STM32 Security features

life.augmented

Introduction

- Purpose : understand all the security blocks available across STM32 families and also experience them when it’s possible.

- This will allow you to understand the capabilities of each STM32 family regarding security.

- This is the necessary first step in order to understand how to combine them to build your system security with STM32.

STM32 security features

- Resources: STM32 Unique ID
- Static Protection
  - WRP
  - PCROP
  - RDP
  - Unique entry boot
  - User Secure Mem / HDP
- Dynamic protection :
  - MPU
  - FIREWALL
  - TrustZone
  - OTFDEC
  - Tamper

configured at code start thanks option bytes

- Crypto HW resources :
  - TRNG
  - CRYPT / AES
  - HASH
  - PKA

Crypto SW resources :

- CryptoLib

STM32 security features

- Resources: STM32 Unique ID
- Static Protection
  - WRP
  - PCROP
  - RDP
  - Unique entry boot
  - User Secure Mem / HDP
- Dynamic protection :
  - MPU
  - FIREWALL
  - TrustZone
  - OTFDEC
  - Tamper

- Crypto HW resources :
  - TRNG
  - CRYPT / AES
  - HASH
  - PKA
- Crypto SW resources :
  - CryptoLib

Flash part is protected from unwanted write or erase

Flash part is protected in execute only acces mode (core or debugging link) : no read or write access

Allow to lock any flash access if debugging link is open or remove any debugging access

Insure the STM32 boot to a specific address whatever happen

Define a portion of flash executed at boot time that can dissapear after execution

STM32 security features

- Resources: STM32 Unique ID
- Static Protection
  - WRP
  - PCROP
  - RDP
  - Unique entry boot
  - User Secure Mem / HDP
- Dynamic protection :
  - MPU
  - FIREWALL
  - TrustZone
  - OTFDEC
  - Tamper

- Crypto HW resources :
  - TRNG
  - CRYPT / AES
  - HASH
  - PKA
- Crypto SW resources :
  - CryptoLib

Memory protection unit allow memory isolation. This is a cortex feature.

Allow to define a security enclave (flash/ram) with a unique software entry point

Cortex M33 mechanism to create a strong isolation between secure and non-secure domain (hardware and software)

On the fly decryption of an external memory access thanks OCTOSPI interface

Allow detection of tamper event

STM32 security features

- Resources: STM32 Unique ID
  - PCROP
  - User Secure Mem / HDP
  - FIREWALL
  - TrustZone
  - OTFDEC
  - Tamper

- Crypto HW resources :
  - TRNG
  - CRYPT / AES
  - HASH
  - PKA
- Crypto SW resources :
  - CryptoLib

True random number generator

Symmetric encryption/decryption accelerator

Hash computation accelerator

Assymmetric cryptography accelerator

API implementation of symetric/assymetric cryptography (CAVP certified)

Security Features by STM32 Series

STM32 Series	Security Features																
	96-Bit Unique ID	FLASH WRP	FLASH PCROP	FLASH RDP	Unique entry point	Secure mem/ HDP	MPU	Firewall	Trustzone	OTFDEC	Tamper	TRNG	CRYPT AES	HASH	PKA	Cryptolib	Arm Cortex®
STM32 **F0**	🟩	🟩		🟩							🟩					🟩	**M0**
STM32 **F1**	🟩	🟩					🟩				🟩					🟩	**M3**
STM32 **F2**	🟩	🟩		🟩			🟩				🟩	🟩	▨	▨		🟩	**M3**
STM32 **F3**	🟩	🟩		🟩			🟩				🟩					🟩	**M4**
STM32 **F4**	🟩	🟩	▨	🟩			🟩				🟩	🟩	▨	▨		🟩	**M4**
STM32 **F7**	🟩	🟩	▨	🟩			🟩				🟩	🟩	▨	▨		🟩	**M7**
STM32 **L0**	🟩	🟩	🟩	🟩			🟩	🟩			🟩	▨	▨			🟩	**M0+**
STM32 **L1**	🟩	🟩	🟩	🟩			🟩				🟩		▨			🟩	**M3**
STM32 **L4**	🟩	🟩	🟩	🟩			🟩	🟩			🟩	🟩	▨	▨		🟩	**M4**
STM32 **L5**	🟩	🟩		🟩	🟩	🟩	🟩		🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	**M33**
STM32 **H7**	🟩	🟩	🟩	🟩	▨	▨	🟩				🟩	🟩	▨	▨		🟩	**M7/M4**
STM32 **G0**	🟩	🟩	🟩	🟩	▨	▨	🟩				🟩	🟩	▨			🟩	**M0+**
STM32 **G4**	🟩	🟩	🟩	🟩	▨	▨	🟩				🟩	🟩	▨			🟩	**M4**
STM32 **WB**	🟩	🟩	🟩	🟩			🟩				🟩	🟩	🟩		🟩	🟩	**M4/M0+**

🟩 Available on all devices

▨ Depends on device part number

life.augmented

Let’s discover those security features !

