

Introduction to STM32 microcontrollers security

Introduction

This application note presents the basics of security in STM32 microcontrollers.

Security in microcontrollers encompass several aspects including protection of firmware intellectual property, protection of private data stored or collected in the device and guarantee of a safe service execution.

The context of IoT has made security even more important. The huge number of connected devices makes them a leading target for attackers and several remote attacks have shown the vulnerabilities of devices through their communication channels. With IoT, the security extends the requirements for confidentiality and authentication to communication channels, through cryptography algorithm deployment.

This document is intended to help building a secure system by applying appropriate countermeasures to different types of attacks.

In a first part, after a quick overview of different types of threats, some examples of typical attacks are presented to show how attackers exploit the different axis of vulnerability in an embedded system.

The next sections focus on the set of hardware and applicative protections that allow thwarting these attacks.

The last sections list all security features available in STM32 series and some guidelines are given to build a secure system.

Table 1. Applicable products

Type	Product Series
Microcontrollers	STM32F0 Series, STM32F1 Series, STM32F2 Series, STM32F3 Series, STM32F4 Series, STM32F7 Series
	STM32G0 Series, STM32G4 Series, STM32H7 Series
	STM32L0 Series, STM32L1 Series, STM32L4 Series, STM32L4+ Series, STM32L5 Series
	STM32WB Series

1 General information

The table below presents a non-exhaustive list of the acronyms used in this document and their definitions.

Table 2. Glossary

Term	Definition
AES	Advanced encryption standard
CCM	Core-coupled memory (SRAM)
CPU	Central processing unit – core of the microcontroller
CSS	Clock security system
DoS	Denial of service (attack)
DPA	Differential power analysis
ECC	Error code correction
FIA	Fault injection attack
FIB	Focused ion beam
GTZC	Global TrustZone® controller
HDP	Secure hide protection
IAP	In-application-programming
IoT	Internet of things
IV	Initialization vector (cryptographic algorithms)
IWDG	Independent watchdog
MAC	Message authentication code
MCU	Microcontroller unit (STM32 Arm® Cortex®-M based devices)
MPCBB	Memory protection block-based controller
MPCWM	Memory protection watermark-based controller
MPU	Memory protection unit
NSC	Non-secure callable
NVM	Non-volatile memory
OTFDEC	On-the-fly decryption
PCROP	Proprietary code readout protection
PKA	Public key algorithm (also named aka asymmetric algorithm)
PVD	Programmable voltage detector
PWR	Power control
ROM	Read only memory – system Flash memory in STM32
RDP	Read protection
RSS	Root secure services
RTC	Real-time clock
SAU	Security attribution unit
SB	Secure boot
SCA	Side channel attack

Term	Definition
SDRAM	Synchronous dynamic random access memory
SFU	Secure firmware update
SPA	Simple power analysis
SRAM	Static random access memory (volatile)
SWD	Serial-wire debug
WRP	Write protection

Documentation references

The **reference manual** of each device gives details on availability of security features and on memory protections implementation.

A **programming manual** is also available for each Arm® Cortex® version and can be used for MPU (memory protection unit) description:

- *STM32L5 Series Cortex®-M33 programming manual* (PM0264)
- *STM32F7 Series and STM32H7 Series Cortex®-M7 processor programming manual* (PM0253)
- *STM32F3 Series, STM32F4 Series, STM32L4 Series and STM32L4+ Series Cortex®-M4 programming manual* (PM0214)
- *STM32F10xxx/20xxx/21xxx/L1xxx Cortex®-M3 programming manual* (PM0056)
- *STM32L0 Series and STM32G0 Series Cortex®-M0+ programming manual* (PM0223)

Refer to the following set of **user manuals** and **application notes** (available on www.st.com) for detailed description of some security features:

- user manual *STM32 crypto library* (UM1924): describes the API of the STM32 crypto library; provided with the X-CUBE-CRYPTOLIB Expansion Package.
- user manual *Getting started with the X-CUBE-SBSFU STM32Cube Expansion Package* (UM2262): presents the SB (secure boot) and SFU (secure firmware update) ST solutions; provided with the X-CUBE-SBSFU Expansion Package.
- application notes *Proprietary Code Read Out Protection on STM32xx microcontrollers* (AN4246, AN4701, AN4758, AN4968): explain how to set up and work with PCROP firmware for the respective STM32L1, F4, L4 and F7 Series; provided with the X-CUBE-PCROP Expansion Package.
- application note *Managing memory protection unit (MPU) in STM32 MCUs* (AN4838): describes how to manage the MPU in the STM32 products.
- application note *STM32WB ST firmware upgrade services* (AN5185)

Note: Arm is a registered trademark of Arm Limited (or its subsidiaries) in the US and/or elsewhere.

2 Overview

2.1 Security purpose

Why protection is needed

Security in microcontrollers is about protecting embedded firmware, data and the system functionality. The need for data protection seems obvious when talking about confidential passwords or personal data.

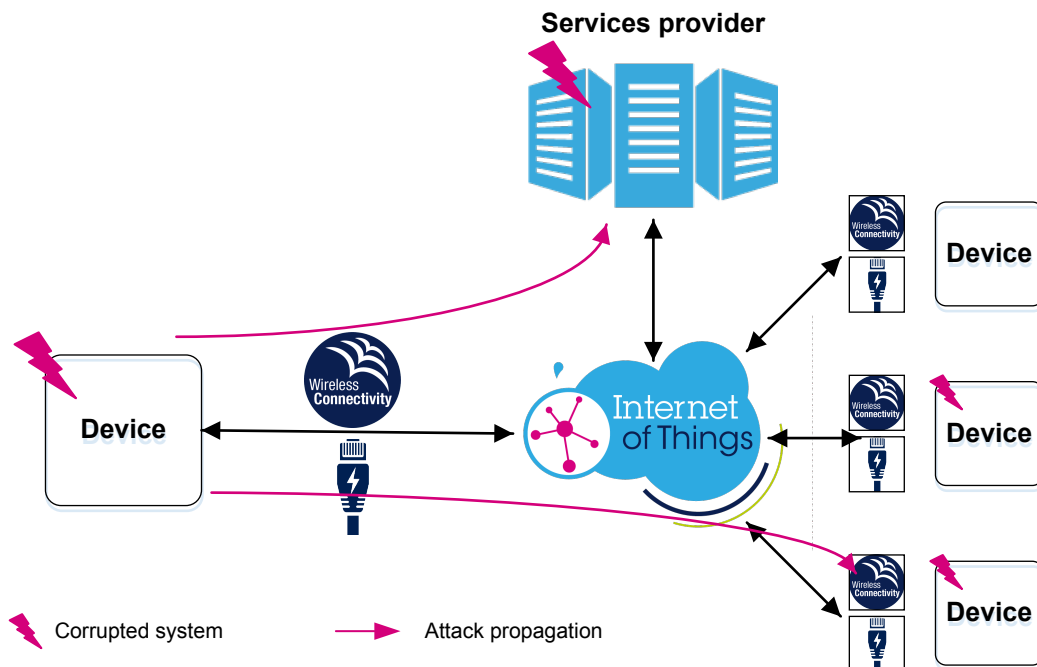
However this is not the only sensitive information. The code is probably where most of the value lies within an embedded application. If an attacker gains access to the binary, he can reverse-engineer the program in an attempt to find further vulnerabilities, bypass licensing and software restrictions. The attacker can copy any custom algorithms or even use it to flash a clone of the hardware, that can then be sold at a reduced price.

Denial-of-service attack (DoS attack) is another major threat when considering protection systems (such as environment: gas, fire or intrusion), detection alarms or surveillance cameras. The system functionality must be robust and reliable.

The requirement for security must not be underestimated even if it adds more complexity to the system. The risk of attacks is real and permanent. Today, the systems built around microcontrollers are everywhere and become as many potential targets for more and more skilled attackers, that expect financial gains from a successful attempt. These gains can be very high, especially if the attack can be propagated to a large scale like in the context of the IoT.

Indeed the IoT, or smart devices, have raised the requirement for security to an even higher level. Connected devices are very attractive for hackers because they are remotely accessible. The connectivity offers an angle of attack through protocol vulnerabilities. In case of a successful attack, a single hacked device can jeopardize the integrity of an entire network (see the figure below).

Figure 1. Corrupted connected device threat



What should be protected

Security is designed for protection of code, data and system functionality. Code protection is set to guarantee the firmware intellectual property and its integrity. Data protections, including cryptographic keys, are needed to guarantee the confidentiality of user data and avoid any identity theft (authentication). This later asset becomes more sensitive with the multiplication of devices used to access paid services (cloud) or any other access-controlled resource. Finally, system functionality itself must be protected to avoid device malfunction or service failure (DoS attack).

The table below presents a non-exhaustive list of assets targeted by attackers.

Table 3. Assets to be protected

Type of attack	Targeted assets	Risks
Data access	Sensor data (such as healthcare data or log of positions) User data (such as ID, PIN, password or accounts) Transactions logs Cryptographic keys	Unauthorized sale of personal data Usurpation Spying Blackmail
Control of device	Device correct functionality Device/user identity	Denial of service Attacks on service providers Fraudulent access to service (cloud)
Reverse engineering	Device hardware architecture/design Software patent/architecture Technology patents	Device counterfeit Software counterfeit Software modification Access to secure areas

Vulnerability, threat and attack

Various means are involved in an unauthorized attempt to access a device, either to steal its content or to take control of it. Whether it is a bug in the firmware, an access let opened (debug) or a weakness in communication protocols, an attacker can exploit any system's vulnerability to succeed. Protection mechanisms have to deal with these different threats. An overview of main attack types are presented in [Section 3 Attack types](#), from the basic ones to the most advanced ones.

The following specific wording is used around security:

- **Asset:** what needs to be protected
- **Threat:** what the device/user need to be protected against
- **Vulnerability:** weakness or gap in a protection mechanism

In summary, an attack is the realization of a threat that exploits a system vulnerability in order to access an asset.

2.2 Building a secure system

In order to counter multiform attacks, several types of security mechanisms must be implemented in a single system and the overall security must be considered at both device (hardware) and application levels.

On one hand, a sensitive application manipulating secrets cannot be trustfully implemented if the device is not protected against external intrusion (such as debug port). On the other hand, a robust device protection is useless if the device is not protected against any potential application bugs or if it allows full device access through its interfaces.

A complete system security implementation involves device protection mechanisms as well as a set of secure applications that depends on system functionality and required robustness (see [Figure 2](#)). The overall system protection level is determined by its least secure node.

Device security features include several memory protection mechanisms, protections against intrusion through external interfaces (including debug), control of boot mode and device environment monitoring. They are detailed in [Section 4 Device protections](#).

Securing applications requires a significant effort in term of firmware development. It requires good software skills and cryptography knowledge. The right tradeoff between development cost and device security requirement must be found.

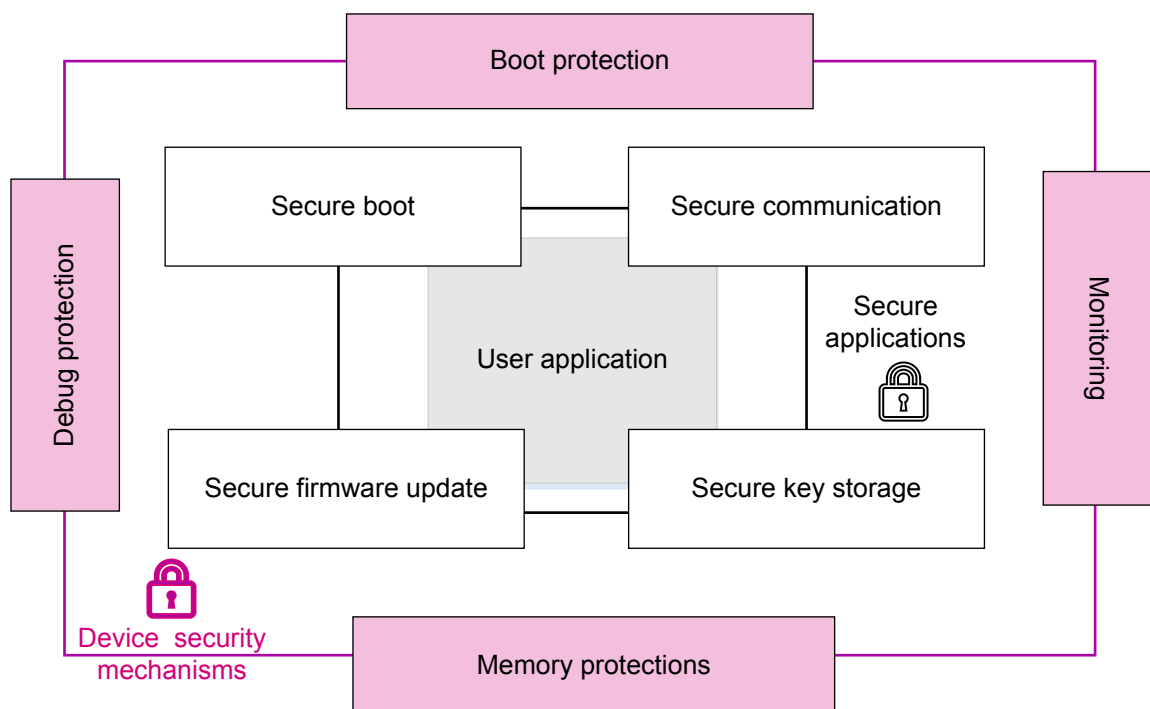
Typical secure applications include the following:

- a secure boot (SB) that guarantees the application integrity and authenticity at startup
- support for secure communication protocols (such as TLS for internet secure communication)
- a secure firmware update (SFU) capability to download a new firmware version in a secure way (authentication/confidentiality/integrity)
- extended capabilities for SKS (secure key storage)

The SBSFU (secure boot and secure firmware update) is detailed in [Section 5 Secure applications](#).

Because most secure applications rely on cryptography tools, the basic concepts are presented in [Appendix A Cryptography - Main concepts](#).

Figure 2. Hardware and application security layers



3 Attack types

This section presents the different types of attack that a microcontroller may have to face, from the most basic ones to attacks worthy a professional engineering. The last part presents typical examples of attacks targeting an IoT system.

Attacks on microcontroller are classified in one of the following types:

- **Software** attack: exploits software vulnerabilities (such as bug or protocol weaknesses).
- **Hardware non-invasive** attack: focuses on MCU interfaces and environment information.
- **Hardware invasive** attack: destructive attack with direct access to silicon

3.1 Introduction

A key rule in security domain is that a successful attack is always possible.

First, there is no absolute protection against unexpected attack. Whatever the security measures taken to protect a system, it is possible that a security breach is found and exploited during the device lifetime. This last point makes necessary to consider how the device firmware is updated all along its lifetime to increase its security (see [Section 5.3 Secure firmware update \(SFU\)](#)).

Secondly, some advanced and expensive techniques exist, deployed by skilled laboratories, to retrieve microcontroller content or even design architecture details. These techniques are briefly presented in [Section 3.3 Hardware attacks](#).

From an attacker point of view, an attack is profitable if the ratio expected revenue/attack cost is as high as possible. The revenue depends on the stolen asset value and on the repeatability of the attack. The cost depends on time and money (equipment) spent to succeed. The security measures intend to raise the cost of an attack, and therefore these measures must be adapted to the asset to be protected.


