

# Security Subsystems for Systems-on-Chip (SoCs)

Common concepts and usage paradigms of security subsystems

## Table of Contents

1	Introduction.....	2	4	Platform security .....	8
1.1	Intended audience .....	2	4.1	System architecture aspects.....	8
2	What is a security subsystem? .....	2	4.2	System security policies .....	9
2.1	Objectives .....	2	4.3	System bring-up .....	9
2.2	Architecture.....	3	4.3.1	Secure boot.....	9
2.3	Industry specifications.....	4	4.4	Runtime integrity protection.....	11
2.4	Hardware Security Module.....	4	4.4.1	Runtime memory protection.....	11
3	Security services.....	5	4.4.2	Security sensors .....	12
3.1	Service requests .....	5	4.5	System lifecycle.....	12
3.2	Key management.....	5	4.5.1	Lifecycles in automotive products.....	12
3.2.1	Key generation.....	5	5	Conclusions.....	13
3.2.2	Key derivation.....	5	6	Appendix: terms and abbreviations .....	13
3.2.3	Key import, export and update.....	6	6.1	Terms and definitions .....	14
3.2.4	Key usage policies.....	6	6.2	Abbreviations .....	14
3.3	Encryption services .....	6	6.3	References.....	15
3.3.1	On-the-fly encryption .....	6			
3.4	Data authentication services .....	7			
3.5	Entity authentication and key agreement services.....	7			
3.6	Random number generator .....	7			
3.7	Monotonic counters.....	8			



## 1 INTRODUCTION

Just about every new device and electronics system entering the market is in one way or another connected to its environment, and often directly or indirectly, to the internet. A complex building control system, for example, may contain many smaller sensor nodes that send data through a wireless connection to a local gateway or through the gateway's internet connection to a backend system. In another example, a connected vehicle, edge nodes provide a range of functions, from seat adjustments to engine control. Each node is interconnected using an in-vehicle network. The network connects to a backend server through a cellular modem, which may also be connected to the vehicle's network gateway. A similar architecture also exists for industrial automation and control systems (IACS).

These modern systems are susceptible to cyberattacks, which increase the importance of mandatory security requirements for many of the integrated circuits (ICs) used in these systems. There are several ways to implement protection, or countermeasures, against cyberattacks. One solution becoming more common is the integration of security subsystems into ICs. This solution can help enforce security policies on the IC itself (i.e, platform security). It also protects assets, including code (software), data, and cryptographic keys when at rest (stored), in use, or in transit (e.g. processed, communicated).

This paper describes common concepts and usage paradigms of security subsystems that are integrated into and are part of a larger microcontroller or system-on-chip (SoC) device. It should be noted that individual implementations may deviate from the descriptions featured here due to the specific needs of an intended IC application.

### 1.1 Intended audience

The intended audiences for this whitepaper are hardware and software architects and design and support engineers who need to understand security subsystems. This whitepaper also provides a good introduction for non-engineers who want to learn more.

The concepts of a security subsystem outlined in this whitepaper are described from a user's perspective. This paper intentionally does not give technical implementation details of the inner workings of security subsystems but instead describes its concepts and usage paradigms. This paper describes the common elements and features of modern security subsystems.

## 2 What is a security subsystem?

A security subsystem is a dedicated subsystem within an IC (i.e., microcontroller or SoC). Several terms refer to such subsystems, including integrated (or on-chip) security subsystems. In the automotive market, they are often referenced as the secure hardware extension (SHE) module or the hardware security module (HSM).

As part of a larger system, a security subsystem provides services to applications and manages and protects cryptographic assets. Due to its integration into the IC system, the security subsystem also provides a way to guard the platform integrity for the entire IC. While security companion ICs (e.g., TPMs, secure elements) may also implement many concepts described in this paper, platform integrity is a state that generally cannot be reached through external (slave) components.

### 2.1 Objectives

The primary objectives for security subsystems are to:

- Provide security services to the application, and support securing application code and data in transit and at rest
- Provide platform security and protect platform integrity as well as the confidentiality and availability of critical assets where needed; these functions includes controlling the application environment in various ways (e.g., secure boot) and enforcing security policies
- Protect cryptographic keys from software attacks by moving the control over critical assets (in particular, cryptographic keys) from the application domain into a separate domain

The exact definition of the functions a security subsystem provides depends on the application and system. It could provide security services, platform security or some other security feature.

## 2.2 Architecture

A microprocessor system with a security subsystem splits into two domains: the application domain and the security subsystem domain. The application domain consists of an environment that provides common resources, such as one or more application processors, RAM, flash, and peripherals. The security subsystem domain hosts or controls security-related assets and services.

