Introduction to STM32 microcontrollers security

Introduction

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

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

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

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

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

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

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

Table 1. Applicable products

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

AN5156 - Rev 1 - October 2018
For further information contact your local STMicroelectronics sales office.

www.st.com

1 Preamble

Table 2. Glossary

Term	Definition
AES	Advanced encryption standard
CPU	Central processing unit – core of the microcontroller
CSS	Clock security system
DoS	Denial of service (attack)
DPA	Differential power analysis
ECC	Error code correction
FIA	Fault injection attack
FIB	Focused ion beam
IAP	In-application-programming
IoT	Internet of things
IV	Initialization vector (crypto algorithms)
IWDG	Independent watchdog
MAC	Message authentication code
MCU	Microcontroller unit (STM32 arm® Cortex®-M based devices)
MPU	Memory protection unit
NVM	Non-volatile memory
PCROP	Proprietary code readout protection
PKA	Public key algorithm – aka asymmetric algorithm
PVD	Programmable voltage detector
ROM	Read only memory – system Flash memory in STM32
RDP	Read protection
RSS	Root secure services
RTC	Real-time clock
SB	Secure Boot
SCA	Side channel attack
SDRAM	Synchronous dynamic random access memory
SFU	Secure Firmware Update
SPA	Simple power analysis
SRAM	Static random access memory (volatile)
SWD	Serial wire debug
WRP	Write protection

Documentation references

Reference manual of each device give details on availability of security features and on memory protections implementation.

Programming manual for each Arm® Cortex® version are also available and can be used for Memory Protection Unit (MPU) description.

- STM32F7 Series Cortex®-M7 processor programming manual (PM0253)

- STM32F3 Series, STM32F4 Series, STM32L4 Series and STM32L4+ Series Cortex®-M4 programming manual (PM0214)

- STM32F1, STM32F2 and STM32L1 Cortex®-M3 programming manual (PM0056)

- STM32L0 Series Cortex®-M0+ programming manual (PM0223)

Refer to the following set of **user manuals** and **application notes** for detailed description of some security features:

- UM1924 "STM32 crypto library"

  This user manual describes the API of the STM32 crypto library. A software expansion package is available with this user manual: X-CUBE-CRYPTOLIB

- UM2262 "Getting started with the X-CUBE-SBSFU STM32Cube Expansion Package"

  Presents ST Secure Boot (SB) and Secure Firmware Update (SFU) solutions. Provided with the software expansion package X-CUBE-SBSFU

- AN4246, AN4701, AN4758, AN4968 "Proprietary Code Read Out Protection on STM32xx microcontroller"

  This set of application notes explains how to setup and work with PCROP firmware for the respective STM32 families: L1, F4, L4 and F7. Linked to the software expansion package X-CUBE-PCROP

- AN4838 "Managing memory protection unit (MPU) in STM32 MCUs"

  This application note describes how to manage the MPU in the STM32 products.

All these documents are available on *www.st.com*.

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

2   Introduction

2.1   Security purpose

Why protection is needed?

Security in microcontrollers is about protecting embedded firmware and data as well as the system functionality. The need for data protection seems obvious when talking about confidential passwords or personal data. However this is not the only sensitive information, the code is probably where most of the value lies within an embedded application. If an attacker can gain access to the binary, he can reverse engineer the program in an attempt to find further vulnerabilities, bypass licensing and software restrictions, copy any custom algorithms or even use it to flash a clone of the hardware, which can then be sold at a reduced price. Denial-of-service attack (DoS attack) is another major threat when considering protection systems such as environment (gas, fire, intrusion…) detection alarms or surveillance cameras. System functionality shall be robust and reliable.

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

Indeed the IoT, or smart devices, have raised the requirement for security to an even higher level. Connected devices are very attractive for hackers because they are remotely accessible. The connectivity offers an angle of attack through protocol vulnerabilities. In case of a successful attack, a single hacked device can jeopardize the integrity of an entire network (Figure 1).

Figure 1. Corrupted connected device threat
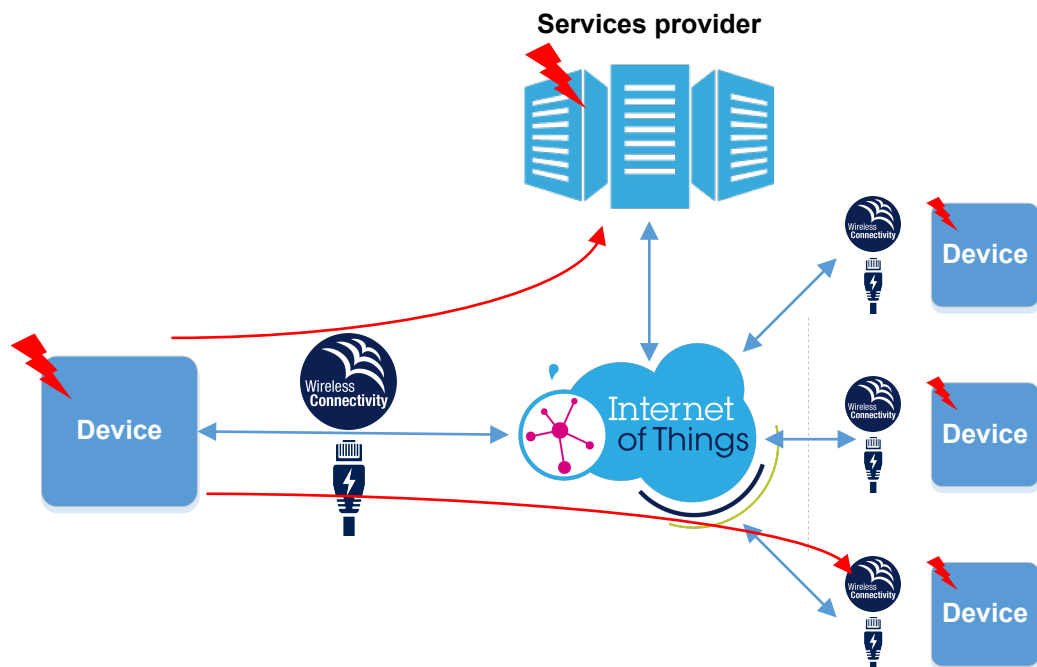
What should be protected:

Security is designed for protection of code, data and system functionality. Code protection is set to guarantee the firmware intellectual property and its integrity. Data protections, including cryptographic keys, are needed to guarantee the confidentiality of user data and avoid identity theft. This later asset becomes more sensitive with

the multiplication of devices used to access paid services (cloud) or any other access-controlled resource. Finally, system functionality itself shall be protected to avoid device malfunction or service failure (DoS attack).

Table 3 presents a non-exhaustive list of assets targeted by attackers.

Table 3. **Assets to be protected**

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

Vulnerability/Threat/Attack

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

A specific wording is used around security:

• Asset: what needs to be protected
• Threat: what we need to be protected against
• Vulnerability: weakness or gap in a protection mechanism

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

2.2 Building a secure system

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

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

A complete system security implementation will involve device protection mechanisms as well as a set of secure applications that depends on system functionality and required robustness (Figure 2).

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

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

• A Secure Boot that guarantees application integrity and authenticity at startup
• A support for secure communication protocols (e.g. TLS for internet secure communication)

- A Secure Firmware Update capability to download a new firmware version in a secure way (authentication / confidentiality / integrity)
- Extended capabilities for Secure Key Storage (SKS)

Secure Boot and Secure Firmware Update (SBSFU) will be detailed in Section 5 Secure applications .

Because most secure applications rely on cryptography tools, basic concepts are presented in Section A Cryptography - Main concepts .