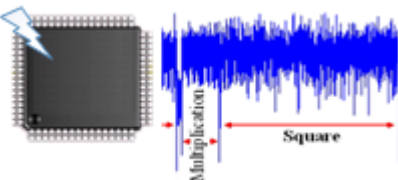
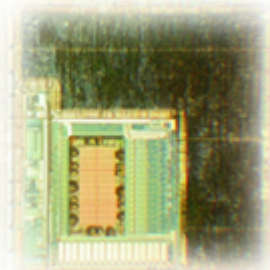
Attack types

The attacks are categorized into two broad categories:

- **Software attacks** are carried by exploiting bugs, protocol weaknesses or untrusted pieces of code for example. Attacks on communication channels (interception or usurpation) can also be considered as part of this category. The software attacks represent the vast majority of cases. Their cost may be very low. They can be widely spread and repeated with huge damage.
- **Hardware attacks** need physical access to the device. The most obvious one exploits the debug port, if it is not protected. But in general, hardware attacks are sophisticated and can be very expensive. They are carried out with some specific materials and require electronics engineering skills. A distinction is made between non-invasive attacks, carried out at board level or chip level without device destruction, and invasive attacks, carried out at device silicon level with package destruction.

The table below gives an overview of the cost and techniques used for each types of attack.

Table 4. Attacks types and costs

Attacks types	Software	Hardware non-invasive	Hardware invasive
			
Scope	Remote or local	Local board and device level	Local device level
Techniques	Software bugs Protocol weaknesses Trojan horse Eavesdropping	Debug port Power Glitches Fault injection Side-channels analysis	Probing Laser FIB Reverse engineering
Cost/expertise	From very low to high, depending on the security failure targeted	Quite low cost. Need only moderately sophisticated equipment and knowledge to implement.	Very expensive. Need dedicated/ heavy equipment and very specific skills.
Objectives	Access to confidential assets (code and data). Usurpation Denial of service	Access to secret data or device internal behavior (algorithm).	Reverse engineering of the device (silicon intellectual property) Access to hidden hardware and software secrets (Flash access)

3.2 Software attacks

Software attacks are carried out on the system by executing a piece of code, named a malware, by the CPU. The malware is intended to take control of the device in order to get access to any resources of the system (such as ID, RAM and Flash memory content or peripheral registers) or to modify its functionality.

This type of attack represents most of device threats for the following reasons:

- The attack cost is low since it does not need specific equipment but a personal computer.
- Many hackers can put their effort together, sharing their expertise and tricks, so that a successful attack is likely to happen if a security breach exists. Furthermore, in case of success, the attack protocol may spread very quickly through the web

The malware can be injected into the device or can already be present (insider threat) in main application firmware through a non-verified or untrusted library for example.

Malwares are of many types and they can be very small and easy to hide.

Here are examples of what a malware can do:

- Access to and modify device configuration (such as option bytes or memory attributes).
- Disable protections.
- Read memory and dump its content for firmware and data cloning.
- Trace or log device data.
- Access to cryptographic items.
- Open communication channel/interface.
- Modify or block the device functionality.

Unless user application is fully trusted, bug-free and isolated, without any means to communicate with external world, software attacks must be considered.

Malware injection

There are various methods for a hacker to inject a piece of code inside the system. The size of the malware depends on the hacker's target but may be very small (few tens of bytes). To be executed, the malware must be injected in the device memory (RAM or Flash memory). Once injected, the challenge is to have it executed by the CPU, which means that the PC (program counter) must branch on it.

Listing malware injection methods consists in listing the various ways of interfacing with the device. These methods can be categorized as follows:

- Basics device access/"open doors":
 - Debug port: JTAG or SWD interface
 - Bootloader: if accessible, can be used to read/write memory content through any available interface.
 - Execution from external memory

These malware injections are easy to counter with simple hardware mechanisms that are described in [Section 4 Device protections](#).

- Application download:
 - Firmware update procedure: a malware can be transferred instead of a new FW.
 - OS with capability to download new applications (and malware)

This second category countermeasure is based on authentication between the device and the server or directly with code authentication. Authentication relies on cryptography algorithms.

- Weaknesses of communication ports and bugs exploitation:
 - Malware injected through the application input data and executed thanks to bugs (such as overflow or lack of data verification)
 - Stack-based buffer overflows, heap-based buffer overflows, jump-to-libc attacks and data-only attacks

This third category is by definition difficult to avoid. Most embedded system applications are coded using low-level languages such as C/C++. These languages are considered unsafe because they can lead to memory management errors leveraged by attackers (such as stack, heap or buffers overflow). **The general idea is to reduce as much as possible what is called the attack surface, by minimizing the untrusted or unverified part of firmware.** One solution consists in isolating the execution and the resources of the different processes. Several solutions are detailed later in this application note.

- Use of untrusted libraries with device back door

This last category is an intentional malware introduction that facilitates device corruption. Today, lot of firmware developments rely on software shared on the web and complex ones can hide Trojan horses. As in previous category, the way to countermeasure this threat is to reduce the surface attack by isolating as much as possible the process execution and protecting the critical code and data.

Brute forcing

This type of attack targets the intrusion of an application protected by a password. A secure device may require a session authentication before accessing services (in the cloud for example) and a human-machine interface (HMI) can be exploited with an automatic process in order to try successive passwords exhaustively.

Some countermeasures are listed below:

- Limit the number of login trials with a monotonic counter (implemented with a timer).
- Increase the delay between two login attempts.
- Add a challenge-response mechanism to break automatic trials.



3.3 Hardware attacks

Hardware attacks require physical access to the device.

A distinction is made between two types of attacks, that differ in cost, time and necessary expertise:

- Non-invasive attacks have only external access to the device (board-level attack) and are not very expensive to carry on.
- Invasive attacks have direct access to device silicon (after de-packing). They are carried out with advanced equipment often found in specialized laboratories. They are very expensive and target very valuable data (Keys or IDs) or even technological patents.

General-purpose microcontrollers are not the best candidates to counter the most advanced physical attacks. If highest protection level is required, it is advised to consider pairing a secure element with the general-purpose microcontroller. Secure elements are dedicated microcontrollers certified as per the latest security standards with specific hardware.

Refer to ST secure microcontrollers web page (www.st.com/en/secure-mcus.html).

3.3.1 Non-invasive attacks

Non-invasive, or board-level, attacks gather techniques that try to retrieve information embedded inside the device (such as code, keys or confidential information) without physical damage (no evidence of the attack). Only accessible interfaces and device environment are used. This last point is important since the owner of the compromised device may even not notice that the secret keys have been stolen. These attacks require moderately sophisticated equipment and engineering skills (such as signal processing).

Debug port access

This is the most basic attack that can be carried out on a device. Disabling debug capability must be the first protection level to consider. Indeed, accessing to debug port or scan chain through JTAG or SWD protocol allows accessing the full internal resources of the device: CPU registers, embedded Flash memory, RAM and peripheral registers.

Countermeasure:

- Debug port deactivation or fuse through [Readout protection \(RDP\)](#)

Serial ports access

Access to communication ports (such as I2C or SPI) may be a weakness that can be exploited. Communication ports can be spied or used as a device entry point. Depending on how the associated protocol are implemented (such as memory address access range, targeted peripherals or read/write operations), an attacker can access to the device resources.

Countermeasures:

- Software:
 - Associated protocol operations must be limited to firmware level, so that no sensitive resources can be read or written.
 - Isolate communication stack from sensitive data.
 - Length of data transfer must be controlled to avoid buffer overflows.
 - Communication can be encrypted with a shared key between the device and the target.
- Hardware:
 - Physical communication port can be buried in multi-layer boards to make it more difficult to access.
 - Unused interface port must be deactivated.

Fault injection: clock and power disturbance/glitch attacks

Fault injection consists in using the device in abnormal environmental conditions to generate malfunctions in the system. A successful attack can lead to serious damages by modifying the program behavior in different ways such as corrupting program state, corrupting memory content, stopping process execution ("stuck-at fault"), skipping instruction, modifying conditional jump or providing unauthorized access.

The typical threats involve tampering with clock (freezing or glitch) and power (under/over voltage or glitch). Since fault injection may be non-intentional, countermeasures are the same as the one used for safety: redundancy, error detection and monitoring.

Countermeasures:

- Software:
 - Redundancy (or re-computation) can be implemented to consolidate conditional jumps in sensitive parts of the code.
- Hardware:
 - Use [Clock security system \(CSS\)](#) if available.
 - Use internal clock sources.
 - Use internal voltage regulators.
 - Use memory error detection (ECC and parity).

Side-channel attacks (SCA)

When a firmware is executed, an attacker can observe the device analog characteristics (such as power consumption, electromagnetic radiations, temperature or activity time). This observation can bring enough information to retrieve secret assets such as data values and/or algorithms implementation. Side-channel based attacks are powerful against cryptographic devices in order to reveal the keys used by the system. SPA (simple power analysis) and DPA (differential power analysis) are typical example of side-channel attack exploiting power consumptions.

Countermeasures:

- Software:
 - Limit key usage: use session random keys when possible.
 - Use protected cryptographic libraries with behavioral randomization (such as delays or fake instructions).
- Hardware:
 - Shields against monitoring can be found in secure elements (STSAFE), but there is no efficient hardware countermeasure embedded in general-purpose microcontrollers.

3.3.2

Silicon invasive attacks

The cost of such attacks is very high; all means are considered to extract information of the device that is destroyed during the process. Carried out with some specific and expensive equipments often found in specialized laboratories, they are within the reach of very qualified engineers and can take hours or days to be successful.

Invasive attacks start with the removal of the device package. Some analysis can be done without eliminating the passivation layer, however investigations with device interaction (probing) require its removal. De-packing can be done by chemical etching, drilling or by a laser cutter. Once the device is opened, it is possible to perform probing or modification attacks.

Some ST microcontrollers dedicated to security offer robustness against such kind of treatments. Refer to ST secure hardware platforms (www.st.com/en/secure-mcus.html).

Reverse engineering

The goal is to understand the inner structure of the device and learn its functionality. This is quite a challenging task with modern devices featuring millions of gates.

The first step is to create a map of the microcontroller. It can be done by using an optical microscope to produce a high-resolution photograph of the device surface. Deeper layers can then be analyzed in a second step, after the metal layers have been stripped off by etching the device.

Micro probing and internal fault injection

Micro probing consists in interacting with the device at metal layer level. Some thin electrodes are used to establish an electrical contact directly with the surface of the device so that the attacker can observe, manipulate, and interfere with it while the device is running.

Device modification

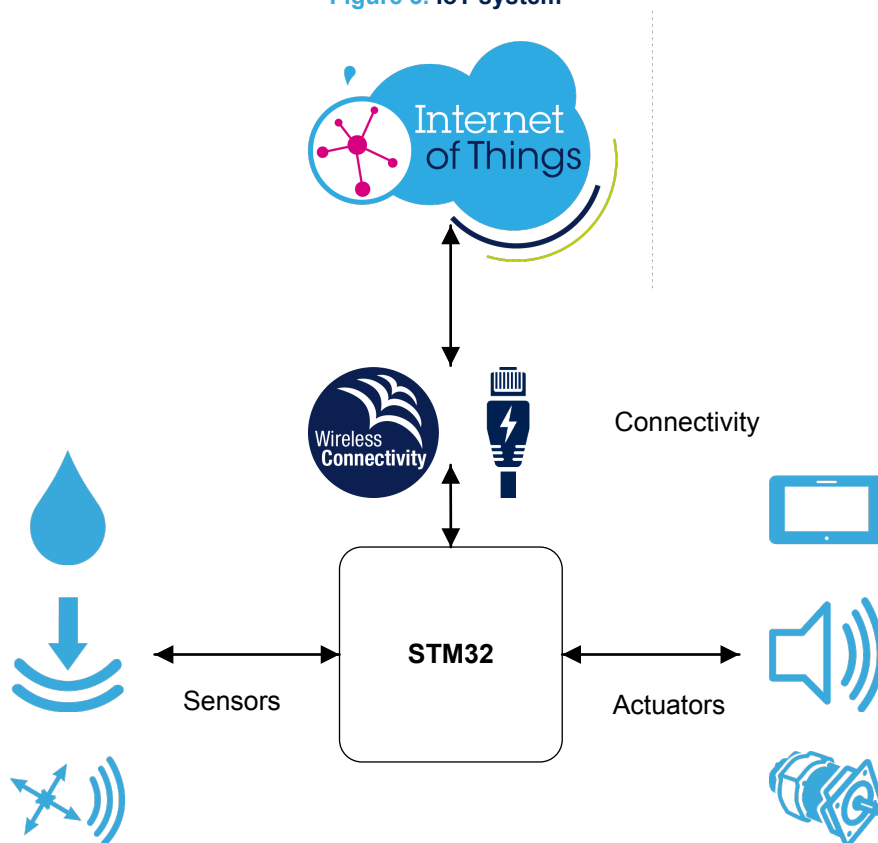
More sophisticated tools can be used to perform attacks. FIB (focused ion beam) workstations, for example, simplify the manual probing of deep metal and polysilicon lines. They also can be used to modify the device structure by creating new interconnection lines and even new transistors.

3.4 IoT system attack examples

This section presents some typical examples of attacks that try to corrupt an IoT system. Fortunately, most of these attacks can be countered by enabling security feature (hardware countermeasures) and secure applicative architecture (software countermeasures). The countermeasures are detailed in the next sections.

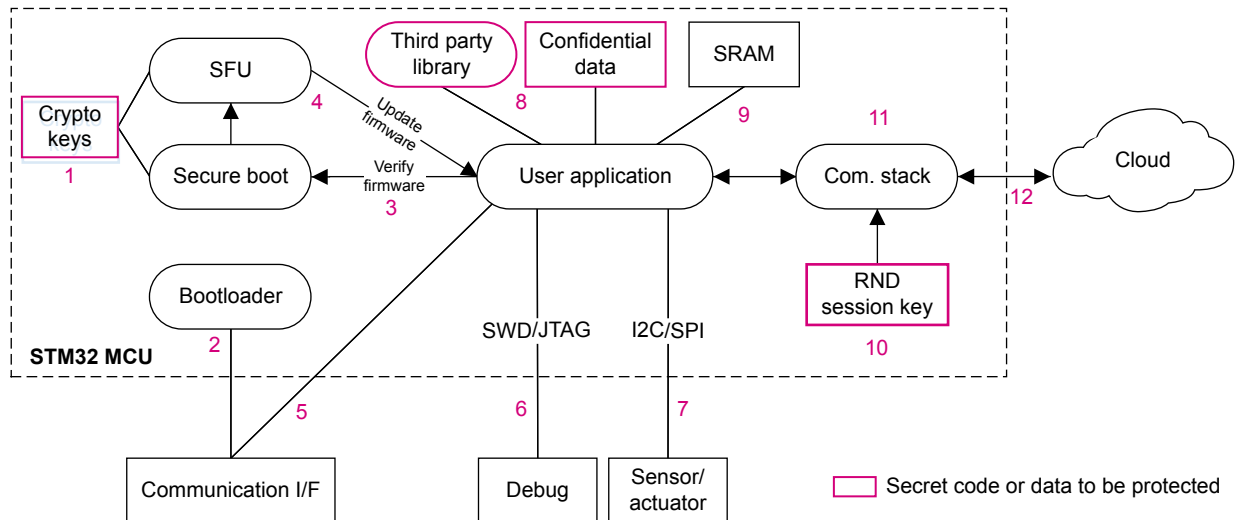
An IoT system is built around a STM32 microcontroller with connectivity systems (such as Ethernet, Wi-Fi®, Bluetooth® Low Energy or LoRa®) and sensors and/or actuators (see the figure below). The microcontroller handles the application, data acquisition and communications with a cloud service. The microcontroller is also responsible for the system maintenance through firmware update and integrity check.