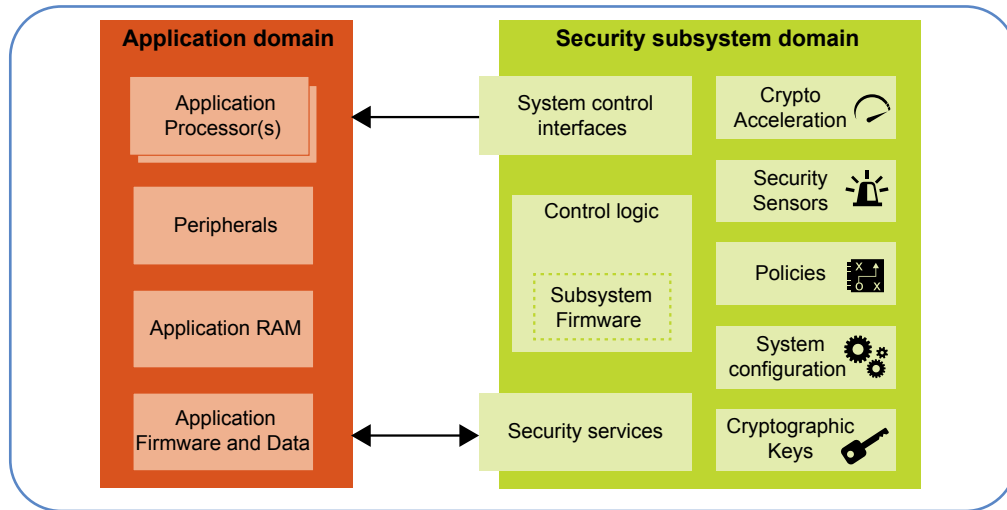


Figure 1: Logical view of security subsystem domain and application domain

The security subsystem is embedded into an IC/SoC and equipped with controls and privileged access to system resources. During system power-on, the security subsystem is one of the first entities to be initialized. During the next steps of system initialization, it assumes the vital role of secure system bring-up.

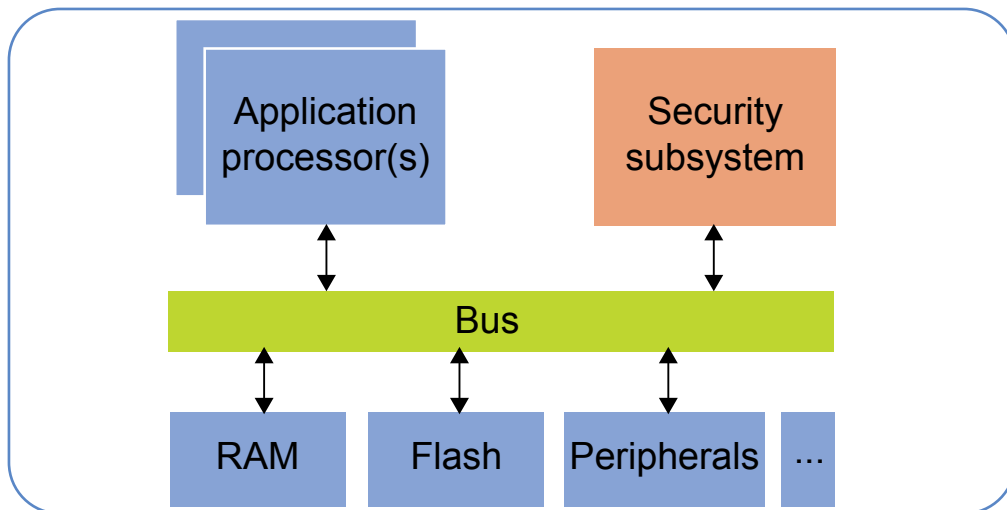


Figure 2: Simplified example of a security subsystem integrated into an IC

For simplicity, the security subsystem is sketched here as a single (monolithic) entity. In practice, the subsystem may have a more distributed (physical) architecture. This architecture consists of several interconnected modules distributed across the IC architecture.

## 2.3 Industry specifications

During the last decade, integrated security subsystems have gained increased popularity and have become a mandatory requirement in some markets and applications.

The automotive market provides a good example for increasingly popular integrated security subsystems. In 2008, the SHE specification <sup>[1]</sup> introduced the concept of a configurable security subsystem for automotive microcontrollers. Within two years, the e-Safety Vehicle intrusion Protected Application's (EVITA's) HSM specification extended this concept into a programmable subsystem in three flavors (full, medium, and light), to address a broader range of use cases. SHE and EVITA have been instrumental in introducing the concept of on-chip security subsystems, including associated usage paradigms such as key usage policies and secure boot, to a broader automotive audience. Since the introduction of the specifications, there has been limited discussion on the need for such subsystems because nearly every customer requirement specification today includes an item on SHE and EVITA compatibility. However, neither specification has been maintained or updated, while automotive systems and applications and the associated security requirements have evolved. Nowadays, vehicle manufacturers are therefore creating their technical specifications, usually based on SHE and EVITA, and are including select aspects of other specifications such as FIPS 140-2 <sup>[2]</sup>, and SAE J3101 <sup>[3]</sup>.

Security subsystems are not specific to automotive. In 2013, Microsoft introduced a security subsystem in the Xbox One's system-on-chip (SoC) that featured a processing core, cryptographic engines, a random number generator, and dedicated memories. This subsystem provides security services to and enforces security policies on the SoC. More recently, Microsoft is marketing a derivative, called Pluton, an IoT security subsystem for Azure Sphere <sup>[4]</sup>.

In the mobile phone markets, the two prevailing operating systems Android and iOS include support for hardware-backed key storage, yet the concept of using (or concealing) cryptographic keys through services is missing. Within the Android ecosystem, there exist further recommendations and reference flows to enable a hardware-supported verified boot process as well as security services implemented in a TEE environment.

Another example is the Arm<sup>®</sup> platform security architecture (PSA) concept, announced in 2017, which proposes a similar architecture for IoT applications. Here, the basic idea is to secure assets by separating them from the application firmware and hardware as well. This security is achieved by introducing a secure processing environment (SPE) for the sensitive assets and the code that manages them. The SPE is isolated from the non-secure processing environment (NSPE), in which the main application and communication firmware executes. The way this split is implemented is different from the SHE-defined process as it permits a logical rather than a physical split.

## 2.4 Hardware Security Module

Security subsystems as described in this whitepaper are sometimes also referred to as (integrated) hardware security modules (HSMs), which is particularly true for the automotive industry due to the EVITA HSM specification. However, the term hardware security module is used differently in other markets and applications, where it may also refer to different solutions including:

- a dedicated tamper-proof physical computing device for mission-critical infrastructure (e.g., PKI)
- a compute module (e.g., an extension card) inside a computer system
- a software implementation (software HSM)
- a stand-alone security coprocessor (e.g., secure element or TPM)
- a subsystem within an IC

To avoid confusion, we refrain from using HSM in the remainder of this whitepaper. The focus of this paper is set on security subsystems (last definition from the above list), which we define as central or distributed security subsystem integrated within a larger IC.

