International Journal of Information Security (2025) 24:72
https://doi.org/10.1007/s10207-024-00972-3

REGULAR CONTRIBUTION

A cluster of patterns for trusted computing

Eduardo B. Fernandez[1] · **Antonio Muñoz**[2]

Published online: 4 February 2025

Abstract
The proliferation of Internet of Things and cyberphysical systems has introduced unprecedented challenges in ensuring the integrity and confidentiality of critical data, making robust security mechanisms essential. There are several mechanisms intended to assure trust with respect to the software loaded into the system and the trustworthiness of the boot process. These mechanisms start from a Root of Trust (RoT), from where all the other trusts, e.g., for components and software are derived. As part of the *RoT*, a Secure Storage is needed. This Secure Storage can be considered as part of the *RoT* or considered a separate component. After a *RoT* is established, a Trusted Boot can be performed. The execution of computational processes can then be supported by using separate execution zones (*Zone Isolation*). More complex trust functions such as remote attestation can be performed by a *Trusted Platform Module (TPM)*. In this paper, we propose security patterns for these components. The abstraction power of patterns can be used to define the basic aspects that each of these components must have, thus serving as reference for designers and for security evaluation.

Keywords Trusted systems · Security patterns · Root of Trust · Trusted processing

1 Introduction

The rapid expansion of the Internet of Things (*IoT*) and cyberphysical systems has revolutionized various industries, enabling seamless connectivity and smart automation. However, the widespread adoption of those technologies has also introduced unprecedented security challenges. As digital systems become ubiquitous in critical applications such as healthcare, industrial control systems, and smart cities, assuring their integrity during their boot-up process becomes paramount; the introduction of impostor software can provide many ways to attack the system. The abstraction power of patterns can be used to define those aspects that each of the software units to be used in the system must have. A pattern is a a solution to a recurring problem in a specific context. We define here a cluster of patterns that together describe the possibilities available to designers to build systems which will only use trusted software. These mechanisms start from a Root of Trust (RoT), from where all the other trusts, e.g., for components and software, are derived. As part of the *RoT*, a Secure Memory is needed, which can be considered as part of the *RoT* or as a separate component. After a *RoT* is established, a Secure Boot can be performed. The execution of computational processes can then be supported by using two or more separate execution zones (*Zone Isolation*). More complex trust functions such as remote attestation can be performed by a *Trusted Platform Module (TPM)*. A *TPM* requires a *RoT* and a Trusted Boot; that is, a *TPM* is a compound pattern (a pattern composed of other patterns). Figure 1 is a pattern diagram showing the relationships between these patterns. The *RoT* provides trust for the Trusted Boot and the TPM. Note that a *TPM* usually includes a *RoT*. The Secure Memory is shown here as part of the *RoT*.

Our contributions here include:

- New patterns for *RoT*, Secure Boot, and Secure Memory.
- Revised patterns for *TPM* and Zone isolation.
- An example of the use of these patterns in a system design, intended to validate them.

The Table 1 includes a comparison of the significant contributions of the proposed work to the state of the art

✉ Antonio Muñoz
anto@uma.es

Eduardo B. Fernandez
fernande@fau.edu

[1] Department of Electrical Engineering and Computer Science, Florida Atlantic University, Boca Raton, USA

[2] Languages and Computar Science Department, University of Malaga, Malaga, Spain

Table 1 Significant advances of the proposed work compared to state-of-the-art (SOTA)

Aspect of contribution	State-of-the-art (SOTA)	Advances in this work
Modular and Interdependent Security Patterns Framework	Research on TPMs or Secure Boot focuses on isolated implementations. Integration of patterns into cohesive clusters enhances modularity and scalability	Introduces interrelated pattern clusters connecting functional dependencies (e.g., RoT foundational for Secure Boot and TPM)
Reusability and Abstraction	Conventional frameworks lack abstraction to bridge theory and practice	Adopts a design pattern approach for abstraction and reusability, adaptable to diverse systems
New Patterns	Studies focus on specific technologies (e.g., TPM or Secure Boot), leaving gaps for modular patterns	Proposes novel patterns (e.g., RoT, Secure Boot, Secure Memory) and extends existing ones (e.g., Zone Isolation, TPM)
Holistic System Security	Lacks end-to-end security, often ignoring interplay between components	Ensures lifecycle security by connecting patterns (e.g., Secure Boot for initialization, Zone Isolation for runtime protection)
Advances in Secure Zone Isolation	Hardware isolation exists but is rarely extensible for secure software design	Zone Isolation pattern ensures execution separation and controlled inter-zone communication
Addressing Challenges in Hardware Security	Hardware security solutions exist but lack integration into frameworks	Integrates tamper-resistant modules to defend against physical and firmware attacks
System Modularity and Integration	Fragmented methods specific to certain applications (e.g., mobile, cloud)	Unified reusable pattern clusters for scalable, integrated architectures
Enhanced Threat Modeling and Mitigation	Threat-countermeasure mappings often lack depth	Detailed mappings align with MITRE ATT&CK and NIST guidelines (e.g., RoT counters side-channel attacks)
Validation and Design Example	Validation focuses on components (e.g., Verified Boot in Android), without system integration	Demonstrates integration of proposed patterns into secure mobile device design

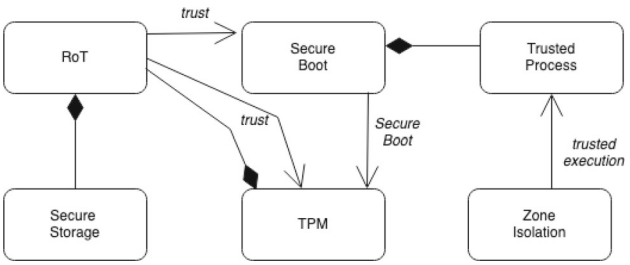

Fig. 1 Pattern diagram of the trust patterns

(SOTA). It highlights advancements in modular security patterns, reusability, novel and extended patterns, holistic system security, detailed threat modeling, and validation through real-world design examples.

To better contextualize the patterns presented in this paper, Fig. 1 illustrates the relationships between key components and their roles within the trusted computing architecture. This diagram provides a high-level view of the Root of Trust (RoT), Secure Storage, Trusted Boot, Zone Isolation, and TPM patterns.

The Root of Trust (RoT) serves as the foundational component, providing a secure foundation for all other patterns. From the RoT, trust is extended to Secure Storage, which ensures the protection of sensitive data and cryptographic keys. Trusted Boot leverages the RoT to validate the integrity of the system boot process. Zone Isolation builds on these components to create separate execution environments, ensuring that sensitive and non-sensitive processes operate independently. Finally, the TPM integrates multiple functions, including attestation, secure storage, and key management, to provide a cohesive Trusted Computing framework. By structuring these patterns into an interconnected cluster, Fig. 1 shows their interdependencies and collaborative functionality. This system-level understanding ensures that designers can effectively use the patterns to build secure and trustworthy systems.

The rest of the article is organized as follows: Sect. 2 presents some background on patterns and trusted computing. Section 3 has the Root of Trust pattern. Section 4 shows the Secure Storage pattern, while Sect. 5 presents the Trusted Boot pattern, Sects. 6 and 7 describe the Zone Isolation and *TPM* patterns, respectively. Section 8 presents an example of the use of these patterns in the design of a complex system and a discussion of some general design aspects. We end in Sect. 9 with some conclusions.

2 Background

2.1 Patterns

As indicated above, a pattern is a solution to a recurring problem in a specific context. All patterns that describe solutions to design problems can be called design patterns, as opposed to patterns that describe other problems such as teaching methodologies. In the context of security, a *security pattern* describes a solution to a security problem by encapsulating good design practices into a reusable and abstract framework.

Abstract Security Patterns (ASPs) describe a conceptual security mechanism that realizes one or more security policies aimed at controlling (stopping or mitigating) a threat or complying with a security-related regulation or institutional policy [25]. To ensure consistency and usability, patterns are described using a structured template composed of several sections. Below, we detail the main components of a pattern's structure, following the extended *POSA template* [17]:

- *Problem* Defines the recurring issue the pattern addresses, such as the need for trusted software loading or secure data storage. It also identifies forces (e.g., trade-offs, limitations) that influence the solution.
- *Context* Describes the conditions and environment where the pattern applies, such as trusted computing environments with sensitive operations.
- *Solution* Outlines the approach to solving the problem, using diagrams (e.g., UML class, sequence) to detail interactions. For instance, Secure Boot solutions rely on a Root of Trust to validate system integrity during startup.
- *Structure* Provides a static view of the pattern using class diagrams. For example, the Root of Trust structure includes Secure Memory, Trusted Processor, and Cryptographic Systems.
- *Dynamics* Describes runtime behavior using sequence diagrams. For example, Secure Boot dynamics illustrate the validation of each boot module in a trusted sequence.
- *Threats and Countermeasures* Identifies potential threats (e.g., tampering, side-channel attacks) and mitigations (e.g., tamper-resistant hardware, cryptographic protections). Countermeasures align with frameworks like NIST and MITRE ATT&CK.
- *Consequences* Highlights the advantages (e.g., flexibility, secure execution) and trade-offs (e.g., performance overhead) of using the pattern. For example, Zone Isolation ensures strong separation but may introduce inter-zone communication overhead.
- *Implementation* Details practical implementation approaches, such as using hardware modules (e.g., TPM) or firmware (e.g., UEFI Secure Boot).
- *Known Uses* Lists real-world applications. For example, Secure Boot is used in platforms like Android Verified Boot [27] and Microsoft BitLocker [34].
- *Related Patterns* Explores complementary or alternative patterns. For instance, the Root of Trust pattern is foundational for Secure Boot and TPM.

Patterns in this paper are grouped into a *cluster*, emphasizing their inter-dependencies. Figure 1 illustrates the relationships, with the Root of Trust as the foundation, extending trust to Secure Boot, Zone Isolation, and TPM. This interconnected approach ensures modularity, scalability, and robust security for trusted computing systems.

2.2 Trusted computing

Trusted Computing (TC) is a foundational technology designed to enforce consistent, expected behaviors in computers through a combination of hardware and software mechanisms [45]. The core objective of TC is to establish trust in a system's behavior by ensuring its integrity, confidentiality, and authenticity. This is achieved through a set of interrelated components and functions that form the backbone of secure systems.

