Introduction to security for STM32 MCUs

Introduction

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

Security in microcontrollers encompasses several aspects including protection of firmware intellectual property, protection of private data in the device, and guarantee of a service execution.

The context of IoT has made security even more important. The huge number of connected devices makes them an attractive target for attackers and several remote attacks have shown the vulnerabilities of device communication channels. With IoT, the security extends the requirements for confidentiality and authentication to communication channels, which often require encryption.

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

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

The subsequent sections focus on the set of hardware and software protections that defend the system from these attacks.

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

Table 1. Applicable products

Type	Product series
Microcontrollers	STM32C0 Series, STM32F0 Series, STM32F1 Series, STM32F2 Series, STM32F3 Series, STM32F4 Series, STM32F7 Series, STM32G0 Series, STM32G4 Series, STM32H5 Series, STM32H7 Series, STM32L0 Series, STM32L1 Series, STM32L4 Series, STM32L4+ Series, STM32L5 Series, STM32U5 Series, STM32WB Series, STM32WBA Series, STM32WL Series, STM32U0 Series, STM32WB0 Series.

1 General information

This document applies to STM32 Arm® Cortex®-core based microcontrollers.

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

The table below presents a nonexhaustive 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)
DRNG	Deterministic random number generator: generates pseudo-random number from input value
DPA	Differential power analysis
ECC	Error code correction
FIA	Fault injection attack
FIB	Focused ion beam
GTZC	Global TrustZone® controller
HDP	Secure hide protection
HUK	Hardware unique key
IAP	In-application programming
IAT	Initial attestation token
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	Nonsecure callable
NVM	Nonvolatile memory
OTFDEC	On-the-fly decryption
OTP	One-time programmable
PCROP	Proprietary code readout protection
PKA	Public key algorithm (also named aka asymmetric algorithm)
PSA	Platform security architecture
PVD	Programmable voltage detector

Term	Definition
PWR	Power control
ROM	Read only memory–system flash memory in STM32
RoT	Root of trust
RDP	Read protection
RSS	Root secure services
RTC	Real-time clock
SAU	Security attribution unit
SB	Secure boot
SCA	Side channel attack
SDRAM	Synchronous dynamic random access memory
SECDED	ECC mode of operation: single error correct, double error detect
SFI	Secure firmware installation
SFU	Secure firmware update
SPA	Simple power analysis
SPE	Secure processing environment
SRAM	Static random access memory (volatile)
SST	Secure storage
STSAFE	Secure element product from STMicroelectronics portfolio
SWD	Serial-wire debug
TF-M	Trusted firmware-M
WRP	Write protection

Documentation references

The **reference manual** of each device gives details on the availability of security features. It also informs about memory protections implementation.

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

- *STM32 Cortex®-M33 MCUs programming manual* (PM0264)
- *STM32F7 series and STM32H7 series Cortex®-M7 processor programming manual* (PM0253)
- *STM32 Cortex®-M4 MCUs and MPUs programming manual* (PM0214)
- *STM32F10xxx/20xxx/21xxx/L1xxxx Cortex®-M3 programming manual* (PM0056)
- *Cortex®-M0+ programming manual for STM32L0, STM32G0, STM32WL, STM32WB, and STM32WB0 series (PM0223)*

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

Ref.	Doc number	Title	Comment
[1]	AN4246	Proprietary code readout protection on STM32L1 series MCUs	Explains how to set up and work with PCROP firmware for the specified MCUs, provided with the X-CUBE-PCROP expansion package.
[2]	AN4701	Proprietary code readout protection on STM32F4 series MCUs	
[3]	AN4758	Proprietary code readout protection on STM32L4, STM32L4+, STM32G4, and STM32WB series MCUs	
[4]	AN4968	Proprietary code readout protection on STM32F72/F73xx MCUs	
[5]	AN4838	Managing memory protection unit (MPU) in STM32 MCUs	Describes how to manage the MPU in the STM32 products.
[6]	AN5185	STMicroelectronics firmware upgrade services for STM32WB series	-
[7]	AN5447	Overview of secure boot and secure firmware update solution on Arm® TrustZone® STM32 MCUs	-
[8]	UM1924	Legacy STM32 crypto library	Describes the API of the STM32 crypto library; provided with the X-CUBE-CRYPTOLIB expansion package.
[9]	UM2262	Getting started with the X-CUBE-SBSFU STM32Cube expansion package	Presents the SB (secure boot) and SFU (secure firmware update) STMicroelectronics solutions; provided with the X-CUBE-SBSFU expansion package.
[10]	AN4730	Using the firewall embedded in STM32L0/L4/L4+ series MCUs	Describes how to access securely sensitive parts of code and data

2 Overview

2.1 Security purpose

Why protection is needed

Security in microcontrollers means protecting embedded firmware, data, and the system functionality. The need for data protection is greatest in the case of cryptographic keys or personal data.

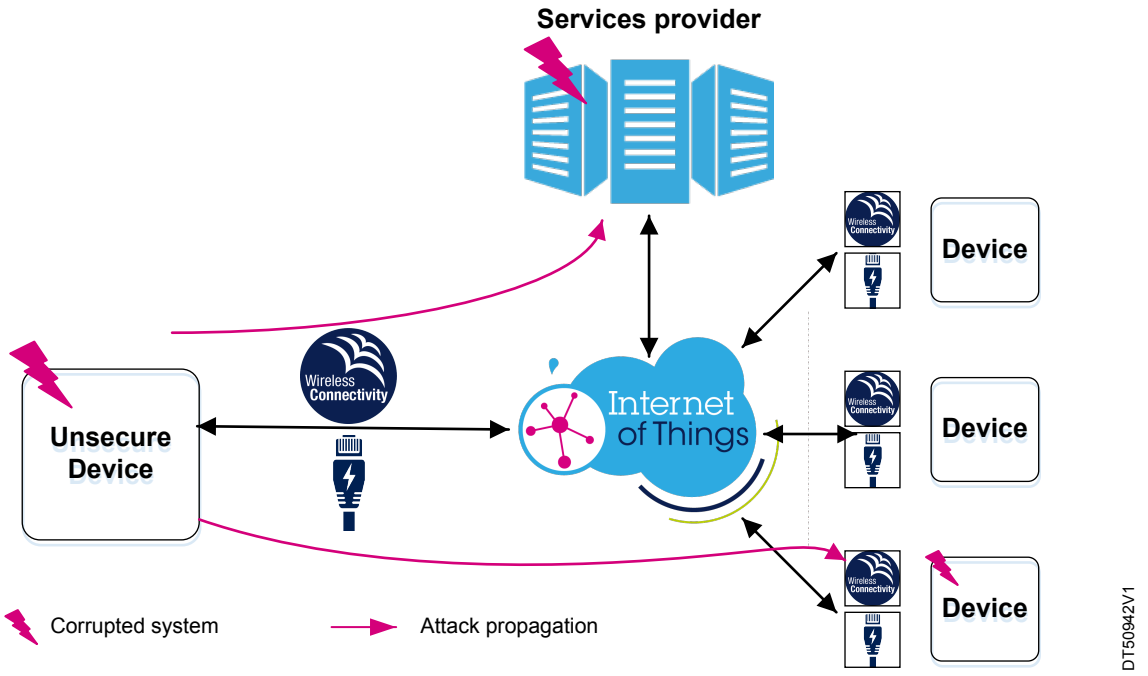
The firmware code is also an important asset. If an attacker gains access to the binary, they can reverse-engineer the program in an attempt to find further vulnerabilities, bypass licensing and software restrictions. The attacker can copy any custom algorithms, or even use it to flash a clone of the hardware. Even in the case of open-source software, it makes sense to attest that the code is authentic, and not replaced by malicious firmware.

Denial-of-service attack (DoS attack) is another major threat when considering protection systems (such as environmental: 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. Today, systems built around microcontrollers become potential targets for more and more skilled attackers who expect financial gains. These gains can be very high, especially if the attack can be propagated to a large scale, as in the context of the IoT. Even if no system is completely secure, it is possible to make the attack more expensive.

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

Figure 1. Corrupted connected device threat



What must be protected

Security cannot be limited to a certain target or asset. It is difficult to protect data if the code binary is exposed. Both the attacks and the protection mechanisms often do not make difference. However it is still useful to summarize the assets and risks.

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

Table 3. Assets to be protected

Target	Assets	Risks
Data	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 (bootloader, malicious application)	Device correct functionality Device/user identity	Denial of service Attacks on service providers Fraudulent access to service (cloud)
User code	Device hardware architecture/design Software patent/architecture Technology patents	Device counterfeit Software counterfeit Software modification Access to secure areas

Vulnerability, threat, and attack

Protection mechanisms have to deal with different threats. The objective is to remove vulnerabilities that could be exploited in an attack. 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.

3 Attack types

This section presents the different types of attack that a microcontroller may have to face, from the most basic ones to very sophisticated and expensive ones. 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 to attack types

A key rule in security 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 it necessary to consider how the device firmware is updated to increase its security (see Section 5.3.2: Secure firmware update (SFU)).

Secondly, in laboratory conditions with proper equipment, it is possible to retrieve microcontroller content, or even its design architecture details. These techniques are briefly presented in Section 3.3: Hardware attacks.

From an attacker's point of view, an attack is most 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, the acquisition of the necessary skills by the attacker, and on money (equipment) spent to succeed.

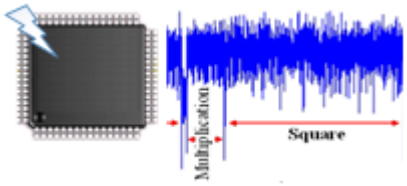
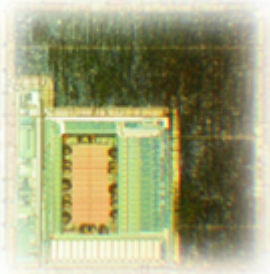
Attack types

While there are more detailed groups and categories of attack, the basic categories are the following ones:

- **Software attacks** are carried by exploiting bugs, protocol weaknesses, or untrusted pieces of code among others. Attacks on communication channels (interception or usurpation) are part of this category. Software attacks represent the vast majority of cases. Their cost may be very low. They can be widely spread and repeated with huge damage. It is not necessary to have a physical access to the device. The attack can be executed remotely.
- **Hardware attacks** need physical access to the device. The most obvious one exploits the debug port, if it is not protected. However, in general, hardware attacks are sophisticated and can be very expensive. They are carried out with specific materials and require electronics engineering skills. A distinction is made between noninvasive attacks (carried out at board or chip level without device destruction), and invasive attacks (carried out at device-silicon level with package destruction). In most cases, such an attack is only profitable if it reveals information that leads to a new and widely applicable remote attack.

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

Table 4. Attacks types and costs

Attacks types	Non-invasive	Semi–invasive	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 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 memory 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 takes 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 nonverified or untrustworthy 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:

- 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 to inject a piece of code inside the system. The size of the malware depends on the 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 to it.

Methods of injecting malware 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

  This 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:
  - Execution of data. Sometimes it is possible to sneak the malware in as data, and to exploit incorrect boundary check to execute it.
  - Stack-based buffer overflows, heap-based buffer overflows, jump-to-libc attacks, and data-only attacks

  This third category is by definition difficult to circumvent. 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. For example, the TF-M includes such a mechanism.
- 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 authentication based on a shared secret. 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.

Interesting countermeasures are listed below:

- Limit the number of login trials with a monotonic counter (implemented with a timer, or if possible, with a backup domain).
- Increase the delay between consecutive login attempts.
- Add a challenge-response mechanism to break automatic trials.

3.3 Hardware attacks

Hardware attacks require a physical access to the device or, often, to several devices in parallel.

The two following types of attacks differ in cost, time, and necessary expertise:

- Non-invasive attacks have only external access to the device (board-level attack) and are moderately expensive (thousands to tens of thousands US dollars in equipment).
- 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 (more than 100k dollars, and often in the range of millions) and target very valuable data (Keys or IDs) or even the protection technology itself.