### 3 Security services

A security subsystem provides security services to the application, to support it in tasks such as encrypting or decrypting chunks of data, verifying the authenticity and integrity of received messages, and associated key management.

A security subsystem also protects confidential keys from disclosure to the application and from an adversary controlling the application. In general, keys reside within the security subsystem and are referenced through an identifier when the application wants to make use of a security service; the keys themselves are not exposed to the application.

Furthermore, access to the security subsystem's services is typically restricted to authenticated applications only (i.e., code that passed secure boot). See section 4.3.1.

#### 3.1 Service requests

To use the security subsystem's services, an application issues a service request. The security subsystem may serve such a request immediately or at a later time. Service requests that cannot be answered immediately are handled as service jobs with the security subsystem. Modern security subsystems can handle multiple service jobs at once.

A service request typically contains information such as source and destination memory pointers and reference to the key that is to be used. Cryptographic keys stored in the security subsystem are referenced through a unique key identifier.

#### 3.2 Key management

The security subsystem manages cryptographic keys and protects their integrity and confidentiality. Cryptographic keys are stored separately from applications in a dedicated memory region, which is either physically protected from access through the application or protected through cryptographic means, e.g., an encrypted region within shared non-volatile memory.

A security subsystem can securely handle and protect cryptographic keys of various types, i.e., keys that are used in symmetric, as well as in asymmetric schemes. All cryptographic keys are organized in slots and, as stated above, addressed through a unique reference; namely, a key (slot) identifier. Some implementations of security subsystems allow the organization of keys into groups and subgroups.

##### 3.2.1 Key generation

A security subsystem allows the (internal) generation of cryptographic keys. To mitigate the risk of key exposure, keys may be generated within a security subsystem and thus always remain within control boundaries of the security subsystem.

Internal key generation may constitute an alternative to key injection, e.g., in situations in which the device is already deployed (in use) or the production environment cannot be trusted to protect the confidentiality of a cryptographic key.

##### 3.2.2 Key derivation

A security subsystem typically also supports key derivation, which is a fundamental concept to limit the exposure of a secret key during use. In a key derivation process, a secret (long-term) key along with one or more additional parameters is input to a key derivation function. Additional parameters for key derivation are typically a composition of static and randomly selected inputs, e.g., a static subject string concatenated with a random nonce value.

Key derivation is a common technique used in communication protocols such as TLS. In TLS 1.3, for example, long-term keys of two communicating entities and session-related information are combined in a key schedule to derive a number of ephemeral keys. After a successful handshake among the two communication partners, these keys are used to protect the confidentiality and authenticity of record (payload) data.

As with all cryptographic primitives, a security subsystem should only implement standardized and proven algorithms for key derivation.

### **3.2.3 Key import, export and update**

As an alternative to local key generation, a security subsystem typically also allows the import and update operations on cryptographic keys to support use cases in which the keys are generated elsewhere (e.g., in a PKI) and securely transferred to the subsystem.

An import operation of a cryptographic key into an unpopulated key slot is typically permitted during some stages of a device's lifecycle (see section 4.4) only. Furthermore, to prevent unauthorized use of the update mechanism, a key update operation typically requires proof of knowledge of either the to-be-updated key in the target key slot or of a cryptographic key associated with the update procedure.

A key export operation is conducted if cryptographic keys are to be exported from the security subsystem. Export of (secret) keys shall only be permitted in protected (e.g., encrypted and authenticated) format, and not in unprotected (e.g., plain text) format, as that would defeat the key protection offered by the subsystem.

### **3.2.4 Key usage policies**

Each cryptographic key has attributes (key flags) assigned to it, which are used to define its key usage policy. Based on the setting of key attributes, a security subsystem may permit or deny usage of security service with a given key. Thus, key attributes are a way to define and enforce usage restrictions on cryptographic keys. The owner of the respective key, e.g., the IC manufacturer (for platform-related keys) or the user of the subsystem (for customer keys) defines key attributes.

For example, a symmetric firmware authentication key stored in the subsystem can be restricted to be used for verification only, but not for the generation of authentication tags. This way, the key cannot be misused to sign and execute a manipulated firmware image, e.g., by malicious (compromised) application code running on the IC containing the subsystem.

Also, key import, update and export capabilities (see the previous section) are typically restricted by policy. The reason for this is that a subsystem typically needs to support a wide variety of use cases and users. Such policy allows users that don't need key export capabilities to restrict (disable) all export capabilities on secret keys, whereas other users that need these capabilities, can leave it enabled.

## **3.3 Encryption services**

Encryption services can be used by the application to protect the confidentiality of application data. A typical example is the protection of data transferred over (unprotected and/or untrusted) communication channels.

Security subsystems generally support encryption and decryption using various cipher modes and schemes, based on symmetric and asymmetric ciphers. Examples include AES-CBC, AES-CCM, RSA and ECIES. Encryption operations can be triggered explicitly, e.g., through security services, or implicitly, through on-the-fly encryption services (where available).

### **3.3.1 On-the-fly encryption**

An on-the-fly encryption service performs encryption or decryption operations without an explicit request from the application, but rather on-demand, when the application reads or writes specific memories or regions therein. Keys used with on-the-fly encryption services are managed by the security subsystem, similar to other cryptographic services. On-the-fly encryption services may be configured during system runtime, or once in an IC system's lifetime as part of the system's persistent configuration.

