud-based or on-premises management solution that checks the UCS
2346 platform hardware for authenticity via the Cisco PProT before allowing it to be claimed or
2347 managed by the Intersight management solution.

2348 Appendix F—IBM Technology Examples

2349 This section describes a number of IBM technology examples that map back to the key concepts
2350 described in previous sections of this document.

2351 F.1 Platform Integrity Verification**2352 F.1.1 Hardware Security Module (HSM)**

2353 HSMs include cryptographic co-processors which additionally provide very strong tamper-
2354 detection, prevention, and response capabilities against physical attacks, such as the IBM 4758
2355 and its successors. IBM HSMs are available as features of IBM Z, LinuxONE, and IBM POWER
2356 Systems [70]. HSMs offer additional layers of protection in cloud deployment use cases [71].

2357 F.1.2 IBM Chain of Trust (CoT)

2358 IBM Z and IBM POWER Systems are equipped with a hardware RoT to enable key security
2359 capabilities, including trusted boot and enhanced data confidentiality.

2360 TPM is a special type of HSM. POWER9 servers include a TPM 2.0. The TPM 2.0 specification
2361 can be found at the Trusted Computing Group (TCG) webpage [72][73].

2362 POWER 9 Enterprise Systems implement hardware and firmware enhancements to make them
2363 even more secure for hybrid cloud deployments. These include a Secure Initial Program Load
2364 (IPL) Process or Secure Boot which only allows platform manufacturer-signed Hostboot and
2365 POWER Hypervisor related firmware, up through and including Partition Firmware, to run on
2366 the system, and a framework to support remote attestation of the system firmware stack through
2367 a hardware TPM.

2368 Secure Boot implements a processor-based chain of trust based in the POWER9 processor
2369 hardware and enabled by the POWER9 firmware stack. Secure Boot provides for a trusted
2370 firmware base to enhance confidentiality and integrity of customer data in a virtualized
2371 environment. POWER9 Trusted Boot provides for measurements of system configuration and
2372 initial program load (IPL) path code, which can be used later as proof to a third party via
2373 attestation of the system's initial IPL path configuration. In order to create a CRTM, a Secure
2374 Boot flow is used which adds cryptographic checks to each phase of the IPL process until
2375 communications with the TPM are established. This flow aims to assert the integrity of all
2376 firmware that is to be executed on the core processors, thereby preventing any unauthorized or
2377 maliciously modified firmware from running. A firmware component verification failure will
2378 prevent the IPL from completing if the component is deemed critical for system functionality. If
2379 the component is not a core critical function, the failed image will not be executed, the IPL will
2380 be allowed to complete, and appropriate notifications will be presented.

2381 Details of IBM POWER Systems' secure and trusted boot implementation can be found in
2382 [74][75]. Details of the IBM Z and LinuxONE secure boot implementation can be found in [76].

2383 **F.2 Software Runtime Protection Mechanisms**

2384 **F.2.1 IBM ROP and COP/JOP Attack Defenses**

2385 The POWER platform added four instructions (hashst, hashchk, hashstp, hashchkp) to handle
2386 ROP in the Power ISA 3.1B starting in the Power10 processor.

2387 **F.3 Data Protection and Confidential Computing**

2388 In current computing environments, applications rely on system software for providing services,
2389 such as managing access to the computing system's resources. System software (at the
2390 minimum) includes an OS but may also include a hypervisor. Conventionally, system software
2391 must be trusted because it has complete control over applications and their data. The OS or the
2392 hypervisor can access or modify the data of any application, or potentially tamper with any
2393 security features implemented by the application without being detected. Consequently, the
2394 underlying software must be part of the Trusted Computing Base (TCB).

2395 In shared environments, customers are forced to trust that the entities that develop, configure,
2396 deploy, and control the system software are not malicious. Customers must also trust that the
2397 systems software is invulnerable to attacks that escalate privilege and compromise the
2398 confidentiality and integrity of the customers' applications. This broad trust requirement is often
2399 difficult to justify and poses a significant risk, especially for customers who adopt public cloud
2400 services. IBM has led efforts across the industry to address such concerns by reducing the size of
2401 the TCB and introducing technologies that make customer data inaccessible to system and cloud
2402 administrators.