Figure 3. IoT system



The figure below shows the system architecture with software and hardware components. Each of them may have weaknesses leveraged by attackers. The potential damages and a list of countermeasures is given for each example.

Figure 4. IoT system attack examples



For each component, the potential damages and a list of countermeasures are given below.

Table 5. Attacks targets

1	Initial provisioning	7	External peripheral access
2	Boot mode	8	Sensitive firmware and data
3	Secure Boot	9	SRAM
4	Firmware Update	10	Random number generation
5	Communication interfaces	11	Communication stack
6	Debug port	12	Communication eavesdrop

1. Initial provisioning

Some systems rely on primary secrets or sensitive data to build further secure applications (SB or SFU). Secrets can be, for example, cryptographic keys (root of trust) that must be kept private. Other sensitive data may only be kept immutable, such as hash values or certificates. All may be stored in the main Flash memory or in a dedicated secured area. The initial provisioning must be done in a trusted environment or must use STM32 secure services (such as SFI, secure firmware install). Once programmed inside the device, the data protection mechanism must be enabled and only authorized process must have access to it.

- *Risks:* firmware corruption or usurpation
- *Countermeasures:*
 - Trusted manufacturer environment
 - Use of secure data provisioning services (SFI)
 - Data protection mechanisms
 - Secure application isolation

2. Boot modification

The purpose of this attack is to use the bootloader as an input port to access to device content. The attack aims at modifying the boot mode and/or the boot address to preempt the user application and to take control of the CPU through the bootloader (via USB DFU, I2C or SPI), the debug port or through a firmware injected in RAM. The boot mode and the address are controlled by device configuration and/or input pin and must be protected for security reason.

- *Risks:* full access of the microcontroller content
- *Countermeasures:*
 - Unique boot entry
 - Bootloader and debug disabled (see [Readout protection \(RDP\)](#))

3. Secure boot (SB)

Robust systems rely on initial firmware integrity and authenticity check before starting the main application. This is the goal of SB application, executed at device boot. As SB is the root of trust of a device, this part of user firmware must be immutable and impossible to bypass thanks to a set of proper protections.

A successful attack on a SB consists in executing a non-trusted application by bypassing the verification and by jumping directly to the malware. It can be done by hardware techniques such as fault-injection. It can also be done by replacing the expected hash value by the hash value of the malware (see **1. Initial provisioning**).

- *Risks:* device spoofing or application modification
- *Countermeasures:*
 - Unique boot entry point to avoid verification bypassed
 - "Immutable code" to avoid SB code modification
 - Secure storage of firmware signature and/or tag value
 - Environment event detection (such as glitch, temp or clock)

4. Firmware update

The firmware update (or upgrade) procedure allows a product owner to propose corrected version of the firmware to ensure the best user experience during device lifetime. However, a firmware update is a fragile process that gives an attacker a good opportunity to enter the device with its own firmware or a corrupted version of the existing firmware.

The process is secured with firmware authentication and integrity verification. A successful attack requires an access to the cryptographic procedure and keys (see **1. Initial provisioning**).

- *Risk:* device firmware corruption
- *Countermeasure:* SFU application with authentication and integrity checks. Confidentiality can also be added by encrypting the firmware in addition to signature.

5. Communication interfaces

Serial interfaces (such as SPI, I2C or USART) are often used either by the bootloader or by custom applications

to exchange data and/or commands with the device. The physical interception of a communication with access to the device port allows an attacker to use the interface as a device entry point. The firmware protocol can also be prone for bugs (like overflow).

- *Risk:* Access to device content
- *Countermeasures:*
 - Make physical bus hard to reach on board.
 - Isolate software communication stacks to prevent them from accessing critical data and operations.
 - Use cryptography for data exchange.
 - Limit range of interface operations protocol.
 - Disable I/F ports when not needed.

6. Debug port

The debug port provides access to the full content of the device: core and peripherals registers, Flash memory and SRAM content. Used for application development, it may be tempting to keep it alive for investigating future bugs. This is the first breach tried by an attacker with physical access to the device.

- *Risk:* full access to the device
- *Countermeasure:* Disable device debug capabilities (see [Readout protection \(RDP\)](#) feature).

7. External peripheral access

An IoT device controls sensors and actuators depending on the global application. An attacker can divert the system by modifying data coming from sensors or by shunting output data going to actuators.

- *Risk:* prevent correct system behavior.
- *Countermeasure:* anti-tamper to detect system intrusion at board level

8. Sensitive firmware and data

Some parts of the firmware need special protection: for example the most sensitive parts of an algorithm or a third-party library. In addition, some data may need enhanced protection if they are considered as valuable assets (cryptographic keys).

The internal memory content must be protected against external accesses (such as debug ports or communication interfaces) and internal accesses (other software processes). The memory attributes and the firewall are the main protections for process and data isolation.

- *Risks:* sensitive firmware copy or data theft
- *Countermeasures:*
 - Execute-only access right (XO).
 - Firewall
 - Memory protection unit
 - Secure area

9. SRAM

The SRAM is the device running memory. It embeds runtime buffers and variables (such as stack or heap) and can embed firmware and keys. The SRAM may not be as protected as the user Flash memory, and its content at runtime may be difficult to control. For these two reasons, an attacker may be tempted to focus his effort on the SRAM. At least three types of attack can be raised against this memory: code (malware) injection, memory corruption through buffer overflow and retrieval of secrets through temporary stored variables.

- *Risks:* buffer overflow, data theft or device control
- *Countermeasures:*
 - Firewall
 - Memory protection unit
 - Secure area

10. Random number generation

Random numbers are often used in cryptography for session key or initialization vector (IV) generation. Indeed, confidentiality of communication relies on the key robustness and a weak random generator may make a secure protocol vulnerable.

An attack tries to exploit some hidden periodicity or structures of a random sequence to guess the secret key and break into communication confidentiality.

A robust random generator depends on the quality of the entropy source (analog).

- *Risk*: reduced security of cryptographic protocols
- *Countermeasure*: use true hardware entropy generator

11. Communication stack

Connectivity protocols (such as Bluetooth, Ethernet, Wi-Fi or LoRa) have complex communication firmware stacks. These stacks, often available in open source, must not always be considered as best-in-class and trusted. A potential weakness can be massively exploited.

- *Risk*: device access (content, control) through network
- *Countermeasures*:
 - Communication process isolation
 - Server authentication

12. Communication eavesdrop

Data exchanges between a device and an IoT service can be eavesdropped, either directly by a compatible RF device or through the network. An hacker may seek for retrieving data, getting device IDs or accessing services.

Cryptography is the answer adopted by all protocols. Several encryption steps are often considered to protect the communication between all the different layers (device, gateway, applications).

- *Risk*: observation and spoofing of network traffic.
- *Countermeasure*: use cryptography version of communication stack (like TLS for Ethernet)

4 Device protections

Security protections described in this section are controlled by hardware mechanisms. They are set either by configuring the device through option bytes, or dynamically by hardware component settings:

- **Memory protection:** main security feature as it is used to protect code and data from internal (software) and external attacks
- **Software isolation:** inter-processes protection to avoid internal attacks
- **Interface protection:** allows to protect device entry points like serial or debug ports
- **System monitoring:** detects device external tampering attempts or abnormal behaviors

4.1 TrustZone for ARMv8-M architecture

Microcontrollers based on ARMv6 or ARMv7 architecture (Cortex-M0, M3, M4 and M7) rely on specific implementations for firmware and resources isolation. These mechanisms, described later in the document, are robust but not that flexible to allow the concurrent execution of secure firmware together with non-secure firmware.

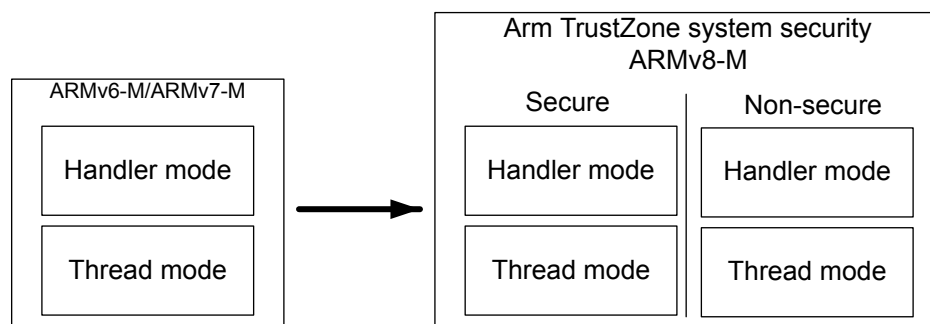
The ARMv8-M architecture brings a new security paradigm in Arm microcontrollers. It implements the TrustZone technology at microcontroller system level, allowing the development of trusted firmware through robust isolation mechanisms at runtime.

The TrustZone technology relies on specific processor (Cortex-M23 or Cortex-M33) with dual registers banks for secure and non-secure domains and on a bus infrastructure (AHB5) propagating secure attribute throughout the whole system (peripherals and memories).

The TrustZone is made for robust and flexible security control at runtime. Switching from secure to non-secure domain and vice-versa is straightforward with few cycle penalty. No need for an hypervisor as in TrustZone for application processors Cortex-A.

Secure modes are orthogonal to the existing modes, Thread and Handler. Thus, there can be a Thread or Handler mode in each secure mode (see the figure below).

Figure 5. ARMv8-M TrustZone execution modes



On typical firmware architecture running on ARMv8 TrustZone, the non-secure domain executes the application and the OS tasks, while the secure domain executes the secure application and the system root-of-trust mechanisms.

4.2 Memory protections

Memory protections are of the highest importance when considering systems security. When containing sensitive code and data, the memories must not be accessible from any unexpected interface (debugging port) or an unauthorized process (internal threat).

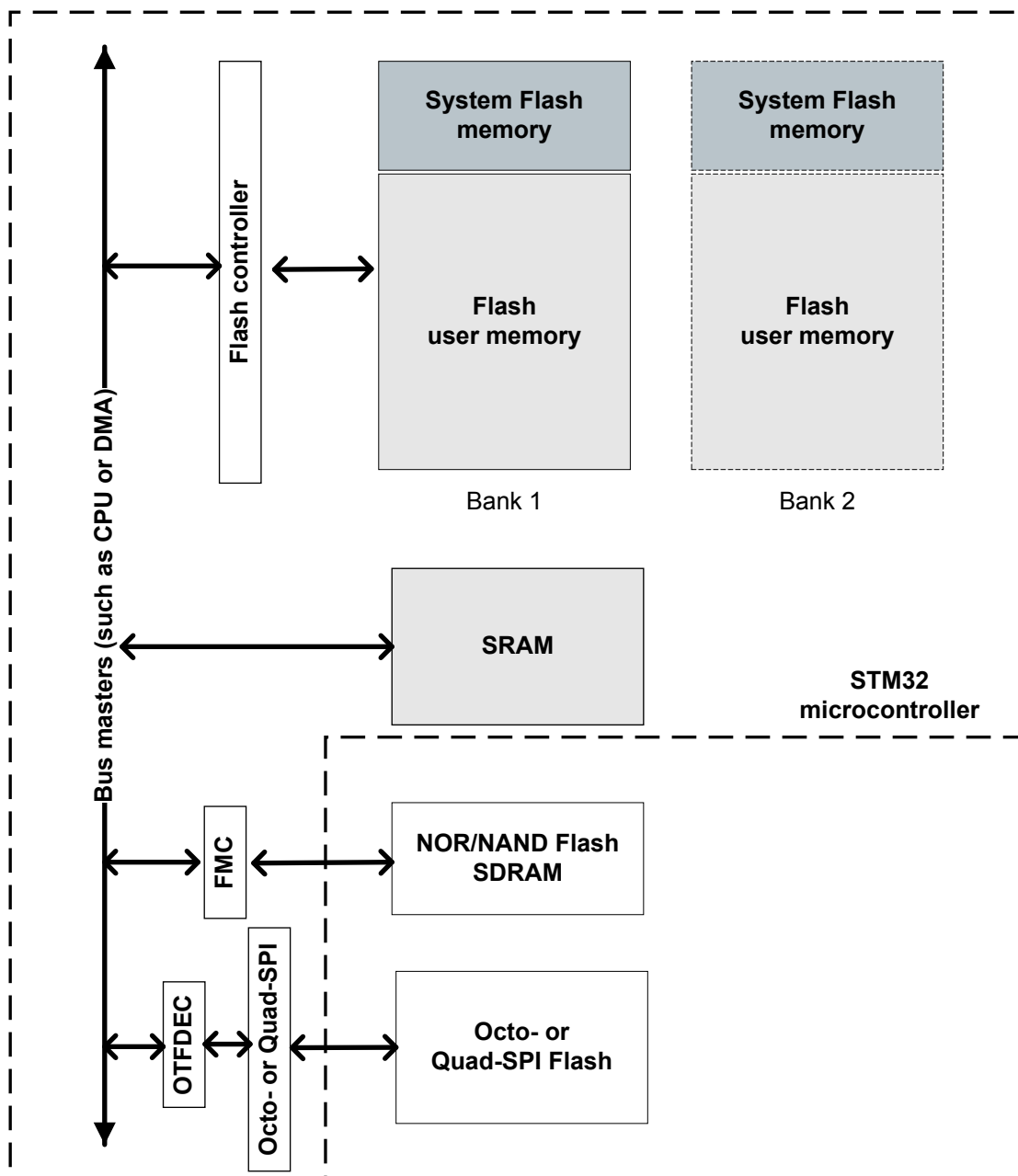
Depending on the asset to be protected (code or data), various mechanisms can be set to establish some protections at the source of the unauthorized access (external port, internal process) or on the memory type to be protected (Flash, SRAM or external memory).

Some access filtering can be performed by the memory interfaces (like Flash controller), the bus controller IP (firewall) or through the core MPU if it is available. Details on proprietary protections (secure hide protection, PCROP, WRP, RDP) can be found in [Section 6 STM32 security features](#).

Embedded Flash memory, embedded SRAM and external memories are used for very different purpose. Their respective protections mechanisms reflect these differences.

The figure below provides a simple view of memories access architecture in a microcontroller.

Figure 6. Memory types



The table below summarizes the particularities of each type of memories and typical protection features.

Table 6. Memory types and associated protection

Memory	Types	Description	Protections
System Flash memory	. Internal . NVM . ROM	ROM part of the Flash memory. Embeds device bootloader and other ST services.	Cannot be updated (erase/written). Some part may also be unreadable.
User Flash memory	. Internal . NVM	Flash memory for user application	Internal protections: <ul style="list-style-type: none"> • RDP • WRP (not for SRAM) • TrustZone • PCROP (not for SRAM) • Firewall • Secure hide protection (not for SRAM) • MPU
SRAM	. Internal . Volatile	Working memory for Stack, heap or buffers. Can be used to execute the firmware downloaded from internal or external non-volatile memories.	
NAND, NOR, Octo- or Quad-SPI Flash memory	. External . NVM	Additional memory for applications or data storage	Cryptography Write protection (on Flash device) TrustZone
SDRAM	. External . Volatile	Additional RAM for application execution	Cryptography

4.2.1 System Flash memory

In STM32 MCUs, the system memory is a read-only part (ROM) of the embedded Flash memory. It is dedicated to the ST bootloader. Some devices may include some secure services (RSS) in this area. This part cannot be modified to guarantee its authenticity and integrity. The bootloader is readable since it does not contain any sensitive algorithm. Some parts of the RSS are hidden and cannot be read by the user.