Where available, on-the-fly encryption services allow applications to access memories (e.g., RAM or flash) in a transparent manner. This access means that an application can perform read or write operations on the memory without explicitly performing encryption or decryption operations. Rather, the application addresses the memory contents like any other (non-encrypted) memory. This technique may, for example, be employed for firmware encryption.

### 3.4 Data authentication services

Data authentication services can be used by the application(s) to protect and/or verify the authenticity and integrity of code and data, by generating or verifying MAC tags or digital signatures. Signing and verification services may be used in various applications such as network communication or firmware verification.

In the case of a symmetric scheme (such as a MAC), the secret key must be known to the verification service inside the security subsystem. However, the key's usage policy may restrict its use to verification only. In such cases, the application is unable to forge a valid MAC tag for an arbitrary message (see also section 3.2.4).

### 3.5 Entity authentication and key agreement services

Entity authentication describes the process of authentication among different entities. In its simplest form, one entity (the prover) provides proof of knowledge of secret value to another entity (the verifier), e.g., by supplying a password. More sophisticated and adequate forms of entity authentication protocols provide mutual authentication with zero-knowledge proofs.

If an application demands secure communication after initial authentication, protocols that implement authentication with key agreement are the preferred choice. Popular examples of such protocols include the handshake protocols of TLS, IKE(v2) as well as various EAP methods.

To perform authentication, either a shared secret or a public/private keypair is required. Security subsystems may be utilized to store such information as part of their key storage database. To support a specific protocol implementation however, additional support from the security subsystem is needed in order to retain secret information within the control of the security subsystem. A security subsystem may implement entity authentication and key agreement as part of a larger protocol support service, such as a TLS or IPsec support stack, or a subset thereof, e.g., as (separate) key derivation and message authentication services.

### 3.6 Random number generator

Security subsystems typically provide services for random number generation. Through such services, an application may obtain high-entropy random numbers from a hardware entropy source. Random numbers are also used by the security subsystem itself, e.g., for the signature generation or when cryptographic keys are generated by the security subsystem.

Security subsystems typically offer two types of random number generators: a true random number generator (TRNG) and pseudorandom number generator (PRNG). A true random number generator is comprised of one or more high-quality entropy sources. A pseudorandom number generator is a deterministic algorithm that produces a limited sequence output based on an input or local state (seed).

Entropy sources used in a TRNG are intrinsically slow and do not always meet an application's requirements. Therefore, the TRNG's output is used to extend an entropy pool. The contents of the entropy pool are used as a seed to a PRNG, which is much faster than a TRNG, and the PRNG's output is made available to an application.

Due to its deterministic nature, a PRNG produces the same result when seeded with a static input. To overcome this limitation, the entropy pool is iterated (and every now and then extended with further data produced by the TRNG) whenever an application obtains data through the PRNG.

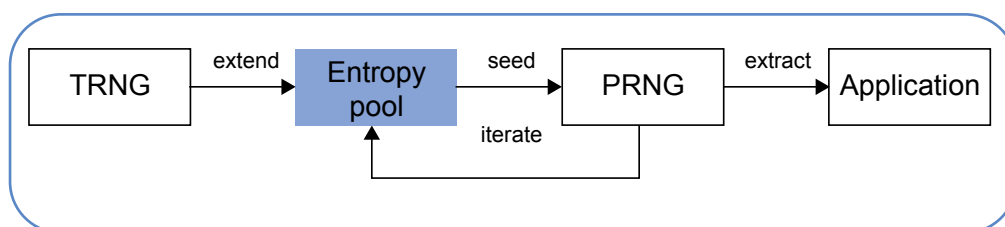


Figure 3: Interaction between TRNG and PRNG functions

Extension of the entropy pool and iteration usually happen asynchronously so that an application may make high bandwidth use of the PRNG without waiting for the next TRNG result.

As part of a random number generator implementation, tests to assess the quality of the entropy may be implemented. Such tests (for example, as defined in BSI AIS-31 <sup>[5]</sup>) are either offline or online tests, meaning the quality of entropy is tested during runtime or offline. Testing and pooling of entropy data inside the context of a security subsystem not only offloads the task, but also helps to obtain reliable and fast sourcing of entropy data.

### **3.7 Monotonic counters**

Security subsystems can also provide monotonic counters. This allows the application to access a trusted state value implemented in the security subsystems. Monotonic counters are useful in various applications, e.g., for logging, freshness assurance, in communication protocols and rollback protection.

Monotonic counter service is a service that protects the state of a counter. Monotonic counters can, as their name implies, not be reset or decremented, but may only be incremented by one (or more).

## **4 Platform security**

Next to providing security services to the application, a security subsystem typically also provides platform security to protect the integrity and availability of critical assets, and their confidentiality, where needed. These assets can be parts of the hardware implementation, as well as code or secrets stored therein. To warrant platform integrity, a security subsystem requires control over parts of the IC architecture and may implement validation mechanisms, such as secure boot (see section 4.3), hardware-enforced isolation and runtime monitoring services (see section 4.4).

### **4.1 System architecture aspects**

A security subsystem must be able to monitor, validate and control the application and its environment to enforce relevant security policies (see 4.2). The security subsystem may, for example, reset the application processor or pause its operation. Violations of the system security policy are typically sanctioned by putting the system into a secure state (reset) and/or restricting access to security services and associated assets. Depending on the criticality of a security policy violation and system policy, security reactions may also result in other actions, such as notifications (e.g., through an interrupt) to an application.

To warrant platform security and to enforce security policies on a larger IC system, the security subsystem requires permanent or temporary control over resources. There are various ways to achieve this control: through configuration during system initialization, resource isolation, specific control channels to an ICs subsystem and many other means. Implementations of IC systems with embedded security subsystem typically employ a selection of various methodologies that are tailored to the system and application needs of a specific IC design.