At the heart of Trusted Computing is the establishment of a *Root of Trust (RoT)*, a reliable and immutable component that serves as the foundation for extending trust to other system elements. The RoT verifies the authenticity and integrity of the software and hardware in the system. It provides a starting point for secure operations, such as Secure Boot and Trusted Boot, which ensure that only validated software components are executed during the system startup process. Key components of TC include:

- *Endorsement Key (EK)* A unique 2048-bit RSA key pair embedded in the hardware during manufacturing. It is essential for secure transactions and serves as a root credential for generating additional keys used in attestation and encryption processes.
- *Memory Curtaining* A mechanism for isolating sensitive memory areas, even from the operating system. This isolation prevents unauthorized access and ensures the confidentiality of critical data, such as cryptographic keys and application states.
- *Sealed Storage* Critical to protecting sensitive data, Sealed Storage binds private data to specific software and hardware configurations. This ensures that data can only be accessed after the integrity of the system has been verified, making it a cornerstone for Digital Rights Management (DRM) and other applications that require data confidentiality.
- *Remote Attestation* A process that allows a system to demonstrate its integrity to an external verifier by gener-

ating and presenting a software certificate. This certificate is based on measurements of the system's state, enabling detection of unauthorized software changes and promoting confidence in the system's operation.
- *TPM* A secure hardware module that performs key management, cryptographic operations, and attestation. It is a critical enabler of Trusted Computing, integrating functions such as secure storage, cryptographic key generation, and measurement logging. The TPM also supports Remote Attestation by securely storing integrity measurements and providing cryptographic proofs to external entities.
- *Trusted Third-Party (TTP)* In systems requiring anonymity, a TTP certifies the *Attestation Identity Key (AIK)* used in Remote Attestation. The TTP ensures that attestation can occur without revealing sensitive information about the attesting device, thereby enhancing privacy.

Trusted Computing Framework Together, these components provide a robust framework for secure computing. For example, the RoT initializes the secure boot process, ensuring that all subsequent modules in the system are trusted. The TPM logs and protects integrity measurements that can then be used for remote attestation. Memory Curtain and Sealed Storage provide runtime protection for sensitive data, ensuring its confidentiality and integrity.

Relevance to Security Patterns Trusted Computing concepts are integral to the security patterns presented in this paper. The RoT serves as the foundational element for the Root of Trust pattern, while the Secure Boot pattern builds upon the RoT to validate system integrity at boot time. The TPM pattern uses Sealed Storage and Remote Attestation to ensure trusted operations, and the Zone Isolation pattern uses Memory Curtaining to enforce strict separation between execution environments.

By leveraging these trusted computing principles, the patterns proposed in this work enable the systematic design of systems that consistently maintain critical security goals such as confidentiality, integrity, and availability across diverse application domains. This integration highlights the synergy between theoretical foundations and practical implementations, ensuring robust and adaptable solutions for secure computing.

3 Root of Trust (RoT)

3.1 Intent

A Root of Trust is a trustable component that defines a starting point for the chain of trust used for establishment and attestation of the software modules being loaded onto a system.

3.2 Context

Modern computing systems have increasing numbers of interconnected devices, their complexity is also increasing, and they are exposed to more elaborate attacks. There is a variety of available software from different origins and unknown level of trust.

3.3 Problem

We need to trust the authenticity of the software we are using. This means that we need a secure way to load and start software onto a computer system. This is called a Boot process or just Boot. We need to make sure that every component of the software we are loading is authentic. Otherwise, hackers could replace or modify some of the software modules and compromise the security of the system.

One of the primary consequences of not having a trusted way to perform the boot of a system is that the boot process becomes vulnerable to attacks. Furthermore, a lack of trust in the system's security properties can undermine interoperability and compatibility with other systems, leading to communication failures and compatibility issues.

Additionally, without authentic software, secure updates and remote attestation become difficult to implement. A secure update mechanism requires a trusted source to verify and authenticate the new software-firmware before installing it. Similarly, remote attestation requires the ability to verify the identity and integrity of the remote device.

The solution to this problem must satisfy the following forces:

- Immutability. The initial module must be immutable; otherwise, its verification functions have no meaning.
- Secure execution. During verification in the boot process, the starting module and the following modules must be protected from interference. Otherwise, attackers can introduce malicious software.
- Trusted Anchor: We need a reliable anchor to launch other applications; this allows applications to know their platform is authentic.

3.4 Threats

A *Root of Trust* (RoT) is susceptible to two types of threats [2]:

3.4.1 Physical threats

Where the attacker has physical access to the device. These include:

- Side-channel attacks (T1-SCA). These attacks are based on power analysis of the signals used by the software modules. According to frameworks such as NIST[42] countermeasures and MITRE ATT&CK[6], these can be mitigated through techniques such as secure hardware design and power trace obfuscation.
- Fault injection attacks (T2-FIA). An attacker manipulates a device's electrical inputs (e.g., voltage or clock). By violating the safe ranges of these operating parameters, a fault can be introduced into the processor, resulting in skipped instructions or corrupted memory transactions. NIST countermeasures emphasize the use of input validation and secure hardware redundancy, while MITRE ATT&CK highlights the importance of monitoring anomalous behavior.
- Probe attacks (T3-PA). These access data by physically probing exposed wires or pins. Secure encapsulation and shielding, as recommended by NIST, can minimize this risk. MITRE ATT&CK suggests employing tamper-evident and tamper-resistant techniques to detect and prevent such attacks.
- Tampering attacks (T4-TA). An attacker may change the results of the verification of cryptographic keys. Frameworks such as NIST suggests the use of tamper detection mechanisms, while MITRE ATT&CK emphasizes the importance of monitoring and logging to detect unauthorized changes.

3.4.2 Software threats (T5-SACA)

Software threats include scenarios where the attacker coexists alongside hardware assets and targets critical software components. These include:

- Application-level malware. This exploits vulnerabilities in software applications to execute unauthorized actions or gain access to sensitive data. Established frameworks like MITRE ATT&CK provide detailed techniques for application-level attacks (e.g., T1059 - Command and Scripting Interpreter [10]) and recommend defenses such as application whitelisting and behavioral anomaly detection. NIST countermeasures emphasize secure software development practices and regular vulnerability assessments.
- System-level malware. This type of malware can reside in the operating system or firmware, allowing the attacker to gain privileged access and persist undetected. The MITRE ATT&CK includes techniques for firmware compromise (e.g., T1542 - Pre-OS Boot [12]) and OS-level persistence (e.g., T1547 - Boot or Logon Autostart Execution [13]). NIST recommends the use of secure boot mechanisms, run-time integrity monitoring, and firmware integrity validation.

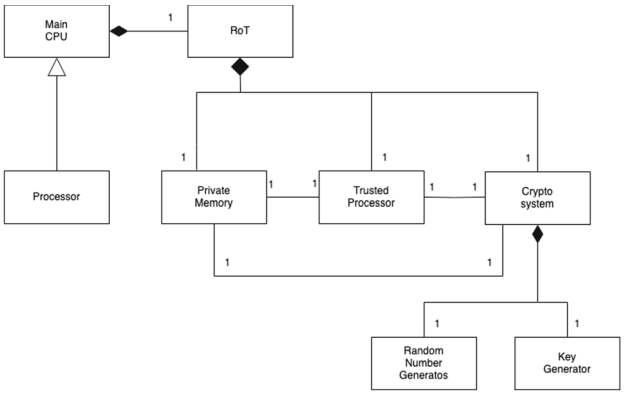

Fig. 2 Class diagram of *Root of Trust (RoT)*

3.5 Solution

Define a module called a *RooT of Trust (RoT)*, protected by hardware. The integrity of the *RoT* is verified and then it sequentially loads the software modules of the system, thus defining a chain of trust. The modules may include the operating system and applications. Control is transferred to the next level only if the integrity of the current layer is verified.

3.5.1 Structure

Figure 2 shows the class diagram of the Root of Trust pattern. It includes Private (*Secure*) Memory to store results of validation and cryptographic keys, a Trusted Processor to execute validation and storage functions, and a Cryptographic System that contains a Random Number Generator and a Key Generator. Class Main CPU indicates some type of processor used to run applications.

3.5.2 Dynamics

The use cases for a *RoT* include: secure boot, issue and store keys, remote attestation, and others. Figure 3 shows a sequence diagram for a secure boot. When the user starts the boot, the initiator module (*RoT*) loads the next module (module1), verifies its authenticity and if positive transfers control to it, module 1 then takes over and performs the same operations on the next module, until a final module, usually the operating system, is reached.

3.6 Countermeasures

Since the defenses to the identified threats are implemented in hardware or firmware, we do not show them as additions to the class diagram of the *RoT*. These defenses include:

- Side-channel attacks (T1-SCA). Commercial offerings such as the *RoT* of Rambus [49, 50] provide electric

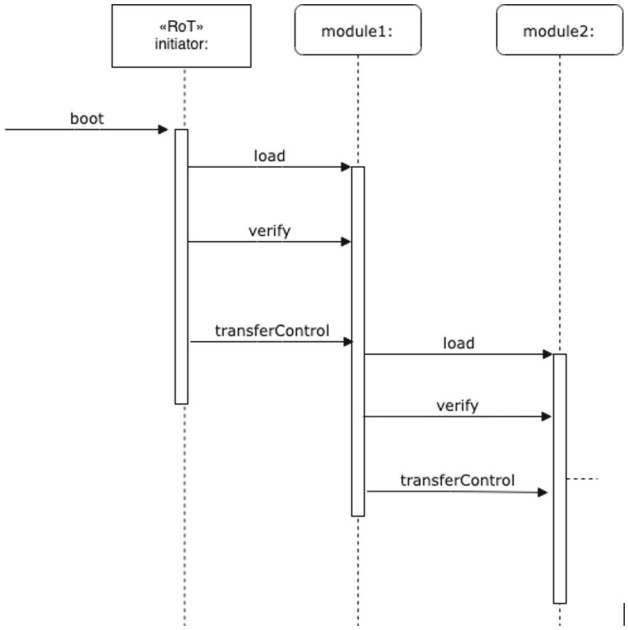

Fig. 3 Sequence diagram for the use case Secure Boot

protection against various side-channel attacks such as *Differential Power Analysis (DPA)*, *Simple Power Analysis (SPA)*, *Simple Electromagnetic Analysis (SEMA)*, *Differential Electromagnetic Analysis (DEMA)*, *Correlation Power Analysis (CPA)*, and *Correlation Electromagnetic Analysis (CEMA)*.