Figure 2. Hardware and application security layers
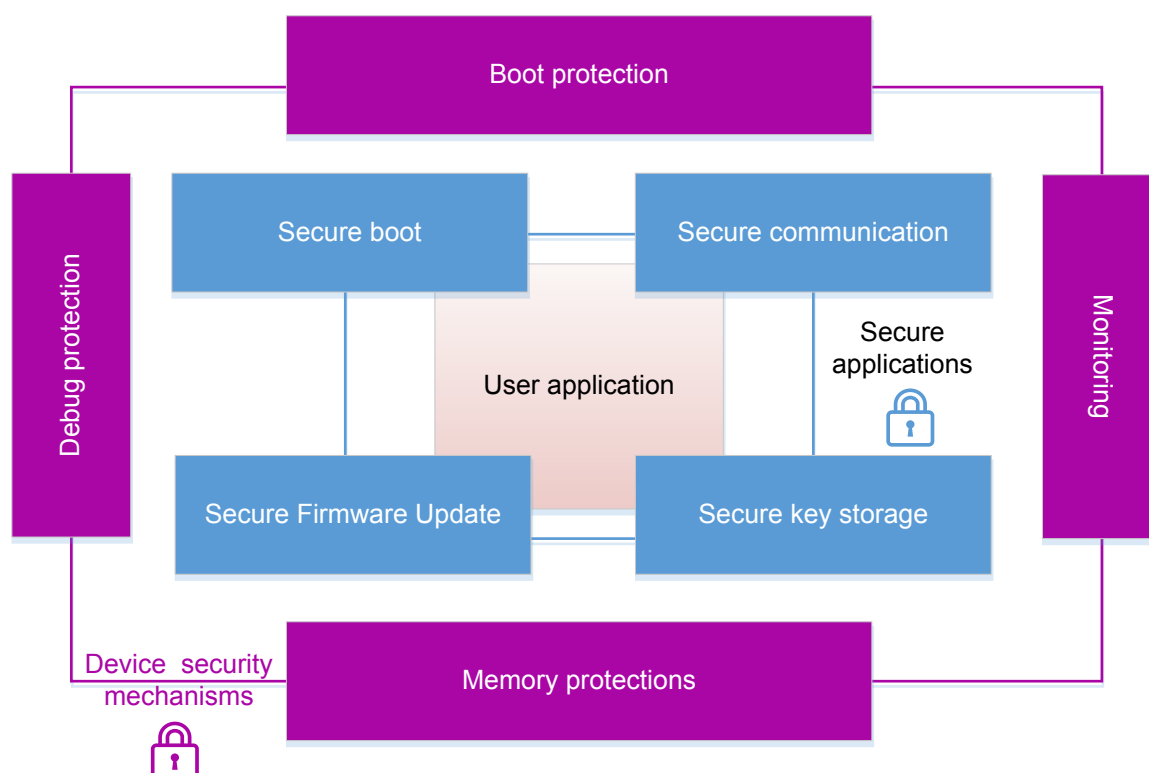
3   Attack types

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

Attacks on microcontroller are classified in one of these three types:

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

3.1   Introduction

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

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

Secondly, we will see in this section that some very advanced and expensive techniques exist to retrieve a microcontroller content.

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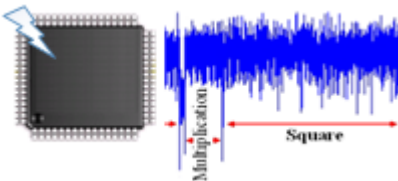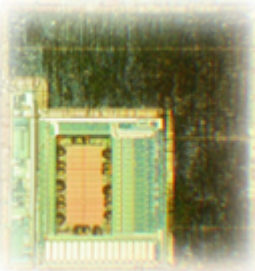
Attack types

The attacks are categorized into two broad categories, the software and the hardware ones.

Software attacks are carried by exploiting bugs, protocol weaknesses, untrusted pieces of code… Attacks on communication channels (interception, usurpation) can also be considered as 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.

Hardware attacks need physical access to the device. Most obvious one consists in exploiting the debug port if it is not protected. But in general, hardware attacks are sophisticated and can be very expensive, they are carried out with some specific materials and require electronics engineering skills. A distinction is made between non-invasive attacks carried out at board level or chip level without device destruction and invasive attacks carried out at device silicon level with package destruction.

Table 4 gives an overview of the cost and techniques used for each types of attack.

Table 4. Attacks types and costs

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

3.2 Software Attacks

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

This type of attack represent most of device threats for two reasons. Firstly, attack cost is low since it does not need specific equipment but a personal computer. Secondly, so many hackers can put their effort together, sharing their expertise and tricks, 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 it can already be present (insider threat) in main application firmware through a non-verified or untrusted library for example.

Malware are of many types and they may be very small and easy to hide. Some examples of what a malware can do:

- Access to and modify device configuration (option bytes, 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 device functionality
- Etc …

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

Malware injection

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

Listing malware injection methods consists in listing the various ways of interfacing with the device. We can group them in four categories:

1. Basics device access / "open doors"

    a. Debug port: JTAG or SWD interface

    b. Bootloader: If bootloader is accessible, it can be used to read/write memory content through any available interface

    c. Execution from external memory

2. Application download

    a. Firmware update procedure: a malware can be transferred instead of a new FW

    b. OS with capability to download new applications (and malware).

3. Weaknesses of communication ports and bugs exploitation:

    a. Malware injected through application input data and executed thanks to bugs (overflow, lack of data verification...)

    b. Stack-based buffer overflows, heap-based buffer overflows, jump-to-libc attacks, and data-only attacks.

4. Use of untrusted libraries with device back door

The first category is easy to counter with simple hardware mechanisms that will be described in Section 4 Hardware protections .

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

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

The last category is an intentional malware introduction that will facilitate 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 intrusion of an application protected by a password. A secure device may require a session authentication before accessing services (in the cloud for example) and a human-machine interface (HMI) can be exploited with an automatic process in order to try successive passwords exhaustively.

Countermeasures:

• Limit the number of login trials with a monotonic counter (implemented with a timer)

• Increase the delay between two login attempts

• Add a challenge-response mechanism to break automatic trials.

3.3 Hardware attacks

Hardware attacks require physical access to the device. A distinction is made between two types of attacks, which differ in cost, time and necessary expertise: non-invasive attacks and invasive attacks. Non-invasive attacks have only external access to the device (board-level attack) and are not very expensive to carry on. Invasive attacks have direct access to device silicon (after de-packing); they are carried out with advanced equipment often found in specialized laboratories, are very expensive and target very valuable data (Keys or IDs) or even technological patents.

General-purpose microcontrollers are not the best candidates to counter the most advanced physical attacks. If highest protection level is required, it is advised to consider pairing a secure element with the general-purpose microcontroller. Secure elements are dedicated microcontrollers certified as per the latest security standards with specific hardware. Refer to ST secure microcontrollers web page: www.st.com/en/secure-mcus.html.

This section introduces briefly the two types of hardware attacks.

3.3.1 Non-invasive attacks

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

Debug port access

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

Countermeasure:

- Debug port deactivation or fuse through Readout protection (RDP) .

Serial ports access

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

Countermeasures:

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

Fault injection: Clock and power disturbance / Glitch attacks

Fault injection consists in using the device in abnormal environmental conditions to generate malfunctions in the system. A successful attack can lead to serious damages by modifying the program behavior in different ways: corrupting program state, corrupting memory content, stopping process execution ("stuck-at fault"), skipping instruction, modifying conditional jump, providing unauthorized access, and so on … Typical threats involve tampering with clock (freezing, glitch) and power (under/over voltage, glitch). Since fault injection may be non-intentional, countermeasures are the same as the one used for safety: redundancy, error detection and monitoring.

Countermeasures:

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

Side-channel attacks (SCA)

When a firmware is executed, an attacker can observe the device analog characteristics: power consumption, electromagnetic radiations, temperature, activity time, etc… The observation of these physical characteristics 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.

Simple Power Analysis (SPA) and Differential Power Analysis (DPA) are typical example of side-channel attack exploiting power consumptions.

Countermeasures:

• Software:

– Limit key usage: use session random keys when possible

– Use protected cryptographic libraries with behavioral randomization (delays, fake instructions...)

• Hardware: shields against monitoring can be found in secure elements (STSAFE), but there is no efficient hardware countermeasure embedded in general-purpose microcontrollers.