Protection attribute on the system Flash memory cannot be modified.

4.2.2 User Flash memory

This is the main user memory, used to store firmware and non-volatile data. It is part of the embedded Flash memory and can be protected by a set of memory protection features available on all STM32 MCUs.

External attacks

The embedded Flash memory is easy to protect against external attacks, unlike external Flash memories. Protections against the debugging port access and the controlled access of connectivity interface provide robust isolation from outside.

Associated protection: RDP to disable debug access

Internal attacks

Internal attack must be taken into account too. An internal read or write access to the memory can come from a malware injected either in the device SRAM or already embedded inside an untrusted library, so that the critical code and data must only be accessible by authorized processes.

Associated protections: PCROP, MPU, firewall, secure hide protection or TrustZone

Protecting unused memory

Write protection must always be set by default on the Flash memory, even on unused area, to prevent either code modification or injection. A good practice also is to fill unused memory with known values such as software interrupt (SWI) op-codes, illegal op-codes, or NOPs.

Associated protections: MPU or WRP

Error code correction (ECC)

The Flash memory may feature ECC that allows error detection and correction (up to 2-bits error detection and 1-bit error correction). More considered as a safety feature, it is worth enabling it if available anyway.

4.2.3 Embedded SRAM

The embedded SRAM is the device working memory. It embeds stack, heap, global buffers and variables at runtime. The SRAM can be accessed as bytes, half-words (16 bits) or full words (32 bits), at maximum system clock frequency without wait state.

Code execution

The part of the firmware that requires faster performances can be downloaded from the user or the external Flash memory and executed from the SRAM. Another reason why executing code from SRAM is when using encrypted external Flash memory: the code is decrypted inside the SRAM before its execution. This requires of course that the SRAM (or a part of it) cannot be accessed by other means to guarantee the code confidentiality. When no code must be executed in the SRAM, it is advised to prevent any malware execution by setting the appropriate attribute (execute never) with the MPU.

Associated protections: MPU or firewall

SRAM cleaning

The SRAM may contain sensitive data or temporary values allowing some secrets retrieving. A typical example is the transfer of a secret cryptographic key from protected Flash memory area in clear text, inside the SRAM. It is highly recommended to clean explicitly the working buffers and variables after the processing of functions manipulating sensitive data.

Note: *In case of reset, the STM32 MCUs allow the automatic erase of the SRAM (refer to each device reference manual). For some products, part of the SRAM is protected against external access or untrusted boot (SRAM boot) when the RDP is set.*

Write protection

The write protection can be used to isolate part of the area from being corrupted by another process or by preventing an overflow attack. An overflow attack consists in writing more data than the targeted buffer size (during a data transfer through interface ports for example). If no boundary checks are performed, the memory address above the buffer is corrupted and a malware can be injected this way. The SRAM write protection is available on some STM32 series only (see the reference manual).

Associated protections: MPU, TrustZone or SRAM write protection (available on some STM32 series only)

Parity check

The parity check on the SRAM allows the control of potential errors word-by-word (32 bits). One extra bit per byte is added to the memory content (data bus width is 36 bits) to increase its robustness, as required for instance by Class B or SIL norms.

4.2.4 External Flash memories

The external Flash memories are connected to the microcontroller through dedicated interfaces (NAND, NOR, Octo- or Quad-SPI). As the embedded Flash memory, the external ones contain code and data, but the external storage raises the problem of confidentiality (content protection) and authentication (device protection). The hardware protection is limited to a write lock, to avoid content erasing or modification. Further protection is brought by cryptography algorithms. The content must be at least signed to avoid execution of unauthenticated firmware. Encryption is required only if the content is confidential.

The embedded code can be either executed in-place or loaded into the SRAM before execution. Execution in-place of encrypted firmware is possible only if the device has on-the-fly decryption capabilities. In the other case, the firmware must be decrypted when loaded into SRAM.

Associated protection: OTFDEC

4.2.5 STM32 memory protections overview

Several STM32 features are available to cover the various cases considered. They are listed in the table below with their respective scope and described in [Section 6 STM32 security features](#).

Table 7. Scope of STM32 embedded memories protection features

Feature	External attack protection	Internal attack protection	Flash memory	SRAM
RDP	Yes	No	Yes	Yes
Firewall	No	Yes	Yes	Yes
MPU	No	Yes	Yes	Yes
PCROP	Yes	Yes (read/write)	Yes	No
WRP ⁽¹⁾	Yes	Yes	Yes	No
Secure hide protection	Yes	Yes	Yes	Yes (for execution) ⁽²⁾
TrustZone	Yes	Yes	Yes	Yes

1. Write protection can be unset when RDP level \neq 2.

2. the SRAM is protected by a secure area only at secure code execution. It must be cleaned before leaving the secure area.

4.3 Software isolation

The software isolation refers to a runtime mechanism protecting different processes from each other (inter-process protection). These processes can be executed sequentially or concurrently (for example tasks of operating system). The software isolation insures that respective stack and working data of each process cannot be accessed by the other processes. This inter-process protection can be extended to the code and non-volatile data as well.

Goals of the software isolation:

- Prevent a process to spy the execution of another sensitive process in order to retrieve data like cryptographic key or part of its code.
- Protect a process execution against a stack corruption due to memory leaks or overflow (incorrect memory management implementation).

This memory protection is achieved thanks to different mechanisms listed in the table below and detailed in [Section 6 STM32 security features](#).

Table 8. Software isolation mechanism

Protection	Type	Isolation
MPU	Dynamic	By privilege attribute ⁽¹⁾
Firewall	Static	By bus address hardware control
Secure hide protection	Static	Process preemption at reset
Dual core	Static	By core ID
TrustZone	Static and dynamic	By secure attribute propagated from the core to all resources

1. The attribute protection is only for CPU access and is not taken into account for other bus master (such as DMA).

4.4 Debug port and other interfaces protection

The debug ports provide access to the internal resources (core, memories and registers) and must be disabled. It is the most basic external attack that is easily avoided by deactivating JTAG (or SWD) ports by a secure and immutable firmware (refer to [Section 5.2 Secure boot \(SB\)](#)), or by permanently disabling the functionality (JTAG fuse in RDP2).

Other serial interfaces can also be used. If the bootloader is available, the device content can be accessed through I2C, SPI, USART or USB-DFU. If the interface is open during the runtime, the applicative transfer protocol must limit its access capabilities (such operation mode or address access range).

Associated STM32 features:

- Read protection (RDP)

- Disable of unused ports.
- Forbid bootloader access (configured by RDP in STM32 devices).

4.5 Boot protection

The boot protection is about securing the very first software instructions in a system. If an attacker succeeds in modifying the device boot address, he may be able to execute his own code, to bypass initial dynamic protections configuration or to access unsecured bootloader applications that give access to the device memory.

A microcontroller usually allows the boot configuration in order to choose between starting at user application, at bootloader application or at the SRAM located firmware. The boot protection relies on a single entry point to a trusted code that can be the user application or a secure service area if available (RSS).

Associated STM32 features:

- Read protection (RDP)
- Unique boot entry
- Secure hide protection (HDP)
- TrustZone

4.6 System monitoring

The monitoring of the device power supply and environment can be set to avoid malfunction and to take corresponding countermeasures. Some mechanisms, like tamper detection, are dedicated to security. Some other mechanisms are primarily used for safety reason but can serve security as well. For example, the detection of a power down or external clock disconnection may be unintentional (safety) but may also reveal an attack (security).

Tamper detection is used to detect system/board level intrusions. The opening of a consumer product enclosure can be detected on an MCU pin and trigger appropriate actions.

Clock security system is used to protect against external oscillator failures. If a failure is detected on the external clock, the oscillator is automatically disabled and the microcontroller switches to the internal clock in order to safely execute the rescue operations.

Power supply and voltage level can be monitored to detect abnormally-low voltage level. Below a certain voltage value, the normal behavior cannot be ensured.

Device temperature can be measured with an internal sensor. The information is feed backed to the device through an internal ADC channel. A monitoring application can take appropriate actions according to the temperature range.

Associated STM32 features:

- Tamper protection (with RTC component)
- Clock security system
- Power supply supervision
- Temperature sensor

5 Secure applications

Secure applications are the other aspect of secure systems, complementary to device hardware protections introduced previously. Secure applications participate in keeping a high level of security all along the device life.

This section defines the **root and chain of trust** concept before presenting the two following typical secure applications:

- **Secure boot (SB):** built to check device state, the SB sets runtime protections and authenticates the firmware before the application execution.
- **Secure firmware update (SFU):** built to allow authenticated firmware update

These applications have a close link with cryptography. All cryptographic schemes are based on the three concepts of secret key, public key and hashing. Basics of cryptography are explained in [Appendix A Cryptography - Main concepts](#).

Note: The user manual [Getting started with the X-CUBE-SBSFU STM32Cube Expansion Package \(UM2262\)](#) provides an implementation example of SB and SFU (www.st.com/en/product/x-cube-sbsfu).

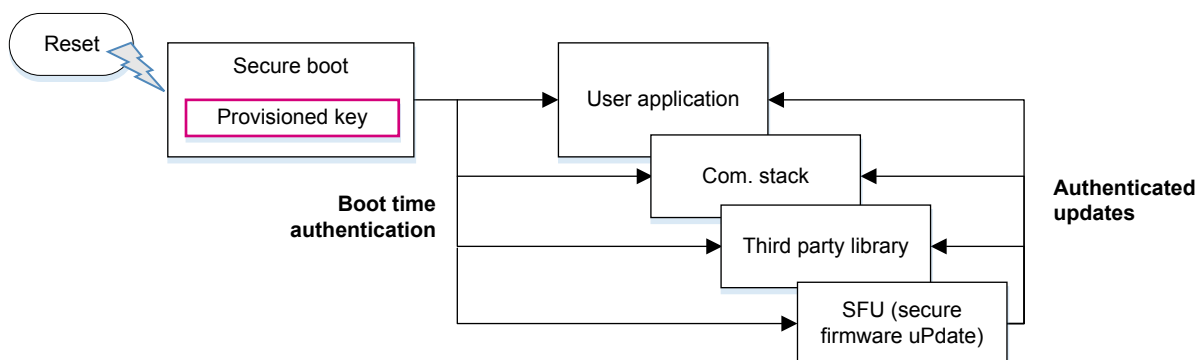
5.1 Root and chain of trust

A chain of trust is built as a set of applicative components in which the security of each component is guaranteed by another component. The root of trust is the very first step of the chain on which everything counts.

In the context of embedded system, the applicative root of trust is the SB application. Its goal is to verify each embedded firmware signature for authentication.

Successive part of the chain of trust may be composed by a SFU that guarantees, in turn, the authenticity of future applications (see the figure below).

Figure 7. Chain of trust



The whole system security depends on all its constituting applicative elements. A single weak element may corrupt the entire system by being the target of an attack.

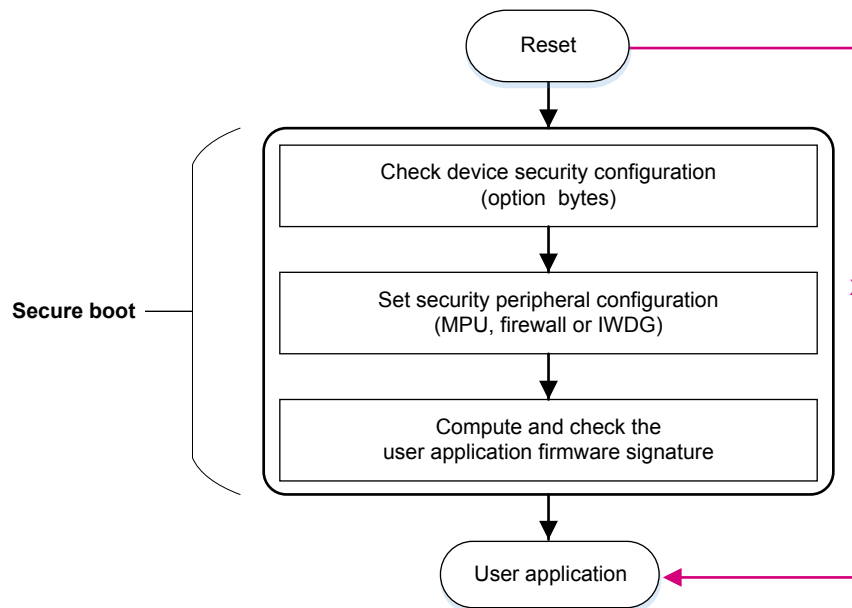
5.2 Secure boot (SB)

The SB application is executed at reset before the user application. It provides first stages of security and is then responsible for ensuring the global chain of trust of the system.

SB main functionalities:

- Check the STM32 security configuration and set up runtime protections.
- Assert the integrity and authenticity of the user application images that are executed (see the figure below).

Figure 8. Secure boot FSM



Checking device security

This part of the SB application checks if static configurations are correct and sets the dynamic ones. Static secure configurations are defined by option bytes (RDP, PCROP, WRP and secure hide protection). Dynamic protections must be programmed (firewall, MPU, tamper detection and IWDG).

Integrity and authenticity check

The firmware integrity is performed by hashing the application image (with MD5, SHA1 or SHA256 hash algorithms) and comparing the digest with the expected one. This way, the application firmware is considered error-free.

An authenticity check is added if the expected tag is encrypted with a key shared between the firmware owner and the device. This key is stored in a protected area of the device.

Protection attributes

The SB firmware must have the following attributes to fulfill its role:

- It must be the device-unique entry point (no bypass).
- Its code must be immutable.
- It must have access to sensitive data (such as certificates or application signatures).

The most sensitive SB part takes benefit from process and data isolation features, like firewall, MPU or secure hide protection. the implementation depends on STM32 series available features.

5.3 Secure firmware update (SFU)

The SFU provides a secure implementation of in-field firmware updates, enabling the download of new firmware images to a device in a secure way.

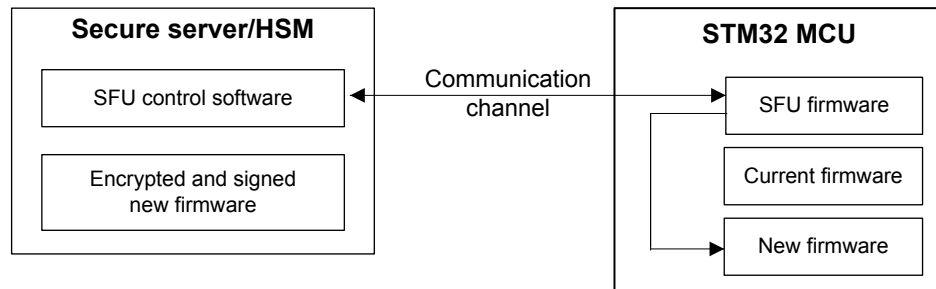
The firmware update is a sensitive operation that must protect two parties:

- the device owner: the goal is to avoid loading a corrupted firmware (intentionally or not) that may definitely damage the device.
- the application owner (OEM): needs to protect his firmware from being cloned or loaded into an unauthorized device.

Architecture

An SFU transfer involves two entities: the firmware owner (OEM) and the device to be updated (see the figure below). As the communication channel is generally considered as non-secure since it is subject to eavesdropping, the overall security responsibility is shared between the sender (firmware owner server) and the receiver (the device).