2403 **F.3.1 IBM Memory Isolation Technology**

2404 While confidential computing (see VM isolation below for examples) and encrypting content in
2405 memory are effective security technologies for achieving memory isolation, using these
2406 technologies in combination with other isolation technologies can provide a robust multi-layered
2407 protection scheme.

2408 One example of such an isolation technology is IBM Z Processor Resource/System Manager
2409 (PR/SM). PR/SM is a type 1 hypervisor integrated with all IBM Z models that transforms
2410 physical resources into virtual resources so that many logical partitions (LPARs) can share the
2411 same physical resources. It provides the ability to divide physical system resources (dedicated or
2412 shared) into isolated logical partitions. Each logical partition operates like an independent system
2413 running its own operating environment. PR/SM enables each logical partition to have dedicated
2414 or shared processors and I/O, and dedicated memory (which you can dynamically reconfigure as
2415 needed). PR/SM provides the security administrator the ability to define a completely secure
2416 system configuration. When the system is defined in such a manner, total separation of the
2417 logical partitions is achieved, thereby preventing a partition from gaining any knowledge of
2418 another partition's operation. This isolation technology is evaluated at Common Criteria
2419 Evaluation Assurance Level (EAL) 5+ [77].

2420 F.3.2 IBM Application Isolation Technology

2421 IBM Hyper Protect Virtual Servers are a new technology based on IBM Secure Service
2422 Containers to protect workloads on IBM Z and LinuxONE throughout the application lifecycle.
2423 The IBM Secure Service Container (SSC) is a container runtime technology enabling you to
2424 quickly and securely deploy software appliances on servers. An appliance is an integration of
2425 OS, middleware, and software components that work autonomously and provide core services
2426 and infrastructures that focus on consumability and security.

2427 The goal of SSC technology is to provide a run-time environment to workloads, including access
2428 to network, storage, and crypto adapters. The primary goal is to protect any data created by the
2429 workload and only give a specific workload instance access to its data. That means even a
2430 second workload of the same type never gets access to any data created by another instance. SSC
2431 only runs inside of an SSC-type logical partition. Firmware disables memory access for SSC-
2432 type logical partitions, and a special bootload/ bootchain ensures that only signed SSC-based
2433 appliances can be started in such a partition. There is no ssh access into a hosting appliance, and
2434 activities can only be triggered through well-defined and tested APIs, always with the claim in
2435 mind, that only a workload has access to its data.

2436 Details of IBM Hyper Protect Virtual Servers can be found here [78].

2437 F.3.3 IBM VM Isolation Technology

2438 Both IBM POWER Protected Execution Facility (PEF) and IBM Secure Execution for Linux
2439 (IBM Z and LinuxONE) are examples of TEEs providing VM/memory isolation. IBM POWER
2440 PEF enables support for secure virtual machines (SVMs) on IBM Power Systems. PEF protects
2441 SVMs from other software while the SVMs are at rest, in transit, and while running. SVMs are
2442 supported by a new mode in the IBM Power Architecture called Ultravisor mode that has higher
2443 privilege than the hypervisor mode. An SVM can run only on systems that support PEF and are
2444 verified by the customer who created the SVM. Each system that supports PEF has a
2445 public/private key pair where the private key is known only to the system (not exposed to the
2446 owner of the system). More details about PEF can be found at [78]. Instructions for how to set up
2447 the PEF-enabled software stack can be found at [79]. The PEF/Ultravisor code is available as
2448 open source at [80].