3.3.2 Silicon invasive attacks

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

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

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

Reverse engineering

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

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

Micro probing and internal fault injection

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

Device modification

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

3.4 IoT system attack examples

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

An IoT system is built around a STM32 microcontroller with connectivity systems (Ethernet, WiFi, BLE, LoRa…) and sensors and/or actuators (Figure 3). The microcontroller handles the application, data acquisition and communications with a cloud service. It is also responsible for the system maintenance through firmware update and integrity check.
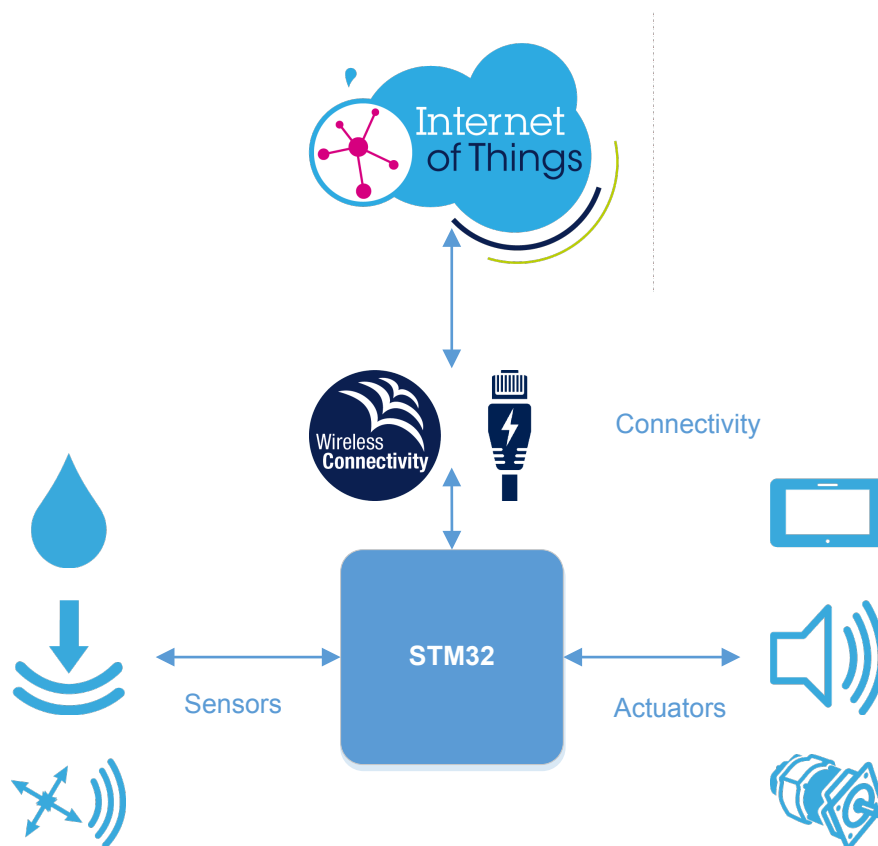
Figure 3. IoT system

Figure 4 shows the system architecture with software and hardware components. Each of them may have weaknesses leveraged by attackers. The potential damages and a list of countermeasures is given for each example.

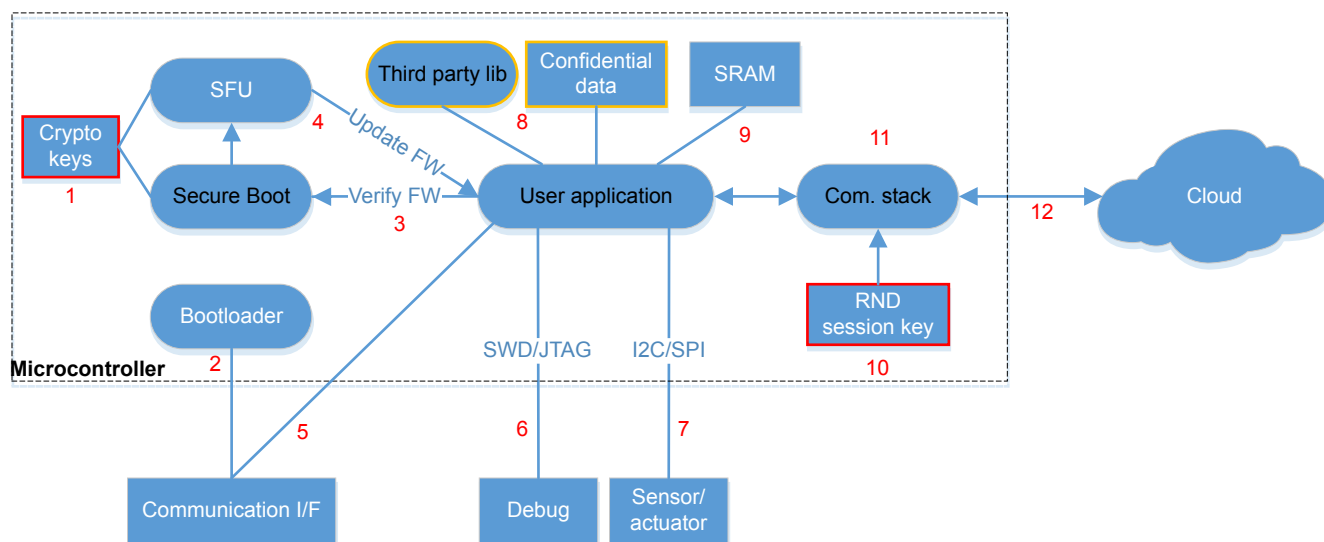**Figure 4. IoT system attack examples**

Table 5. Attacks targets

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

1. Initial provisioning

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

- Risks: firmware corruption, usurpation…
- Countermeasures:
  - Trusted manufacturer environment
  - Use secure data provisioning services: SFI
  - Data protection mechanisms
  - Secure application isolation

2. Boot modification

The purpose of this attack is to use the bootloader as an input port to access to device content. The attack aims at modifying the boot mode and/or the boot address to preempt the user application and to take control of the CPU through the bootloader (via USB DFU, I2C, 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 shall be protected for security reason.

- Risks: full access of the microcontroller content
- Countermeasures:
    - Unique boot entry
    - Bootloader and debug disabled – Readout protection (RDP)

3. Secure Boot

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

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

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

4. Firmware Update

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

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

- Risks: device firmware corruption
- Countermeasure: Secure Firmware Update application with authentication and integrity checks. Confidentiality can also be added by encrypting the firmware in addition to signature.

5. Communication interfaces

Serial interfaces (SPI, I2C, USART…) are often used either by the bootloader or by custom applications to exchange data and/or commands with the device. The physical interception of a communication with access to the device port allows an attacker to use the interface as a device entry point. The firware protocol can also be prone for bugs (overflow…).

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

6. Debug port

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

- Risk: full access to the device
- Countermeasure: disable device debug capabilities (see Readout protection (RDP) feature)

7. External peripheral access

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

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

8. Sensitive firmware and data

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

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

- Risk: sensitive firmware copy, data theft…
- Countermeasures: execute-only access right (XO), firewall, memory protection unit, secure area…

9. SRAM

The SRAM is the device running memory. It embeds runtime buffers and variables (stack, heap…) and can embed firmware and keys. The SRAM may not be as protected as the user Flash memory and its content at runtime may be difficult to control. These are two reasons why an attacker may focus his efforts 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.

- Risk: buffer overflow, data theft, device control…
- Countermeasures: firewall memory protection unit, secure RAM usage

10. Random number generation

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

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

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

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

11. Communication stack

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

- Risk: device access (content, control) through network
- Countermeasures: communication process isolation, server authentication

12. Communication eavesdrop

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

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

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

4 Hardware protections

These security protections are controlled by hardware mechanisms. They are set either by configuring the device through option bytes, or dynamically by hardware component settings.

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

4.1 Memory protections

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

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

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

Embedded flash memory, embedded SRAM and external memories are used for very different purpose. Their respective protections mechanisms reflect these differences. Figure 5 provides a simple view of memories access architecture in a microcontroller.