General-purpose microcontrollers are not the best candidates to counter the most advanced physical attacks. If a highest protection level is required, 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.

3.3.1 Non-invasive attacks

Non-invasive, or board-level attacks try to bypass the protection without physical damage (device kept functional). Only accessible interfaces and device environment are used. 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)
- Life-cycle management using product state (where this technology succeeded the RDP)

Serial port access

Access to communication ports (such as I2C or SPI) may hide 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 potentially gain access to the device resources.

Countermeasures:

- Software:
  - Associated protocol operations must be limited by the firmware level, so that no sensitive resources can be read or written.
  - Isolate communication stack from sensitive data.
  - Length of data transfer must be checked 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 outside the parameters defined in the datasheet to generate malfunctions in the system. A successful attack can modify 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 may be non-intentional, countermeasures are the same as the one used for safety: redundancy, error detection and monitoring.

Countermeasures:

- Software:
  - Check function return values.
  - Use strict comparisons when branching.
  - Make sure that no code was skipped in critical parts by incrementing a dedicated variable in each branch with prime number and check for the expected value.
  - Use non-trivial values as true and false (avoid comparing to 0 or -1, try complex values with high mutual Hamming distance).
- 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 running 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 consumption.

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 usually no efficient hardware countermeasure embedded in general-purpose microcontrollers (except for the SAES in STM32H5 and STM32U5 devices, the hardware protection in limited in general-purpose microcontrollers. The protection level is reflected by the certification level achieved by the device. See Section 5.6: Product certifications for more details).

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. The attacker needs to obtain a substantial quantity of devices to be successful. Carried out with expensive equipments often found in specialized laboratories, they require a high level of skills and knowledge, as well as time.

Invasive attacks start with the removal of the device package. An initial analysis can be done without eliminating the passivation layer, however investigations with device interaction (probing) require its removal. De-packaging 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.

Several ST microcontrollers dedicated to security offer robustness against such kind of treatments. These are not part of the STM32 family and are out of scope of this document. 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 analyze 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.

Reading the data

When using the electron microscope, the data, represented by an electric charge, becomes visible. It is possible to read the whole device memory.

Micro probing and internal fault injection

Micro probing consists in interacting with the device at metal layer level. 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

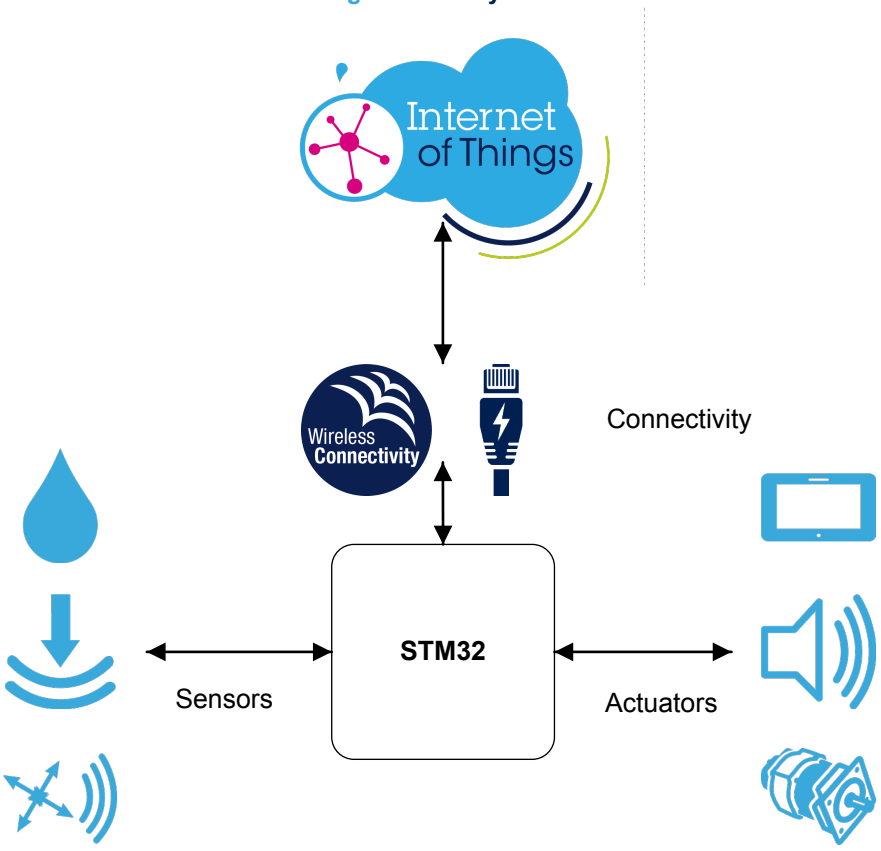
More sophisticated tools can be used to perform attacks. FIB (focused ion beam) workstations, for example, simplify the manual probing of deep metal and polysilicon lines. They also can be used to modify the device structure by cutting existing or creating new interconnection lines and even new transistors.

3.4 IoT system attack examples

This section presents typical examples of attacks on an IoT system. Fortunately, most of these attacks can be countered by enabling security feature (hardware countermeasures) and secure application 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 may also be responsible for the system maintenance through firmware update and integrity check.

Figure 2. IoT system



3.5 List of attack targets

The following sections list the possible attack targets.

Initial provisioning

The cryptographic data for root of trust for the chain of security must be injected to the SoC in a controlled trusted way. Whether it is a key, a certificate or a hash initial value, it must remain immutable and/or secret. 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
  - use of OTP memory

Boot modification

The purpose of this attack is to use the bootloader 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.

- *Risks*: full access of the microcontroller content
- *Countermeasures*:
  - unique boot entry
  - bootloader and debug disabled (see Section 6.2: Readout protection (RDP))

Secure boot (SB) or Trusted Firmware-M (TF-M)

Robust systems rely on initial firmware integrity and authenticity check before starting the main application. As the root of trust of a device, this part of user firmware must be immutable and impossible to bypass.

A successful attack 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 (refer to the *Initial provisioning* section at the beginning of this chapter).

- *Risks*: device spoofing or application modification
- *Countermeasures*:
  - unique boot entry point to avoid verification bypass
  - "immutable code" to avoid SB code modification
  - secure storage of firmware signature and/or tag value
  - environment event detection (such as power supply glitch, temperature or clock speed)

Firmware update

The firmware update 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 gives an attacker an opportunity to enter the device with its own firmware or a corrupted version of the existing firmware.

The process must be secured with firmware authentication and integrity verification. A successful attack requires an access to the cryptographic procedure and keys (refer to the *Initial provisioning* section at the beginning of this chapter).

- *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.

Communication interfaces

Serial interfaces (such as SPI, I2C or USART) are used either by the bootloader or by applications to exchange data and/or commands with the device. The interception of a communication 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.
  - Disable I/F ports when not needed.
  - Check inputs carefully.

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 Section 6.2: Readout protection (RDP)).

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*: incorrect system behavior.
- *Countermeasure*: anti-tamper to detect system intrusion at board level

Sensitive firmware and data

Some parts of the firmware need special protection: for example the cryptographic algorithm or a third-party library. In addition, selected 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 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
  - encryption of external memory

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. While in the non-volatile memory, the secrets may be stored as encrypted, when loaded to the SRAM, they need to be present in plain view to be used. In the same time, the SRAM usually holds communication buffers. 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

Random number generation

Random numbers are often used in cryptography for session key, cryptographic nonce or initialization vector (IV) generation. Weak random generator may make any secure protocol vulnerable.

A software attack tries to exploit an hidden periodicity or structures of a random sequence to guess the secret key and break into communication confidentiality. An hardware attack attempts to disable the entropy source, or weaken the statistic randomness of the output.

A robust random generator depends on both the quality of the entropy source (analog) and the subsequent processing in digital.

- *Risk*: reduced security of cryptographic protocols
- *Countermeasure*:
  - Use true hardware entropy generator.
  - Use tests on the RNG output, and verify statistic properties of produced random numbers.
  - Take full advantage of the error detection and heath check mechanisms available on the device RNG.

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 trusted. A potential weakness can be massively exploited.

- *Risk*: device access (content, control) through network
- *Countermeasures*:
  - communication process isolation
  - server authentication
  - secure firmware update to patch bugs

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 can be adopted by all communication 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 of cryptographic version of the 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, used to protect code and data from internal (software) and external attacks
- **Software isolation**: inter-processes protection to avoid internal attacks
- **Interface protection**: used to protect device entry points like serial or debug ports
- **System monitoring**: detects device external tampering attempts or abnormal behaviors

4.1 Configuration protection

Any persistent security configuration is stored in a dedicated area of the non-volatile memory, called option bytes (OB). The option bytes are protected with redundancy, and, in case of perceived vulnerability, they are stored as a “magic value”. Besides rules imposed by security logic (OB can only be changed under specific conditions), there are rules to determine the default value in case of OB corruption.

Note: Magic value is a pattern of usually 8 bits that must be exactly matched for the value to be recognized as valid, common examples are 0xB4 or 0xC3 for TRUE or FALSE.

Details about rules to modify the option bytes are detailed in each reference manual. The configuration is generally frozen by raising the RDP level. Additional rules apply case-by-case.

Each OB value is preconfigured to a reset value in production. This value is usually the least secure, open option. In case of an error detected in OB loading (OBL, happens on every POR or request), the configuration is set to a default value. This default value usually sets the highest possible security level to prevent leaking information from damaged microcontroller.

It is useful to analyze the impact of a configuration error on an application, and to plan for potential recovery. Highest number of corrupted option bytes however result of errors in programming. If the OB are modified in controlled environment, the threat of unintentional device locked-up is very low.

4.2 TrustZone® for Armv8-M architecture

Microcontrollers based on Armv6 or Armv7 architecture (Cortex-M0, M3, M4, and M7) rely mostly on software implementations for firmware and resource isolation. These mechanisms, described later in the document, are robust but not that flexible to allow the concurrent execution of secure firmware together with nonsecure firmware.

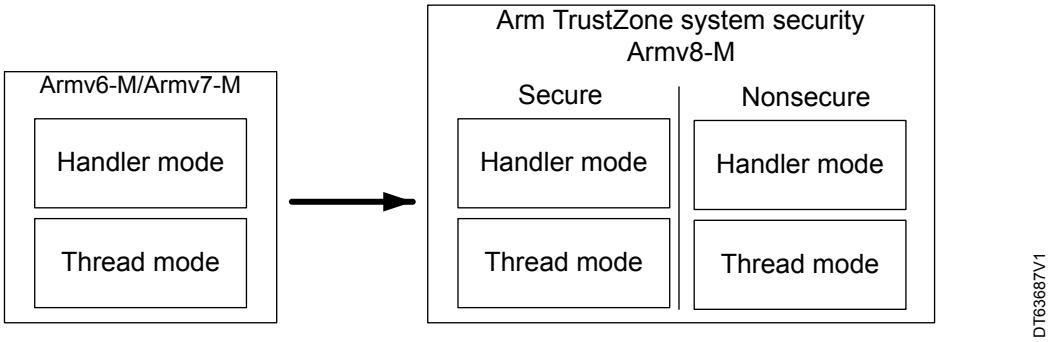
The Armv8-M architecture brings a new security paradigm in Arm microcontrollers. It implements the TrustZone® technology at microcontroller system level, allowing the development of trusted firmware through a robust isolation at runtime.

The TrustZone® technology relies on a processor (Cortex-M23 or Cortex-M33) separation for secure and nonsecure domains and on a bus infrastructure (AMBA®) 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 nonsecure domain and vice versa is straightforward with few cycle penalties. There is no need for a Secure monitor 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 3. Armv8-M TrustZone® execution modes



On typical firmware architecture running on Armv8 TrustZone®, the nonsecure domain executes the application and the OS tasks, while the secure domain executes the secure application and the system root-of-trust mechanisms.