2449 IBM Secure Execution for Linux is a hardware-based security technology that is built into the
2450 IBM z15T and LinuxONE III generation systems. It is designed to provide scalable isolation for
2451 individual workloads to help protect them from not only external attacks, but also insider threats.
2452 Secure Execution provides isolation between a kernel-based VM (KVM) hypervisor host and
2453 guests in virtual environments. This level of vertical isolation is designed to remove the ability
2454 for administrators to have total visibility into the sensitive workloads being hosted on VMs and
2455 individual containers. Secure Execution provides hardened access restrictions to protect
2456 intellectual property and proprietary secrets while allowing administrators to manage and deploy
2457 workloads as black boxes and continue normal job functions. Secure Execution also helps
2458 enterprises provide isolation between individual multi-tenant workloads running on a shared
2459 logical partition. Details of IBM Secure Execution for Linux can be found in [81].

2460 IBM POWER PEF and IBM Secure Execution for Linux leveraged several key innovations led
2461 by IBM Research towards trusted execution such as SecureBlue [82] and SecureBlue++
2462 [83][84], which laid the foundations of secure application isolation.

2463 **F.3.4 IBM Cryptographic Acceleration Technology**

2464 IBM Z, LinuxONE, and IBM Power offer integrated PCIe-attached HSMs. Examples include
2465 IBM 4767 offered as IBM Z and LinuxONE feature CryptoExpress5s, IBM Power Systems
2466 features EJ32 and EJ33 and IBM 4769 offered as IBM Z and LinuxONE feature
2467 CryptoExpress7s and soon to be offered in IBM Power Systems [85][86].

2468 On IBM Z and LinuxONE, the CryptoExpress features provide the flexibility to support different
2469 types of workloads and may be configured as a cryptographic accelerator, an IBM Common
2470 Cryptographic Architecture cryptographic coprocessor, or an IBM Enterprise Public Key
2471 Cryptography Standards (PKCS) #11 (EP11) cryptographic coprocessor [87].

2472 IBM Z and LinuxONE also provide the CP Assist for Cryptographic Functions (CPACF) for
2473 high-performance on-core cryptographic acceleration for symmetric and asymmetric
2474 cryptographic operations. In conjunction with the IBM CryptoExpress adapter, keys that reside
2475 in the HSM may be exported in an encrypted fashion and used by CPACF so that the key is
2476 never in the clear in hypervisor, OS, or application memory [88].

2477 **F.4 Remote Attestation Services**

2478 **F.4.1 IBM Platform Attestation Tooling**

2479 The IBM TPM Attestation Client Server Framework from IBM Research is open-source tooling
2480 to perform platform attestation [89].

2481 **F.4.2 IBM Continuous Runtime Attestation**

2482 Continuous monitoring of an application agent is enabled by extending measurements
2483 throughout its runtime, not just at startup. For example, when enabled, the Integrity
2484 Measurement Architecture (IMA) in the Linux kernel will continuously extend runtime
2485 measurements [90]. These measurements can be attested periodically to a verification service,
2486 which not only checks for unexpected changes to the application agent, but also monitors its
2487 dynamic behavior [91].

2488 **Appendix G—Acronyms and Abbreviations**

2489 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

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	European Telecommunications Standards Institute Network Functions
SEC	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

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

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
SCR TM	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

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

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

2490 **Appendix H—Glossary**

2491	Asset Tag	Simple key value attributes that are associated with a platform (e.g., location, company name, division, or department).
2492		
2493		
2494	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.
2495		
2496		
2497		
2498		
2499	Confidential Computing	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.
2500		
2501		
2502		
2503	Cryptographic Accelerator	A specialized separate coprocessor chip from the main processing unit where cryptographic tasks are offloaded to for performance benefits.
2504		
2505		
2506	Hardware-Enabled Security	Security with its basis in the hardware platform.
2507	Platform Trust	An assurance in the integrity of the underlying platform configuration, including hardware, firmware, and software.
2508		
2509		
2510	Root of Trust (RoT)	A starting point that is implicitly trusted.
2511	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.
2512		
2513		
2514		
2515	Trusted Execution Environment (TEE)	An area or enclave protected by a system processor.