In order to operate independently and in complete isolation from application domains, the security subsystem requires exclusive access to its own resources and isolation from other components of the IC system.

Communication between the security subsystem and its environment happens through well-defined interfaces, such as service interfaces described in section 3.



## 4.2 System security policies

System security policies describe system behavior, usage conditions and usage restrictions. Security policies are configured as part of a device provisioning process, as part of an update procedure or at runtime (e.g., during initialization).

The system security policy is usually comprised of multiple configuration options that affect peripherals, subsystems or an entire IC. Here are some examples:

- Debug access: Configuration options to describe access modalities for debug functionality
- Secure boot: Configuration of secure boot features of the system, e.g., to define sanctions and system behavior in case of secure boot failure
- System memories: Configuration to manage the partitioning of, and access to, memories
- Firmware protection: Options for firmware readout protection on devices with internal flash, or options for firmware encryption for devices using an external flash
- Access to security services: Options to restrict access to the security subsystem's services, e.g., the option to enforce a rule that only authenticated applications (i.e., code that passed secure boot) can access those services and to protect access by adversarial code.

The above list is by no means exhaustive. There are many more options available that are specific to a particular IC system.

System configuration is typically also tied to the system's state and lifecycle configuration. Depending on the system implementation, some restrictions are automatically applied during certain lifecycles. Lifecycle configuration may as well restrict further re-configuration of the system. For example, when the device is put into an in-field lifecycle, it may not permit further reconfiguration of some or all options.

It should be noted that general-purpose devices such as microprocessors allow manifold options for platform configuration. For example, security functionality may require explicit enablement, or device debug functionality may need to be disabled through configuration. IC system configuration options should therefore be selected wisely, not to allow unintended side effects.

## 4.3 System bring-up

Similar to software applications, hardware units in IC systems also require some form of initialization. There are peripheral components to configure, memory controllers to set up and many other application-specific subsystems to initialize before software applications may even start executing on a complex IC system.

As part of such an initialization process, a security subsystem may validate platform integrity and perform necessary actions to securely initialize an IC system, e.g., by configuring memory barriers. The validation process can be extended to also verify the integrity and authenticity of firmware, the latter commonly being referred to as secure boot (see the next section).

### 4.3.1 Secure boot

A major milestone to achieve protection of an application is secure boot. In a secure boot process, the authenticity of the application firmware is verified through cryptographic validation. Only if the verification confirms the authenticity of the application firmware, the security subsystem permits access to its services and allows the application to execute on the application processor(s). In case the verification fails, the security subsystem may reset the system or apply other sanctions.

The term secure boot originates from the PC industry. In other markets and industries, different terminology may be used, including verified boot, measured boot and authenticated boot, to more specifically indicate that the code's authenticity and integrity is verified during boot. Also, the term secure boot may be used to imply sometimes that the confidentiality of the code is (also) protected. In this whitepaper, we limit the scope of secure boot to the verification of authenticity and integrity only.

#### 4.3.1.1 Boot stages

As part of its configuration, secure boot may be set up to verify a (small) subset of the application firmware, e.g., the application bootloader. Further stages of software (e.g., OS and applications) can be verified by applying the secure boot concept during further software initialization to establish a chain of trust, starting from (hardware-assisted) secure boot down to each component of a software stack. Subsequent application of the verification principle is referred to as chain verification.

In chain verification, the software can make use of security services provided by the security subsystem as shown in Figure 4:

- 1) As part of a secure boot verification, the bootloader application is verified
- 2) After successful verification, the application processor is commanded to start execution of the bootloader application, e.g., by releasing the reset signal
- 3) The bootloader makes use of a verification service provided by the security subsystem
- 4) The bootloader hands execution to the firmware, typically by issuing a JUMP instruction to the start address of the firmware in memory

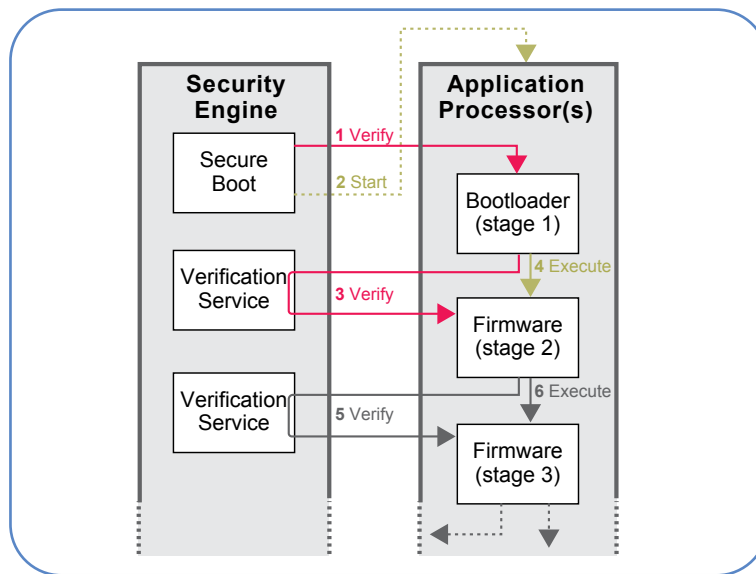


Figure 4: Chain verification process assisted by secure boot and signature verification services

#### 4.3.1.2 Root-of-trust for secure boot

The first boot code, i.e., the first code that starts after power-on reset (“Secure Boot” in Figure 4), is an essential part of the Trusted Code Base (TCB) and acts as root-of-trust (RoT) for the secure boot process and the further system. As such, this code shall be protected against modification, which is typically achieved by storing it in a read-only memory (ROM). Furthermore, this initial boot code shall be kept as simple (small) as possible and be thoroughly reviewed for bugs and vulnerabilities. Such review can be a combination of using automated tools to detect common coding weaknesses and using manual code reviews to identify specific (remaining) weaknesses.