Figure 9. Secure server/device SFU architecture



Application

From OEM side, a secure server is maintained that is responsible for sending the encrypted (if confidentiality is required) and signed firmware to an authenticated device.

The SFU application running on device is in charge of the following:

- authentication and integrity checking of the loaded image before installing it
- decrypting the new firmware if confidentiality is required
- checking the new firmware version (anti-rollback mechanism)

Protection attributes

As the SB application, the SFU must have the following attributes:

- Its code must be immutable.
- It must have access to secrete (private or shared key) or/and sensitive (public key) data.
- It must be executed in a safe way (isolated) to avoid inter-process interactions.

6 STM32 security features

This section presents all the STM32 features that can be gathered to meet the different security concepts presented in previous sections and to achieve a high level of security.

6.1 Overview

Static and dynamic protections

A distinction can be made depending on whether protection features are static or dynamic:

- **Static** protections refer to features that are set with option bytes. Their configuration is retained at power off. Static protections are RDP, PCROP, WRP, BOR and secure hide protection (when available). All these features are set through permanent device configuration with option bytes.
- **Dynamic** (or run time) protections do not retain their status at reset. They have to be configured at each boot (for example during first stage of [Secure boot \(SB\)](#)).
Dynamic protections provided by STM32 are MPU, tamper detection and firewall.
Other dynamic protections are related to both security and safety. An abnormal environment behavior may be accidental (safety) or intentional, in order to carry out an attack. These protections include clock and power monitoring systems, memory integrity bits and independent watchdog (IWDG).

Security features by STM32 series

The tables below list the available features according to STM32 series.

Table 9. Security features for STM32Fx Series

Feature	STM32F0	STM32F1	STM32F2	STM32F3	STM32F4	STM32F7
Cortex core	M0	M3	M3	M4	M4	M7
RDP additional protection	Backup registers	N/A ⁽¹⁾	+ backup SRAM	+ backup registers	+ backup SRAM	+ backup SRAM
WRP	By sectors (4 Kbytes)	By pages (4 K or 8 Kbytes)	By sectors (16 K, 64 K or 128 Kbytes)	By sectors (4 Kbytes)	By sectors (16 K, 64 K or 128 Kbytes)	By sectors (16 K, 64 K, 128 K or 256 Kbytes)
PCROP	No	No	No	No	By sectors	By sectors
HDP	No	No	No	No	No	No
Firewall	No	No	No	No	No	No
MPU	No	yes	Yes	Yes	Yes	Yes
Unique boot entry	No	No	No	No	No	No
Tamper detection	Yes	Yes	Yes	Yes	Yes	Yes
IWDG	Yes	Yes	Yes	Yes	Yes	Yes
Device ID (96 bits)	Yes	Yes	Yes	Yes	Yes	Yes
Hardware crypto	No	No	AES, HASH, TRNG	No	AES, HASH, TRNG	AES, HASH, TRNG

1. The RDP in STM32F1 Series is only for Flash memory protection. RDP is set (RDP1) or unset (RDP0). RDP Level 2 is not implemented.

Table 10. Security features for STM32Lx Series

Feature	STM32L0	STM32L1	STM32L4 STM32L4+	STM32L5
Cortex core	M0	M3	M4	M33 with TrustZone
RDP additional protection	+ EEPROM	+ EEPROM	+ backup registers + SRAM2	RDP four levels + backup registers + SRAM2
WRP	By sectors (4 Kbytes)	By sectors (4 Kbytes)	By area with 2-Kbyte granularity One area per bank	Up to four protected areas with 2-K or 4-Kbyte granularity
PCROP	By sectors	By sectors	By area with 8-byte granularity One area per bank	No
HDP⁽¹⁾	No	No	No	Up to two secure hide areas (HDP) inside the TrustZone secure domain
Firewall	Yes	No	Yes	No
MPU	Yes	Yes	Yes	Yes
Unique boot entry	No	No	No	Yes
Tamper detection	Yes	Yes	Yes	Yes
IWDG	Yes	Yes	Yes	Yes
Device ID (96 bits)	Yes	Yes	Yes	Yes
Hardware crypto	AES	AES	AES, HASH, TRNG	AES, OTFDEC, HASH, PKA, TRNG

1. HDP is known as secure user memory, sticky area or securable memory depending on the product.

Table 11. Security features for STM32H7, STM32G0, STM32G4 and STM32WB Series

Feature	STM32H7	STM32G0	STM32G4	STM32WB
Cortex core	M7	M0+	M4	M4 and M0+
RDP additional protection ⁽¹⁾	+ backup SRAM	+ backup registers	+ backup registers + CCM-SRAM ⁽²⁾	+ backup registers + SRAM2
WRP	By sectors (128 Kbytes)	By area with 2-Kbyte granularity Two areas available	By page (2 K or 4 Kbytes)	By page (4 Kbytes)
PCROP	By area with 256-byte granularity One area per bank	By area with 512-byte granularity Two areas available	By area with 64-bit or 128-bit granularity Up to two areas ⁽³⁾	By area with 2-Kbyte granularity Up to two areas
HDP ⁽⁴⁾	Yes (secure user memory)	Yes (securable memory area)	Yes (securable memory area)	Yes (dedicated to CM0+ firmware only)
Firewall	No	No	No	No
MPU	Yes	Yes	Yes	Yes (CM4)
Unique boot entry	Yes	Yes (boot lock feature)	Yes	No
Tamper detection	Yes	Yes	Yes	Yes
IWDG	Yes	Yes	Yes	Yes
Device ID (96 bits)	Yes	Yes	Yes	Yes
Hardware crypto ⁽¹⁾	AES, HASH, TRNG	AES, TRNG	AES, TRNG	AES, PKA, TRNG

1. Depends on device part number.

2. CCM: core-coupled memory

3. The number of area depends on device category and dual/single bank configuration.

4. HDP is known as secure user memory, sticky area or securable memory depending on the product.

6.2 Readout protection (RDP)

Readout protection is a global Flash memory protection allowing the embedded firmware code to be protected against copy, reverse engineering, dumping, using debug tools or code injection in SRAM. The user must set this protection after the binary code is loaded to the embedded Flash memory.

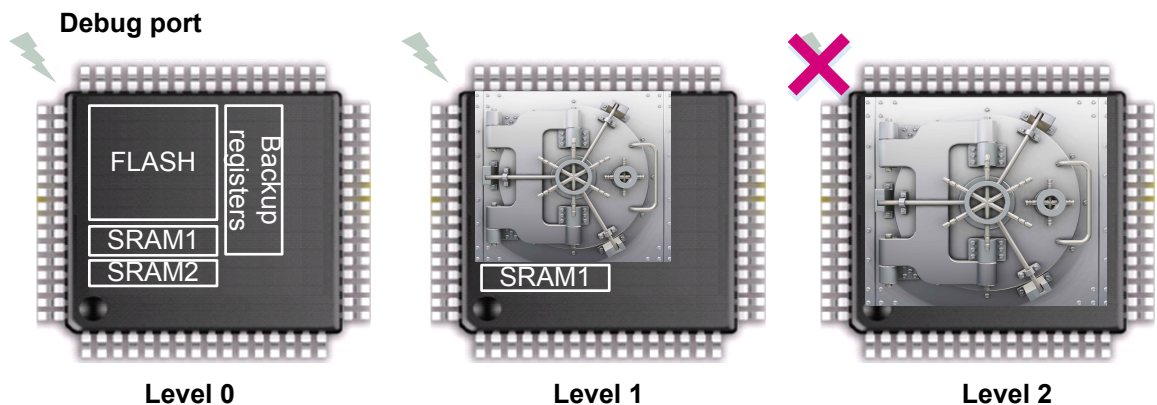
Readout protection applies to all STM32 series for:

- the main Flash memory
- the option bytes (level 2 only)

Depending on the STM32 series, additional protections may be available including:

- backup registers for real-time clock (RTC)
- backup SRAM
- EEPROM

Figure 10. Example of RDP protections (STM32L4 Series)



The RDP levels (0, 1 and 2) are defined as follows:

- **Level 0:** this is the default RDP level. The Flash memory is fully open and all memory operations are possible in all boot configurations (debug features, boot from RAM, boot from system memory bootloader, boot from Flash memory). There is no protection in this configuration mode that is appropriate only for development and debug.
- **Level 1:** this level corresponds to the activation of the read protection Level 1. Flash memory accesses (read, erase, program) or SRAM2 accesses via debug features (such as serial-wire or JTAG) are forbidden, even while booting from SRAM or system memory bootloader. In these cases, any read request to the protected region generates a bus error.
However, when booting from Flash memory, accesses to both Flash memory and to SRAM2 (from user code) are allowed.
- **Level 2:** When RDP Level 2 is activated, all protections provided in Level 1 are active and the MCU is fully protected. The RDP option byte and all other option bytes are frozen and can no longer be modified. The JTAG, SWV (single-wire viewer), ETM, and boundary scan are all disabled.

On devices built on TrustZone architecture, a fourth RDP level is available:

- **Level 0.5:** non-secure debug only. All read and write operations (if no write protection is set) from/to the non-secure Flash memory are possible. The debug access to secure area is prohibited. Debug access to non-secure area remains possible.

RDP level regression

RDP can always be leveled up. A level regression is possible with the following consequences:

- Regression from RDP Level 1 to RDP Level 0 leads to a Flash memory mass erase and the erase of SRAM2 and backup registers.
- Regression from RDP Level 1 to RDP Level 0.5 leads to a partial Flash memory erase: only the non-secure part is erased.
- Regression from RDP Level 0.5 to RDP Level 0 leads to a Flash memory mass erase and the erase of SRAM2 and backup registers.

In RDP level 2, no regression is possible.

Internal Flash memory content updating on an RDP protected STM32 microcontroller

In RDP Level 1 or 2, the Flash memory content can no longer be modified with an external access (bootloader or booting from SRAM). However, modifications by an internal application are always possible. This can be performed through a SFU application or (even if it is not advised from a security point of view) from a simple in-application-programming process (IAP).

The table below summarizes the RDP protections.

Table 12. RDP protections

Area	RDP Level	Boot from user Flash			Debug or boot from SRAM or from bootloader		
		Read	Write	Erase	Read	Write	Erase
Flash main memory	0	Yes	Yes	Yes	Yes	Yes	Yes
	1	Yes	Yes	Yes	No	No	No
	2	Yes	Yes	Yes	N/A	N/A	N/A
System memory	0	Yes	No	No	Yes	No	No
	1	Yes	No	No	No	No	No
	2	Yes	No	No	N/A	N/A	N/A
Option bytes	0	Yes	Yes	Yes	Yes	Yes	Yes
	1	Yes	Yes	Yes	Yes	Yes	Yes
	2	Yes	No	No	N/A	N/A	N/A
Other protected assets ⁽¹⁾	0	Yes	Yes	Yes	Yes	Yes	Yes
	1	Yes	Yes	N/A	No	No	No
	2	Yes	Yes	N/A	N/A	N/A	N/A

1. Backup registers/SRAM

When should RDP be used

On a consumer product, RDP must always be set at least at Level 1. This prevents basic attacks through the debug port or through the bootloader. However, in RDP Level 1, there is a risk of service denial caused by a Flash memory mass erase, following a return to RDP Level 0.

The RDP Level 2 is preferred and is even mandatory to implement a SB and SFU application with correct security level (such as immutable code or JTAG debugger attach on reset). The counterpart is that the RDP Level 2 can prevent a device configuration update, for instance after a customer return.

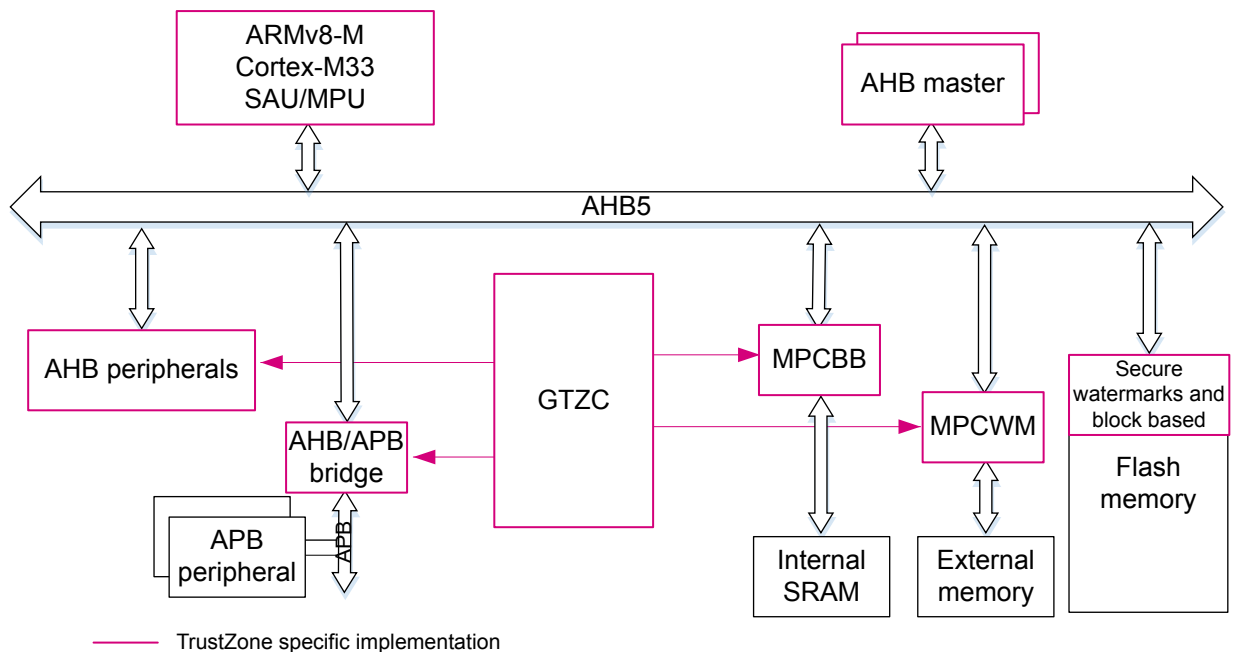
Note: RDP is available on all STM32 series.

6.3 TrustZone

The ARMV8-M TrustZone architecture defines two domains at system level: secure and non-secure. The full memory-map space is split into secure and non-secure areas. This includes all memory types (Flash, SRAM and external memories), as long as all peripherals that can be shared (with specific context for each domain) or dedicated to one domain or the other.

At system level, the isolation between secure and non-secure domains relies on the following hardware mechanisms (see the figure below).

- specific core architecture (ARMV8-M Cortex-M33) with a dual-bank register for secure and non-secure domains
- hardware unit responsible for the attribution of secure and privilege attribute (SAU)
- bus infrastructure that propagates the secure and privilege attributes of any transaction (AHB5)
- some dedicated hardware blocks managing the split between the two domains (GTZC to define security attribute for memories and peripherals/Flash interface)

Figure 11. TrustZone implementation at system level


6.3.1 Core state

The core state depends on the region of the current running code. When the code runs from a secure region, the core is in secure state. Otherwise, the core is in non-secure state.

6.3.2 Secure attribution unit (SAU)

The SAU is an hardware unit coupled to the core (as the MPU). The SAU is responsible for setting the secure attribute of the AHB5 transaction. The security attribute of a transaction is fixed by the targeted address of a memory-mapped resource (memory areas or peripherals). Depending on the SAU configuration, an address transaction is tagged as NSC (secure, non-secure callable) or non-secure. The NSC is a sub-domain of the secure domain, that allows a gateway to be defined for non-secure code to access the secure domain at a specific entry point.