Figure 5. Memory types
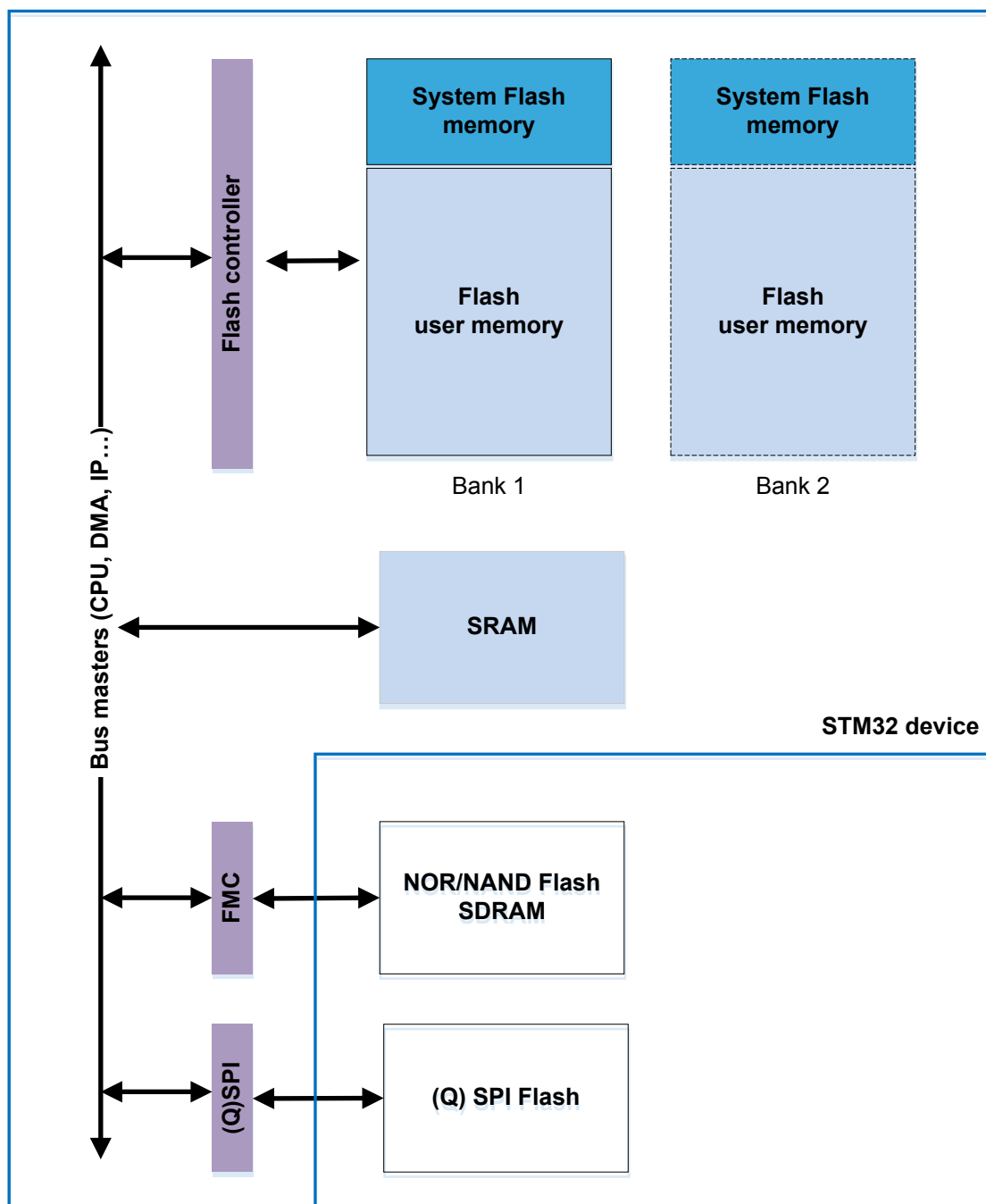
Table 6summarizes the particularities of each type of memories and typical protection features.

Table 6. Memory types and associated protection

Memory	Types	Description	Protections
System Flash	• Internal NVM • ROM	ROM part of Flash memory. Embeds device bootloader and other services from STMicro.	Cannot be updated (erase/written) Some part may also be unreadable.
User Flash	• Internal NVM	Flash memory for user application	Internal protections: • RDP • WRP (not for SRAM) • PCROP (not for SRAM) • Firewall • Secure User Area (not for SRAM) • MPU
SRAM	• Internal Volatile	Working memory for Stack, heap, buffers… Can be used to execute FW downloaded from internal or external non-volatile memories.	
NAND/NOR/ (Quad) SPI Flash	• External NVM	Additional memory for applications or data storage.	Cryptography Write Protection (on flash device)
SDRAM	• External Volatile	Additional RAM for application execution	Cryptography

4.1.1 System Flash memory

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

Protection attribute on System Flash memory cannot be modified.

4.1.2 User Flash memory

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

External attacks

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

Associated protections: RDP to disable debug access

Internal attacks

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

Associated protections: PCROP, MPU, firewall, Secure User Area

Protecting unused memory

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

Associated protections: MPU, WRP

Error Code Correction

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

4.1.3 Embedded SRAM

This is the device working memory; it embeds stack, heap, global buffers and variables at runtime. 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

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

Associated protections: MPU, firewall

SRAM cleaning

SRAM content may contain sensitive data or temporary values allowing some secrets retrieving. A typical example would be the transfer of a secret cryptographic key from protected flash area in clear text inside the SRAM. It is highly recommended to clean explicitly the working buffers and variables after the processing of functions manipulating sensitive data. Note that in case of reset, STM32 products allow the automatic erase of the SRAM (refer to each device reference manuals). Note that for some products, part of SRAM is protected against external access or untrusted boot (SRAM boot) when RDP is set.

Write protection

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

Associated protections: MPU and SRAM write protection (available on some STM32 series only)

Parity check

Parity check on SRAM allows controlling potential error word-by-word (32 bits). One extra bits per byte is added to the memory content (data bus width is 36 bits) to increase its robustness as required for instance by Class B or SIL norms.

4.1.4 External Flash memories

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

Embedded code can be either executed in-place or loaded into device 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 shall be decrypted when loaded into SRAM.

4.1.5 STM32 memory protections overview

Several STM32 features are available to cover the various cases considered. They are listed in Table 7 with their respective scope. They will be described in Section 6 STM32 Security features.

Table 7. Scope of STM32 embedded memories protection features

Feature	External attack protection	Internal attack protection	Flash	SRAM
RDP	Yes	No	Yes	Yes
Firewall	No	Yes	Yes	Yes
MPU	No	Yes	Yes	Yes
PCROP	Yes	Yes (Read/write)	Yes	No
WRP[1]	Yes	Yes	Yes	No

Feature	External attack protection	Internal attack protection	Flash	SRAM
Secure User Area	Yes	Yes	Yes	Yes (for execution)[2]

1. *Write protection can be unset when RDP level ≠ 2.*

2. *SRAM is protected by secure area only at secure code execution. It shall be cleaned before leaving secure area.*

4.2 Software isolation

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

Software isolation has two goals:

1. Prevent a process to spy execution of another sensitive process in order to retrieve data like cryptographic key or part of its code.

2. Protect process execution against stack corruption due to memory leaks or overflow (incorrect memory management implementation).

This memory protection is achieved thanks to different mechanisms (see Table 8) that will be detailed in Section 6 STM32 Security features .

Table 8. Software isolation mechanism

Protection	Type	Isolation
Memory protection unit (MPU)	Dynamic	By privilege attribute [1]
Firewall	Static	By bus address hardware control
Secure user area	Static	Process preemption at reset
Dual core	Static	By core ID

1. *Attribute protection is only for CPU access and is not taken into account for other bus master (DMA, ...)*

4.3 Debug port and other interfaces protection

Debug ports provide access to internal resources (core, memories and registers) and shall be disabled. It is the most basic external attack and is easily avoided by deactivating JTAG (or SWD) ports by a secure and immutable firmware (refer to Section 5.2 Secure Boot), or by permanently disabling the functionality (JTAG fuse in RDP2).

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

Associated STM32 features:

•   Read protection (RDP)

•   Disable of unused ports

•   Forbid bootloader access (configured by RDP in STM32 devices)

4.4 Boot protection

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

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

Associated STM32 features:

•   Read protection (RDP)

•   Unique boot entry

•   Secure User Memory

4.5 System monitoring

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

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

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

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

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

Associated STM32 features:

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

5 Secure applications

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

- **Secure Boot (SB):** built to check device state, set runtime protections and authenticate firmware before application execution
- **Secure Firmware Update (SFU):** built to allow authenticated firmware update

Note: *These applications have a close link with cryptography. All cryptographic schemes are based on the three concepts of secrete key, public key and hashing. Basics of cryptography are explained in Section A Cryptography - Main concepts.*

Note: *STMicroelectronics provides a user manual together with an implementation example of Secure Boot and Secure Firmware Update. Please refer to UM2262 "Getting started with the X-CUBE-SBSFU STM32Cube Expansion Package" at www.st.com/en/product/x-cube-sbsfu.*

5.1 Root and chain of Trust

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

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

Successive part of the chain of trust may be composed by a Secure Firmware Update that will guarantee in turn the authenticity of future applications (Figure 6).

Figure 6. Chain of trust
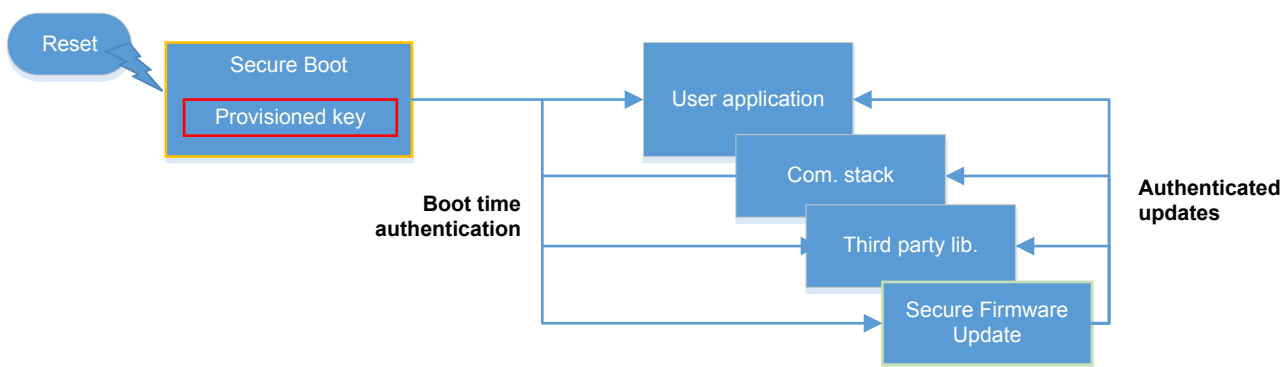
Whole system security depends on all its constituting applicative elements. A single weak element may corrupt the entire system by being the target of an attack.