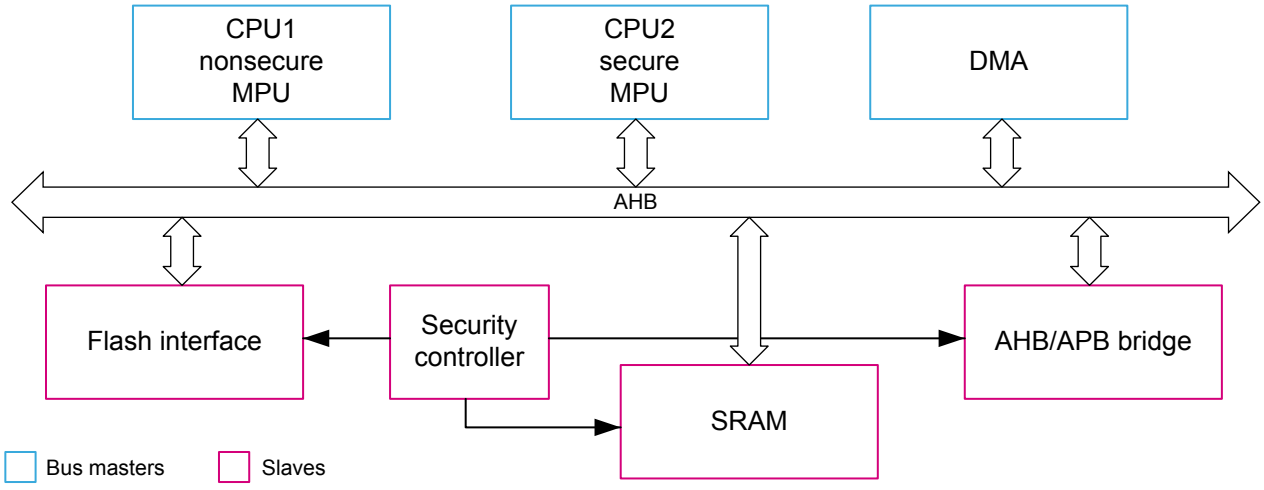
4.3 Dual-core architecture

In dual-core products, one core can act as secure while the other is nonsecure. Some products, such as dual-core STM32WL devices in particular, feature a hardware support to propagate the secure attribute to memory and peripherals, ensuring that robust runtime isolation implementation is possible.

A dedicated security controller is added to the dual-core STM32WL devices to facilitate the isolation.

Instead of using attribution units, the secure NV memory dedicated to securable CPU2 is defined in the flash memory interface configuration. Key peripherals such as the DMA must convey the secure context (refer to the user manual *Getting started with STM32CubeWL for STM32WL series* (UM2643) for details about this hybrid architecture).

Figure 4. Simplified diagram of dual-core system architecture



4.4 Memory protections

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

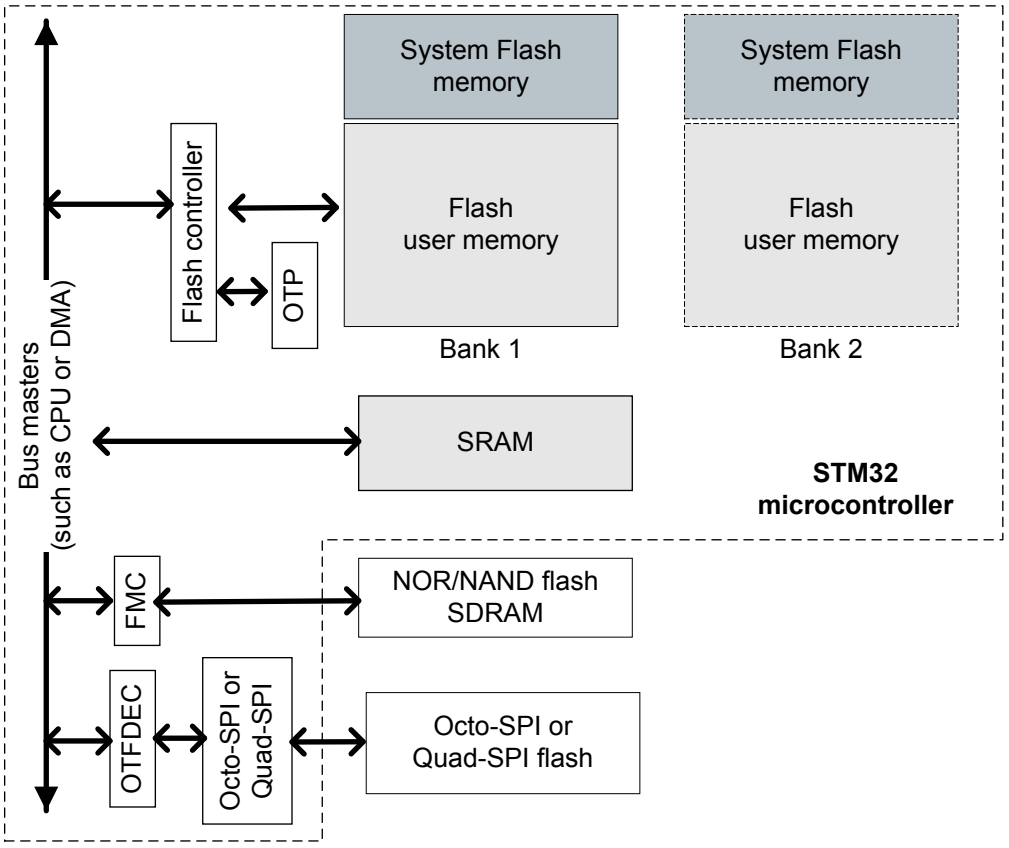
Depending on the asset to be protected (code or data), various mechanisms can be set to establish protections at the source of the unauthorized access (external port, internal process) or on the memory type to be protected (flash, SRAM, or external memory).

Part of the access filtering can be performed by the memory interfaces (like flash controller), the bus controller peripheral (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 designed for different purposes. Their respective protections mechanisms reflect these differences.

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

Figure 5. Memory types



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

Table 5. 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). A part may also be unreadable.
User flash memory	. Internal . NVM	Flash memory for user application	Internal protections: • RDP • WRP (not for SRAM)

Memory	Types	Description	Protections
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.	• TrustZone • PCROP (not for SRAM) • OTP (not in SRAM) • Firewall • Secure hide protection (not for SRAM) • MPU
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

4.4.1 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 include additional 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.

The protection attribute on the system flash memory cannot be modified.

4.4.2 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. Disabling the debugging port access with RDP and the controlled access of connectivity interface provide sufficient isolation from outside.

Associated protection: RDP to disable debug access

Internal attacks

An internal read or write access to the memory can come from a malware injected either in the device SRAM or 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 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 sometimes feature ECC that allows error detection and correction (up to 2-bit error detection and 1-bit error correction). More considered as a safety feature, it also works as a complementary protection against fault injection.

4.4.3 Embedded SRAM

The embedded SRAM is the device working memory. It is used for 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 to execute code from the SRAM is when using encrypted external flash memory on devices without on-the-fly decryption: the code is decrypted inside the SRAM before its execution. Appropriate memory protections must then be enabled on the SRAM address range containing the code. 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 can 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 immediately 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 the reference manual). For some devices, 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. This protection is only featured by the SRAM regions, which are used primarily for code execution (this protection is not practical for data).
The SRAM write protection is available for SRAM2 region on some STM32 MCUs only
(refer to Section 6.1: Overview of security features and to the reference manual).

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

Parity check and ECC

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. ECC is more sophisticated, with SECDED functionality, but only available for SRAM on certain MCU devices. Integrity protections based on redundancy often cannot be disabled.

4.4.4 External flash memories

The external flash memories are connected to the microcontroller through dedicated interfaces (NAND, NOR, Octo-SPI, 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. If the decrypted code or parts of it are not protected from readout (RDP2), then the confidentiality of the code is violated. It is also recommended to combine encryption with integrity protection.

Associated protection: OTFDEC

4.4.5 STM32 memory protections

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 6. Scope of STM32 embedded memory 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(1)	Yes	Yes (read/write)	Yes	No
WRP(2)	Yes	Yes	Yes	No
HDP	Yes	Yes	Yes	Yes (for execution)(3)
TrustZone®	Yes	Yes	Yes	Yes

1. *Support of this feature in Armv6-M products is limited as constant data cannot be stored in NVM.*
2. *Write protection can be unset when RDP level ≠ 2.*
3. *The SRAM is protected by a secure area only at secure code execution. It must be cleaned before leaving the secure area.*

4.5 Software isolation

The software isolation refers to a runtime mechanism protecting different processes from each other (interprocess protection). These processes can be executed sequentially or concurrently (for example tasks of operating system). The software isolation in the SRAM ensures that respective stack and working data of each process cannot be accessed by the other processes. This interprocess protection can be extended to the code in the flash memory and nonvolatile data as well.

Goals of the software isolation:

- Prevent a process to spy the execution of another sensitive process.
- Protect a process execution against a stack corruption due to memory leaks or overflow (incorrect memory management implementation).

This memory protection can be achieved through different mechanisms listed in the table below, and detailed in Section 6: STM32 security features .

Table 7. 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(2)
TrustZone®/Security attribute	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).*
2. *Reading the CPUID indicates which CPU is currently executing code. An example can be found in the HAL_GetCurrentCPUID function.*

4.6 Debug port and other interface protection

The debug ports provide access to the internal resources (core, memories, and registers) and must be disabled in the final device. 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 5.3.1: Secure boot (SB)), or preferably 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 application transfer protocol must limit its access capabilities (such as operation mode or address access range).

Associated STM32 features:

- read protection (RDP)
- disable of unused ports
- bootloader access forbidden (configured by RDP in STM32 devices)

4.7 Boot protection

The boot protection secures the very first software instructions in a system. If an attacker succeeds in modifying the device boot address, he/she can execute his/her 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

4.8 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. 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 can be unintentional (safety) but can 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. Internal tamper sensors are capable of detecting irregular voltage, temperature, or other parameters.

Clock security system is used to protect against external oscillator failures. If a failure is detected on the external clock, the microcontroller switches to the internal clock in order to safely execute. The interrupt signal allows the firmware to react to the clock failure event.

Power supply and voltage level can be monitored to detect abnormally-low voltage level. Below a certain voltage value, the normal behavior cannot be guaranteed and it may be the sign of a fault injection attack.

Device temperature can be measured with an internal sensor. The information is feedbacked to the device through an internal ADC channel. A monitoring application can take appropriate actions according to the temperature range. Increased temperature may be part of a fault injection attack scheme.

Associated STM32 features:

- tamper protection (with RTC component)
- clock security system
- power supply supervision
- temperature sensor

5 Secure applications

In order to create a secure system, the hardware features must be used in a secure firmware architecture implementation. An industry standard solution is the PSA, proposed by Arm for the IoT ecosystem. The STMicroelectronics proprietary solution is Secure boot (SB) and Secure firmware update (SFU). It is possible to use Secure firmware installation (SFI) to securely provision blank devices in manufacturing.

This section defines the **root and chain of trust** concept before presenting the following typical secure applications implementing the features listed below:

- Secure boot
- Secure firmware update
- Secure storage
- Cryptographic services

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 document [9] provides an implementation example of SB and SFU (www.st.com/en/product/x-cube-sbsfu).*
- *The user manual 'Getting started with STM32CubeL5 TF-M application' (UM2671) describes an example of TF-M implementation with the STM32L5 Series MCU.*
- *The user manual 'Getting started with STM32CubeU5 TF-M application' (UM2851) describes an example of TF-M implementation with the STM32U5 Series MCU.*

5.1 Secure firmware install (SFI)

In a mass production scenario, there may be concerns to get secure binaries to the parts without exposing them to an untrusted environment.

In the SFI scenario, the binaries are encrypted using STM32 Trusted Package Creator software tool and sent to HSM within the production facility, to install the code in the microcontrollers.

Note: See AN5391, AN5054, and AN4992 for more information.