The SAU is configurable by firmware. It can be configured at boot for a fixed configuration or can be dynamically modified by a secure firmware.

Note:

A security attribute cannot be less secure (by security order: secure > NSC > non-secure) than a default attribute set by hardware through an IDAU (implementation defined secure attribute). Refer to implementation details of each device in reference manual.

Address aliasing

From one hand, the security attribute is set depending on the resource address, that is fixed. On the other hand, a memory-mapped resource can be set either as secure or non-secure, depending on the application. To overcome this apparent contradiction, two addresses are given to each memory-mapped resources: one used when the resource must be accessed in secure mode, one used in non-secure mode. This mechanism is called address aliasing.

The address aliasing allows also all peripherals access to be grouped in only two regions instead of multiple scattered regions. Finally, the SAU splits the memory-mapped resources in the following regions:

- Peripherals secure/non-secure regions
- Flash memory secure/non-secure regions
- SRAM secure/non-secure regions
- External memory secure/non-secure regions

Refer to device reference manual for the detailed configuration.

6.3.3 Memories and peripherals protections

The SAU defines the transaction security attribute and the bus infrastructure propagates this attribute towards the targets. The targets (memories and peripherals) are protected by hardware mechanisms that filter the access depending on the secure and privilege attributes.

There are two types of peripherals in the TrustZone system architecture:

- **TrustZone-aware** peripherals: connected directly to the AHB or APB bus, with a specific TrustZone behavior such as a subset of secure registers. The access filtering control is included in these peripherals
- **Securable** peripherals: protected by an AHB/APB firewall gate controlled by the GTZC to define security properties

TrustZone-aware peripherals are the ones with a bus master role (DMAs), the GTZC, the Flash memory controller and others peripherals with a fundamental role within the system: PWR, RTC and system configuration block. The remaining system peripherals are securable.

GTZC

The GTZC defines the secure and privilege access state of securable peripherals, embedded SRAM and external memories:

- Peripherals can be set as secure or non-secure (exclusively), privileged or non-privileged.
- Embedded SRAM is protected by blocks of 256 bytes through the MPCBB block.
- External memories are protected by regions (watermark: start @ and length). The number of protected regions depends on the memory types (NAND, NOR or OCTOSPI).

Note: The Flash memory, that is TrustZone-aware, has its own watermark registers, independent from the GTZC.

6.3.4 Summary of transaction types

The table below summarizes the different access types in the system, depending on the core state, the transaction attribute and the resource (memory or peripheral) configuration.

Table 13. Transaction types

Core state	Transaction attribute (SAU)	Resource configuration	Access granted
Secure	Secure/NSC	Secure	Yes
		Non-secure	No
	Non-secure	Secure	No
		Non-secure	Yes
Non-secure	Secure/NSC	Secure	No
		Non-secure	No
	Non-secure	Secure	No
		Non-secure	Yes

When should TrustZone be used

The TrustZone is used to allow flexible and robust firmware and peripheral isolation at runtime. In addition of typical secure applications (SB or SFU), the TrustZone can offer access to runtime secure services (such as key storage).

Note: The TrustZone is available on STM32L5x Series only and is enabled when the TZEN option bit is set.

6.4 Flash memory write protection (WRP)

The write protection feature is used to protect the content of specified memory area against erase or update. For Flash memory technology, an update must be considered as filling with zeros.

For instance, the write protection can be set on a page or a sector of a Flash memory to prevent its alteration during a firmware or data update. It can also be set by default on the unused memory area to prevent any malware injection. Its granularity is linked to the page or sector size.

When should WRP be used

This protection must be used, in particular when write operations are foreseen within the application. This is the case if data storage or code update operations are expected. WRP prevents wrong accesses due to unsafe functions causing unexpected overflows.

Note: Write protection is available on all STM32 series.

6.5 Execute-only firmware (PCROP)

Part of the STM32 Flash memory can be configured with an 'execute-only' attribute. The firmware stored in such configured area can only be fetched by the CPU instruction bus. Any attempt to read or write this area is forbidden. The protection applies against both internal (firmware) accesses as well as external (debug port) accesses. In STM32, this feature is named proprietary code readout protection (PCROP).

The PCROP is a static protection set by option bytes. The number of protected areas and their granularity depends on the STM32 series (see Table 9, Table 10 and Table 11). When PCROP is in use, care must be taken to compile the firmware with the execute-only attribute (refer to user compiler options).

When should PCROP be used

PCROP is used to protect third-party firmware (intellectual property) as well as the most sensitive parts of the user firmware.

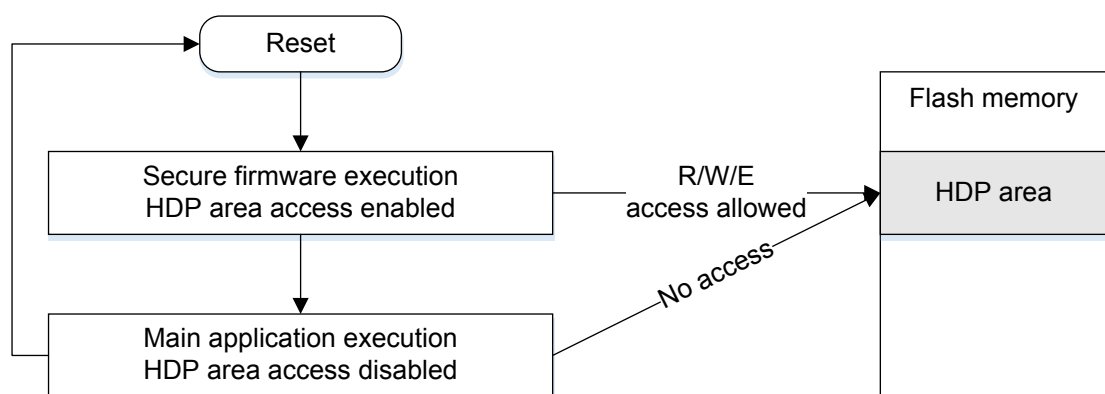
Note: PCROP is available on all STM32 Series listed in Table 1.

6.6 Secure hide protection (HDP)

Some STM32 devices supports the HDP memory concept. The HDP, named secure hide protection on STM32L5 Series, is also known as secure user memory on STM32H7 Series, or securable memory on STM32G0 Series.

An HDP area is a part of the Flash memory that can be accessed only once, just after a device reset. The HDP targets sensitive applications that embed or manipulate confidential data and that must be securely executed at boot. Once the application is executed, the HDP area is closed and cannot be accessed anymore by any mean (see the figure below).

Figure 12. HDP protected firmware access



The HDP is a static protection configured by option bytes. Once set, the CPU boots on the firmware embedded in this area, whatever the boot configuration set by boot pin or boot address.

Thanks to this protection, the user can provide secure services to the device at boot time. The user SB and SFU applications are typical examples that must reside in this secure area.

When should HDP be used

The HDP is suited for secure applications that must only be executed after reset, like SB or SFU.

This protection is available in STM32H7, STM32G0, STM32G4 and STM32L5 Series with slight differences in its implementation and name (refer to dedicated reference manuals for details).

6.7 Firewall

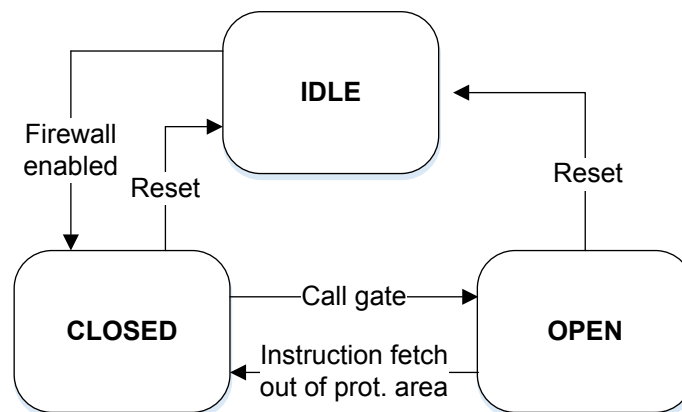
The firewall is a hardware protection peripheral controlling the bus transactions and filtering accesses to three particular area: a code area (Flash memory), a volatile data area (SRAM) and a non-volatile data area (Flash memory). The protected code is accessible through a single entry point (the call-gate mechanism explained below). Any attempt to jump and try to execute any of the functions included in the code section without passing through the entry point, generates a system reset.

The firewall is part of the dynamic protections. It must be set at startup (for example by a SB application).

Call gate mechanism

The firewall is opened by calling a 'call-gate' mechanism: a single entry point that must be used to open the gate and to execute the code protected by the firewall. If the protected code is accessed without passing through the call gate mechanism, then a system reset is generated. If any instruction is fetched outside the protected area, the firewall is closed (see the figure below).

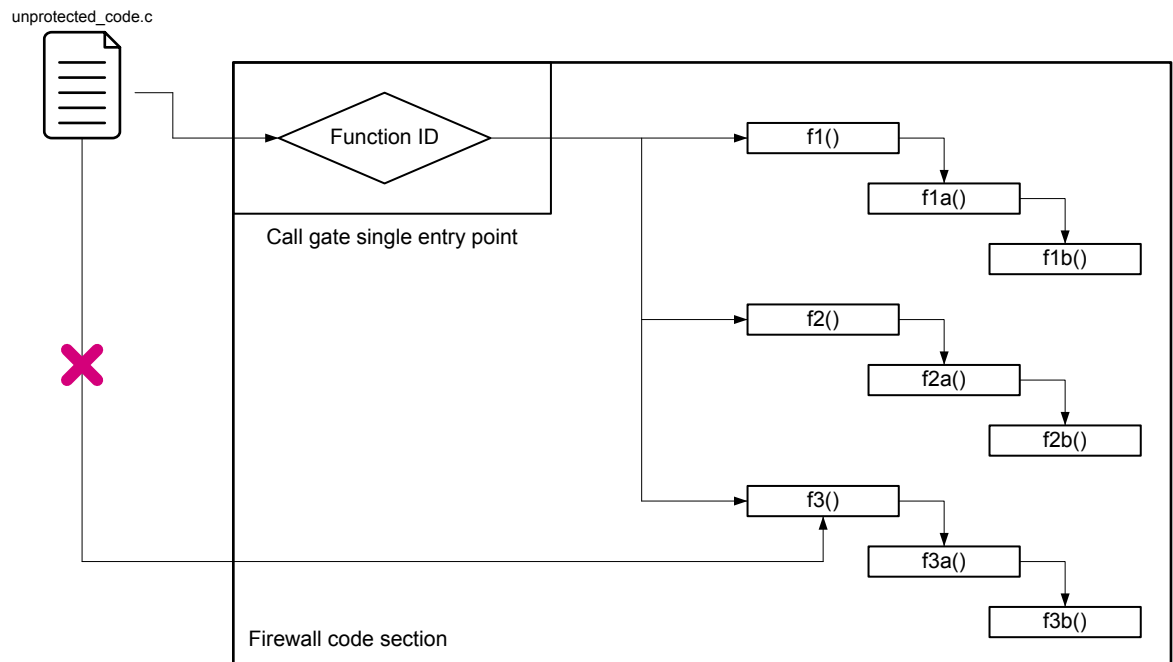
Figure 13. Firewall FSM



Since the only way to respect the call gate sequence is to pass through the single call gate entry point, a mechanism must be provided in order to support application calling multiple firewall-protected functions from unprotected code area (such as encrypt and decrypt functions). A parameter can be used to specify which function to execute (such as `CallGate(F1_ID)` or `CallGate(F2_ID)`). According to the parameter, the right function is internally called.

This mechanism is represented in the figure below.

Figure 14. Firewall application example



When should firewall be used

The firewall protects code, data and volatile data. Unlike secure memory, the firewall is used to protect code and data at runtime. The protected code can always be called as soon as a call-gate mechanism is respected.

Note: A firewall is available on STM32L0 and STM32L4 Series only.

6.8 Memory protection unit (MPU)

The MPU is a memory protection mechanism that allows specific access rights to be defined for any memory-mapped resource of the device: Flash memory, SRAM and peripheral registers. This protection is dynamically managed at runtime.

Note: MPU attributes are only set for CPU access. Other bus masters requests (such as DMA once) are not filtered by the MPU and must be deactivated if they are not needed.

Region access attributes

The MPU splits the memory map into several regions, each having its own access attribute. Access right can be set as Executable, Not executable(XN), Read-Write (RW), Read Only (RO) or No Access.

Note: There are other attributes set by the MPU for each region: shareable, cacheable and bufferable. These are not linked to security and are not considered here. Refer to applicable device programming manual or to the application note Managing memory protection unit (MPU) in STM32 MCUs (AN4838).

Privileged and unprivileged modes

On top of the access attribute, the Arm Cortex-M architecture defines two execution modes, allowing a process to run in either privileged or unprivileged mode. For each region, the access attribute can be set independently for each mode.

The table below shows the different cases supported by mixing modes and access attributes.

Table 14. Attributes and access permission managed by MPU

Privileged mode attribute	Unprivileged mode attribute	Description
Execute Never (XN) ⁽¹⁾		Code execution attribute
No access	No access	All accesses generate a permission fault.
RW	No access	Access from a privileged software only
RW	RO	Written by an unprivileged software generate a permission fault .
RW	RW	Full access
RO	No access	Read by a privileged software only
RO	RO	Read only, by privileged or unprivileged software

1. *Execute Never (XN) attribute is set by region and is valid for both modes. It can be used to avoid SRAM code injection for example.*

The code executed in privileged mode can access additional specific instructions (MRS) and can also access Arm core peripheral registers (such as NVIC, DWT or SBC). This is useful for OS kernels or pieces of secure code that require access to sensitive resources that are otherwise inaccessible to unprivileged firmware.

Secure process isolation strategy

At reset, the privilege mode is the default one for any process. Therefore the SB application is executed in privileged mode. Then the idea is to isolate secure processes (such as SB, OS kernel, Key manager or SFU) from unsecured or untrusted processes (user applications).

Table 15. Process isolation

Firmware type	Mode	Resources access
Secure firmware (such as SB or OS kernel)	Privileged	Full access
All remaining firmware	Unprivileged	MPU controlled access: No access, RO, RW

An OS kernel can manipulate MPU attributes dynamically to grant access to specific resources depending on the currently running task. Access right may be updated each time the OS switches from one task to another.

When should MPU be used

The MPU is used at runtime to isolate sensitive code and/or to manage access to resources according to the process currently executed by the device. It requires good programming skills to manage the switching from one mode to another.

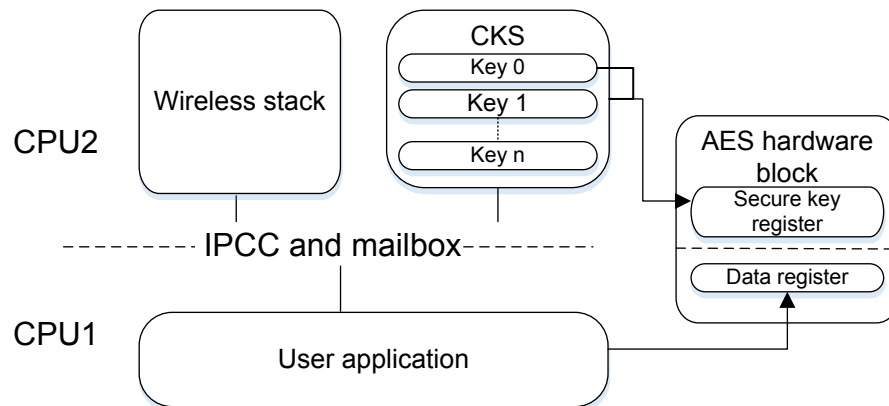
Note: The MPU is available on all STM32 series, except the STM32F0 Series (see the various programming manuals for more details).