- Fault injection attacks (T2-FIA). Hardware or firmware can be designed to compensate for fault injection attacks. Hardware isolation is another solution.
- Tampering attacks (T3-TA). The hardware can be built to avoid tampering. Hardware isolation is another solution.
- Software attacks on critical assets (T5-SACA). Cryptographic protection of critical assets protects against software attacks. The private memory of the *RoT* can have access controls as shown in Lohr et al. [31].
- Probe attacks (T3-PA) can be mitigated by hardware designs that incorporate physical barriers and shielding techniques to prevent access to sensitive circuitry. Additionally, encryption ensures that any data intercepted through probing remains unintelligible.

3.7 Implementation

The *Root of Trust (RoT)* can be implemented as a software unit [30], as a hardware module, or as part of a Trusted Execution Environment (TEE) [2]. In general, hardware implementations provide more security. Each *RoT* implementation has its strengths and weaknesses, and the selection of the *RoT* implementation depends on the specific security requirements of the system. Some of the *RoT* implementations that are commonly used in the industry are:

- *Rambus Hardware Root of Trust (Rambus, Inc.)* [50] A silicon-based hardware root of trust falls into two categories: fixed function and programmable. Essentially, a fixed-function root of trust is a state machine. These are typically compact and designed to perform a specific set of functions like data encryption, certificate validation and key management. These compact, state machine-based root of trust solutions are particularly well suited for *Internet of Things (IoT)* devices. A hardware-based programmable root of trust is built around a CPU. It can execute a more complex set of security functions. A programmable root of trust is versatile and upgradable, able to run new cryptographic algorithms and secure applications to meet evolving attack vectors. A *Trusted Platform Module (TPM)* 7 is a hardware component that provides *RoT* functionality as well as other functions.
- *Trusted Execution Environment (TEE) TEE* is a secure environment that is isolated from the main operating system (Sect. 6). They can be used to execute *RoT*s.
- Zheng et al. [59] consider the *TPM* connection to the rest of the system to be insecure and propose a dual architecture for trusted booting using a *RoT*.
- *Hardware Security Modules (HSM) HSMs* [3] are specialized hardware devices that are designed to provide secure key storage and cryptographic operations. *HSMs* are commonly used in financial institutions, government agencies, and other organizations that require high levels of security. These provide secure *RoTs* by storing cryptographic keys in a tamper-resistant hardware device.
- *DesignWare tRoot Secure* This is a hardware-based *RoT* from *Synopsis* that they announce as highly secure, emphasizing that it can provide a scalable platform to deliver diverse security functions and applications. Like other implementations, *Synopsis tRoot* [54] uses a secure separate processor to host the public key of the root of trust with its own key management and storage.

The set of required roots defined by *Trusted Computing Group (TCG)* offers essential functionalities to describe the trustworthiness aspects of a platform [28]. Although it is impossible to ascertain if a *Root of Trust* behaves correctly, it is feasible to understand how these roots are implemented. Certificates play a crucial role in ensuring the trustworthiness of a root's implementation. For instance, a certificate may specify the manufacturer and the *evaluated assurance level (EAL)* of a *TPM*, instilling confidence in the Roots of Trust incorporated into the *TPM*. Moreover, a certificate from a platform manufacturer can provide assurance that the *TPM* is correctly installed on a compliant machine, making the

platform's *Root of Trust* trustworthy. TCG mandates three Roots of Trust in a trusted platform:

- *Root of Trust* for Measurement (*RTM*): The RTM is responsible for reliably initiating measurements of the platform's state during the boot process or system initialization. It ensures that integrity measurements are taken and recorded for critical system components, enabling the evaluation of the platform's trustworthiness. This is the first critical step in the chain of trust.
- *Root of Trust* for Storage (*RTS*): The RTS provides a secure mechanism for storing integrity measurements and cryptographic keys. It ensures that sensitive information, such as platform configuration and security status, is protected from unauthorized access or tampering, thereby maintaining data confidentiality and integrity.
- *Root of Trust* for Reporting (*RTR*): The RTR is responsible for securely reporting the stored integrity measurements and cryptographic proofs to an external verifier. This process, known as attestation, allows third parties to assess the trustworthiness of the platform and its compliance with security policies.

While existing hardware and software solutions for *RoT* are well established, the conceptual utility of the *RoT* pattern lies in its ability to generalize these implementations into a reusable framework. The pattern provides a high-level abstraction that can standardize design principles and guide the development of new trusted systems. In addition, reverse engineering and conformance testing can be used to verify that existing implementations adhere to the pattern's structure and principles, ensuring consistency and reliability across platforms.

Certificates also play a critical role in the validation of *RoT* implementations. For example, platform vendors issue certificates that specify compliance with security standards such as *Evaluated Assurance Level (EAL)*. This certification process ensures that the *RoT* behaves as intended and that its integration into the platform is trustworthy.

By providing a structured framework, the *RoT* pattern allows system designers to adapt its principles to different implementations, bridging the gap between theoretical abstractions and practical applications while maintaining robust security.

3.8 Known uses

- Mobile devices, e.g., *Motorola* [51], *Huawei* [29], and others, use a *RoT* to establish trust in the device's boot process. This is critical for maintaining the security and integrity of mobile devices, which are often used to access sensitive information.
- A *RoT* is commonly used in cloud computing systems to provide a secure foundation for virtual machines and containers. In this context, a *RoT* is used to establish trust between the cloud provider and the customer [46], as well as between different customers sharing the same infrastructure. By establishing trust, *RoT* can help prevent attacks such as virtual machine escape [33], in which an attacker gains access to sensitive data by exploiting vulnerabilities in the virtualization layer.
- *RoT* is used on the *Internet of Things (IoT)* to establish trust between *IoT* devices and the cloud or other backend systems [47]. By establishing trust, *RoT* can help prevent attacks such as device spoofing and man-in-the-middle attacks, which are common in those environments.
- A *RoT* is also used in many other types of systems, including automotive systems [48], industrial control systems [57], and critical infrastructure [21]. In these systems, the *RoT* is used to establish trust in the system's firmware, software, and hardware components, and to ensure that only authorized updates are installed.
- Other known uses of *RoTs* include *on-the-fly* drive encryption [32], rootkit detection [46], identification of possible unauthorized modifications at both the OS and application level [57], prevention of improper reads or writes of unauthorized programs in memory, and they even have some presence in hardware-based *digital rights management (DRM)* solutions [18].

3.9 Consequences

The *RoT* pattern provides the following advantages:

- *Immutability* As the *RoT* is implemented in hardware it can be protected against illegal modification.
- *Secure execution* The *RoT* can provide secure execution. A Secure Boot process validates each loaded module and protects the system against invalid software.
- *Trusted Anchor* A *RoT* provides a reliable anchor upon which to launch other applications, with the premise that there are, at least to some extent, guarantees of platform trustworthiness.
- *Secure storage* The results of verification and the cryptographic keys used for protection can be stored in the private memory of the *RoT* or in a separate module (Sect. 4).
- *Flexibility*. The functions of the *RoT* do not depend on the implementation of the application processors.

Its liabilities include:

- The *Root of Trust* pattern introduces additional complexity and overhead into a system, as additional validation

steps and checks may be required. This complexity can be difficult to manage and can introduce potential vulnerabilities if not properly designed and implemented. It can also impact system performance and require additional resources.
– Potential for compromise of the initial trust anchor. If the *RoT* is compromised, the entire system's security can be compromised. This can occur if the *RoT* is implemented using software, as software is inherently vulnerable to attack.
– Potential for failure of the *RoT* itself. If the *RoT* fails, the entire system can become untrusted. This can occur if the *RoT* is not designed with sufficient fault tolerance.

4 Secure storage

4.1 Intent

Control access to stored data based on authorization rules and proof of integrity of the requester.

4.2 Context

Trusted computing environments where highly sensitive information is kept.

4.3 Problem

Hardware is considered trusted, but we need to make sure that only authorized and unmodified software can access highly sensitive information, e.g., secret keys.

The solution to this problem must satisfy the following forces:

Security. We need to control access to storage; otherwise, an attacker would be able to read and modify application data, change secret keys, or use these keys to access sensitive data.

Flexibility. We want to allow only legitimate modifications to the operating systems or applications.

4.4 Threats

– Impostor access to secure storage (T6-IASS). An attacker with a stolen certificate attempts to read or modify sensitive data. Frameworks such as MITRE ATT&CK categorize these attacks under credential access techniques, particularly T1552 [58] (Unsecured Credentials). Countermeasures recommended by NIST include enforcing strong access controls, encrypting sensitive data in storage, and employing certificate revocation mechanisms to mitigate unauthorized access.
– A compromised operating system (T7-COS) attempts to retrieve secret keys. MITRE ATT&CK provides insights into these attacks under the category T1555 [14] (Credentials from Password Stores) and T1570 [16] (Lateral Tool Transfer), highlighting how attackers can exploit operating system-level weaknesses to gain access to secure key storage. NIST suggests measures such as runtime integrity verification, isolating key storage in hardware-based security modules, and continuously monitoring for signs of compromise at the operating system level.

4.5 Solution

A Root Key is used to cryptographically protect data and other keys. This Root Key is used by the Rook Key Control to allow access to data.

4.5.1 Structure

Figure 4 shows the class diagram of this pattern. A Root Key protects the data. The Root Key Control encrypts and decrypts data and verifies the integrity of the software trying to access the data.

4.5.2 Dynamics

Figure 5 shows the use case "Access data in a secure storage". When an application requests access to some data, the Root Key Control verifies its integrity and the integrity of its execution environment. If successful, the Root Key Control uses the Root Key to decrypt the requested data. Now the application can access the data.

4.6 Countermeasures

– Impostor access to secure storage (T6-IASS): An impostor can be authenticated but it cannot pass the integrity test.
– A compromised operating system (T7-COS) will have changes; because of this lack of integrity its attempt will be rejected by the Root Key Control.

4.7 Implementation

The same issues discussed for the *RoT* apply to this pattern.