5.2 Secure Boot

Secure Boot application is executed at reset before user application. It provides first stages of security and is then responsible for ensuring the global chain of Trust of the system.

The two main functionalities of Secure Boot are:

1. Checking STM32 security configuration and setting up runtime protections
2. Asserting the integrity and authenticity of the user application image(s) that will be executed (Figure 7)

Figure 7. Secure Boot FSM
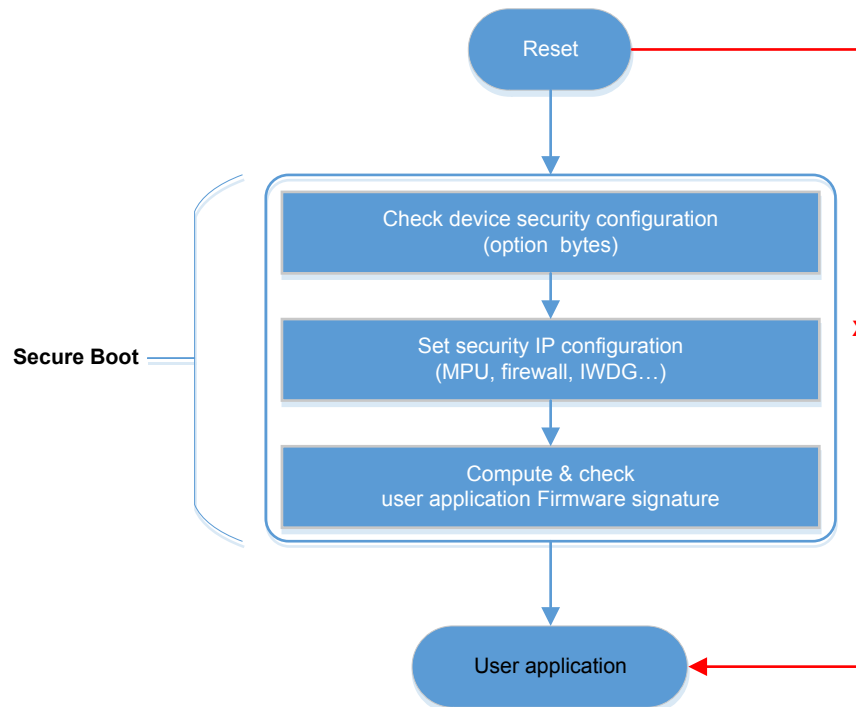
Checking device security

This part of Secure Boot application checks if static configurations are correct and sets the dynamic ones. Static secure configurations are defined by option bytes: RDP, PCROP, WRP and secure area. Dynamic protections shall be programmed: firewall, MPU, tamper detection, IWDG…

Integrity and authenticity check

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.

Authenticity check is added by if the expected tag is encrypted with a share key between the firmware owner and the device. This key is stored in a protected area of the device.

Protection attributes

Secure Boot firmware shall have the following attributes to fulfill its role:

•   It must be the device unique entry point (no bypass)

•   Its code must be immutable

•   It shall have access to sensitive data: Certificates, application signatures…

Most sensitive part of Secure Boot will take benefit from process and data isolation features like firewall, MPU or Secure User Area. Implementation depends on STM32 series available features.

5.3   Secure Firmware Update

Secure Firmware Update (SFU) provides a secure implementation of in-field firmware updates, enabling the download of new firmware images to a device in a secure way.

Firmware update is a sensitive operation that must protect two parties: the device owner and the application (OEM) owner. From the device side, the goal is to avoid loading a corrupted firmware (intentionally or not) which might definitely damage the device. On the other side, the application owner 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 Figure 8). As the communication channel is generally considered as non-secure since it is subject to eavesdropping, the overall security responsibility is shared between the sender (firmware owner server) and the receiver (the device).

Figure 8. Secure server/device SFU architecture
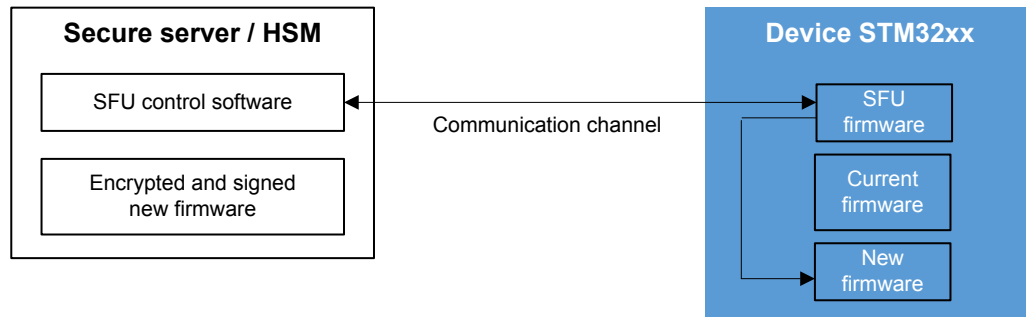
Application

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

SFU application running on device is in charge of:

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

Protection attributes

As Secure Boot application, SFU shall have the following attributes:

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

6 STM32 Security features

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

6.1 Overview

Static and dynamic protections

A distinction can be made depending on whether protection features are static or dynamic. Static protections refer to features that are set with option bytes. Their configuration is retained at power off. Dynamic, or run time, protections do not retain their status at reset, so they have to be configured at each boot (for example during first stage of Secure Boot).

Static protections are RDP, PCROP, WRP, BOR and secure user area (when available). All these features are set through permanent device configuration with option bytes.

Dynamic protections provided by STM32 are MPU, tamper detection and firewall.

Other dynamic protections are related to both security and safety. An abnormal environment behavior may be accidental (safety) or intentional, in order to carry out an attack. These protections include clock and power monitoring systems, memory integrity bits and independent watchdog (IWDG).

Security features by STM32 series

Following tables (Table 9 , Table 10 and Table 11) list the available features according STM32 series.

Table 9. Security features for STM32Fx series

	F0	F1	F2	F3	F4	F7
Cortex	M0	M3	M3	M4	M4	M7
RDP additional protection [1]	backup registers	N/A [2]	+ backup SRAM	+ backup registers	+ backup SRAM	+ backup SRAM
WRP	By sectors (4K)	By pages (4K or 8K)	By sectors (16K, 64K or 128K)	By sectors (4K)	By sectors (16K, 64K or 128K)	By sectors (16K, 64K, 128K or 256K)
PCROP	No	No	No	No	By sectors [1]	By sectors [1]
Secure area	No	No	No	No	No	No
Firewall	No	No	No	No	No	No
MPU	No	yes	Yes	Yes	Yes	Yes
Unique Boot Entry [1]	No	No	No	No	No	No
Tamper detection	Yes	yes	Yes	Yes	Yes	Yes
IWDG	Yes	yes	Yes	Yes	Yes	Yes
Device ID 96 bits	Yes	yes	Yes	Yes	Yes	Yes
HW Crypto [1]			TRNG, HASH, AES		TRNG, HASH, AES	TRNG, HASH, AES

1. *Depends on device part number.*

2. *RDP in STM32F1 is only for Flash protection. RDP is set (RDP1) or unset (RDP0). RDP level 2 is not implemented.*

Table 10. **Security features for STM32Lx series**

	L0	L1	L4
Cortex	M0	M3	M4
RDP additional protection [1]	+ EEPROM	+ EEPROM	+ backup registers + SRAM2
WRP	By sectors (4K)	By sectors (4K)	By area with 2K bytes granularity One area per bank
PCROP	By sectors	By sectors [1]	By area with 8 bytes granularity One area per bank
Secure area	No	No	No
Firewall	Yes	No	Yes
MPU	Yes	Yes	Yes
Unique Boot Entry [1]	No	No	No
Tamper detection	Yes	Yes	Yes
IWDG	Yes	Yes	Yes
Device ID 96 bits	Yes	Yes	Yes
HW Crypto [1]	AES	AES	TRNG, HASH