SFI is supported on STM32L4, STM32L5, STM32U5, STM32H5, and STM32H7 series.

5.2 Root and chain of trust

The principle of root and chain of trust is common to many secure systems. It is obviously scalable ad libitum, inherently efficient and also flexible.

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 anchor at the beginning of the chain on which the overall security depends.

The secure boot implementation must be the single entry point to the device, start after reset with immutable code in secure mode. It then authenticates a subsequent functionality and executes the next part of the firmware that enables the additional functionality required to securely attest the following chain link. For example, it configures volatile memory protection, so that a secure storage service can use it.

5.3 STMicroelectronics proprietary SBSFU solution

Secure boot and secure firmware update are complementary security concepts. The associated model implementation can be found in the X-CUBE-SBSFU package.

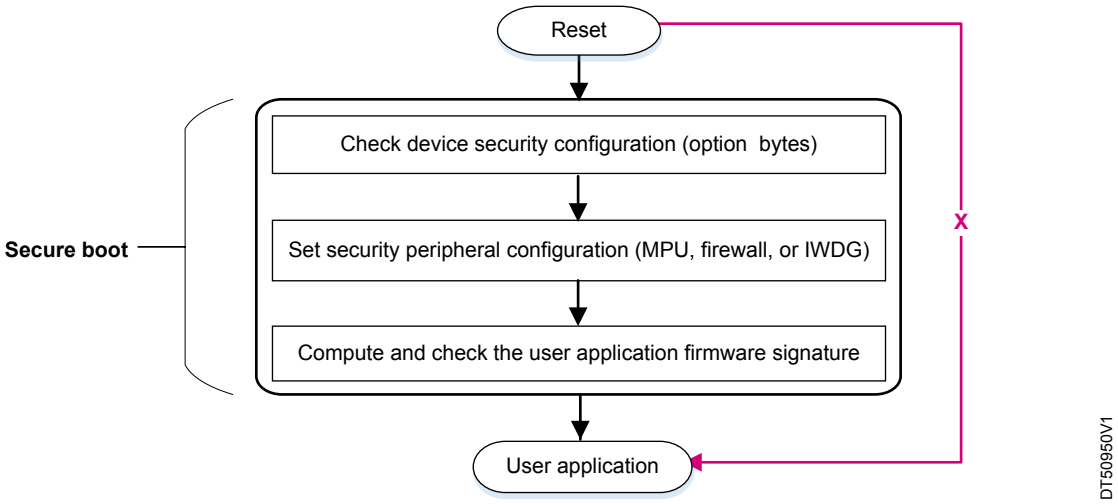
5.3.1 Secure boot (SB)

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 6. Secure boot FSM



Device security check

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 HDP). 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, MP, U or secure hide protection. the implementation depends on the STM32 available features.

5.3.2 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.

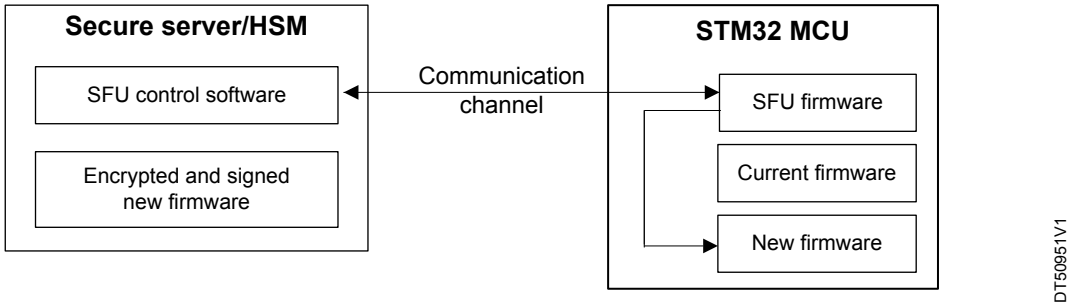
The firmware update is a sensitive operation that must protect two parties:

- the device owner: the goal is to avoid loading a corrupted firmware (intentionally or not) that can 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 nonsecure since it is subject to eavesdropping, the overall security responsibility is shared between the sender (firmware owner server) and the receiver (the device).

Figure 7. 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)

5.3.3 Configurations

The ST proprietary SBSFU is very configurable. The most important configuration option is the choice to use a single or dual image handling of application code. Each has a separate example. Single image leaves more space for application code. Two or more images add some advanced features to the image handling.

The second most important option is the cryptographic scheme selection. There are usually the following choices:

- ECDSA asymmetric cryptography for firmware verification with AES-CBC or AES-CTR symmetric cryptography for firmware encryption
- ECDSA asymmetric cryptography for firmware verification without firmware encryption
- X509 certificate-based ECDSA asymmetric cryptography for firmware verification without firmware encryption
- AES-GCM symmetric cryptography for both firmware verification and encryption

For more details, see the document [9] or the document *Integration guide for the X-CUBE-SBSFU STM32Cube Expansion Package* (AN5056).

5.4 Arm TF-M solution

Arm trusted firmware existed for chips based on Cortex-A when the secure Cortex-M33 core was introduced with Armv8-M architecture. A more compact TF-M open source implementation of PSA standard was provided as a reference secure firmware framework.

For STMicroelectronics MCUs that take advantage of the Armv8 architecture (such as STM32H5, STM32L5, and STM32U5 devices), the SBSFU is replaced with the TF-M based solution.

For a documentation on TF-M itself, refer to the UM2851, and use Arm resources as well as the code comments.

For guidance on TF-M integration on the STM32L5 and STM32U5 devices, refer to the user manuals *Getting started with STM32CubeL5 TF-M application* (UM2671) or *Getting started with STM32CubeU5 TF-M application* (UM2851).

Refer to document [7] for a detailed comparison when migrating from X-CUBE-SBSFU package SBSFU to TF-M.

Table 8. **Basic feature differences of TrustZone-based secure software**

Feature	SBSFU for TrustZone®	TF-M
RoT services	Immutable RoT	Immutable RoT + updatable RoT
Cryptographic key management	Static keys only	Key storage hierarchy with HUK root key
Secure storage	Absent	Internal and external
NV counter	No	Yes

Both alternatives are based on TF-M and MCU boot, but while SBSFU intends to replicate familiar features of X-CUBE-SBSFU while retaining most flash memory space for user code, TF-M offers more functionality. Some of that can be dropped to gain memory space. For the STM32H57x line, the Secure manager, a closed-source implementation of TF-M, offers a convenient and express way to adopt certified secure solutions.

5.5 Secure manager

The goal of the Secure manager is to guide customers to a certified secure software solution. Secure manager implements the PSA secure APIs (similar to the TF-M) and by using them, the user nonsecure application can form a secure system with certified attack resistance.

The Secure manger application kit is distributed in the form of encrypted binary and is free to use on compatible STM32H5 MCUs.

Secure manager can expand its secure functionality by using modules.

5.6 Product certifications

Different secure applications often require certain certifications, proving their capability to perform in a secure manner. Independent or government agencies grant the certification status to either an MCU application or a combination of MCUs with secure firmware after testing it against the evaluation goals.

The certifications and evaluations related to STM32 microcontrollers include, but are not limited to:

- PSA certified (platform security architecture), governed by Arm, focused on IoT security, MCU certification, three levels of assessment
  - STM32L4 devices are certifiable up to Level 1.
  - STM32L5 devices with TF-M are certifiable up to Level 2.
  - STM32U5 and STM32H5 devices with TF-M are certifiable up to Level 3.
  - To achieve Arm PSA certifiable security level, refer to the user manual *STM32U585 security guidance for PSA Certified™ Level 3 with SESIP Profile* (UM2852).
- SESIP (security evaluation standard for IoT platforms), international methodology adopted by several major security evaluation labs, five levels
  - Systems using SBSFU or TF-M are compliant to Level 3 with STM32L4, STM32L4+, STM32L5, STM32H5, and STM32U5 devices.
- PCI (payment card information), important security standard focusing on point of sale (POS) applications
  - Good record of successful evaluation of systems, using for example, STM32L4 devices

Note: FIPS (Federal Information Processing Standards) is a set of standards published by NIST, some of which (FIPS 140, SP800) are related to security or cryptography.

Table 9. Certifications coverage

MCU series	Crypto conformance certification	Security system certification	Security system certification with physical resistance (invasive)
STM32G0/ STM32G4	FIPS CAVP, SP800-90A	PSA L1	-
STM32L4			
STM32L4+	FIPS CAVP, SP800-90B		
STM32H7			
STM32L5		PSA L2, SESIP L3	
STM32WBA		Yes	PSA L3, SESIP L3 (with TF-M)
STM32U5			
STM32H5			PSA L3, SESIP L3 (with Secure manager)

6 STM32 security features

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

6.1 Overview of security features

6.1.1 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 (or product state), PCROP, WRP, BOR, OTP, and secure hide protection (when available).
- **Dynamic** (or run time) protections do not retain their status at reset. They have to be configured at each boot (for example during Secure boot (SB)).
  Dynamic protections provided by STM32 include 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).

6.1.2 Security features by STM32 devices

Following tables are intended for reference and the prospects of secure code porting between STM32 products. Presence or absence of a particular feature may not be indicative of achievable security. For that reason, refer to the available security certifications for the particular product.

Table 10. Security features for STM32C0, STM32F0/1/2/3/4, STM32G0/4 devices

Feature	STM32C0	STM32F0	STM32F1	STM32F2	STM32F3	STM32F4	STM32G0	STM32G4
Cortex core	Cortex-M0+	Cortex-M0	Cortex-M3	Cortex-M3	Cortex-M4	Cortex-M4	Cortex-M0+	Cortex-M4
RDP additional protection	Bad OBL recovery	Backup registers	2 level RDP only	Backup SRAM	Backup registers	Backup SRAM	Backup registers	Backup registers, CCMSRAM
Flash WRP	By area with 2-Kbyte granularity, two areas available	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 area with 2-Kbyte granularity, two areas available	By page (2 K or 4 Kbytes)
SRAM WRP	No	No	No	No	No	No	No	CCM SRAM, with 1-Kbyte granularity
PCROP	By area with 256-byte granularity, one area per bank	No	No	No	No	By sectors	By area with 512-byte granularity, two areas available	By area with 64- or 128-bit granularity, up to two areas
HDP	Yes (securable memory area)	No	No	No	No	No	Yes (securable memory area)	
Firewall	No	No	No	No	No	No	No	No
MPU	Yes	No	Yes(1)	Yes	Yes(2)	Yes	Yes	Yes
OTP	1 Kbyte	No	No	Yes	Yes	512 bytes	1 Kbyte	1 Kbyte
UBE(3)	Yes (boot lock feature)	No	No	No	No	No	Yes (boot lock feature)	Yes
Internal tamper detection	No	No	No	No	No	No	Yes	Yes
Hardware crypto(4)	No	No	No	AES, HASH	No	AES, HASH	AES	
RNG	No	No	No	SP800-90-A	No	SP800-90-A		

Feature		STM32C0	STM32F0	STM32F1	STM32F2	STM32F3	STM32F4	STM32G0	STM32G4
Secure software	SBSFU	No	No	No	No	No	Yes	Yes	Yes
	TF-M	No	No	No	No	No	No	No	No
	KMS	No	No	No	No	No	No	No	No

1. *Only XL density devices feature the MPU.*
2. *MPU is not universally supported in STM32F3 series. Refer to the product datasheet to confirm availability.*
3. *Unique boot entry.*
4. *For most devices with hardware crypto capability, there is a direct equivalent without it.*

Table 11. Security features for STM32L0/1/4/4+, STM32WB, STM32WBA, STM32WB0x, STM32WL devices