4.8 Known uses

There are various implementations of *Secure Storage*:

– **Cell Processor Secure Storage** The Cell processor [52] provides secure storage accessible only when the processor is in a *'secure state'*. In this state, the software running

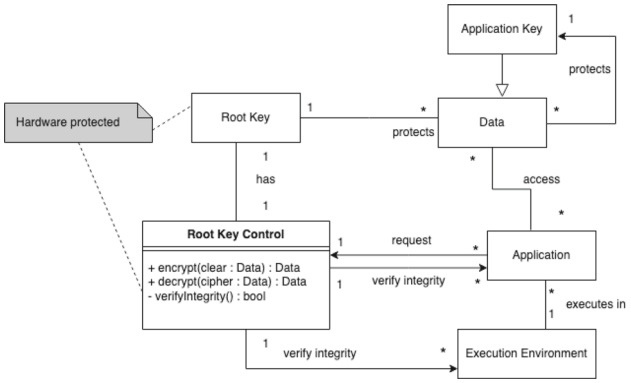

Fig. 4 Class diagram of the Secure Storage pattern (from [31])

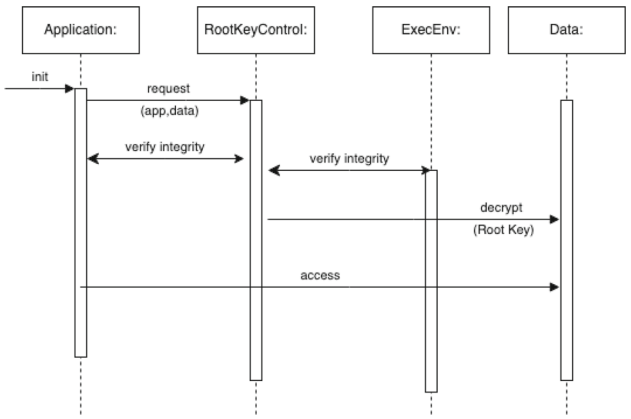

Fig. 5 Sequence diagram of use case "Access data in a secure storage"

on the processor is measured and validated through a secure boot mechanism. This design ensures that secure storage is available only to verified and trusted software.

- **Rambus Hardware Root of Trust** The Rambus Hardware Root of Trust [50], developed by Rambus, Inc., includes a secure storage unit as part of its architecture. This secure storage is designed to protect cryptographic keys, sensitive data, and other critical assets in a hardware-isolated environment.
- **TPM Sealing** The Trusted Computing Group (TCG) specifies a sealing mechanism [55], implemented in the TPM. This functionality allows a key, usable only within the TPM, to have its usage restricted by defining specific values for Platform Configuration Registers (PCRs). Since the PCRs securely store integrity measurements recorded during the authenticated boot process, the key's usage can be restricted to software that successfully passes integrity verification.
- **OMTP TR1 Recommendation** The Open Mobile Terminal Platform (OMTP) TR1 specification [44] outlines detailed requirements for secure storage in mobile devices, ensuring compliance with stringent security standards.

4.9 Consequences

This pattern has the following advantages:

Security The countermeasures can stop the identified threats.

Flexibility Applications that pass the integrity checks of the Root Key Control can amake changes to system software.

Liabilities include: Software updates are more difficult because of the need to show integrity before accessing storage.

5 Secure boot

5.1 Intent

Establish and ensure the integrity, authenticity, and trustworthiness of the boot process in computing systems.

5.2 Context

A computing environment involving connected devices and handling sensitive or potentially sensitive information. We assume we have a root of trust to start the loading.

5.3 Problem

Systems are often threatened at a fundamental level during the boot process, where unauthorized or compromised code can be executed, leading to serious security breaches. A vulnerable boot sequence could allow a tampered operating system or malicious boot loader to be loaded. Such breaches at the boot level compromise the integrity of the entire system, often before the operating system itself starts. In addition, the presence of rootkits or bootkits that embed themselves within the boot process can evade detection by standard security measures and gain deep system access. These threats are of particular concern because they can persist undetected through system reboots and anti-virus scans. In addition, the execution of unauthorized or unverified software during system startup poses significant risks, including compromising system security and data integrity. This is exacerbated in environments where regulatory compliance is critical, as unauthorized software can lead to non-compliance issues. The fundamental nature of these threats requires a mechanism to ensure that only verified and trusted software components are executed during the critical boot phase.

5.4 Solution

The solution to this problem must satisfy the following forces:

- Tamperproof. The initial module must be immutable; otherwise, its verification functions have no meaning.
- Secure execution. During its verification process in the boot process, the starting module and the following modules must be protected from interference. Otherwise, attacks can introduce malicious software.
- Hardware Integrity: Ensuring the underlying hardware hasn't been tampered with, as compromised hardware can undermine the security of the entire system.
- Software Supply Chain Attacks: Mitigating risks associated with the software supply chain, such as the inclusion of malicious code or unauthorized modifications during the software development or distribution process.
- Compatibility and Flexibility: Balancing security with compatibility for various software and hardware, ensuring that security measures do not unduly restrict legitimate software or hardware functionality.
- User Trust and Compliance: Maintaining user trust and meeting regulatory compliance standards, especially in industries where data security and privacy are paramount.

5.5 Threats

Secure Boot is susceptible to several types of threats. These threats, when mapped to established frameworks such as NIST countermeasures and MITRE ATT&CK, highlight specific attack vectors and potential mitigations:

- *Firmware Corruption or Tampering (T8-FCT)* The integrity of Secure Boot can be compromised if the firmware itself is corrupted or tampered with, allowing malicious code to bypass security checks. This aligns with MITRE ATT&CK technique T1542.001 [7] (Firmware Corruption) and NIST-recommended controls, including firmware integrity verification and the use of digitally signed firmware updates.
- *Exploitation of Vulnerabilities in Secure Boot Process (T9-EoVSB)* Potential vulnerabilities within the Secure Boot process, such as flawed verification algorithms or insecure handling of cryptographic keys, can be exploited to circumvent security measures. MITRE ATT&CK techniques T1542.001 [7] (Pre-OS Boot) o T1542.002 [36] (Component Firmware) apply here, and NIST advises secure algorithm design, robust key management, and penetration testing.
- *Unauthorized Key Installation (T10-UKI)* The installation of unauthorized cryptographic keys into the Secure Boot key database can lead to the booting of untrusted software, thereby undermining the security guarantees of Secure Boot. This threat is related to MITRE ATT&CK technique T1556.001 [38] (Credential Manipulation via Key Installation). NIST recommends implementing strict access controls and auditing key management activities.
- *Bootloader Exploits (T11-BE)* Exploits targeting the bootloader can potentially bypass Secure Boot restrictions, allowing for the execution of unauthorized code at system startup. MITRE ATT&CK technique T1542.003 [37] (Bootkit) addresses such exploits, and NIST countermeasures include using a TPM to secure the bootloader.
- *Physical Access Attacks (T12-PAA)* Attackers with physical access to the hardware can attempt to manipulate the Secure Boot settings or hardware configuration to disable or circumvent the Secure Boot mechanism. This aligns with MITRE ATT&CK technique T1562.001 [8] (Impair Defenses: Disable or Modify Tools).
- *Social Engineering Attacks (T13-SEA)* These involve tricking authorized personnel into disabling Secure Boot or making insecure configurations, thus compromising the security of the boot process. MITRE ATT&CK technique T1566 [15] [15] (Phishing) is relevant, and NIST advises awareness training and policy enforcement to mitigate such risks.
- *Supply Chain Attacks (T14-SUCA)* Compromise in the supply chain, such as the inclusion of compromised components or software, can undermine the efficacy of Secure Boot by introducing malicious elements before deployment. This is mapped to MITRE ATT&CK technique T1195 [11] (Supply Chain Compromise), with NIST recommending supply chain risk management (SCRM) practices.
- *Malware Targeting Secure Boot Components (T15-MTSBC)* Malware specifically designed to target and exploit weaknesses in the components involved in Secure Boot can compromise the boot process. This corresponds to MITRE ATT&CK technique T1542.003 [37] (Bootkit), and NIST suggests endpoint protection and real-time monitoring.
- *Insider Threats (T16-IT)* Malicious insiders within an organization could potentially disable or misconfigure Secure Boot settings, leading to a compromise in system integrity. MITRE ATT&CK technique T1562.001 [8] (Impair Defenses: Disable or Modify Tools) applies. This technique highlights how adversaries, including insiders, can disable or modify security settings, such as Secure Boot, to evade detection and compromise system integrity. NIST advises implementing role-based access control (RBAC) and auditing administrative activities, emphasizing tamper-proof audit trails and referencing CyberArk [20] as an example.
- *Outdated or Weak Cryptographic Algorithms (T17-OWCA)* Utilizing outdated or weak cryptographic algorithms in Secure Boot can make the system susceptible to cryptographic attacks, rendering the Secure Boot process ineffective. This risk aligns with MITRE ATT&CK technique T1600.001 [9] (Weaken Encryption: Reduce Key Space). NIST emphasizes regular updates to cryp-

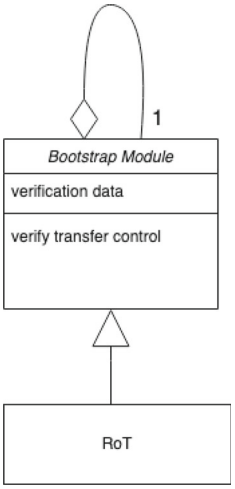

Fig. 6 Class diagram of the Secure Boot pattern

tographic libraries and compliance with the latest standards, such as FIPS 140-2 [43], to mitigate this threat.

5.6 Solution

The Boostrap Modules define a chain of trust. One of these modules is the Root of Trust. Each module is connected to the following module in the chain; it verifies it, stores the results of verification, and if successful transfers control to the next module.

Trusted Boot refers to a security process that ensures a computer or device's operating system and firmware are authenticated and have not been tampered with or altered during the startup process. This process ensures each component in the boot process is validated for integrity, but it does not prevent untrusted software from running after the verification. In essence, "Secure Boot" and "Trusted Boot" (known as specializations of SB) address the need for stronger, more resilient defenses against advanced persistent threats (APTs) that target the boot process and system integrity.

The primary challenges that both mechanisms aim to address are multifaceted. Firstly, they counteract the threat of sophisticated malware, such as rootkits, which can be embedded deeply into the system at the boot level.