1. *Depends on device part number.*

Table 11. **Security features for STM32Hx and STM32WB series**

	H7	STM32G0
Cortex	M7	M0+
RDP additional protection [1]	+ backup SRAM	+ backup registers
WRP	By sectors (128KB)	By area with 2K bytes granularity. Two areas available.
PCROP	By area with 256 bytes granularity One area per bank	By area with 512 bytes granularity. Two areas available.
Secure area	Yes (Secure User Memory)	Yes (Securable memory area)
Firewall	No	No
MPU	Yes	Yes
Unique Boot Entry	Yes	Yes (Boot lock feature)
Tamper detection	Yes	Yes
IWDG	Yes	Yes
Device ID 96 bits	Yes	Yes
HW Crypto [1]	TRNG, HASH, AES	TRNG, AES

1. *Depends on device part number.*

6.2 Readout protection (RDP)

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

Readout protection applies to all STM32 series for:

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

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

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

Figure 9. Example of RDP protections (STM32L4 series)
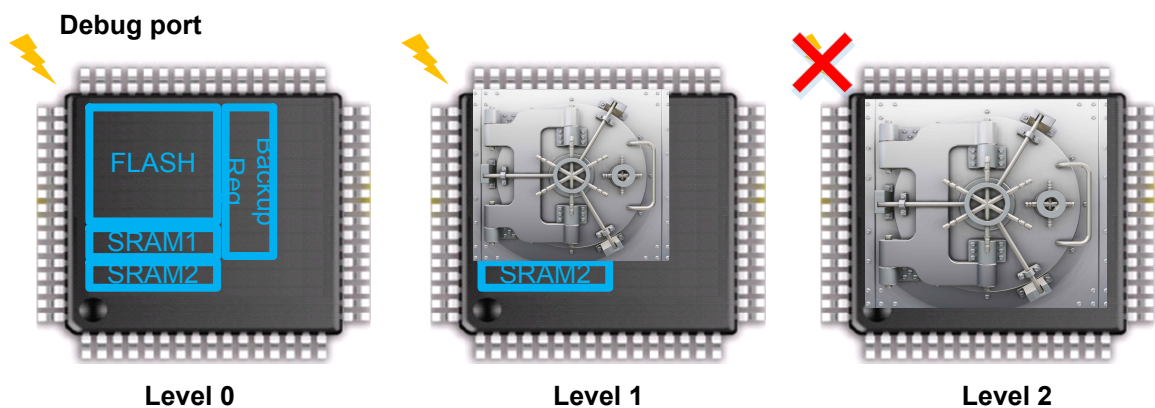
Three RDP levels (0, 1 and 2) are defined:

- **Level 0**: this is the default RDP level. The Flash memory is fully open and all memory operations are possible in all boot configurations (debug features, boot from RAM, boot from system memory bootloader, boot from Flash memory). There is no protection in this configuration mode that is appropriate only for development and debug.
- **Level 1**: this level corresponds to the activation of the read protection Level 1. Flash memory accesses (read, erase, program) or SRAM2 accesses via debug features (such as Serial Wire or JTAG) are forbidden, even while booting from SRAM or system memory bootloader. In these cases, any read request to the protected region will generate a bus error.

  When booting from Flash memory, however, accesses to both the Flash memory and to the SRAM2 (from user code) ar

  Disabling RDP Level 1 protection by re-programming RDP option byte to Level 0 leads to a Flash memory mass erase a

- **Level 2**: When RDP Level 2 is activated, all protections provided in Level 1 are active and the MCU is fully protected. The RDP option byte and all other option bytes are frozen and can no longer be modified. The JTAG, SWV (single-wire viewer), ETM, and boundary scan are all disabled.

Internal Flash memory content updating on an RDP protected STM32 microcontroller

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

Table 12 summarizes the RDP protections.

Table 12. **RDP protections**

Area	RDP Level	Boot from user Flash			Debug or Boot from SRAM or from bootloader		
		Read	Write	Erase	Read	Write	Erase
Flash main memory	0	Yes	Yes	Yes	Yes	Yes	Yes
	1	Yes	Yes	Yes	No	No	No
	2	Yes	Yes	Yes	N/A	N/A	N/A
System memory	0	Yes	No	No	Yes	No	No
	1	Yes	No	No	No	No	No
	2	Yes	No	No	NA	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/	N/A

1. *Backup registers/SRAM*

When should RDP be used:

On a consumer product, RDP should always be set at least at level 1. It prevents basic attacks through the debug port or through the bootloader. In RDP level 1 however, there is a risk of service denial caused by a Flash mass erase, following a return to RDP level 0.

RDP level 2 is preferred and is even mandatory to implement a Secure Boot and Secure Firmware Update application with correct security level (immutable code, JTAG debugger attach on reset...). The counterpart is that RDP 2 can prevent a device configuration update, for instance after a customer return.

RDP is available on all STM32 series.

6.3 Flash memory write protection (WRP)

The write protection feature is used to protect the content of specified memory area against erase or update.

Note: *For Flash memory technology, an update should 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 WRP should be used:

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

Write protection is available on all STM32 series.

6.4 Execute-only firmware (PCROP)

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

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

When should PCROP be used:

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

6.5 Secure User Memory

Some STM32 devices supports the Secure User Memory feature. This protection targets sensitive applications that embed or manipulate confidential data and that shall be securely executed.

The code, data and execution isolation of secure user firmware is ensured by a hardware mechanisms. The secure firmware can only be executed at reset. It is executed during boot, whatever the boot mode configuration. The protection consists in implementing a "unique boot entry" feature. Once executed, the secure user part is closed and cannot be accessed anymore by any mean (Figure 10).

Figure 10. Secure user firmware access
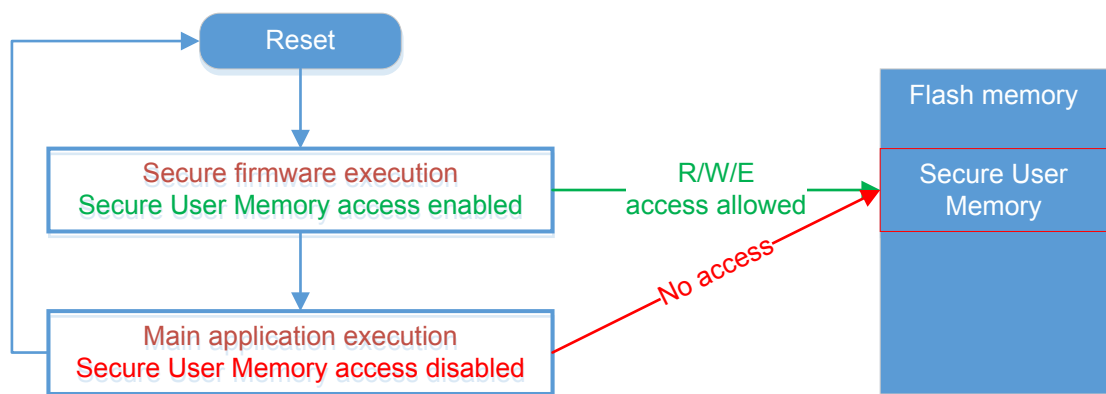
The Secure User Memory protection makes possible for the user to provide secure services to the device at boot time. The user Secure Boot or Secure Firmware Update (SFU) applications are typical examples that should reside in this secure area.

Secure User Memory is a static protection configured by option bytes. Once set, the CPU will boot on the firmware embedded in this area whatever the boot configuration set by boot pin or boot address.

When should secure user area be used:

The secure user area is suited for secure applications that should only be executed after reset, like Secure Boot or Secure Firmware Update.

This protection is available in STM32H75x series and STM32G0x series.

For STM32G0 the secure user memory is call "Securable memory area" and is placed at the beginning of the user Flash memory (0x08000000).

6.6 Firewall

The firewall is a hardware protection peripheral controlling the bus transactions and filtering accesses to three particular area: a code area (Flash), a volatile data area (Flash) and a non-volatile data area (SRAM). The protected code is accessible through a single entry point (the call gate mechanism explain 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 has to be set at startup, for example by a Secure Boot 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 (Figure 11).