Feature		STM32L0	STM32L1	STM32L4	STM32L4+	STM32WB	STM32WBA	STM32WB0x	STM32WL
Cortex core		Cortex-M0	Cortex-M3	Cortex-M4		Cortex-M4/ Cortex-M0+	Cortex-M33	Cortex-M0+	Cortex-M4/ Cortex-M0+
RDP additional protection		EEPROM		Backup registers, SRAM2			RDP four levels,backup registers, SRAM	No	Backup registers, SRAM2
Flash WRP		By sectors (4 Kbytes)		By area with 2-Kbyte granularity, one area per bank		By area with 4-Kbyte granularity, two areas available	Two areas, defined by page range	By area with 2-Kbyte granularity, two areas available	
SRAM WRP		No	No	SRAM2, with 1-Kbyte granularity				No	SRAM2, with 1-Kbyte granularity
PCROP		By sectors		By area with 8-byte granularity, one area per bank		By area with 2-Kbyte granularity, up to two areas	No		By area with 1-Kbyte granularity, two areas available
HDP		No	No	No	No	Yes (dedicated to Cortex-M0+ firmware only)	Yes	No	Yes
Firewall		Yes	No	Yes	Yes	No	No		Yes
MPU		Yes	Yes	Yes	Yes	Yes (Cortex-M4)	Yes	Yes	Yes
OTP		No	No	1 Kbyte					
UBE[1]		No	No	No	No	No	Yes (lockable secure and NS address)	No	Yes (boot lock feature)
Internal tamper detection		No	No	No	No	No	Yes	No	Yes
Hardware crypto[2]		AES		AES, HASH	AES, HASH, (some PKA)	AES, PKA	AES, HASH, PKA	AES[3], PKA	
RNG		No	No	SP800-90-A	SP 800-90-B	SP800-90-A	800-90-B	SP 800-90-B	
Secure software	SBSFU	Yes	Yes	Yes	Yes	Yes	Yes	Yes[4]	Yes
	TF-M	No	No	No	No	No	Yes	No	No
	KMS	No	No	Yes	Yes	No	No	No	No

1. *Unique boot entry.*
2. *For most devices with hardware crypto capability, there is a direct equivalent without it.*

3. *For STM32WB0, the AES is only available on the radio peripheral.*
4. *Secure boot solution separated from the usual X-CUBE-SBSFU package.*

Table 12. Security features for STM32L5, STM32U0, STM32U5, STM32H503/5, STM32H7R/S, STM32H72x/73/74x/75, STM32H7Ax/7Bx, STM32F7 devices

Feature		STM32L5	STM32U0	STM32U5	STM32 H503	STM32H5	STM32 H7R/S	STM32 H72x/73x	STM32 H74x/75x	STM32 H7Ax/7Bx	STM32F7
Cortex core		Cortex-M33	Cortex-M0+	Cortex-M33			Cortex-M7				
RDP additional protection		RDP four levels, backup registers, SRAM2	Backup registers, SRAM2	RDP four levels, backup registers, SRAM3	Product state instead of RDP		Product state instead of RDP, backup SRAM	Backup SRAM, backup registers, OTFDEC	Backup SRAM, backup registers	Backup SRAM, backup registers, OTFDEC	Backup SRAM
Flash WRP		Up to four protected areas with 2-K or 4-Kbyte granularity	Two areas per bank defined by page range				By sectors (8 Kbytes)	By sectors (128 Kbytes)		By group of 4 8-Kbyte sectors	By sectors (16 K, 64 K, 128 K, or 256 Kbytes)
SRAM WRP		SRAM2, with 1-Kbyte granularity	No	SRAM2, with 1-Kbyte granularity			No	No	No	No	No
PCROP		No (replaced by TrustZone)	No	No (replaced by TrustZone)	No	No (replaced by TrustZone)	No	By area with 256-byte granularity	By area with 256-byte granularity, one area per bank		By sectors
HDP		Up to two secure hide areas (HDP) inside the TrustZone secure domain	Yes, with second stage extension	Up to two secure hide areas (HDP) inside the TrustZone secure domain	3-stage temporal isolation, one per bank			Yes (secure user memory, with 256-byte granularity)			No
Firewall		No (replaced by TrustZone)	No	No (replaced by TrustZone)	No	No (replaced by TrustZone)	No	No	No	No	No
MPU		Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
OTP		512 bytes	1 Kbyte	512 bytes	2 Kbytes		1 Kbyte	No	No	No	No
UBE[1]		Yes (boot lock feature)			Yes	Yes	Yes (boot lock feature)	Yes (unique entry point in secure access)			No
Internal tamper detection		Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Hardware crypto[2]		AES, HASH, PKA	AES	AES, HASH, PKA	HASH	AES, HASH, OTFDEC, PKA	AES, HASH, PKA	AES, DES, HASH, OTFDEC			AES, HASH
RNG		SP 800-90-B							SP800-90-A[3]	SP 800-90-B	SP800-90-A[3]
Secure software	SBSFU	No	Yes	No	No	No	Yes	Yes	Yes	Yes	Yes
	TF-M	Yes	No	Yes	No	Yes	-	No	No	No	No
	KMS	No	No	No	No	Yes	-	No	No	No	No

1. *Unique boot entry.*
2. *For most devices with hardware crypto capability, there is a direct equivalent without it.*
3. *RNG v2.0, can be used as a seed source for DRBG.*

6.2 Readout protection (RDP)

The 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.

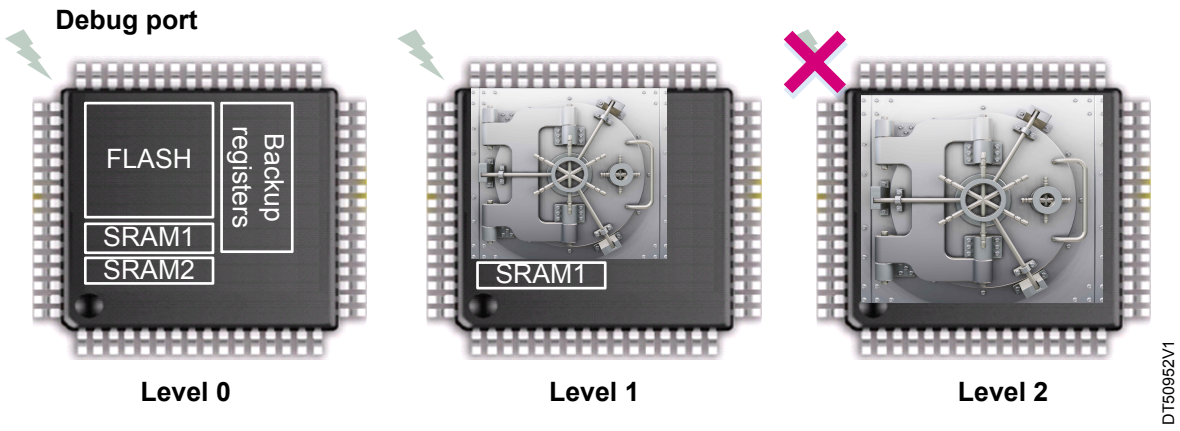
The RDP applies to all STM32 devices for:

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

Depending on the STM32 device, additional protections are available, including:

- backup registers for real-time clock (RTC)
- backup SRAM
- Nonvolatile memories

Figure 8. Example of RDP protections (STM32L4 series)



The RDP levels are defined as follows:

- **Level 0**(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**
  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**
  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.

A fourth RDP level is available for devices built on Armv8 architecture:

- **Level 0.5**(for nonsecure debug only)
  All read and write operations (if no write protection is set) from/to the nonsecure flash memory are possible. The debug access to secure area is prohibited. Debug access to nonsecure 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 nonsecure 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, with exception of preconfigured OEM keys in STM32Ux series, no regression is possible.

Internal flash memory content updating on an RDP protected STM32 MCU

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. Practical implementations of such firmware updates are SFU (secure firmware update) and IAP (in-application-programming). See examples in related documents AN4657, AN5056, AN5544, and AN5447 to learn more.

Regression locked by OEM keys

The STM32U0 and STM32U5 series introduce the possibility to define secret keys (passwords) to allow controlled regression (for example for fault analysis purposes). The keys are programmed in similar way as a regular option byte, and to allow regression, they need to be presented to the debug interface in a dedicated connection mode, which prevents concurrent connection for normal debug session.

There are separate keys for regression from RDP2 and from RDP1.

Regression is supported in the STM32CubeProgrammer utility both in GUI and CLI.

The table below summarizes the RDP protections.

Table 13. RDP protections

Area	RDP level	Boot from user flash memory			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 to use the RDP

On a consumer product, the 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 mandatory to implement an application with higher security level (such as immutable code). The drawback is that the RDP level 2 can prevent a device examination, for instance after a customer return.

The RDP level 0.5 is used to debug a nonsecure application, while protecting contents within secure area boundaries from debug access. Refer to section 'Development recommendations using TrustZone®' of the application note *Arm® TrustZone® features on STM32L5 and STM32U5 series* (AN5347) for more information about this protection.

Note: The RDP is available on all STM32 device, unless succeeded by the lifecycle management product state (see Section 6.3).

6.3 Lifecycle management–product state

The addition of RDP 0.5 into the RDP mechanism used by the STM32 enabled the necessary isolation between secure and nonsecure development. However, the RDP does not allow going further in the user experience with the adoption of new development and OEM manufacturing models. The RDP has been replaced by the product state, a more refined lifecycle management system, on the STM32H5 devices as a pilot project. The product state is also an answer to the needs of customers requesting a state that is effectively an RDP2 to the outside world and it allows them to perform a regression in the controlled environment. A similar provision was also added to the STM32U5 series, but the product state enabled finer control over delegating the debugging rights in TrustZone enabled products. It is sufficient for microcontrollers with no TrustZone isolation to use only two states, open and closed, for example, the STM32H7Rx/7Sx which implements the product state this way.

The new lifecycle management defines a set of permitted states, and the possible transition between them, just as in the case of the RDP, but there are more states defined for the following:

- Provision with immutable root of trust code.
- Clear separation of nonsecure and secure environment.
- Use certificates for controlled regression.
- Use certificates for temporary debug sessions.

For applications that do not need these new options, there are still product states corresponding to the original RDP values.

The main available product states are:

- Open: roughly equivalent to RDP 0
- Provisioning: marks an ongoing installation of iRoT
- iRoT-provisioned: roughly comparable to RDP 0.5
- TZ-closed: state in which debugging of nonsecure code is permitted
- Closed: equivalent of RDP2, but the regression is possible (only with a valid authorization)
- Locked: final state with no possibility of further transition (like RDP2)

6.4 Unique boot entry

If the secure boot firmware is located in the user flash, it is important to prevent the attacker from diverting the boot sequence from the intended progression. This is the goal of the unique boot entry or BOOT_LOCK.

The RDP2 on some products prevent booting from other memory. The BOOT_LOCK goes further by also working in RDP1 and working with the product state life-cycle management. In some devices the BOOT_LOCK allows the possibility to lock the boot to a specific address.

Even if RDP2 is set; the unique boot entry provides additional security assurance against glitch attacks.

A unique boot entry is available on STM32Ux, STM32Gx, STM32Hx, STM32Cx series, and in STM32WBA wireless devices. Implementation details are available in each device reference manual, on the sections about boot configuration and flash.

6.5 One-time programmable (OTP)

The OTP is a dedicated, isolated area in the flash memory, which can be only written on or locked out, preventing any modification. It is usually a smaller area compared to the size of the user flash memory.

This feature is very useful for lifecycle management, provisioning, personalization, or configuration. Once the OTP is written, there is no method of erasing data without physically damaging the device. No restriction is implicitly put on reading written data.

Note: The OTP is available on most STM32 devices (refer to Section 2: Overview for more details).