5.6.1 Structure

Figure 6 shows the class model of this pattern. Each Bootstrap Module is connected to its successor. It keeps verification data and verifies the authenticity of that module. After the verification is successful, it transfers control to that module. The sequence is started by the Root of Trust, the initial module.

5.6.2 Dynamics

Figure 6 shows the sequence diagram for the use case Secure Boot, illustrating a practical example of how the pattern could be implemented in a real system. The *RoT* starts the boot process. The *RoT* and each Boot Module verify the authenticity of the next module and transfer control to it. The Operating System is the last module loaded and is in charge of loading the applications.

In the first verification approach, cryptographic hash functions, like SHA-256, are used to compute hash values of program binaries, which are then compared to reference values. Any mismatch indicates modifications to the binary. The second approach involves cryptographically signing each program binary with a vendor-specific signature key. The binary's signature can be verified to ensure it remains unmodified after its generation.

The chain of trust is built through a sequence of integrity checks, wherein each stage validates the next. If an integrity violation is detected, execution is halted, and the system stops or invokes a backup module [4]. The root of trust for the entire chain of integrity measurements lies in the very first boot module, which must be protected against unauthorized modifications. Additionally, the integrity verification data, such as hash reference values or signature verification keys, also requires protection.

5.7 Countermeasures

Since the defenses to the identified threats are implemented in hardware or firmware, we do not show them as additions to the class diagram of the Secure Boot. These defenses include:

- *Firmware Corruption or Tampering (T8-FCT)* Secure Boot counters this threat by implementing integrity checks and cryptographic signing of firmware. This ensures that any unauthorized modifications to the firmware are detected, maintaining the integrity of the Secure Boot process.
- *Exploitation of Vulnerabilities in Secure Boot Process (T9-EoVSB)* To combat potential vulnerabilities, Secure Boot is regularly updated and patched. This, along with comprehensive security testing and vulnerability assessments, helps identify and mitigate any weaknesses that could be exploited.
- *Unauthorized Key Installation (T10-UKI)* Strict access controls and authentication mechanisms are employed to manage the Secure Boot key database. This is reinforced by regular audits and monitoring to detect and prevent unauthorized key installations.
- *Bootloader Exploits (T11-BE)* The bootloader is kept up-to-date and securely coded to prevent exploits. Advanced security measures like code signing and integrity verification are implemented for additional protection.
- *Physical Access Attacks (T12-PAA)* Hardware-based security measures, such as Trusted Platform Modules (TPMs), are used to safeguard Secure Boot settings.

Additionally, tamper-evident hardware designs help detect and prevent physical tampering.

– *Social Engineering Attacks (T13-SEA)* Regular security awareness training for personnel is conducted to mitigate the risk of social engineering. Policies against unauthorized changes to Secure Boot configurations are established and enforced.
– *Supply Chain Attacks (T14-SUCA)* The countermeasure includes rigorous supply chain security practices, vetting of suppliers, and secure handling of components and software. Cryptographic verification of software and components from suppliers is also a crucial part of this strategy.
– *Malware Targeting Secure Boot Components (T15-MTSBC)* Deployment of advanced malware detection and prevention tools, regular updates of Secure Boot components, and monitoring for signs of compromise form the core of this countermeasure.
– *Insider Threats (T16-IT)* To address insider threats, the least privilege access controls are implemented, along with regular audits of Secure Boot configurations. Behavioral monitoring is also used to detect and mitigate actions by malicious insiders.
– *Outdated or Weak Cryptographic Algorithms (T17-OWCA)* Secure Boot constantly updates its cryptographic algorithms to align with current best practices and employs strong, industry-standard cryptographic algorithms for all operations.

5.8 Implementation

Figure 7 shows an implementation-oriented sequence diagram for a Secure Boot. Upon the system's initialization, the hardware commences the boot sequence by powering up, thereby activating the foundational phase where the Root of Trust (RoT) is established. Subsequently, the Unified Extensible Firmware Interface (UEFI), an integral element of the *RoT*, assumes control to conduct an integrity check of the system BIOS, ensuring the primary firmware is authentic and unbreached. Following the firmware's validation, the process transitions control to the bootloader, exemplified by GRUB, whose integrity is meticulously validated against the *RoT* to confirm its inviolability. The bootloader, in its trusted state, proceeds to load the operating system kernel, meticulously verifying the operating system's signature prior to transferring control; this step is crucial to ascertain that the kernel is the genuine, untampered version intended for the system. With the kernel securely loaded, the operating system takes the helm, embarking on a thorough verification and loading of its components and drivers. This rigorous inspection ascertains the security of each element, safeguarding the system against compromised components. Culminating the secure boot process, the operating system undertakes the initialization and management of applications. At this juncture, applications are granted execution privileges solely after stringent authentication, verified by the preceding security layers, thus ensuring a fortified commencement of the system's operations.

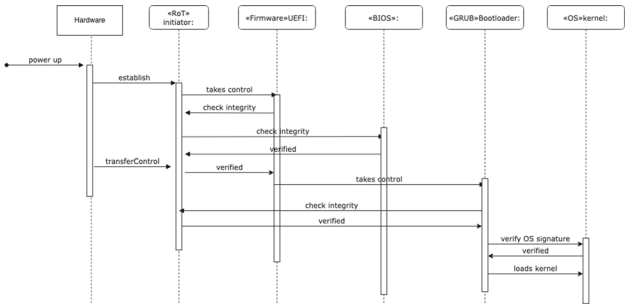

Fig. 7 Sequence diagram for the use case "Secure Boot"

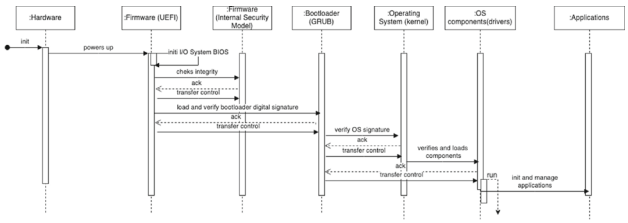

Fig. 8 Secure data access sequence diagram

Diagram from Fig. 8 depicts the secure data access procedure involving an application, Root Key Control, Execution Environment (ExecEnv), and the Data entity. The process begins with the application initiating a request, followed by integrity verifications by the Root Key Control and ExecEnv. Upon successful verification, the ExecEnv decrypts the root key, granting the application access to the data while maintaining the integrity and security of the system.

The UEFI, replacing traditional BIOS, initiates the Secure Boot by validating its integrity with Root of Trust (RoT) support, crucial for system security. This involves checking the UEFI's digital signature against RoT's security elements like cryptographic keys. Upon passing this verification, the UEFI assesses the bootloader, such as GRUB for Linux, verifying its authenticity through RoT's cryptographic keys. GRUB then validates the operating system kernel's signature, ensuring it's secure and permitted to run. Once verified, the kernel oversees loading and verifying OS components and drivers with RoT's backing, ensuring system integrity. The operating system then controls, running applications and managing services securely, completing the Secure Boot process and establishing a ready and secure system. This sequential process from hardware start-up to application launch forms a comprehensive chain of trust, securing the boot process by verifying each component's trustworthiness.

5.9 Known uses

- Arbaugh et al. presented a bootstrap system that used a secure boot process [4].
- *Windows Trusted Boot* Microsoft Windows implements Trusted Boot to safeguard the boot process after Secure Boot has verified the initial boot sequence [35]. Trusted Boot in Windows begins where Secure Boot ends, with the Windows bootloader verifying the Windows kernel's digital signature. It then verifies all other components of the startup process, including boot drivers and Early Launch Anti-Malware (ELAM) drivers. If tampering is detected, it prevents the component from loading, potentially repairing it to restore system integrity.
- *Unified Extensible Firmware Interface (UEFI) Secure Boot* This is a protocol provided by UEFI firmware and is a standard step in the Secure Boot process, which is a part of the Trusted Boot methodology. UEFI Secure Boot ensures that only signed, verified bootloaders and kernels are allowed to be executed, preventing unauthorized code from running at boot time.
- *Measured Boot* Measured Boot [31], often integrated with *Trusted Platform Module (TPM)* technology, takes measurements of each component of the boot process and stores this information in the TPM. The integrity of these measurements can be later verified to ensure that the boot process has not been compromised.
- *tboot (Trusted Boot)* [59] An open-source implementation that works with Intel's Trusted Execution Technology (TXT) and *TPM* to provide a secure boot environment for Linux. tboot can ensure that the platform boots into a known good state by verifying the integrity of the launch environment and the OS.
- *Linux Trusted Boot* Trusted Boot on Linux systems often involves the use of a *Trusted Platform Module (TPM)*. The boot process, in this case, includes a *TPM* taking measurements of the BIOS/EFI layer and creating cryptographic hashes for each binary image, which are then stored in the TPM's Platform Configuration Registers (PCRs). This ensures the integrity of the boot process and can also support remote attestation, allowing external verification of the system's boot integrity.
- *Android Verified Boot (AVB)* Android devices use Verified Boot to ensure the integrity of the device's software. AVB involves checking the cryptographic signatures of each partition before mounting them to ensure they match the signatures provided by the device manufacturer. This process is enforced by the bootloader and is critical for Android's security model.
- *Chrome OS Verified Boot* Chrome OS implements a Verified Boot process that checks each stage of the boot process for tampering. This includes the firmware, kernel, and other critical components. If any modifications are detected, Chrome OS attempts to repair itself by reverting to a known good state.

These known uses demonstrate the broad application of Trusted Boot processes across various platforms and devices, each leveraging cryptographic verification to ensure the integrity and trustworthiness of the system from the initial power-on to the launching of the operating system and applications.