6.9 Cryptographic key storage (CKS)

The STM32WB Series are dual-core devices with one core (CPU1) for user application and another core (CPU2) dedicated to the wireless real-time aspect execution (either Bluetooth Low Energy or thread protocols). The Flash memory used by CPU2 is secured from CPU1 or external access. Communication between the two cores is ensured by a mailbox and an inter-process channel control hardware block (IPCC).

In addition to the wireless stack execution, CPU2 offers a secure storage service for cryptographic keys used with a dedicated AES hardware block (refer to the figure below). The key register of this block is accessible only to CPU2, preventing access to the key by an untrusted process running on CPU1 or by the debug port.

After keys have been provisioned inside the secure area, the user application can use them by calling a secure load service with an index referencing the key and no more the key itself.

Figure 15. Dual-core architecture with CKS service


When should CKS be used

CKS must be used when a user application relies on AES encryption or decryption. Provisioned or ephemeral keys can be stored in a secure area, so that no other internal process or external access can read their value.

Note: CKS is available on STM32WB Series only.

6.10 Anti-tamper

The anti-tamper is a system level protection, used to detect physical tampering attempts on the system and to take related countermeasures. A tamper event is detected by a level transition on dedicated device pins. This event can be used to wake up the core in order to take appropriate actions (such as memory erase or alarms). The tamper detection is associated to the real-time clock peripheral (RCC). This peripheral embeds backup registers that may contain sensitive data such as master key for example. These registers may be reset if a tamper attempt is detected.

When should anti-tamper be used

It must be used upon system intrusion detection (in consumer products sealed enclosures for example).

Note: The tamper detection is available on all STM32 series.

6.11 Clock security system (CSS)

The CSS is designed to detect a failure of an external clock source (a crystal for example). A loss of clock source may or may not be intentional. In any case, the device must take an appropriate actions to recover. The CSS trigs an interrupt to the core in such event.

In case the external clock source drives the main system clock, the CSS switches the system to an internal clock source.

When should CSS be used:

The CSS must be used when an external clock is used.

Note: The CSS is available on all STM32 series.

6.12 Power monitoring

Some attacks may target the microcontroller power supply, for instance to make the system non-functional (Dos). A loss of power supply may also denote an attempt to freeze the device state in order to access the internal memory content.

The STM32 devices embed a programmable voltage detector (PVD) that can detect a drop of power. The PVD allows the configuration of a minimum voltage threshold, below which an interrupt is generated so that the core can take appropriate actions

When should PVD be used

the PVD must be used as soon as a sensitive application is running and is likely to leave some confidential data in the working memory (SRAM). A memory cleaning can be launched in case of power down detection.

Note: The PVD is available on all STM32 series.

6.13 Memory integrity hardware check

The error code correction (ECC) and parity checks are safety bits associated to the memory content. The ECC is associated to the Flash memory and allows recovering from a single bit error or detecting up to two erroneous bits on each Flash memory word. A simple parity check allows the detection of a single error bit on the SRAM 32-bit words.

When should ECC and parity bits be used

ECC and parity checks are mostly used for safety reasons. The ECC can also be used to prevent some invasive hardware attacks.

6.14 Independent watchdog (IWDG)

The IWDG is a free running down counter that can be used to trigger a system reset when the counter reaches a given timeout value. The IWDG can be used to provide a solution in case of malfunctions or deadlock in the running code. The IWDG is clocked by its own independent low-speed clock (LSI), so that it stays active even in the event of a main clock failure.

When should IWDG be used

IWDG can be used in order to break deadlocks or to prevent abnormally long download from a communication channel. It can also be used to control execution time of critical code (such as decryption or Flash programming).

Note: The IWDG is available on all STM32 series.

6.15 Device ID

Each STM32 device has a unique 96-bit identifier providing an individual reference for any device in any context. These bits can never be altered by the user.

The unique device identifier can be used for direct device authentication or used, for instance, to derive a unique key from a master OEM key.

6.16 Cryptography

As described in [Section 5](#), cryptography algorithms are essential to secure an embedded system. The cryptography ensures confidentiality, integrity and authentication of data or code. For efficiently supporting these functions, most STM32 series offer microcontroller options with embedded hardware cryptography peripherals. These hardware blocks allow the cryptographic computations (such as hashing or symmetric algorithms) to be accelerated. For devices with no such specific hardware acceleration, the STM32 cryptographic firmware library provides a full software implementation of a large set of cryptographic algorithms.

6.16.1 Hardware accelerators

The following cryptographic hardware peripherals are available in STM32 devices:

- **True random generator**
 - Hardware-based peripheral providing a physical noise source. Used to generate strong session keys.
- **AES accelerator**
 - Encryption/decryption
 - 128- or 256-bit keys
 - Several chaining modes (such as ECB, CBC, CTR or GCM)
 - DMA support
- **PKA accelerator**
 - Acceleration of RSA, DH and ECC over GF(p) operations, based on the Montgomery method for fast modular multiplications
 - Built-in Montgomery domain inward and outward transformation
- **HASH accelerator**
 - MD5, SHA1, SHA224, SHA256
 - FIPS compliant (FIPS Pub 180-2)
 - DMA support

Note: In case the AES block is used for encryption or decryption, the access to its registers containing the key must be protected and cleaned after usage (MPU).

In addition to these hardware blocks, some STM32 series (STM32L5 Series, specific part numbers of the STM32H7 Series) embed hardware on-the-fly decryption. This feature allows direct and fast decryption of ciphered content embedded in external memories.

- **On-the-fly decryption engine (OTFDEC)**
 - On-the-fly 128-bit decryption
 - AES in counter (CTR) mode, with 128-bit key length

Refer to [Section 6.17](#) for further details on OTFDEC engine.

6.16.2 Software library

The STM32 X-CUBE-CRYPTOLIB is the software library running on STM32 devices. It is available for free download at www.st.com/en/product/x-cube-cryptolib. The STM32 X-CUBE-CRYPTOLIB V3.1.0 is available with full firmware implementation, compiled for Cortex-M0, M0+, M3, M4, and M7 cores.

The X-CUBE-CRYPTOLIB supports the following algorithms:

- DES, 3DES with ECB and CBC
- AES with ECB, CBC, OFB, CCM, GCM, CMAC, KEY wrap, XTS
- Hash functions: MD5, SHA-1, SHA-224, SHA-256, SHA-384, SHA-512
- Other: ARC4, ChaCha20, Poly1305, Chacha20-Poly1305
- RSA signature with PKCS#1v1.5
- ECC with Key generation, Scalar multiplication (basis of ECDH) & ECDSA + ED25519 and Curve 25519

Note: The X-CUBE-CRYPTOLIB V3.1.0 is supported by all STM32 series with hardware acceleration (AES and/or HASH).

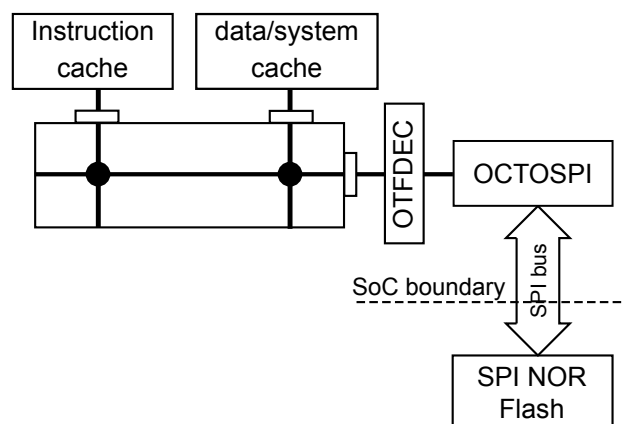
6.17 On-the-fly decryption engine (OTFDEC)

The external memory content (code and data) cannot be protected with traditional read/write protections. The way to protect the content is to encrypt it and to decrypt it inside the device before its usage.

One way consists in downloading the external memory content inside the SRAM, decrypting it and executing the code and/or using the data. There are two drawbacks with this method: it introduces a delay that may not be acceptable, and it may use a large amount of SRAM, depending on the content.

The OTFDEC offers the possibility to decrypt the content directly without any latency penalty and without the need for SRAM allocation. The OTFDEC is a hardware block that decrypt on-the-fly bus (AHB) traffic based on the read-request address information. It is used with Octo-SPI interface (see the figure below).

Figure 16. Typical OTFDEC usage in a SoC



The OTFDEC uses the AES-128 in counter mode, with a 128-bit key to achieve the lowest possible latency. Up to four independent and non-overlapping, encrypted regions can be defined (4-Kbyte granularity), each with its own key.

When OTFDEC should be used

The OTFDEC is used when an external memory is used by the system.

Note: *The OTFDEC is available on STM32L5 Series only.*

7 Guidelines

Several security mechanisms exist (hardware and software) in order to build a secure system. Some of them must be adopted systematically, whatever the user application. It is the case of the RDP feature, that prevents basic access to the Flash memory by disabling the debug port. Other features must be selected depending on user application and the required security level.

This section helps defining the adapted set of security features to put in place, depending on the system use-cases.

Use-cases are gathered in four main groups: protection against external (1) and internal (2) threats, security maintenance (3) and other use-cases related to cryptography (4) (see the table below).

Table 16. Security use cases

1 Device protection against external threats: RDP protection, tamper detection, device monitoring	
-	1.1 Device configuration (option bytes, not supposed to be modified ever)
	<ul style="list-style-type: none"> Use RDP Level 2. This closes the device from any external access.
	1.2 Remove debug capability for the device.
	<ul style="list-style-type: none"> Use RDP Level 2 for permanently disabling the debug.
	1.3 Protect a device against a loss of external clock source (crystal).
	<ul style="list-style-type: none"> Enable clock security system (CSS).
	1.4 Detect a system-level intrusion.
	<ul style="list-style-type: none"> Use tamper detection capability of the RTC peripheral.
	1.5 Protect a device from code injection.
	<ul style="list-style-type: none"> Use RDP. Isolate communication port protocol with MPU, firewall or HDP. Limit communication port protocol access range. Use write protection on empty memory areas (Flash memory and SRAM).
2. Code protection against internal threats: TrustZone, PCROP, MPU, firewall and HDP	
-	2.1 Protect the code against cloning.
	<ul style="list-style-type: none"> Use RDP Level 1 or 2 against external access. Use PCROP on most sensitive parts of the code against internal access. Use OTFDEC to secure code stored in the external memory.
	2.2 How to protect secret data from other processes
	<ul style="list-style-type: none"> Use firewall to protect both code and data. Use MPU to protect secret data area from being read. Use HDP in case data must only be used at reset. Use secure domain of TrustZone if available.
	2.3 Protect code and data when not fully verified or trusted libraries are used.
	<ul style="list-style-type: none"> Use PCROP to protect user most sensitive code. Use firewall to protect user sensitive application (code, data and execution). Use MPU and de-privilege the untrusted library. Use IWDG to avoid any deadlock. Use secure domain of TrustZone if available.
3. Device security check and maintenance: integrity checks, SB, SFU	

-	3.1 Check code integrity.
	<ul style="list-style-type: none"> Hash firmware code at reset and compare to expected value. Enable ECC on the Flash memory and parity check on the SRAM.
	3.2 Security checks or embedded firmware authentication
	<ul style="list-style-type: none"> Implement SB application with cryptography. Protect SB application secret data (refer to previous sections). Guarantee unique boot entry on SB application: <ul style="list-style-type: none"> Use HDP if available. Use RDP Level 2 and disable boot pin selection.
	3.3 Securely update the firmware in the field.
	<ul style="list-style-type: none"> Implement a SFU application with cryptography. Apply relevant secure memory protection around the SFU secret data (refer to previous sections).
4. Communication and authentication: cryptography	
-	4.1 Communicate securely.
	<ul style="list-style-type: none"> Use or implement secure communication stacks relying on cryptography for confidentiality and authentication (such as TLS for Ethernet).
	4.2 Use the ST AES/DES/SHA cryptographic functions with STM32 devices.
	<ul style="list-style-type: none"> Use only official software implementation by ST with STM32 X-CUBE-CRYPTOLIB.
	4.3 Accelerate AES/DES/SHA cryptographic functions.
	<ul style="list-style-type: none"> Use device with cryptographic hardware peripheral together with official STM32 X-CUBE-CRYPTOLIB. Use OTFDEC to access AES-ciphered code in the external memory without latency penalty.
	4.4 Generate true random data.
	<ul style="list-style-type: none"> Use hardware true random generator embedded in the STM32 devices.
	4.5 Uniquely identify ST microcontrollers.
	<ul style="list-style-type: none"> Use STM32 microcontrollers 96-bits UID.
	4.6 Authenticate a product device.
	<ul style="list-style-type: none"> Embed a shared encryption key in the device and exchange encrypted message.
	4.7 Uniquely authenticate a product device.
	<ul style="list-style-type: none"> Embed a device private key and its certificate in the device and exchange encrypted message.
	4.8 Authenticate communication servers.
	<ul style="list-style-type: none"> Embed a shared encryption key in the device and exchange encrypted message. Embed server public key in the device and exchange encrypted message.

8 Conclusion

This document introduces the concept of security in systems built around STM32 microcontrollers. Security is about protecting embedded firmware and data as well as the device functionality.

Implementing security involves development costs and constraints. The counter-measures must be adapted to the device own value and to the possible risks and damages related to a denial of service.

Using the set of STM32 robust hardware security features, together with a secure application implementation ensure system security against multiform attacks all along product lifetime. The most basic and easy to setup security features should systematically be adopted, some others are more complex to handle and must be considered as soon as possible in the project development for efficient implementation.

The STM32 series offer a wide range of features to build such secure systems.

Appendix A Cryptography - Main concepts

Integrity, authentication and confidentiality

The objectives of cryptography are threefold:

- Confidentiality: protection of sensitive data against unauthorized read accesses
- Authentication: guarantee of the message sender identity
- Integrity: detection of any message corruption during transmission

To meet these objectives, all secure data flows rely on more or less complex combinations of the below algorithms:

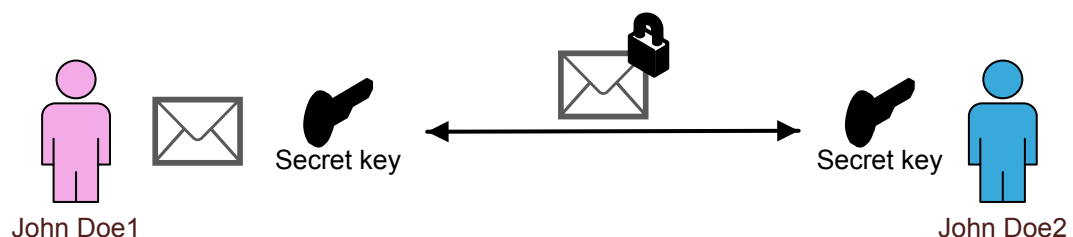
- Secret key/symmetric cryptography
- Public key/asymmetric cryptography
- Hashing

These algorithms are described in the below sections.