6.6 TrustZone®

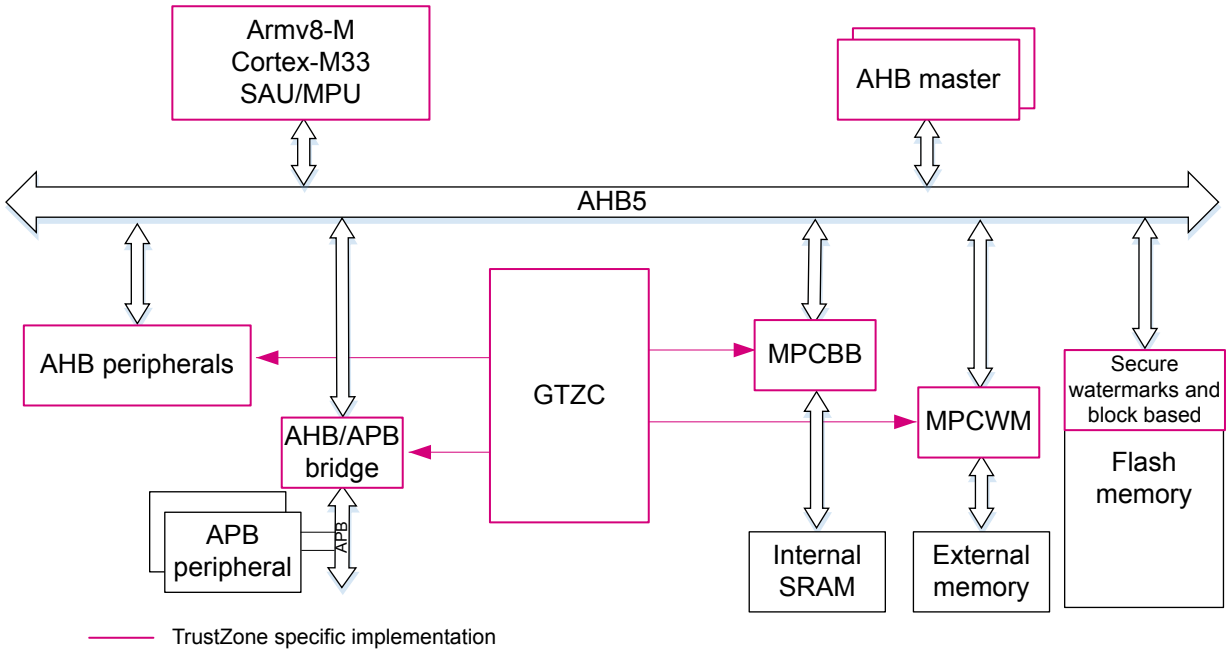
This section describes the main features of the TrustZone® architecture. For further information, refer to the application note *Arm® TrustZone® features on STM32L5 and STM32U5 series* (AN5347), and to the device reference manual.

The Armv8-M TrustZone® architecture defines two domains at system level: secure and nonsecure. The full memory-map space is split into secure and nonsecure areas. This includes all memory types (flash memory, SRAM, and external memories), as well 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 nonsecure domains relies on the following hardware mechanisms (see Figure 9):

- specific core architecture (Armv8-M Cortex-M33) with a dual execution domain for secure and nonsecure domains, and an implementation defined attribution unit (IDAU) to assert address range security status
- secure attribution unit (SAU) is used to refine settings of the IDAU
- bus infrastructure that propagates the secure and privilege attributes of any transaction (AHB5)
- dedicated hardware blocks managing the split between the two domains (GTZC to define security attribute for internal SRAMs and external FSMC/OCTOSPI memories, and peripherals)

Figure 9. TrustZone® implementation at system level



6.6.1 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 nonsecure state.

6.6.2 Secure attribution unit (SAU)

The SAU is a hardware unit coupled to the core (as the MPU). The SAU is responsible for setting the security 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 is tagged as secure, nonsecure callable (NSC), or nonsecure. The NSC is a subdomain of the secure domain, that allows a gateway to be defined for nonsecure code to access the secure domain at a specific entry point.

The SAU is configurable by secure 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 modified to be less secure (by security order: secure > NSC > nonsecure) than a default attribute set by hardware through an IDAU (implementation defined secure attribute).
Refer to implementation details of each device in the reference manual.

Address aliasing

The security attribute is set depending on the fixed resource address. However, a memory-mapped resource can be set either as secure or nonsecure, depending on the application. To overcome this apparent contradiction, two addresses are assigned to each memory-mapped resource: one used when the resource must be accessed in secure mode, one used in nonsecure mode. This mechanism is called address aliasing.

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

- peripherals secure/nonsecure regions
- flash memory secure/nonsecure regions
- SRAM secure/nonsecure regions

Refer to device reference manual for the detailed configuration.

6.6.3 Memory and peripheral 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 privileged 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, system configuration).
The remaining system peripherals are securable.

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

- Peripherals can be set as secure or nonsecure (exclusively), privileged or unprivileged using TZSC.
- Embedded SRAM is protected by blocks of 256 bytes through the MPCBB.
- External memories are protected by regions (watermark: start and length). The number of protected regions depends on the memory types (NAND, NOR, or OCTOSPI).
- Illegal access events lead to secure interrupts generated by TZIC.

Note: The flash memory security attribute is defined through secure watermark option bytes, and/or flash memory interface block-based registers.

6.7 Flash memory write protection (WRP)

The write protection feature is used to protect the content of the 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 to use the WRP

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. The WRP prevents wrong accesses due to unsafe functions causing unexpected overflows.

Note: The WRP is available on all STM32 devices.

6.8 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 an STM32 device, 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 device (see Section 6.1.2: Security features by STM32 devices). When the PCROP is in use, care must be taken to compile the firmware with the execute-only attribute (refer to user compiler options). Refer to the document [3] for more details.

In particular, the ARMv6-M instruction set has difficulty working with this constraint. The compiler must be set not to place constants and literal pools within the program, otherwise even the code execution can eventually trigger the protection.

When to use the PCROP

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

Note: The PCROP is available on all STM32 devices listed in Table 1, except on TrustZone-enabled devices, where it is superseded by another protection mechanism.

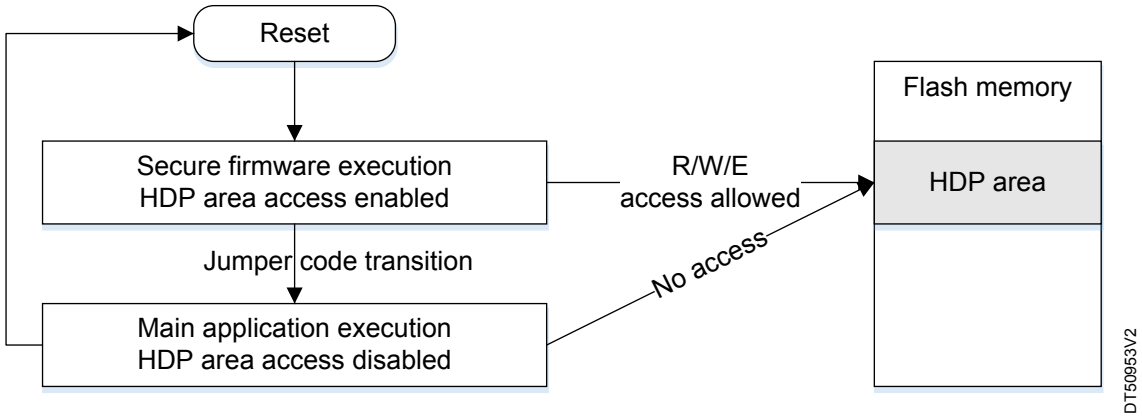
6.9 Secure hide protection (HDP)

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

An HDP area is a part of the user 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 transition is executed, the HDP area is closed, and cannot be accessed anymore by any means (see the figure below).

The code that facilitates the HDP transition is usually an STMicroelectronics code located in the RSS. Code in SRAM is an alternative.

Figure 10. 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, independent of the boot configuration set by boot pin or boot address. HDP can be single stage or use a monotonic counter to gradually cover more memory as the secure boot progresses (STM32H5). Some devices implement a dynamic HDP expansion. This means that the HDP can be extended by the number of sectors using the register value, without programming the OB (STM32H5, STM32U0).

When to use the HDP

The HDP is suited for a code that must only be executed after reset, like secure boot for root of trust. It should be used in synergy with the boot lock feature.

Note: The HDP is available in STM32H7, STM32G0, STM32G4, STM32L5, STM32U0, STM32U5, and STM32H5 devices, with slight differences in its implementation and name (refer to the reference manuals for details).

6.10 Firewall

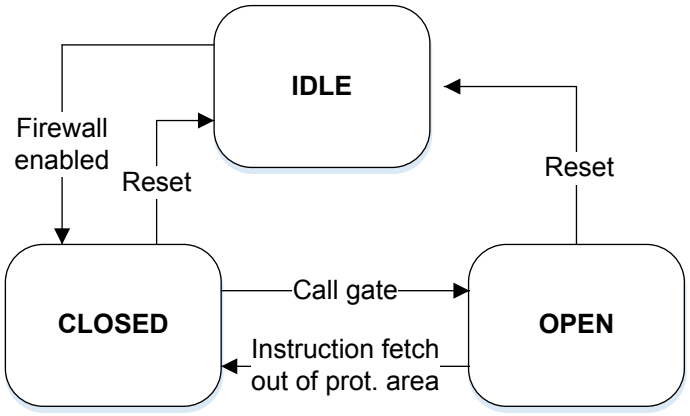
The firewall is a hardware protection peripheral controlling the bus transactions and filtering accesses to three particular areas: a code area (flash memory), a volatile data area (SRAM) and a nonvolatile data area (flash memory). The protected code is accessible through a single entry point (the call-gate 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 an SB application).

Call gate mechanism

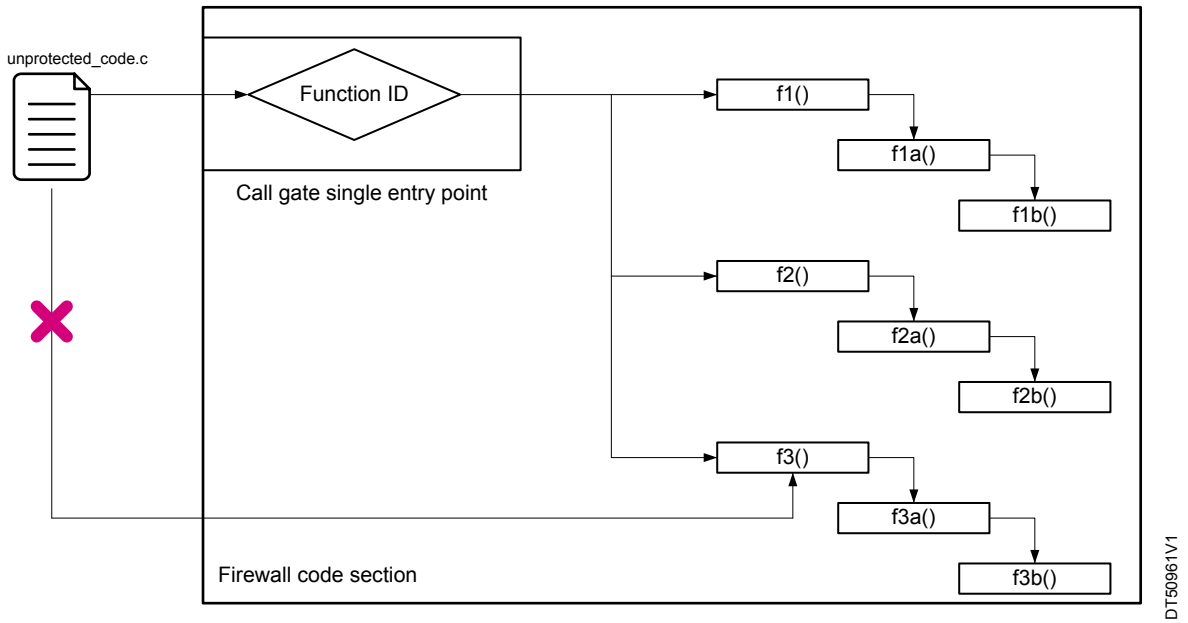
The firewall is opened by calling a 'call-gate' mechanism: a single entry point that must be used to open the gate, and to execute the code protected by the firewall. If the protected code is accessed without passing through the call gate mechanism, then a system reset is generated. If any instruction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

Figure 11. 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 12. **Firewall application example**



When to use the firewall

The firewall protects both code and data. The protected code can always be called as long as a call gate mechanism is respected.