5.10 Consequences

The pattern provides the following advantages:

- *Tamperproof Design* Our security pattern ensures the immutability of the initial module offers the advantage of safeguarding the integrity of verification functions. Since these functions are foundational to the security system, their inviolability is crucial. This advantage directly counteracts the force of needing a tamperproof system, ensuring that the initial verification mechanisms remain effective and unaltered.
- *Secure Execution Benefit* By protecting the initial and subsequent modules from interference during the verification process in the boot process, this security pattern ensures secure execution. This is particularly vital in preventing the introduction of malicious software during critical phases of operation. It addresses the force of secure execution, ensuring that the system remains intact and uncompromised throughout its operation.
- *Hardware Integrity Preservation* This advantage in this context lies in its ability to verify and ensure the integrity of the underlying hardware. This is crucial since compromised hardware can undermine the entire system's security. By addressing this force, the pattern helps in maintaining a robust foundation for overall system security.
- *Verification Rigor* Establishes a stringent verification mechanism to prevent unauthenticated module execution. This advantage speaks to the need for rigorous validation of software component integrity and origin, bolstering system security against unauthorized access or manipulation.
- *Key Security* Highlights the importance of protecting cryptographic keys with the highest security standards to avoid system-wide vulnerabilities. Addressing the need for key security ensures that cryptographic mechanisms, essential for data protection and secure communications, remain uncompromised.
- *Physical Safeguarding* Ensures hardware components are protected from physical tampering, addressing the risk posed by physical interference. This advantage is vital

for upholding the trustworthiness and security posture of the entire system.
- *Compatibility and Flexibility Balance* Balances security measures with the need for compatibility with various software and hardware, ensuring security implementations do not restrict legitimate functionality. This directly addresses the challenge of integrating robust security without compromising on system usability and flexibility.
- *User Trust and Compliance Fulfillment* By enhancing system security and integrity, the pattern significantly boosts user trust and aids in meeting regulatory compliance, especially critical in data-sensitive industries. This advantage ensures the system not only remains secure but also aligns with user expectations and legal standards, addressing the force of user trust and compliance.

The pattern has the following liabilities:

- There is some overhead in verifying signatures.
- Extra complexity (maintenance, scalability, upgrading).

6 Zone isolation

6.1 Intent

Provide two execution environments: one is intended to support the standard functions of phones: make phone calls, take pictures, or store lists of contacts. The other is intended for secure business functions such as accessing databases, using development environments, and employee email. The separation is performed by creating two types of strictly separated virtual processors.

6.2 Context

Corporate databases contain data that may be valuable because they include information about business plans, customer medical records, and similar. We want our employees to have secure access to these resources according to their business functions. People want to use their smart phones and tablets for all kinds of applications, e.g. digital payments as well as all their private functions such as phone calls, interactions with friends, and similar. When using mobile devices to perform work functions we require a highly secure execution environment; for example, to store access rights, cryptographic keys, and trusted processes [56].

6.3 Problem

Mixing employees' private functions with company information may leak valuable company information. A malicious user who has compromised the private zone could get data that can help her attack the work zone. A malicious application run by an unsuspecting user could also collect similar information. We need to keep these two environments strictly separated with controlled interactions.

The following forces apply to the possible solution:

- *Function requirements* Different zones may be used differently which may impose different requirements on them; for example, real-time requirements for phone calls or cameras. We should be able to provide different types of zones to accommodate the needs of different applications.
- *Isolation* We want to make sure that the different zones are strongly isolated. When an operating system in a zone crashes or it is penetrated by a hacker, the effects of this situation should not propagate to other operating systems running on different zones. For example, an attacker should not be able to access data in another zone.
- *Flexibility* If there are significant changes in the threats or the type of applications or system software, we may need to modify the environment to implement different defense mechanisms.
- *Performance overhead* We would like to have minimum performance overhead to perform this isolation.
- *Trust* When the devices are part of some ecosystem we need to create trusted environments. It would be valuable to use the secure zone as a source of trust attestation. System activities such as bootstrapping also require a trusted root.
- *Controlled interzone communications* Both zones need to communicate and there must be a way to perform this communication in a secure way.

7 Trusted platform module (TPM)

We show only its basic sections and new sections on Implementation and Known Uses. The rest can be found in [39].

7.1 Intent

TPM provides assured software execution by verifying that the hardware and software are legitimate and can be trusted before execution takes place. A chain of trust and an integrated set of cryptographic keys are fundamental for that purpose. It may also (or instead) be intended to store secret keys and perform encryption or decryption on request.

7.2 Context

Many cyber-physical systems are safety-critical (patient monitoring, vehicle navigation, water purification) and a

false operating system or other system software could be catastrophic. We need a way to attest these systems to make them trustworthy.

7.3 Problem

Having a way to certify that a computing platform is trusted and executes only trusted software is necessary for many types of applications. The lack of such a facility would make the processing and use of critical information very risky. Similarly, in many applications dealing with sensitive data we need a place to securely store private keys; otherwise the integrity of the whole system cannot be guaranteed.

The solution to this problem is affected by the following forces:

- *Attestation* We need a way to attest that a platform is legitimate and that any software executing in it is also legitimate. Otherwise, we cannot guarantee that the software execution is secure.
- *Authenticated Boot Process* A mechanism is required to log the boot sequence, thereby establishing a transitive trust chain. The integrity of all subsequent processing is dependent on this chain. Sealed-Bound Key Mechanism. It's necessary to have a system that ensures keys for encryption and decryption are securely stored within a trusted platform. This requires binding the creation of these keys to the state of the platform.
- *Secure cryptography* We need ways to generate keys in a secure way so we can use them to encrypt/decrypt documents and data. In any other case, the entirety of the matter would be subject to uncertainty.
- *Maintaining Data Integrity* It is imperative to have the capability to confirm that data has not been altered without proper authorization. Failing to do so would compromise the reliability of our data processing.
- *Assured Secure Processing* It is essential to have mechanisms in place that guarantee software does not illicitly leak or alter information during execution, especially when handling critical documents.

7.4 Implementation

TPMs have been traditionally implemented as a set of discrete chips soldered to a computer's motherboard. This implementation allows a manufacturer to evaluate and certify the TPM, separate from the rest of the system. Nevertheless, recent implementations integrate TPM functionality into the same chipset as other platform components while still providing logical separation similar to discrete TPM chips. Currently, five different types of TPM in its newest version 2.0 implementations can be considered [29] and [5]. Chakraborty [19] have built an implementation using a Subscriber Identity Module (SIM), appropriate for mobile devices. This article also compares the properties of different TPM implementations with respect to their security of TPM, applicability, and deployability.

7.5 Known uses

- IBM's password manager uses it for storing keys.
- Microsoft windows management instrumentation uses TPM for cryptographic co-processing.
- Intel's Trusted eXecution Technology.
- AMD's Secure Technology rely on a hardware TPM.
- Microsoft Bitlocker uses it to release disk encryption credentials only to a trustworthy bootloader.
- Browsers such as Chrome make use of TPM for different purposes.

8 Related patterns

This section explores patterns related to the ones discussed in this paper, highlighting their connections, roles, and the contexts in which they can be applied. These patterns complement or extend the capabilities of the patterns proposed here, offering designers a broader framework for building secure and trustworthy systems.

- *Service Certification Based on Trusted Computing (TC)* [41]: This pattern leverages Trusted Computing mechanisms to certify services by ensuring that the software operating on a platform does not leak or alter sensitive data. Core features include cryptographic primitives, secure key generation, and a Dynamic Root of Trust (DRT). This aligns with the Root of Trust (RoT) and Secure Storage patterns discussed in this paper, emphasizing their roles in providing foundational trust for service certification.
- *Enterprise Service Bus* [24]: A key element in integrating services across architectures, this pattern defines a common communication bus accessible by various services. It complements the TPM by acting as a secure integration point for attested and authorized services.
- *Authenticator* [22]: After verifying the identity of a subject, this pattern grants access based on registration. It parallels the Secure Boot and Trusted Boot patterns by focusing on identity validation as part of the system initialization and access control processes.
- *Authorizer* [22]: Responsible for managing resource access control, this pattern is crucial for implementing Role-Based Access Control (RBAC). It complements Zone Isolation by ensuring that interactions between isolated zones follow strict authorization policies, thereby enhancing security.

– *TPM* [39]: This pattern verifies the legitimacy of hardware and software before execution, ensuring assured software execution. It directly relates to Secure Boot and Trusted Boot by serving as a central component in establishing and maintaining the chain of trust.
– *Zone Isolation Using Virtual Processors* [23]: This pattern creates isolated execution environments, each with dedicated resources for specific functions, such as secure business operations or standard user functions. It complements the Secure Boot and RoT patterns by ensuring that isolated zones maintain integrity and do not interfere with each other.
– *Secure Boot and Secure Memory Patterns* [31]: Presented by H. Lohr et al., these patterns provide mechanisms to maintain the integrity of a system's state and establish secure links between hardware and software components. These complement the RoT, TPM, and Secure Storage patterns by ensuring that system integrity is preserved across all stages of operation.

These patterns form a cohesive framework that designers can use in conjunction with the proposed cluster of patterns to address a wide range of security requirements in trusted computing environments.

9 Example

It is important for system designers to have an integrated set of security functions that can provide protection for all system aspects. It is also useful to have alternatives that allow designers to select the most convenient security mechanism. As an example, we will consider a designer attempting to build a highly secure mobile device, inspired by [19]. The need for identity attestation in the device and the need for having secure processes such as Secure Boot suggests the use of a TPM. The TPM can be built as part of the SIM. A component of the TPM can be used as Root of Trust to provide a Secure Boot to load the software of the device. A TEE can be used to protect the processes of the software that implements the TPM. The device's data can be protected using Secure Storage. The Trusted Process can be protected by using a TEE. Figure 9 shows a pattern diagram of the structure of this phone. The points on the side of the SIM indicate that we can use other patterns to structure the architecture of the mobile device. We note also that there is no pattern for a SIM, see the Conclusions.

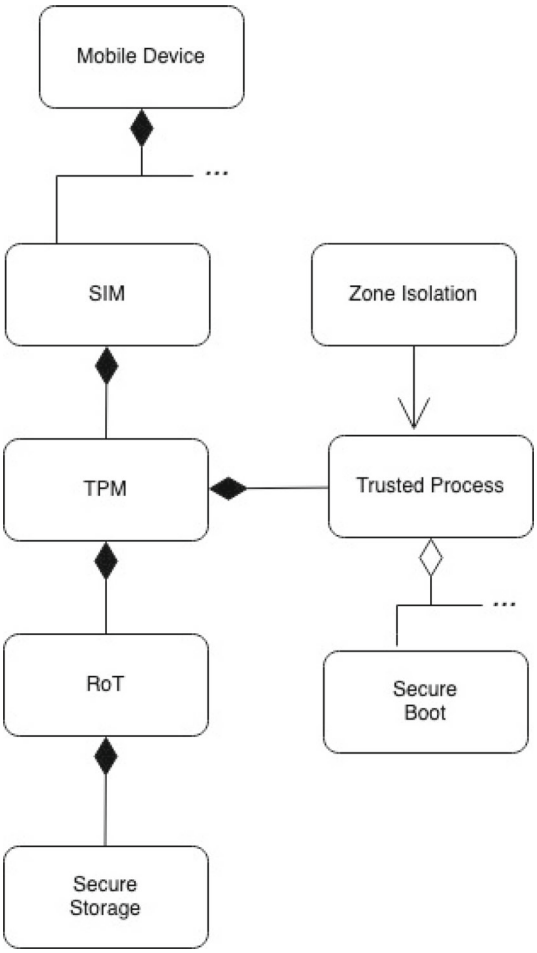

Fig. 9 Pattern diagram of the portable device design

10 Related work

The field of trusted computing has been explored extensively in literature, with significant contributions addressing various components, mechanisms, and their integration. However, the use of security patterns as a framework for describing and designing these components has received limited attention, leaving a gap that our work aims to address.