A.1 Secret key algorithms

This family of algorithms ensures confidentiality by ciphering a clear plain text with a secret key shared between the transmitter and the receiver. This technique is referred to as symmetric cryptography because the same key is used for ciphering and deciphering.

Figure 17. Symmetric cryptography



The inherent weakness of these algorithms is the key sharing between both parties. It may not be an issue in secure environments (such as manufacturing plants) but when both parties are distant, the key transfer becomes a challenge.

Among all secret key algorithms, block-based algorithms are very common since they can be efficiently accelerated by hardware or software parallel implementations. Typical AES (advanced encryption standard) algorithms operate on clear blocks of 128 bits. They produce ciphered blocks of the same length using keys of 128, 192 or 256 bits. The different ways to chain consecutive blocks are called “mode of operations”. They include cipher block chaining (CBC), counter mode (CTR) and Galois counter mode (GCM).

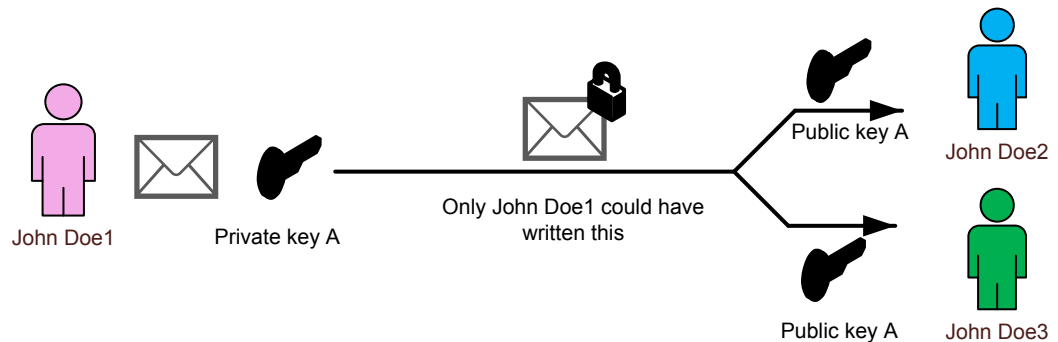
Since these algorithms are deterministic, they always mix input data with a random value, known as nonce, used only for one session as initialization vector.

A.2 Public key algorithms (PKA)

This class of algorithms is based on a pair of keys. One key, the private one, is never exchanged with any remote system, while the other key, the public one, can be shared with any party. The relationship between both keys is asymmetric (asymmetric cryptography):

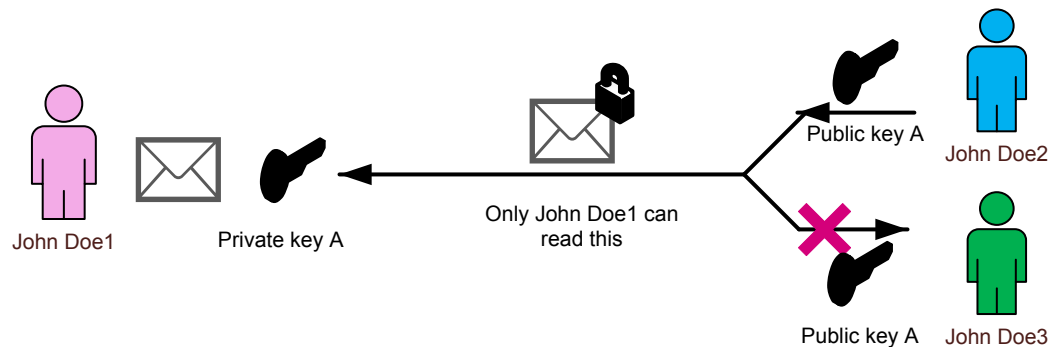
- A message encrypted by the private key can be read by any party with the public key. This mechanism ensures a strong authentication of the sender since the private key has never been shared. Digital signatures are based on this mechanism.

Figure 18. Signature



- A message encrypted by the public key can only be read by the private key owner.

Figure 19. PKA encryption



The main use of public key algorithms is authentication.

It is also used to resolve the “key sharing” issue of symmetric cryptography. However, this comes at the cost of more complex operations, increased computation time and bigger memory footprint.

RSA and elliptic curve cryptography (ECC) are the most common asymmetric algorithms.

Hybrid cryptography

Common secure transfer protocols (such as Bluetooth and TLS) rely on both algorithm types. This scheme is known as hybrid cryptography:

- Asymmetric cryptography is used first, in order to solve the symmetric key-sharing problem. A session key is exchanged by the public key owner to the private key owner.
- Transfer confidentiality is then provided by a symmetric algorithm using the session key.

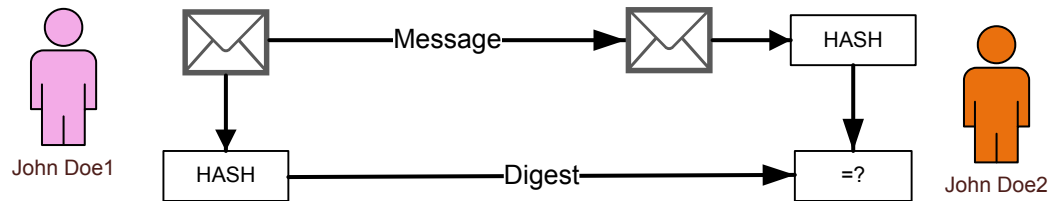
A.3

Hash algorithms

Hash algorithms guarantee the message integrity. They generate a unique fixed-length bitstream from a message called the digest. Any difference in the input message produces a totally different digest. The digest cannot be reversed to retrieve the input message.

Hashing can be used independently from message encryption.

Figure 20. Message hashing



The difference with classic CRC is the robustness due to operations that are more complex and a much higher digest length: up to 512 bits instead of 16 or 32 bits. As an example, CRC are reserved for fast integrity checks during data transfers. Digest length makes them virtually unique and ensures that no collision occurs.

Typical algorithms are the MD5 (128-bit digest), SHA-1 (160-bit digest), SHA-2 (224-, 256-, 384-, or 512-bit digest) and SHA-3 (224-, 256-, 384-, or 512-bit digest).

A.4 MAC or signature and certificate

MAC and signature

The message authentication code (MAC) and signature add authentication to integrity by encrypting the message hash. The difference between MAC and signature is that MAC generation uses a symmetric key algorithm (Figure 21) while signature uses the message sender private key (Figure 22).

The signature adds non-repudiation dimension to authentication:

- A private key is not supposed to be revoked (its lifetime goes beyond the transfer operation.) while a secret key may have a limited lifetime (limited to this transfer).
- The private key used for signature is never shared, its security is higher than a secret key.

Figure 21. MAC generation with secret key algorithm

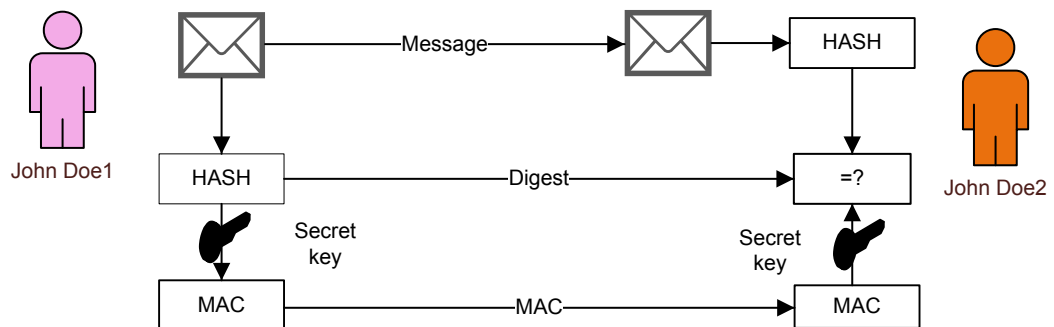
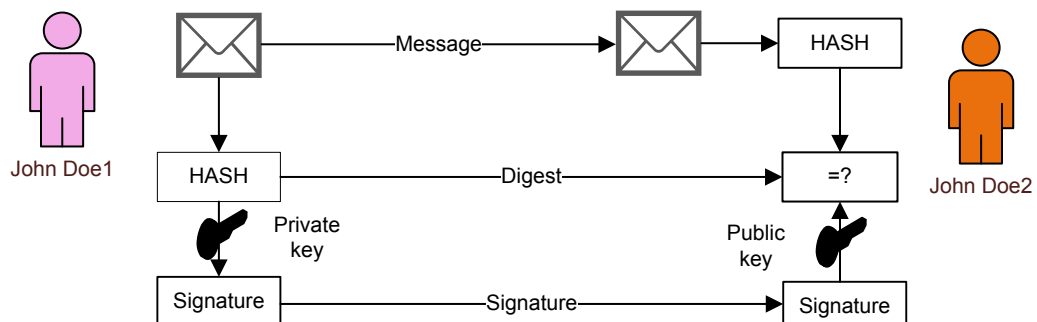


Figure 22. Signature generation with public key algorithm



Certificate

A certificate is related to public key algorithms. It authenticates the public key in an asymmetric transfer. It is used to counteract usurpation by an attacker that could substitute the right public key by his own key. A certificate consists in the public key signed by a certificate authority (CA) private key. This CA is considered as fully trusted. In addition to the public key, the certificate also contains version numbers, validity period and some IDs.

Revision history

Table 17. Document revision history

Date	Version	Changes
17-Oct-2018	1	Initial release.
25-Feb-2019	2	<p>Updated:</p> <ul style="list-style-type: none"> Table 1. Applicable products Section 1 General information Table 11. Security features for STM32H7, STM32G0, STM32G4 and STM32WB Series Figure 9. Example of RDP protections (STM32L4 Series) Section 6.6 Firewall <p>Added:</p> <ul style="list-style-type: none"> Section 6.8 Cryptographic key storage (CKS)
7-Oct-2019	3	<p>Updated:</p> <ul style="list-style-type: none"> Table 1. Applicable products Section introduction renamed overview. Table 2. Glossary Section hardware protections renamed device protections Figure 6. Memory types Table 6. Memory types and associated protection Section 4.2.4 External Flash memories Table 7. Scope of STM32 embedded memories protection features Table 8. Software isolation mechanism Section 4.5 Boot protection Section 5 Secure applications: Table 9, Table 10 and Table 11 Section 6.2 Readout protection (RDP) Section 6.6 Secure hide protection (HDP) Section 6.16 Cryptography Section 7 Guidelines Some colors removed on all figures <p>Added:</p> <ul style="list-style-type: none"> Section 4.1 TrustZone for ARMv8-M architecture Section 6.3 TrustZone Section 6.17 On-the-fly decryption engine (OTFDEC)

Contents

1	General information	2
2	Overview	4
2.1	Security purpose	4
2.2	Building a secure system	5
3	Attack types	7
3.1	Introduction	7
3.2	Software attacks	8
3.3	Hardware attacks	10
3.3.1	Non-invasive attacks	10
3.3.2	Silicon invasive attacks	11
3.4	IoT system attack examples	12
4	Device protections	17
4.1	TrustZone for ARMv8-M architecture	17
4.2	Memory protections	17
4.2.1	System Flash memory	19
4.2.2	User Flash memory	19
4.2.3	Embedded SRAM	19
4.2.4	External Flash memories	20
4.2.5	STM32 memory protections overview	20
4.3	Software isolation	21
4.4	Debug port and other interfaces protection	21
4.5	Boot protection	22
4.6	System monitoring	22
5	Secure applications	23
5.1	Root and chain of trust	23
5.2	Secure boot (SB)	23
5.3	Secure firmware update (SFU)	24
6	STM32 security features	26
6.1	Overview	26

6.2	Readout protection (RDP)	28
6.3	TrustZone	30
6.3.1	Core state	31
6.3.2	Secure attribution unit (SAU)	31
6.3.3	Memories and peripherals protections	31
6.3.4	Summary of transaction types	32
6.4	Flash memory write protection (WRP)	32
6.5	Execute-only firmware (PCROP)	33
6.6	Secure hide protection (HDP)	33
6.7	Firewall	33
6.8	Memory protection unit (MPU)	35
6.9	Cryptographic key storage (CKS)	36
6.10	Anti-tamper	37
6.11	Clock security system (CSS)	37
6.12	Power monitoring	37
6.13	Memory integrity hardware check	38
6.14	Independent watchdog (IWDG)	38
6.15	Device ID	38
6.16	Cryptography	38
6.16.1	Hardware accelerators	38
6.16.2	Software library	39
6.17	On-the-fly decryption engine (OTFDEC)	39
7	Guidelines	41
8	Conclusion	43
Appendix A	Cryptography - Main concepts	44
A.1	Secret key algorithms	44
A.2	Public key algorithms (PKA)	44
A.3	Hash algorithms	45
A.4	MAC or signature and certificate	46
	Revision history	48

List of tables

Table 1.	Applicable products	1
Table 2.	Glossary	2
Table 3.	Assets to be protected	5
Table 4.	Attacks types and costs	8
Table 5.	Attacks targets.	13
Table 6.	Memory types and associated protection	19
Table 7.	Scope of STM32 embedded memories protection features	21
Table 8.	Software isolation mechanism	21
Table 9.	Security features for STM32Fx Series.	26
Table 10.	Security features for STM32Lx Series.	27
Table 11.	Security features for STM32H7, STM32G0, STM32G4 and STM32WB Series.	28
Table 12.	RDP protections	30
Table 13.	Transaction types.	32
Table 14.	Attributes and access permission managed by MPU.	36
Table 15.	Process isolation	36
Table 16.	Security use cases.	41
Table 17.	Document revision history	48

List of figures

Figure 1.	Corrupted connected device threat	4
Figure 2.	Hardware and application security layers	6
Figure 3.	IoT system	12
Figure 4.	IoT system attack examples	13
Figure 5.	ARMv8-M TrustZone execution modes	17
Figure 6.	Memory types	18
Figure 7.	Chain of trust	23
Figure 8.	Secure boot FSM	24
Figure 9.	Secure server/device SFU architecture	25
Figure 10.	Example of RDP protections (STM32L4 Series)	29
Figure 11.	TrustZone implementation at system level	31
Figure 12.	HDP protected firmware access	33
Figure 13.	Firewall FSM	34
Figure 14.	Firewall application example	35
Figure 15.	Dual-core architecture with CKS service	37
Figure 16.	Typical OTFDEC usage in a SoC	39
Figure 17.	Symmetric cryptography	44
Figure 18.	Signature	45
Figure 19.	PKA encryption	45
Figure 20.	Message hashing	46
Figure 21.	MAC generation with secret key algorithm	46
Figure 22.	Signature generation with public key algorithm	46

IMPORTANT NOTICE – PLEASE READ CAREFULLY

STMicroelectronics NV and its subsidiaries ("ST") reserve the right to make changes, corrections, enhancements, modifications, and improvements to ST products and/or to this document at any time without notice. Purchasers should obtain the latest relevant information on ST products before placing orders. ST products are sold pursuant to ST's terms and conditions of sale in place at the time of order acknowledgement.

Purchasers are solely responsible for the choice, selection, and use of ST products and ST assumes no liability for application assistance or the design of Purchasers' products.

No license, express or implied, to any intellectual property right is granted by ST herein.

Resale of ST products with provisions different from the information set forth herein shall void any warranty granted by ST for such product.

ST and the ST logo are trademarks of ST. For additional information about ST trademarks, please refer to www.st.com/trademarks. All other product or service names are the property of their respective owners.

Information in this document supersedes and replaces information previously supplied in any prior versions of this document.

© 2019 STMicroelectronics – All rights reserved