Note: A firewall is available on STM32L0 and STM32L4 devices only. Refer to the application note AN4730 for more details. The firewall functionality is replaced by TrustZone.

6.11 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 master requests (such as DMA) 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. This application note does not cover the whole complexity of the MPU. This section provides only an introduction and high-level overview. Refer to applicable programming manual, or to the document [5].

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. *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 requiring access to sensitive resources that are otherwise inaccessible to unprivileged firmware.

Secure process isolation strategy

At reset, the privileged mode is the default one for any process. The SB application is then executed in privileged mode. 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 can be updated each time the OS switches from one task to another.

When to use the MPU

The MPU is used at runtime to isolate sensitive code, and/or to manage access to resources depending on the process currently executed by the device. This feature is useful especially for advanced embedded operating systems that incorporate security in their design.

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

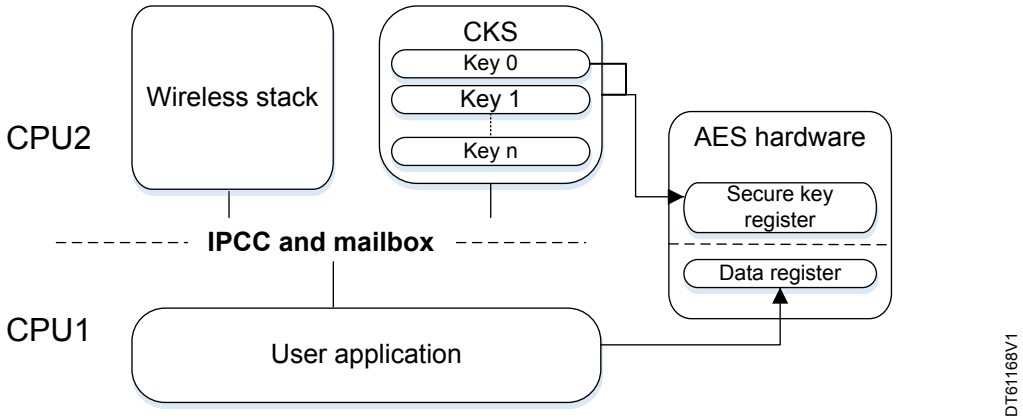
6.12 Customer key storage (CKS)

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, ZigBee, or thread protocols). The flash memory used by CPU2 is isolated from the CPU1 or external access. Communication between the two cores is ensured by a mailbox and an interprocess channel control hardware block (IPCC).

In addition to the wireless stack execution, the CPU2 offers a secure storage service for cryptographic keys used with a dedicated AES hardware peripheral (see Figure 13). The AES key register is accessible only to the CPU2, preventing access to the key by an untrusted process running on the CPU1, or by the debug port.

After the 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 13. **Dual-core architecture with CKS service**



When to use the CKS

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

Note: The CKS is available on STM32WB series only.

6.13 Antitamper (TAMP)/backup registers (BKP)

The antitamper is a system level protection, used to detect physical tampering attempts on the system.
An external tamper event is detected by a level transition on dedicated device pins. Internal tamper sensors can check voltage, temperature, or clock. This event can be used to wake up the core in order to take appropriate actions (such as memory erase or alarm).

This TAMP peripheral includes backup registers with contents preserved by V_{BAT}, along with the real-time clock (RTC). These registers can be reset if a tamper attempt is detected.

On some STM32 devices, this peripheral is known as backup registers (BKP). On recent devices, it has evolved with additional features, such as monotonic counter, or secure section for TrustZone® secure area.

When to use the antitamper

It must be used for system intrusion detection (in consumer products sealed enclosures for example).
The monotonic counter is a countermeasure against tampering with the RTC.

Note: The external tamper detection is available on all STM32 devices. More information about tamper functionality usage is available, for example, in AN4759.

6.14 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 is intentional or not. In any case, the device must take appropriate actions to recover. The CSS triggers an interrupt to the core in such event.

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

When to use the CSS

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

Note: The CSS is available on all STM32 devices.

6.15 Power monitoring (PVD)

Some attacks target the MCU power supply to cause errors that lead to a failure of security countermeasures. A loss of power supply sometimes denotes 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 appropriate actions are implemented.

When to use the PVD

The PVD must be used as soon as a sensitive application runs, 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 devices.

6.16 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 memory words, used to recover from a single-bit error, or to detect up to two erroneous bits on each flash memory or SRAM word (32- to 256-bit word depending on the memory type). Refer to the AN5342 for more details.
- A simple parity check allows the detection of a single error bit on the SRAM words where ECC is not implemented.

When to use ECC and parity check

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

Note: This integrity protection is available on all devices except STM32F1 and STM32L1. STM32H7 devices are champion in ECC protection.

6.17 Independent watchdog (IWDG)

The IDWG is a free running down counter that can be used to trigger a system reset when the counter reaches a given timeout value. The IDWG can be used to provide a solution in case of malfunctions or deadlock in running code. The IDWG 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 to use the IWDG

The IWDG can be used to break deadlocks. It can also be used to control execution time of critical code (such as decryption or flash memory programming).

Note: The IDWG is available on all STM32 devices.

6.18 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 for instance to derive a unique key from a master OEM key.

6.19 Cryptography

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

The cryptography related features for each product can be identified via their root part number, specifically on the second number or letter after the series identifier. For example the STM32L4**8**6 and STM32H7**B**3 devices have crypto hardware while the STM32L4**7**6 and STM32H7**A**3 have to use software library to implement cryptography, and it can be known by their naming.

6.19.1 Hardware accelerators

The following cryptographic peripherals are available in STM32 devices:

- TRNG (true random generator)
  - hardware-based peripheral providing a physical noise source. Used to feed a DRNG with a seed, or directly for random numbers, depending on device.
    Note: For details about the RNG validation, refer to the application note 'STM32 microcontroller random number generation validation using the NIST statistical test suite' (AN4230).
- AES accelerator
  - encryption/decryption
  - 128- or 256-bit keys
  - several modes of operation (such as ECB, CBC, CTR, or GCM)
  - DMA support

  Note: When the AES is used for encryption or decryption, the access to its registers containing the key must be protected and cleaned after use (the MPU can be configured to restrict access to memory with keys).
- 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: Many low-power STM32 devices feature hardware cryptography accelerators. The amount of energy needed to complete cryptographic computations with them is lower than the amount of energy required to treat them via software processing.

6.19.2 CryptoLib software library

The STM32 X-CUBE-CRYPTOLIB software library runs on all STM32 devices. It is available for free download at www.st.com/en/product/x-cube-cryptolib. The version V4 is available with full firmware implementation, compiled for Cortex-M0, Cortex-M0+, Cortex-M3, Cortex-M33, Cortex-M4, and Cortex-M7.

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) and ECDSA + ED25519 and Curve 25519

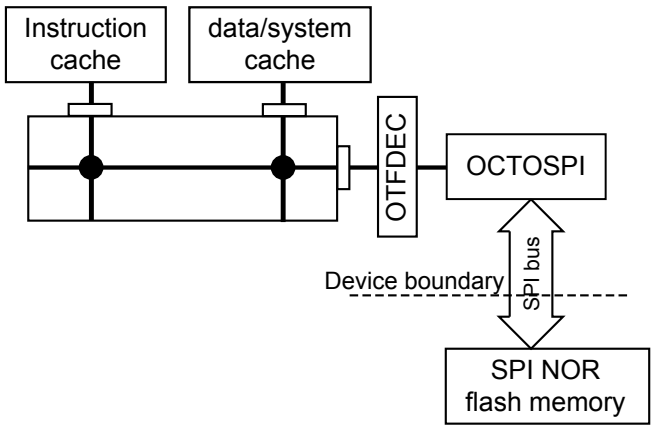
6.20 On-the-fly decryption engine (OTFDEC)

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

One solution is to download the external memory content inside the SRAM, to decrypt it, to execute the code, and/or to use data. There are two drawbacks with this method: it introduces a delay that may not be acceptable, and it uses a large amount of SRAM, depending on the content.

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

Figure 14. **Typical OTFDEC configuration**



The OTFDEC uses the AES-128 CTR mode, with a 128-bit key to achieve a latency below 12 system bus cycles. Up to four independent and nonoverlapping encrypted regions can be defined (4-Kbyte granularity), each with its own key.

When to use the OTFDEC

The OTFDEC is used when an external memory is used by the system. For TrustZone® capable MCUs, the decryption keys can only be made accessible through the secure mode. See the application note *How to use OTFDEC for encryption/decryption in trusted environment on STM32H73/H7B MCUs* (AN5281) for more details.

Note: The OTFDEC is available on STM32H5, STM35H7, STM32L5, and STM32U5 devices only.

7 Guidelines

Secure systems can take advantage of many security supporting hardware feature. Some are useful for any system, and need little change in the application code to be activated and fully functional. 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, depending on the system use-cases. The 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)
	• Use RDP level 2. This closes the device from any external access.
	1.2 Remove debug capability for the device.
	• Use RDP level 2 for permanently disabling the debug.
	1.3 Protect a device against a loss of external clock source (crystal).
	• Enable clock security system (CSS).
	1.4 Detect a system-level intrusion.
	• Use tamper detection capability of the RTC.
	1.5 Protect a device from code injection.
	• Use the RDP. • Isolate communication port protocol with the 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.
	• Use RDP level 1 or 2 against external access. • Use PCROP on most sensitive parts of the code against internal read access. • Use OTFDEC to secure code stored in the external memory.
	2.2 Protect secret data from other processes.
	• 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.
	• 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.
	• 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
	• Implement SB application with cryptography. • Protect SB application secret data (refer to previous sections).

-	• Guarantee unique boot entry on SB application: – Use HDP if available. – Use RDP level 2 and disable boot pin selection.
	3.3 Securely update the firmware in the field.
	• 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.
	• 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.
	• Use only official software implementation by ST with STM32 X-CUBE-CRYPTOLIB.
	4.3 Accelerate AES/DES/SHA cryptographic functions.
	• 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 random data.
	• Use RNG embedded in the STM32 devices.
	4.5 Uniquely identify ST microcontrollers.
	• Use STM32 96-bit unique ID.
	4.6 Authenticate a product device.
	• Embed a shared encryption key in the device, and exchange encrypted message.
	4.7 Uniquely authenticate a device.
	• Embed a device private key and its certificate in the device, and exchange encrypted message.
	4.8 Authenticate communication servers.
	• 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

No system can be made secure by simply enabling security features in the hardware. Security must be rooted in the architecture of the complete solution.

The threats must be identified, the countermeasures correctly designed and implemented in synergy with other security features.

As security demands considerable resources, it is important to correctly evaluate the risks, and spend the resources efficiently, keeping in mind the cost of attack and the value of the protected asset.

The concept of root of trust is pivotal because it uses a more hierarchic and centralized approach, as opposed to attempting to apply security ad hoc.

With the STM32 microcontrollers, the embedded and IoT security is very cost-effective and robust.

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:

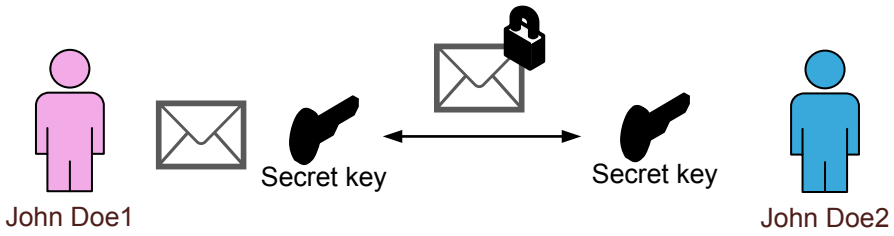
- Secret key/symmetric cryptography
- Public key/asymmetric cryptography
- Hashing

These algorithms are described in this appendix.