#### 4.3.1.3 Firmware verification options

Several options are available to cryptographically assure the authenticity of (part of) a given firmware image. Common options include signature schemes (e.g., PKCS#1-RSASSA <sup>[6]</sup>, ECDSA or EdDSA) and message authentication schemes (e.g., CMAC <sup>[7]</sup> or HMAC <sup>[8]</sup>).

Signature schemes are generally the preferred choice in situations where key confidentiality cannot be assured, as signature verification does not involve any secrets (rather, it involves a public key). This may be the case if the production environment cannot be trusted.

Signature schemes are computationally intensive. Therefore, the security subsystem may offer the option to verify the signature only once (during firmware installation) and replace it with a cryptographic hash or MAC tag. This will significantly reduce the latency for secure boot (firmware verification). However, such a shortcut may only be taken if the implementation has means to adequately protect the hash value or the secret key that is used for MAC tag verification. This generally requires a dedicated memory region that is controlled by the security subsystem.

#### **4.3.1.4 Sequential boot vs. parallel boot**

Secure boot comes in many flavors due to various configuration and implementation options. The most common way to securely initialize a system is verification prior to execution of the application firmware (sequential secure boot).

However, sequential verification may not be the best option for time-critical applications. In such cases, a parallel secure boot mode may be a viable option, which means that execution of the code is being started (executed) while authentication takes place in parallel and that corrective action (e.g., reset of the application core) is taken if the authentication fails. Therefore, the parallel boot option provides a different tradeoff between latency and security, as it allows a shorter time-to-availability for critical functions, while providing a short window of opportunity for an attacker due to the delayed authentication.

On many multiprocessor systems, secure boot options can be distinctively configured for individual application processors, allowing some cores to use sequential boot mode while others use the parallel boot mode.

#### **4.3.1.5 Confidentiality protection**

Beyond being vital to system integrity, the firmware is a high-value asset that may require confidentiality protection in addition to integrity and authenticity protection.

In devices that store firmware in an on-chip flash area, the firmware is guarded against many non-invasive attacks through readout protection. Conducting invasive attacks is possible; however, the effort and cost of invasive attacks typically exceed the asset's value. For that reason, on-chip firmware readout protection resembles adequate protection in many cases.

When the risk of invasive attacks against on-chip flash is deemed unacceptable, or when external flash is used, the security subsystem may need to provide additional protection mechanisms such as on-the-fly decryption and verification services to protect the integrity of firmware stored in NVM.

### **4.4 Runtime integrity protection**

As previously indicated, additional protection mechanisms may be needed to maintain a secure state after boot, i.e., during the execution of the application. Modern security subsystems therefore may feature runtime integrity protection mechanisms such as regular verification of (configurable) memory regions to validate platform integrity or security sensors to monitor the IC.

#### **4.4.1 Runtime memory protection**

Runtime memory protection is a security service that may be utilized to verify the integrity of application code and data at runtime. Integrity checks are initiated periodically or triggered through events. The service may be utilized to protect code and data from unauthorized modification.

Runtime memory protection may be used to verify the integrity of critical application code or data. Generally, runtime memory protection makes sense for memories whose contents remain unchanged throughout longer periods of time. Examples to which this may apply include operating system code, interrupt vector tables, interrupt handlers or monitoring software.

#### 4.4.2 Security sensors

Security sensors aim at protecting from local adversaries that conduct physical attacks (such as glitching using a laser or EMFI) on the microprocessor or the security subsystem.

To prevent false-positive detection, implementations with security sensors may not directly initiate an action after a single-source sensor event. Instead, sensor events from multiple sources are evaluated and correlated. A single sensor event may, however, result in a temporary sanction (e.g., pause in operations), while a series of correlated events can result in a platform reset.

Sanctions and threshold levels for sensor events can typically be configured as part of the system configuration. In addition, a security subsystem may also be configured to inform the application about sensor events, e.g., through interrupts and/or status registers.

#### 4.5 System lifecycle

IC systems employ a concept of lifecycles which reflects the distinct steps of IC production and manufacturing of the device into which the IC is embedded. During IC production, the IC is tested and embedded into its package by the IC manufacturer. Some initial configuration and firmware may be provisioned to the device during IC manufacturing.

When an IC arrives at a system manufacturer, it is in a lifecycle configuration in which it accepts requests for further programming, debugging and configuration. This state is also referred to as an open or virgin mode. Depending on the implementation, the IC may also be pre-provisioned with credentials supplied by the IC manufacturer, e.g., firmware, keys and configuration.

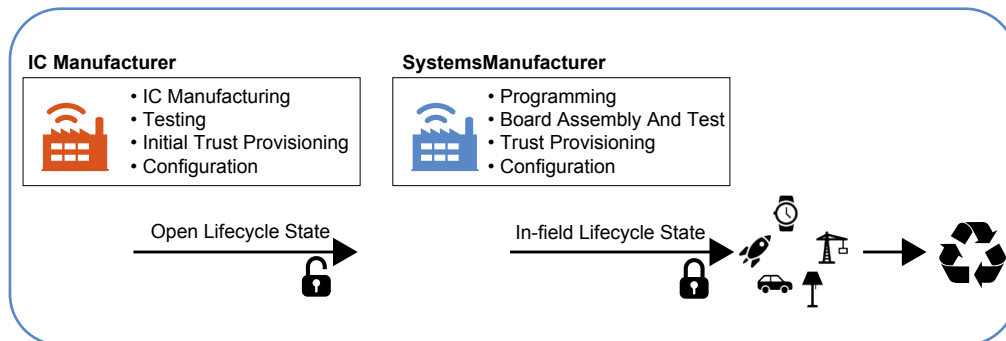


Figure 5: Common device lifecycle

During system manufacturing, the device is programmed and configured to another lifecycle state, which is commonly referred to as the in-field lifecycle. At this stage, the device may be configured (through a security policy; see section 4.2) to accept no further configuration or programming.