Figure 11. Firewall FSM
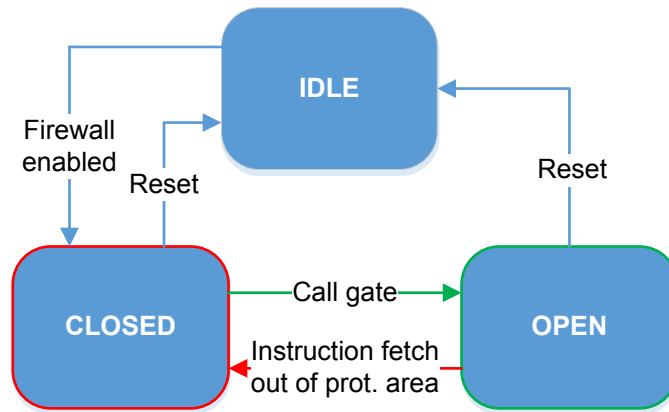
Since the only way to respect the call gate sequence is to pass through the single call gate entry point, a mechanism must be provided in order to support application calling multiple firewall-protected functions from unprotected code area (e.g. encrypt and decrypt functions). A solution is to use a parameter to specify which function to execute, for instance CallGate(F1_ID), CallGate(F2_ID), and so on. According to the parameter, the right function is internally called. This mechanism is represented in Figure 12.

Figure 12. Firewall application example
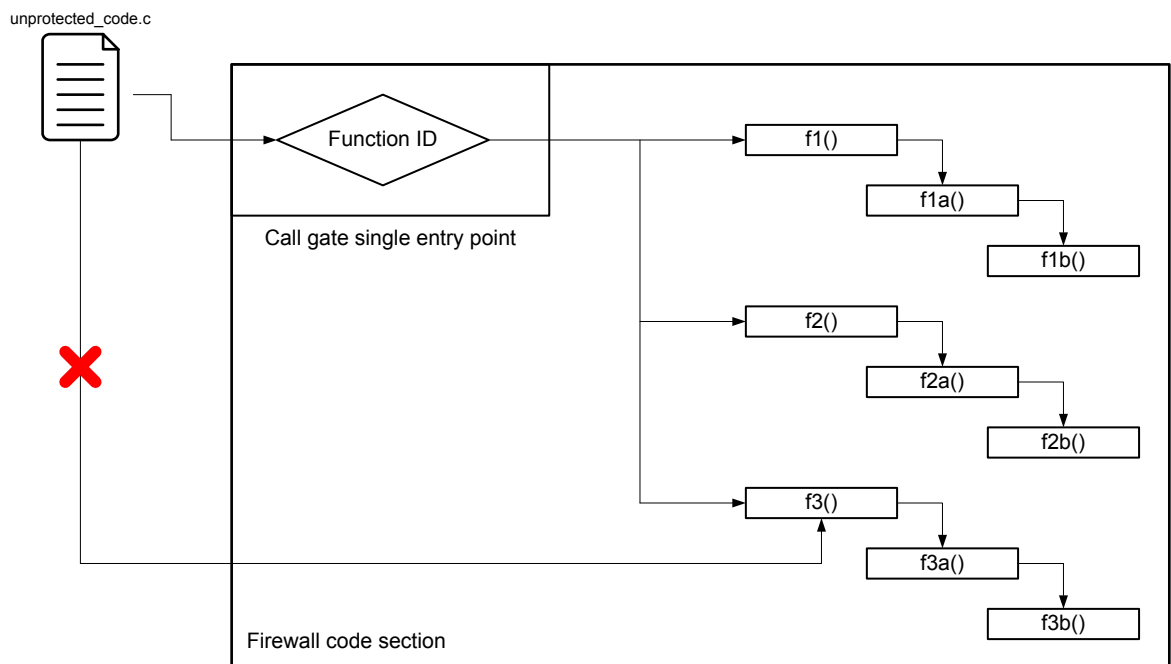
When should Firewall be used

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

A firewall is available on STM32L0 and STM32L4 series.

6.7 Memory protection unit (MPU)

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

Note: *MPU attributes are only set for CPU access. Other bus masters (DMAs...) requests are not filtered by the MPU and they shall 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 MPU for each region: shareable, cacheable and bufferable. These are not linked to security and are not considered here. Please refer to applicable device programming manual or to AN4838 "Managing memory protection unit (MPU) in STM32 MCUs".*

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. Table 13 shows the different cases supported by mixing modes and access attributes.

Table 13. Attributes and access permission managed by MPU

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

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

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

Secure process isolation strategy

At reset, the privilege mode is the default one for any process. Therefore the Secure Boot application is executed in privileged mode. Then the idea is to isolate secure processes (Secure Boot, OS kernel, Key manager, Secure Firmware Update…) from un-secured or un-untrusted processes (user applications).

Table 14. Process isolation

Firmware type	Mode	Resources access
Secure firmware (Secure Boot, OS kernel…)	Privileged	Full access
All remaining firmware	Unprivileged	MPU controlled access: No access, RO, RW

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

When should MPU be used:

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

MPU is available on all STM32 series with the exception of STM32F0. Detailed description of MPU is available on Cortex-M programing manuals.

6.8 Anti-Tamper

The anti-tamper is a system level protection. It is used to detect physical tampering attempts on the system and to take related counter measures. A tamper event is detected by a level transition on dedicated device pin(s). This event can be used to wake up the core in order to take appropriate actions (memory erase, alarms…).

Tamper detection is associated to the real-time clock peripheral. This peripheral embeds backup registers that may contain sensitive data such as master key for example. These registers may be reset if a tamper attempt is detected.

When should anti-tamper be used

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

Tamper detection is available on all STM32 series.

6.9 Clock security system

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

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

When should CSS be used:

When an external clock is used.

CSS is available on all STM32 series

6.10 Power monitoring

Some attacks may target the microcontroller power supply, for instance to make the system non-functional (Dos). A loss of power supply may also denote an attempt to freeze the device state in order to access internal memory content. The STM32 devices embeds a programmable voltage detector (PVD) that can detect a drop of power. The PVD allows to configure a minimum voltage threshold below which an interrupt will be generated so that the core can take appropriate actions

When should PVD be used:

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

6.11 Memory integrity hardware check

The error code correction (ECC) and parity checks are safety bits associated to the memory content. The ECC is associated to the Flash memory and allows recovering from a single bit error or detecting up to 2 erroneous bits on each Flash memory word. A simple parity check allows the detection of a single error bit on the SRAM 32-bit words.

When should ECC and parity bits be used:

This is mostly used for safety reasons. The ECC can also be used to prevent some invasive hardware attacks.

6.12 IWDG

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

When should IWDG be used:

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

6.13 Device ID

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

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

6.14 Cryptography implementation

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

Hardware IP

The following cryptographic hardware peripherals are available in STM32 catalog:

- **True random generator**
  - Hardware-based peripheral providing a physical noise source. Used to generate strong session keys
- **AES accelerator**
  - Encryption/decryption
  - 128/256 bits keys
  - DMA support
- **HASH accelerator**
  - MD5, SHA1, SHA224, SHA256
  - FIPS compliant (FIPS Pub 180-2)
  - DMA support

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

SW library

STM32 X-CUBE-CryptoLib is the software library running on STM32 devices. It is available for free download at www.st.com/en/product/x-cube-cryptolib.

STM32 X-CUBE-CryptoLib V3.1.0 is available with full firmware implementation, compiled for Cortex® M0, M0+, M3, M4, and M7 cores. It supports the following algorithms:

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

STM32 hardware acceleration crypto library V3.1.0 is supported by all STM32 series with hardware acceleration (AES and/or HASH).

7 Guidelines

Several security mechanisms exist (hardware and software) in order to build a secure system. Some of them should be adopted systematically, whatever the user application. It is the case of **RDP** feature, which prevents basic access to the flash memory by disabling the debug port. Other features shall be selected depending on user application and the required security level. The purpose of this section is to help choosing the adapted set of security features that should be put in place depending on the system use-cases.

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

Table 15. 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 will close 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 RTC peripheral
1.5 Protect a device from code injection
• Use RDP • Isolate communication port protocol with MPU, firewall or Secure User Memory • Limit communication port protocol access range • Use write protection on empty memory areas (Flash and SRAM)
2. Code protection against internal threats: PCROP, MPU, firewall and Secure User Memory
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 access
2.2 How to 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 Secure User Memory in case data should only be used at reset
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 un-trusted library • Use IWDG to avoid any deadlock

3. Device security check and maintenance: integrity checks, Secure Boot, Secure Firmware Update	
	3.1 Check code integrity
	• Hash firmware code at reset and compare to expected value • Enable ECC on Flash memory and parity check on SRAM
	3.2 Security checks or embedded firmware authentication

	• Implement Secure Boot application with cryptography
	• Protect Secure Boot application secret data (refer to previous sections)
	• Guarantee unique boot entry on Secure Boot application
	    – Use secure user area if available
	    – Use RDP level 2 and disable boot pin selection

	3.3 Securely update the firmware in the field
	• Implement a Secure Firmware Update 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 (e.g. TLS for Ethernet)
	4.2 Use ST AES/DES/SHA cryptographic functions with STM32 devices
	• Use only official software implementation by ST with STM32 X-CUBE-CryptoLib library
	4.3 Accelerate AES/DES/SHA crypto functions
	• Use device with crypto hardware IP together with official STM32 X-CUBE-CryptoLib library
	4.4 Generate true random data
	• Use hardware true random generator embedded in STM32 devices
	4.5 Uniquely identify ST microcontrollers
	• Use STM32 microcontrollers 96-bits UID
	4.6 Authenticate a product device
	• Embed a shared encryption key in the device and exchange encrypted message
	4.7 Uniquely authenticate a product 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

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

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

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

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

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

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

These algorithms are described in the below sections.