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 15. 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).

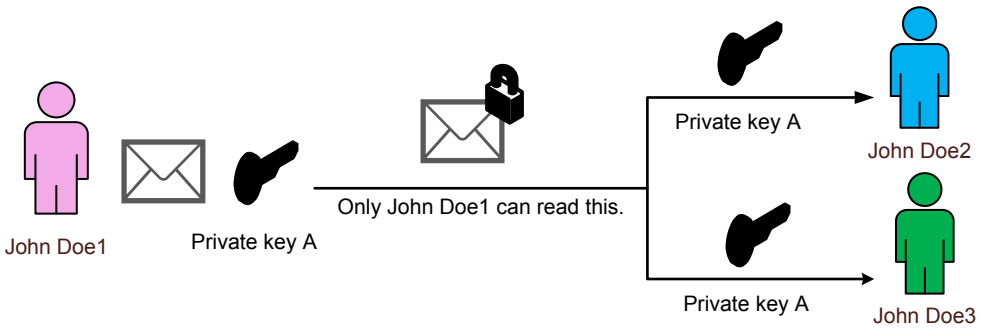
Since these algorithms are deterministic, they always mix input data with a random value, known as nonce, used only for one session as initialization vector.

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):

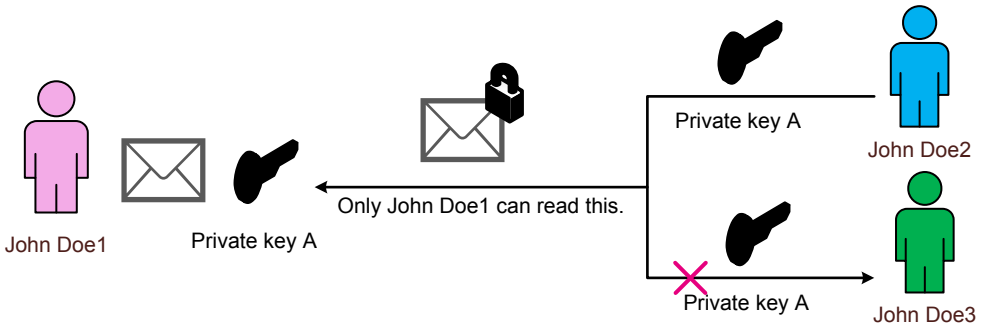
- A message encrypted by the private key can be read by any party with the public key. This mechanism ensures a strong authentication of the sender since the private key has never been shared. Digital signatures are based on this mechanism.

Figure 16. Signature



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

Figure 17. 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:

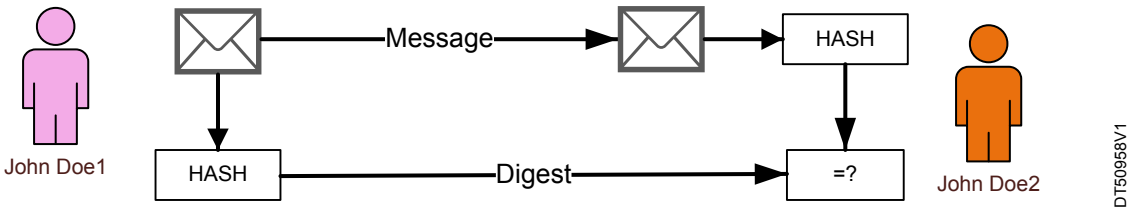
- Asymmetric cryptography is used first, in order to solve the symmetric key-sharing problem. A session key is exchanged by the public key owner to the private key owner.
- Transfer confidentiality is then provided by a symmetric algorithm using the session key.

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 18. 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 the signature add authentication to integrity by encrypting the message hash. The difference between MAC and signature is that the MAC generation uses a symmetric key algorithm (Figure 19), while the signature uses the message sender private key (Figure 20).

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 secrete key.

Figure 19. MAC generation with secrete key algorithm

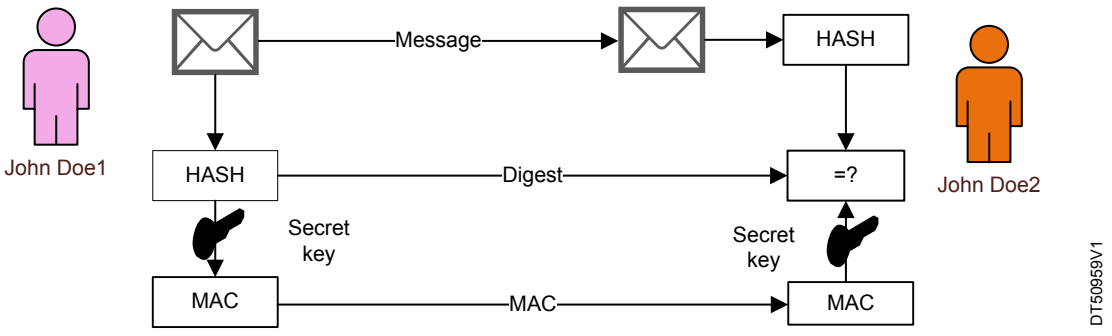
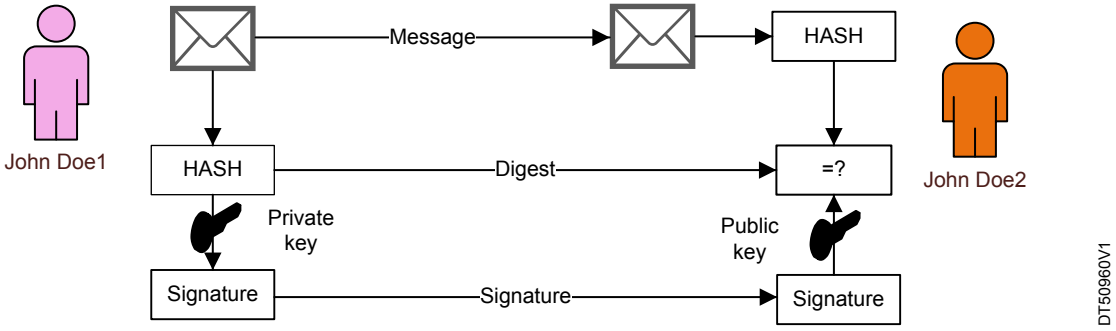


Figure 20. **Signature generation with public key algorithm**



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 substitutes 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	Updated: • 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 Added: • Section 6.8 Cryptographic key storage (CKS)
7-Oct-2019	3	Updated: • *Table 1. Applicable products* • Section "Introduction" renamed "Overview" • *Table 2. Glossary* • Section "Hardware protections" renamed "Device protections" • *Figure 4. Memory types* • *Table 5. Memory types and associated protection* • *Section 4.2.4 External Flash memories* • *Table 6. Scope of STM32 embedded memories protection features* • *Table 7. 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 6.7 Secure hide protection (HDP)* • *Section 6.17 Cryptography* • *Section 7 Guidelines* • Some colors removed on all figures Added: • *Section 4.1 TrustZone® for Armv8-M architecture* • *Section 6.4 TrustZone* • *Section 6.18 On-the-fly decryption engine (OTFDEC)*
21-Feb-2020	4	• Updated Section *Introduction* • Added acronyms in *Table 2. Glossary* • Updated *Section 2 Overview* and *Section 3 Attack types* • Restructured *Section 3.4 IoT system attack examples* (added *Section 3.5 List of attack targets*) • Updated *Section 4 Device protections* • Updated and restructured *Section 5 Secure applications* • Added *Section 5.3 Arm TF-M solution* • Updated *Section 6 STM32 security features*, *Section 7 Guidelines* and *Section 8 Conclusion*
06-Nov-2020	5	Updated: • Document's scope to add STM32WL Series • *Table 1. Applicable products* • *Section 1 General information* • *Section 3.1 Introduction to attack types* • *Section 3.2 Software attacks* • *Section 3.3.1 Non-invasive attacks*

Date	Version	Changes
06-Nov-2020	5 (cont'd)	Updated: • *Section 3.3.2 Silicon invasive attacks* • *Section 4.1 TrustZone® for Armv8-M architecture* • *Table 5. Memory types and associated protection* • *Section 5.3 Arm TF-M solution* • *Table 8. Basic feature differences* • *Section 6.1 Security features overview including updates in all the tables* • *Section 6.2 Readout protection (RDP)* • *Section 6.4 TrustZone* Added: • *Section 4.2 Dual-core security* • *Section 6.3 One-time programmable (OTP)*
07-Jul-2021	6	Updated: • Document's scope to add STM32U5 Series • Table 1. Applicable products • Section 3.3.1 Non-invasive attacks • Section 4.3.3 Embedded SRAM • Section 4.3.4 External Flash memories • Section 5 Secure applications • Table 9. Security features for STM32Fx Series • Table 10. Security features for STM32Lx and STM32U5 Series • Table 11. Security features for STM32H7, STM32G0, STM32G4, STM32WB and STM32WL Series • Section 6.3 One-time programmable (OTP) • Section 6.6 Execute-only firmware (PCROP) • Section 6.8 Firewall • Section 6.9 Memory protection unit (MPU) • Section 6.17 Cryptography • Section 6.17.1 Hardware accelerators • Section 6.17.2 CryptoLib software library Added: • Section 5.4 Product certifications
13-Jan-2023	7	Updated: • Document scope to add STM32C0 and STM32H5 Series • Section 1 General information • Debug port access and SCA in Section 3.3.1 Non-invasive attacks • Random number generation and Communication eavesdrop in Section 3.5 List of attack targets • New Section 4.1 Configuration protection • Introduction of Section 5.2 ST proprietary SBSFU solution • New Section 5.2.3 Configurations • Section 5.3 Arm TF-M solution • Section 6.1 Overview of security features • Last note in Section 6.2 Readout protection (RDP) • New Section 6.3 Lifecycle management – product state • Section 6.7 Execute-only firmware (PCROP)
22-Mar-2023	8	Updated: • Section 1 General information • Section 4.1 Configuration protection • Section 4.2 TrustZone® for Armv8-M architecture • Table 6. Scope of STM32 embedded memory protection features • Table 7. Software isolation mechanism • Section 5.4 Arm TF-M solution • Section 5.5 Product certifications • Table 9. Security features for STM32C0, STM32F0/1/2/3/4, STM32G0/4 devices • Section 6.2 Readout protection (RDP) • Section 6.5 TrustZone® • Section 6.7 Execute-only firmware (PCROP) • Section 6.12 Antitamper (TAMP)/backup registers (BKP) • Section 6.18 Cryptography

Date	Version	Changes
		• Section 7 Guidelines • Section 8 Conclusion Added: • Section 5.1 Secure firmware install (SFI)
13-Oct-2023	9	Updated: • Table 4. Attacks types and costs • Section 5.6: Product certifications • Section 6.3: Lifecycle management–product state • Section 6.9: Secure hide protection (HDP) • STM32WBA added to Table 11. Security features for STM32L0/1/4/4+, STM32WB, STM32WBA, STM32WL devices • STM32H7R/S added to Table 12. Security features for STM32L5, STM32U5, STM32H503/ Added: • Section 5.5: Secure manager
11-Mar-2024	10	Updated: • Document title from *Introduction to STM32 microcontrollers security* to *Introduction to security for STM32 MCUs*. • Added STM32U0 Series and STM32WB0 Series to the document scope and updated Table 1. Applicable products. • Added footnotes to *Hardware crypto* row on Table 10, Table 11 and Table 12. • Table 11: added STM32WB0x column. • Table 12: table header, *RNG* row, added STM32U0 column. • RDP level regression. • Section 6.3: Lifecycle management–product state: main available product states list. • Section 6.9: Secure hide protection (HDP) inlcuding Figure 10. HDP protected firmware access. • Note on When to use the firewall. • Section 6.19.1: Hardware accelerators: added note. Added: • Regression locked by OEM keys • Section 6.4: Unique boot entry

Contents

List of tables

List of figures