The in-field lifecycle may only be left under certain conditions. A transition into another lifecycle from in-field commonly entails erasure of all firmware data, security assets and configuration, setting the device back into the previous open lifecycle, or in a further failure analysis lifecycle. The conditions under which a transition out of the in-field lifecycle is allowed are different among implementations of different semiconductor products and/or may be specified through device configuration.

A transition to a failure analysis state may even render the part useless for further normal use, e.g., by irreversibly disabling some of the functionality needed during normal operation. This limits the possibilities for an attacker to misuse this lifecycle to e.g., reconfigure the device.

### 4.5.1 Lifecycles in automotive products

The E&E architecture of modern vehicles can be comprised of over a hundred distinct electronic control units (ECUs) that various suppliers develop and manufacture. In the automotive industry, various suppliers typically conduct engineering and development activities while the final assembly of the vehicle happens on OEM grounds.

In an automotive production lifecycle, an IC becomes part of a larger vehicle subsystem (ECU) during tier-1 production. After assembly, the ECU is tested, keys are provisioned, and firmware is programmed. Before the ECU leaves the tier-1 production facilities, the IC's lifecycle is advanced into a further lifecycle, enabling the first protection mechanisms to protect Tier-1 firmware and keys.

At the OEM's production line, the device, as part of a larger system (ECU), is deployed into the vehicle and provisioned with further configuration and keys. Before the readily assembled vehicle leaves the OEM production facilities, the IC's lifecycle is once again advanced to a next stage to enable all protection mechanisms of the device.

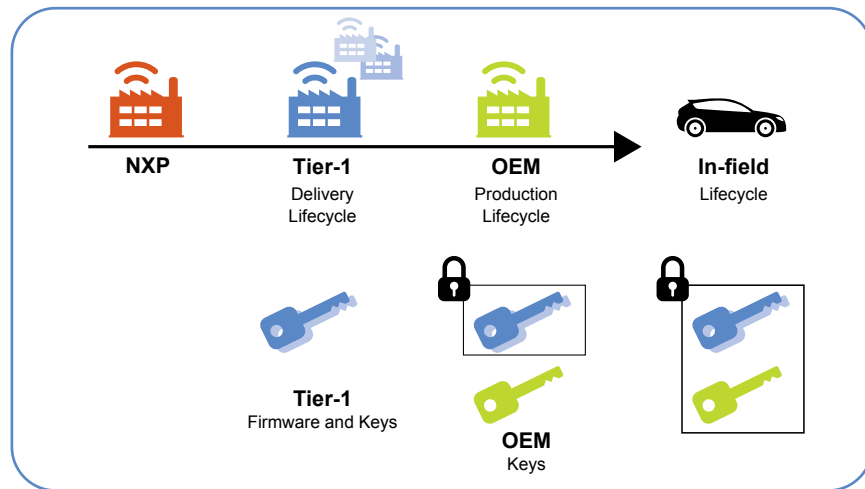


Figure 6: Device lifecycle in automotive applications

To accommodate such distributed development and manufacturing activities between Tier-1s and OEMs, the lifecycle model for automotive ICs extends the standard lifecycle model by a secondary production lifecycle stage that is tailored to the needs during OEM production and vehicle assembly. In the secondary (OEM production) lifecycle stage, the device allows partial configuration and provisioning of additional assets while protecting the integrity and confidentiality of supplier-owned assets.

## 5 Conclusions

As part of a larger system, a security subsystem provides services to applications, manages and protects cryptographic assets, and provides platform security features including secure boot and security policy enforcement. Security subsystems are a leap step with regards to platform integrity and key management, filling a gap to serve system-level protection needs that cannot be addressed through software solutions or external components. In contrast to conventional security solutions, which are often point-solutions that address the needs of a very specific use case, modern security subsystems set the foundation to enable holistic platform security. Embedded in general purpose processors and microcontrollers, security subsystems are serving security needs of a large variety of different applications and use cases.

During the last decade, such integrated security subsystems have gained increased popularity and have become a mandatory requirement in some markets and applications. The key enabler that led to traction in these markets can be clearly identified, namely standardization of software APIs and hardware functionality. Examples include the AUTOSAR software stack and SHE/Evita specifications in the automotive market, and the mobile phone ecosystem which have partially adopted similar concepts with the recent extensions of API definitions to support hardware-based keystores and reference flows for a hardware-anchored verified boot process.

What can be seen from above examples, is that standardization is key to adoption of integrated security subsystem concepts. Although standardized solutions are not yet visible in all markets, the requirements for enhanced platform security are clearly visible today. Some promising steps have already been taken, suggesting it is only a matter of time until we see broad and standardized adoption of enhanced platform security concepts.

## 6 Appendix: terms and abbreviations

For the purposes of this paper, the following terms, definitions and abbreviations apply.