A.1 Secret key algorithms

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

Figure 13. Symmetric cryptography
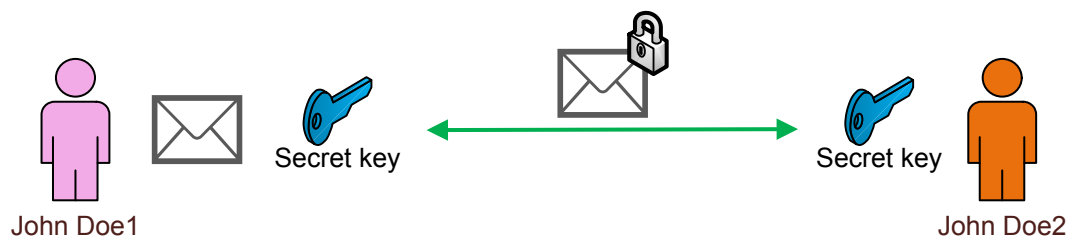
The inherent weakness of these algorithms is the key sharing between both parties. It may not be an issue in secure environments (such as manufacturing plants) but when both parties are distant, the key transfer becomes a challenge.

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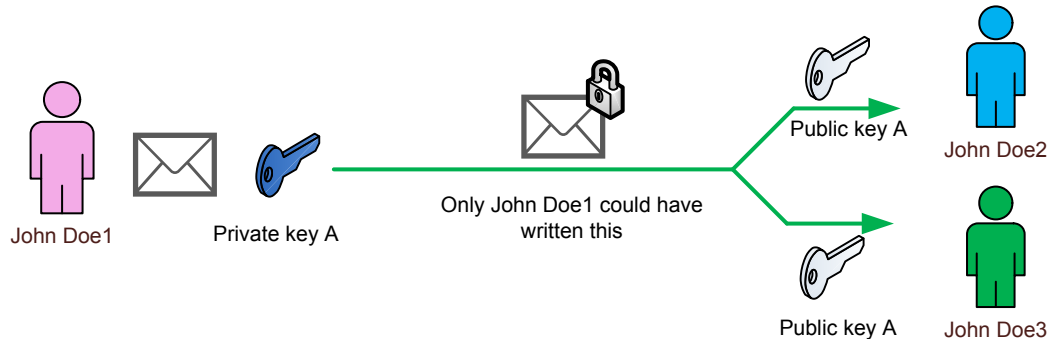
- A message encrypted by the public key can only be read by the private key owner.

Figure 15. PKA encryption
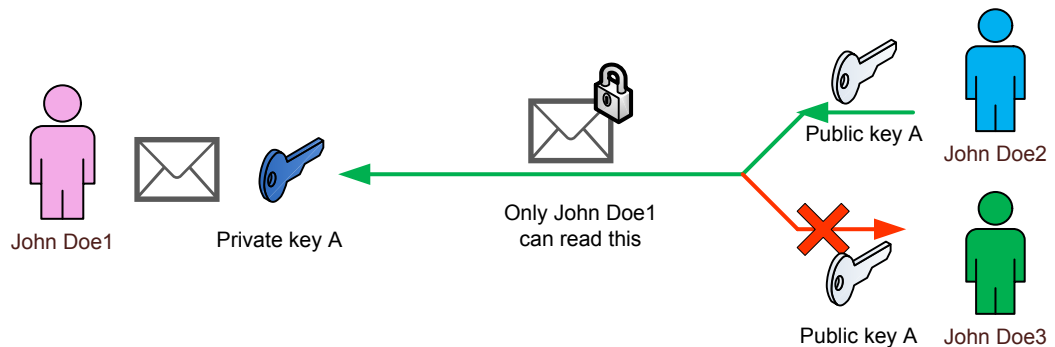
The main use of public key algorithms is authentication.

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:

1. 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.
2. 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 16. Message hashing
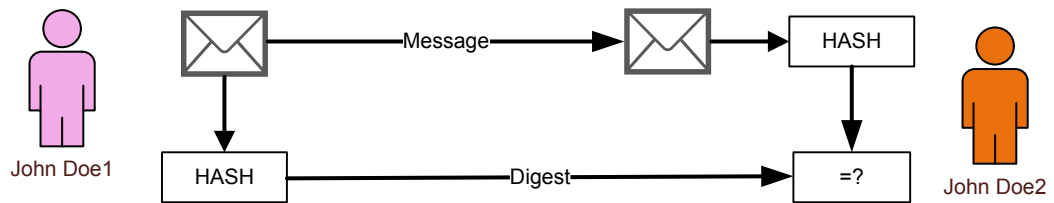
The difference with classic CRC is the robustness due to operations that are more complex and a much higher digest length: up to 512 bits instead of 16 or 32 bits. As an example, CRC are reserved for fast integrity checks during data transfers. Digest length makes them virtually unique and ensures that 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

Message authentication code (MAC) and signature add authentication to integrity by encrypting the message hash. The difference between MAC and signature is that MAC generation uses a symmetric key algorithm (Figure 17) while signature uses the message sender private key (Figure 18).

Signature add 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 17. MAC generation with secrete key algorithm
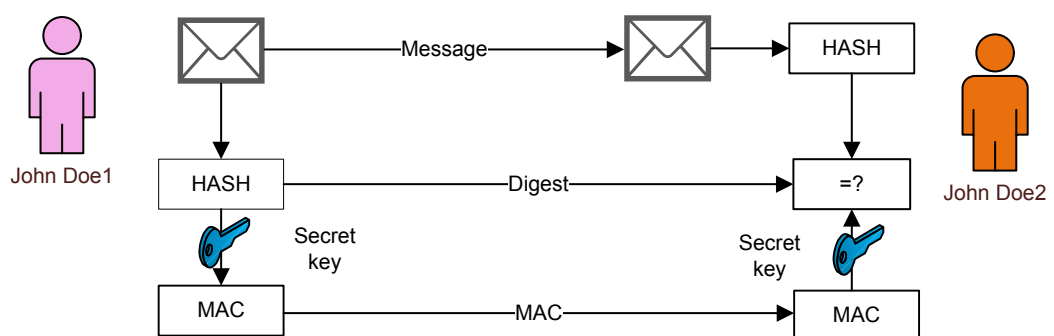
Figure 18. **Signature generation with public key algorithm**
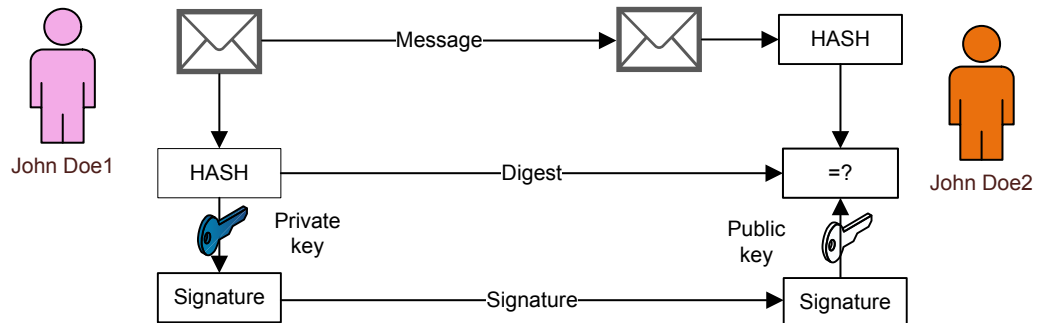
Certificate

A certificate is related to public key algorithms. It authenticates the public key in an asymmetric transfer. It is used to counteract usurpation by an attacker that could substitute the right public key by his own key. A certificate consists in the public key signed by a certificate authority (CA) private key. This CA is considered as fully trusted.

In addition to the public key, the certificate also contains version numbers, validity period and some IDs.

Revision history

Table 16. Document revision history

Date	Version	Changes
17-Oct-2018	1	Initial release.

Contents

List of tables

List of figures

IMPORTANT NOTICE – PLEASE READ CAREFULLY