Oppliger and Rytz [45] provided a comprehensive survey of trusted computing, focusing on its potential to resolve critical security issues in modern systems. While their work covers foundational concepts, it does not mention the use of patterns as a means to structure or standardize trusted computing architectures.

Recent contributions by Ali et al. [2] narrow the focus to specific trusted devices, offering insights into their operational mechanisms and security features. These studies provide valuable details on the hardware and software integration of trusted computing components, such as Trusted Execution Environments (TEEs) and Trusted Platform Modules (TPMs). However, they do not explore the potential of security patterns for abstracting and standardizing these designs.

The concept of a Root of Trust (RoT) has been extensively studied as a foundational element in trusted computing. RoT

Table 2 Cluster of patterns in trusted computing (complete)

Pattern	POSA fields
RoT	Intent: Establish a secure foundation for a chain of trust
	Context: Complex systems with varying trust levels
	Problem: Unauthorized software during boot compromises integrity
	Solution: Define a hardware-protected module that verifies the integrity of software modules in a chain of trust. Each component is loaded only after verification
	Threats: T1-SCA: Side-channel attacks
	T2-FIA: Fault injection attacks
	T3-TA: Physical tampering of hardware
	T4-HSA: Hardware spoofing attacks
	T5-SACA: Malware targeting boot sequence
	Countermeasures: T1-SCA: Secure hardware design, power trace obfuscation
	T2-FIA: Hardware redundancy, error detection
	T3-TA: Tamper-proof enclosures, physical monitoring
	T4-HSA: Strong authentication for hardware devices
	T5-SACA: Cryptographic protection of software modules
	Consequences: Advantages: Immutability, Secure execution, Trusted Anchor, Secure storage, RoT and Flexibility
	Liabilities: Introduces complexity, performance overhead, and dependency on initial trust anchor
	Implementation: Rambus Hardware RoT [50], TPMs [5], TEE [40], HSM [3], DesignWare tRoot [54]
	Known Uses: Mobile devices, IoT devices, cloud computing, automotive systems, industrial controls, critical infrastructure, on-the-fly drive encryption and rootkit detection
Secure Storage	Intent: Protect sensitive data based on requester integrity
	Context: Trusted environments storing high-confidentiality data
	Problem: Unauthorized access or modification of secure data
	Solution: Use a Root Key to cryptographically protect data and verify access requests based on software integrity
	Threats: T6-IASS: Impostor access using stolen credentials or certificates
	T7-COS: Operating systems compromised to retrieve secure keys
	Countermeasures: T6-IASS: Multi-factor authentication, certificate revocation
	T7-COS: Hardware-isolated key storage, integrity checks
	Consequences: Advantages: Ensures secure storage, prevents unauthorized access, supports flexibility for authorized software updates
	Liabilities: Adds complexity to updates, requires robust key management
	Implementation: The same issues discussed for the *RoT* apply to this pattern
	Known Uses: The Cell processor, The Rambus Hardware Root of Trust, TPM Sealing, Open Mobile Terminal Platform (OMTP) TR1 secure storage
Secure Boot	Intent: Ensure integrity and authenticity during boot sequence
	Context: Systems requiring trusted initialization
	Problem: Malicious code compromises system integrity
	Solution: Implement a chain of trust where each boot module verifies the next, starting from a Root of Trust
	Threats: T8-FCT: Firmware corruption or tampering
	T9-EoVSB: Exploitation of Secure Boot vulnerabilities
	T10-UKI: Unauthorized key installations in Secure Boot database
	T11-BE: Bootloader exploits bypassing protections
	T12-PAA: Physical access attacks circumventing Secure Boot
	T13-SEA: Social engineering attacks to misconfigure Secure Boot
	T14-SUCA: Supply chain attacks introducing malicious components
	T15-MTSBC: Malware Targeting Secure Boot Components
	T16-IT: Insider Threats

Table 2 continued

Pattern	POSA fields
	T17-OWCA: Outdated or Weak Cryptographic Algorithms
	Countermeasures: T8-FCT: Cryptographic firmware signing and updates
	T9-EoVSB: Penetration testing, secure algorithm design
	T10-UKI: Strict access controls, database audits
	T11-BE: TPM-based bootloader validation
	T12-PAA: Physical tamper detection, secure hardware
	T13-SEA: Employee training, configuration policy enforcement
	T14-SUCA: Supplier vetting, cryptographic component validation
	T15-MTSBC: Deploy advanced malware detection and prevention tools
	T16-IT: Least privilege AC, audits and Secure Boot configurations
	T17-OWCA: Constant update of cryptographic algorithms
	Consequences: Advantages: Prevents unauthorized execution, maintains system integrity, mitigates supply chain risks
	Liabilities: Adds boot-time delays, requires robust cryptographic management
	Implementation: UEFI Secure Boot, TPM integration, Android Verified Boot, Chrome OS Verified Boot
	Known Uses: Windows Trusted Boot, Linux Trusted Boot, Chrome OS Verified Boot
Zone Isolation	Intent: Provide isolated zones for secure and non-secure processes
	Context: Mixed-use environments requiring data separation
	Problem: Data leakage or unauthorized cross-zone access
	Solution: Use hardware or virtual processors to enforce strict isolation between execution zones
	Threats: T15-XZDA: Cross-zone data leakage or unauthorized access
	T16-MCZ: Malicious applications in compromised zones
	Countermeasures: T15-XZDA: Controlled communication, memory isolation
	T16-MCZ: Secure context switching and zone integrity monitoring
	Consequences: Advantages: Enhances security through strict isolation, supports flexible zone management
	Liabilities: Adds performance overhead, requires sophisticated management
	Implementation: ARM TrustZone, virtualization technologies
	Known Uses: Corporate devices, secure banking, IoT systems
TPM	Intent: Enable tamper-proof authentication and cryptographic operations
	Context: Critical systems requiring trusted execution
	Problem: Compromised hardware or software impacts trust
	Solution: Use a hardware module to perform cryptographic operations and store secure keys, ensuring system integrity
	Threats: T17-WCA: Use of weak cryptographic algorithms
	T18-UKC: Unauthorized key compromise
	Countermeasures: T17-WCA: Regular cryptographic library updates, compliance with standards
	T18-UKC: Tamper-proof hardware, secure key management
	Consequences: Advantages: Enhances secure storage, supports remote attestation, provides robust cryptographic support
	Liabilities: Adds hardware cost, requires proper integration with other components
	Implementation: TPM 2.0, SIM-based TPMs, integrated TPMs
	Known Uses: BitLocker, Chrome, IBM password management

provides the basis for extending trust to other system components, such as Secure Boot, Trusted Boot, and TPMs [30]. Most existing work emphasizes the technical implementation of RoT and its security capabilities [26]. Nevertheless, there is a lack of research exploring RoT within a pattern-oriented framework, which could provide reusable design templates for system architects.

Zone isolation is another crucial aspect of trusted computing, particularly for applications requiring strict separation of processes, such as mobile devices and embedded systems. Sun et al. [53] investigated hardware-assisted isolated computing environments, demonstrating their importance in enhancing security. However, these studies often focus on implementation-specific details and do not address the design principles that could make these solutions reusable and adaptable across different systems.

The role of TPMs in trusted computing has been well-documented, particularly in facilitating complex trust functions such as remote attestation and secure storage [1, 31]. While TPMs provide critical functionality for secure computing, existing research focuses predominantly on their technical details and practical applications. The integration of TPMs into a pattern-based framework for trusted computing remains an open area of research.

Despite the wealth of knowledge in trusted computing, the literature has largely overlooked the potential of security patterns to abstract and generalize the design of trusted computing systems. Security patterns offer a reusable and structured approach to design, enabling designers to address recurring problems in trusted computing with proven solutions. This gap underscores the novelty of our work, which introduces a cluster of security patterns for trusted computing. These patterns provide a unified framework for describing the architecture and functional capabilities of trusted components, thereby bridging the gap between theoretical concepts and practical implementation.

Despite the wealth of knowledge in trusted computing, the literature has largely overlooked the potential of security patterns to abstract and generalize the design of trusted computing systems. Security patterns offer a reusable and structured approach to design, enabling designers to address recurring problems in trusted computing with proven solutions. This gap underscores the novelty of our work, which introduces a cluster of security patterns for trusted computing. These patterns are designed to bridge the gap between theoretical concepts and practical implementations by providing a modular and reusable framework for trusted computing components. By leveraging the abstraction power of patterns, our work aims to equip designers and evaluators with a flexible and effective approach for building secure and trustworthy systems. This contribution not only advances the theoretical foundations of trusted computing but also paves the way for more robust and adaptable system designs in practice.

11 Conclusions

We have presented a cluster of patterns about trusted computing. The patterns in the cluster present alternatives to designers when assembling a software system that requires handling highly sensitive information. Each pattern includes in its template information useful to decide which pattern is more appropriate given the type of application and its security requirements. A design example shows the reasoning in selecting patterns. As we indicate in Sect. 10, this is the only work that brings the use of patterns to the design of systems that require trusted computing (Table 2).

Acknowledgements The authors acknowledge that they received no funding for this study. As co-authors, we wish to acknowledge the pivotal contributions of Dr. Eduardo B. Fernandez to this manuscript. Tragically, Dr. Fernandez passed away before the completion of this work. His dedication to advancing scientific knowledge, his profound insights, and his unwavering commitment to excellence were instrumental in shaping this study. This publication is a tribute to his enduring legacy and a reflection of his invaluable role in its realization. I am deeply honored to have collaborated with him and to present this work in his memory.