### 6.1 Terms and definitions

Term	Definition
Asymmetric/symmetric cryptography scheme	In the field of cryptography, there generally exist two classes of cryptographic schemes, namely symmetric and asymmetric schemes. Symmetric schemes make use of a secret key that is shared among all entities that perform operations using the cryptographic scheme. In asymmetric schemes, there generally exists a pair of keys that are called private and public keys. A private key is used to perform signature generation and decryption operations, whereas the public key is used to perform signature verification and encryption operations.
Cryptographic hash function	One-way compression function that maps an arbitrary-length input (e.g., firmware data or a message) to a fixed-length output (the hash value). By definition, a secure cryptographic hash function aims to achieve security objectives preimage resistance and second preimage resistance. In layman's terms, this means that it is practically infeasible to find (construct) another input (firmware) that results in the same output (hash value). Therefore, it is very hard to modify a firmware image without being detected by a mismatch in the (stored) hash value.
Digital signature	A digital signature is the result of a signing operation using an asymmetric algorithm. The verifier uses a public key for signature verification and the sender is required to know the corresponding private key in order to generate a valid signature.
Embedded software	Software that is part of the system supplied by the control manufacturer and which is not accessible for modification by the user
Firmware	Program code (software) and data stored in an electronic unit. This could therefore indicate low-level driver code, as well as application code and data that is stored in an embedded system such as a vehicle ECU.
MAC	Short for message authentication code, referring to a function, operation or algorithm that implements a one-way, symmetric message authentication function.
MAC tag	A MAC tag is the result of a MAC operation in which the sender and verifier are using a (symmetric) MAC function with a shared secret key and a message.
Private key, public key	Private and public keys are used in asymmetric schemes. In such schemes, the private key remains with the key owner, the public key is shared with communication partners.
Secret key	A secret key is a confidential key used for symmetric encryption schemes.
Software	Program code

### 6.2 Abbreviations

Abbreviation	Definition
ECU	Electronic Control Unit
EVITA	EVITA was a project co-funded by the European Union within the Seventh Framework Programme for research and technological development. Its objective was to design, verify, and prototype architecture for automotive on-board networks where security-relevant components are protected against tampering, and sensitive data are protected against compromise.
HIS	Herstellerinitiative Software (German for 'OEM software initiative') was an interest group consisting of the car manufacturers Audi, BMW, Daimler AG, Porsche, and Volkswagen [9]. This group created the SHE specification.
HSM	Short for hardware security module. In the automotive context, this is a security unit, which can be a standalone device, or a subsystem that is integrated in a microcontroller as described in this whitepaper. Outside the automotive domain, this term has a different meaning. Please also see section 2.4.
IACS	Industrial automation and control systems
NAD	Network access device. A device (modem) that connects a system to a larger communication network such as the internet.
RoT	The root of trust (RoT) is one or more entities or assumptions beyond the scope of a system that are implicitly trusted, e.g., mathematical assumptions, but also hardware or software implementations.
SHE	The secure hardware extension (SHE) is an on-chip extension to any given microcontroller. The SHE specification essentially describes one specific way to implement a security subsystem. The SHE specification was created in 2008 by the HIS workgroup [9]. The SHE specification has been re-published through AUTOSAR in 2019 [1].
SoC	System on Chip; generally referring to a larger IC system, comprised of multiple application cores
TCB	The trusted computing base (TCB) of a system is the set of all hardware and software components that are critical to the security of that system. Bugs or vulnerabilities inside the TCB might jeopardize the security properties of the entire system.

### 6.3 References

- [1] AUTOSAR, "Specification of Secure Hardware Extensions," AUTOSAR (HIS), 2019 (2008).
- [2] "FIPS PUB 140-2 Security Requirements for Cryptographic Modules," NIST, 2001.
- [3] "Requirements for Hardware-Protected Security for Ground Vehicle Applications," SAE, 2020.
- [4] T. Chen, "Guarding Against Physical Attacks - The Xbox One Story," in Platform Security Summit, 2019.
- [5] W. Killmann and W. Schindler, "A proposal for: Functionality classes for random number generators," Bundesamt für Sicherheit in der Informationstechnik (BSI), Bonn, 2011.
- [6] M. e. al, "PKCS #1: RSA Cryptography Specifications Version 2.2," IETF, 2016.
- [7] M. Dworkin, "Recommendation for Block Cipher Modes of Operation:," 2005.
- [8] "RFC 2104 HMAC: Keyed-Hashing for Message Authentication," [Online]. Available: <https://tools.ietf.org/html/rfc2104>.
- [9] "HIS workgroup homepage, archived by waybackmachine," [Online]. Available: [https://web.archive.org/web/20110824235712/http://portal.automotive-his.de/index.php?option=com\\_content&task=view&id=7&Itemid=18](https://web.archive.org/web/20110824235712/http://portal.automotive-his.de/index.php?option=com_content&task=view&id=7&Itemid=18).

## How to Reach Us:

Home Page: [www.nxp.com](http://www.nxp.com)

Web Support: [www.nxp.com/support](http://www.nxp.com/support)

### USA/Europe or Locations Not Listed:

NXP Semiconductors USA, Inc.

Technical Information Center, EL516

2100 East Elliot Road

Tempe, Arizona 85284

+1-800-521-6274 or +1-480-768-2130

[www.nxp.com/support](http://www.nxp.com/support)

### Europe, Middle East, and Africa:

NXP Semiconductors Germany GmbH

Technical Information Center

Schatzbogen 7

81829 Muenchen, Germany

+44 1296 380 456 (English)

+46 8 52200080 (English)

+49 89 92103 559 (German)

+33 1 69 35 48 48 (French)

[www.nxp.com/support](http://www.nxp.com/support)

### Japan:

NXP Japan Ltd.

Yebisu Garden Place Tower 24F,

4-20-3, Ebisu, Shibuya-ku,

Tokyo 150-6024, Japan

0120 950 032 (Domestic Toll Free)

[www.nxp.com/jp/support/](http://www.nxp.com/jp/support/)

### Asia/Pacific:

NXP Semiconductors Hong Kong Ltd.

Technical Information Center

2 Dai King Street

Tai Po Industrial Estate

Tai Po, N.T., Hong Kong

+800 2666 8080

[support.asia@nxp.com](mailto:support.asia@nxp.com)