Funding Funding for open access publishing: Universidad de Málaga/CBUA

Data availability: Data will be made available on request.

Declarations

Conflict of interest The corresponding author declares that there is no Conflict of interest on behalf of all authors.

Compliance with ethical standards The authors of this study have not conducted any research with humans or animals.

References

1. Alam, M., Ali, T., Khan, S., Khan, S., Ali, M., Nauman, M., Alghathbar, K.: Analysis of existing remote attestation techniques. Secur. Commun. Netw. **5**(9), 1062–1082 (2012)
2. Ali, U., Omar, H., Ma, C., Garg, V., Khan, O.: Hardware root-of-trust implementations in trusted execution environments. IACR Cryptol. ePrint Arch. **2023**, 251 (2023)
3. Amael, J.T., Natan, O., Istiyanto, J.E.: High-security hardware module with PUF and hybrid cryptography for data security (2024). arXiv preprint arXiv:2409.09928
4. Arbaugh, W.A., Farber, D.J., Smith, J.M.: A secure and reliable bootstrap architecture. In: Proceedings of the IEEE Symposium on Security and Privacy, pp. 65–71. IEEE Computer Society Press (1997)
5. Arthur, W., Challener, D., Goldman, K.: A Practical Guide to TPM 2.0: Using the New Trusted Platform Module in the New Age of Security. Springer (2015)
6. MITRE ATT&CK: MITRE ATT&CK (2021). https://attack.mitre.org
7. MITRE ATT&CK: Sub-technique t1542.001: system firmware (2024). Accessed 26 Nov 2024
8. MITRE ATT&CK: Sub-technique t1562.001: disable or modify tools (2024). Accessed 26 Nov 2024
9. MITRE ATT&CK: Sub-technique t1600.001: weaken encryption: reduce key space (2024). Accessed 26 Nov 2024
10. MITRE ATT&CK: Technique t1059: command and scripting interpreter (2024). Accessed 26 Nov 2024
11. MITRE ATT&CK: Technique t1195: supply chain compromise (2024). Accessed 26 Nov 2024
12. MITRE ATT&CK: Technique t1542: pre-os boot (2024). Accessed 25 Nov 2024
13. MITRE ATT&CK: Technique t1547: boot or logon autostart execution (2024). Accessed 25 Nov 2024
14. MITRE ATT&CK: Technique t1555: credentials from password stores (2024). Accessed 25 Nov 2024
15. MITRE ATT&CK: Technique t1566: phishing (2024). Accessed 26 Nov 2024
16. MITRE ATT&CK: Technique t1570: lateral tool transfer (2024). Accessed 25 Nov 2024
17. Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P., Stal, M.: Pattern-Oriented Software Architecture: A System of Patterns, vol. 1. Wiley (1996)
18. Chabaud, F.: Setting hardware root-of-trust from edge to cloud, and how to use it. In: C&ESAR, pp. 115–130 (2022)
19. Chakraborty, D., Hanzlik, L., Bugiel, S.: {simTPM}: user-centric {TPM} for mobile devices. In: 28th USENIX Security Symposium (USENIX Security 19), pp. 533–550 (2019)
20. CyberArk Software Ltd: Cyberark: leader in identity security (2024). Accessed 26 Nov 2024
21. Ehret, A., Moore, P., Stojkov, M., Kinsy, M.A.: Hardware root-of-trust support for operational technology cybersecurity in critical infrastructures. In: 2023 IEEE High Performance Extreme Computing Conference (HPEC), pp. 1–7. IEEE (2023)
22. Fernandez, E.B.: Security patterns in practice: building secure architectures using software patterns. Wiley Series on Software Design Patterns (2013)
23. Fernandez, E.B., Forneron, J.: A security pattern for zone isolation using virtual processors in mobile and embedded systems. In: 12th Latin American Pattern Languages of Programs Conference, Valparaiso, Chile, November (2018)
24. Fernandez, E.B., LaRed, M.D.: Patterns for the secure and reliable execution of processes. In: Proceedings of the 15th International Conference on Pattern Languages of Programs (PLoP 2008), Nashville, TN, October (2008)
25. Fernandez, E.B., Yoshioka, N., Washizaki, H., Yoder, J.: Abstract security patterns and the design of secure systems. Cybersecurity (2022)
26. Fiorin, L., Palermo, G., Lukovic, S., Catalano, V., Silvano, C.: Secure memory accesses on networks-on-chip. IEEE Trans. Comput. **57**(9), 1216–1229 (2008)
27. Google: Android verified boot (AVB) (2013). Accessed 3 Dec 2024
28. Hoeller, A., Toegl, R.: Trusted platform modules in cyber-physical systems: on the interference between security and dependability. In: 2018 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW), pp. 136–144 (2018)
29. Huawei: Emui 8.0 security technical white paper. https://consumer-img.huawei.com/content/dam/huawei-cbg-site/en/mkt/legal/privacy-policy/EMUIOctober2017
30. Li, Y., Cheng, Y., Gligor, V., Perrig, A.: Establishing software-only root of trust on embedded systems: facts and fiction. In: Christianson, B. et al. (eds.) Security Protocols 2015, LNCS 9379, pp. 50–68 (2015)
31. Lohr, H., Sadeghi, A-R., Winandy, M.: Patterns for secure boot and secure storage in computer systems. In: 2010 International Conference on Availability, Reliability and Security (ARES 2010), pp. 569–573 (2010)
32. Maene, P.: Lightweight roots of trust for modern systems-on-chip (2019)
33. Mao, J., Zhu, H., Fan, J., Li, L., Chang, X.: Towards trust proof for secure confidential virtual machines (2024). arXiv preprint arXiv:2405.01030
34. Microsoft: Bitlocker drive encryption (2007). Accessed 3 Dec 2024
35. Microsoft: Secure boot. https://learn.microsoft.com/en-us/windows-hardware/design/device-experiences/oem-secure-boot Accessed 2023
36. MITRE ATT&CK: Technique T1542.002: component firmware (2024). Accessed 26 Nov 2024
37. MITRE ATT&CK: Technique T1542.003: BOOTKIT (2024). Accessed 25 Nov 2024
38. MITRE ATT&CK: Technique T1556.001: credential dumping: LSASS memory (2024). Accessed 25 Nov 2024
39. Muñoz, A., Fernandez, E.B.: TPM, a pattern for an architecture for trusted computing. In: Proceedings of the European Conference on Pattern Languages of Programs (EuroPLoP), vol. 2020, pp. 1–8 (2020)
40. Muñoz, A., Ríos, R., Román, R., López, J.: A survey on the (in)security of trusted execution environments. Comput. Secur. (2023)
41. Muñoz, A., Lopez, J.: A security pattern for cloud service certification. ScienceDirect **1**(1), 1–7 (2018)
42. National Institute of Standards and Technology: Framework for Improving Critical Infrastructure Cybersecurity, Version 1.0 (2014). Accessed 25 Nov 2024
43. National Institute of Standards and Technology: FIPS 140-2: security requirements for cryptographic modules (2001). Accessed 26 Nov 2024
44. Open Mobile Terminal Platform Group (OMTP): Open mobile terminal platform group (OMTP) (2015). Accessed 26 Nov 2024
45. Oppliger, R., Rytz, R.: Does trusted computing remedy computer security problems? IEEE Secur. Priv. **3**(2), 16–19 (2005)
46. Parmar, P., Bhavsar, M.: Achieving trust using rot in IAAS cloud. Procedia Comput. Sci. **167**, 487–495 (2020)
47. Parthipan, L., Chen, L., Newton, C.J.P., Li, Y., Liu, F., Wang, D.: Drot: a decentralised root of trust for trusted networks. In: International Conference on Information and Communications Security, pp. 683–701. Springer (2023)
48. Plappert, C., Lorych, D., Eckel, M., Jäger, L., Fuchs, A., Heddergott, R.: Evaluating the applicability of hardware trust anchors for automotive applications. Comput. Secur. **135**, 103514 (2023)

49. Rambus: Hardware root of trust: everything you need to know. Accessed 25 Nov 2024
50. Inc. Rambus. Root of trust rt-600 series: security anchored in hardware (2021). https://www.rambus.com/security/root-of-trust/rt-600-series/
51. Sabt, M., Achemlal, M., Bouabdallah, A.: Trusted execution environment: what it is, and what it is not. In: 2015 IEEE Trustcom/BigDataSE/Ispa, vol. 1, pp. 57–64 (2015)
52. Shimizu, K., Nusser, S., Plouffe, W., Zbarsky, V., Sakamoto, M., Murase, M.: Cell broadband engine^{TM} processor security architecture and digital content protection. In: Proceedings of the 4th ACM International Workshop on Contents Protection and Security, pp. 13–18 (2006)
53. Sun, H., Sun, K., Wang, Y., Jing, J., Wang, H.: Trustice: hardware-assisted isolated computing environments on mobile devices. In: 2015 45th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, pp. 367–378. IEEE (2015)
54. Synopsys, Inc: DesignWare tRoot Secure Hardware Root of Trust. Accessed 1 Dec 2024
55. Trusted Computing Group: TCG TPM Specification, Version 1.2, Revision 103 (2007). Accessed 26 Nov 2024
56. Winter, J.: Trusted computing building blocks for embedded linux-based arm trustzone platforms. In: STC'08, pp. 21–30, Fairfax, VA, USA. ACM (2008)
57. Wu, Y., Skipper, G., Cui, A.: Uprooting trust: Learnings from an unpatchable hardware root-of-trust vulnerability in siemens s7-1500 plcs. In: 2023 IEEE International Symposium on Hardware Oriented Security and Trust (HOST), pp. 179–190. IEEE Computer Society (2023)
58. Zahid, S., Mazhar, M.S., Abbas, S.G., Hanif, Z., Hina, S., Shah, G.A.: Threat modeling in smart firefighting systems: aligning MITRE ATT&CK matrix and NIST security controls. Internet Things **22**, 100766 (2023)
59. Zheng, C., Li, J., Yao, X.: Design and implementation of trusted boot based on a new trusted computing dual-architecture. Comput. Secur. **127**, 103095 (2023)