STM32 Unique ID

STM32 Unique ID

- Unique Device Identifier installed at the ST factory
  - Provides a reference number unique for any STM32

- The Unique ID is suited for:
  - Generating a serial number via an algorithm
  - Cryptographic Key derivation

- 96bits length in general composed of :
  - Lot number
  - Wafer ID
  - coordinate X of the die
  - coordinate Y of the die

STM32 Unique ID

- Tips : to get a unicity across all STM32 family Device unique ID should be combined with MCU device ID

- Tips : all the bits of the Unique Id are not used
  - Some bits will be always 0 for a given product
  - X and Y data has limited range
  - Lot id are coded in ASCII (number and capital letter)

There is possibility to shrink some how this ID but details need to be requested as could be different from a product regarding an other.

Security Features by STM32 Series

STM32 Series	Security Features																
	96-Bit Unique ID	FLASH WRP	FLASH PCROP	FLASH RDP	Unique entry point	Secure mem/ HDP	MPU	Firewall	Trustzone	OTFDEC	Tamper	TRNG	CRYPT AES	HASH	PKA	Cryptolib	Arm Cortex®
STM32 **F0**	🟩	🟩		🟩							🟩					🟩	**M0**
STM32 **F1**	🟩	🟩					🟩				🟩					🟩	**M3**
STM32 **F2**	🟩	🟩		🟩			🟩				🟩	🟩	▨	▨		🟩	**M3**
STM32 **F3**	🟩	🟩		🟩			🟩				🟩					🟩	**M4**
STM32 **F4**	🟩	🟩	▨	🟩			🟩				🟩	🟩	▨	▨		🟩	**M4**
STM32 **F7**	🟩	🟩	▨	🟩			🟩				🟩	🟩	▨	▨		🟩	**M7**
STM32 **L0**	🟩	🟩	🟩	🟩			🟩	🟩			🟩		▨			🟩	**M0+**
STM32 **L1**	🟩	🟩	🟩	🟩			🟩				🟩		▨			🟩	**M3**
STM32 **L4**	🟩	🟩	🟩	🟩			🟩	🟩			🟩	🟩	▨	▨		🟩	**M4**
STM32 **L5**	🟩	🟩		🟩	🟩	🟩	🟩		🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	**M33**
STM32 **H7**	🟩	🟩	🟩	🟩	▨	▨	🟩				🟩	🟩	▨	▨		🟩	**M7/M4**
STM32 **G0**	🟩	🟩	🟩	🟩	▨	▨	🟩				🟩	🟩	▨			🟩	**M0+**
STM32 **G4**	🟩	🟩	🟩	🟩	▨	▨	🟩				🟩	🟩	▨			🟩	**M4**
STM32 **WB**	🟩	🟩	🟩	🟩			🟩				🟩	🟩	🟩		🟩	🟩	**M4/M0+**

🟩 Available on all devices

▨ Depends on device part number

life.augmented

Hands-on

Hands-on

- Purpose : check the unique ID of STM32L476RG and possibility to shrink it.

Hands-on

- Check the Unique ID description in the reference manual

  https://www.st.com/en/microcontrollers-microprocessors/stm32l476rg.html

- For example for STM32L476RG, RM0351 :

  Base address: 0x1FFF 7590

  Address offset: 0x00

  Bits 31:0 UID[31:0]: X and Y coordinates on the wafer

  Address offset: 0x04

  Bits 31:8 UID[63:40]: LOT_NUM[23:0] Lot number (ASCII encoded)

  Bits 7:0 UID[39:32]: WAF_NUM[7:0] Wafer number (8-bit unsigned number)

  Address offset: 0x08

  Bits 31:0 UID[95:64]: LOT_NUM[55:24] Lot number (ASCII encoded)

Hands-on

Address offset: 0x00 : 0x002F003E X and Y coordinates on the wafer

Address offset: 0x04 : 0x4233500E

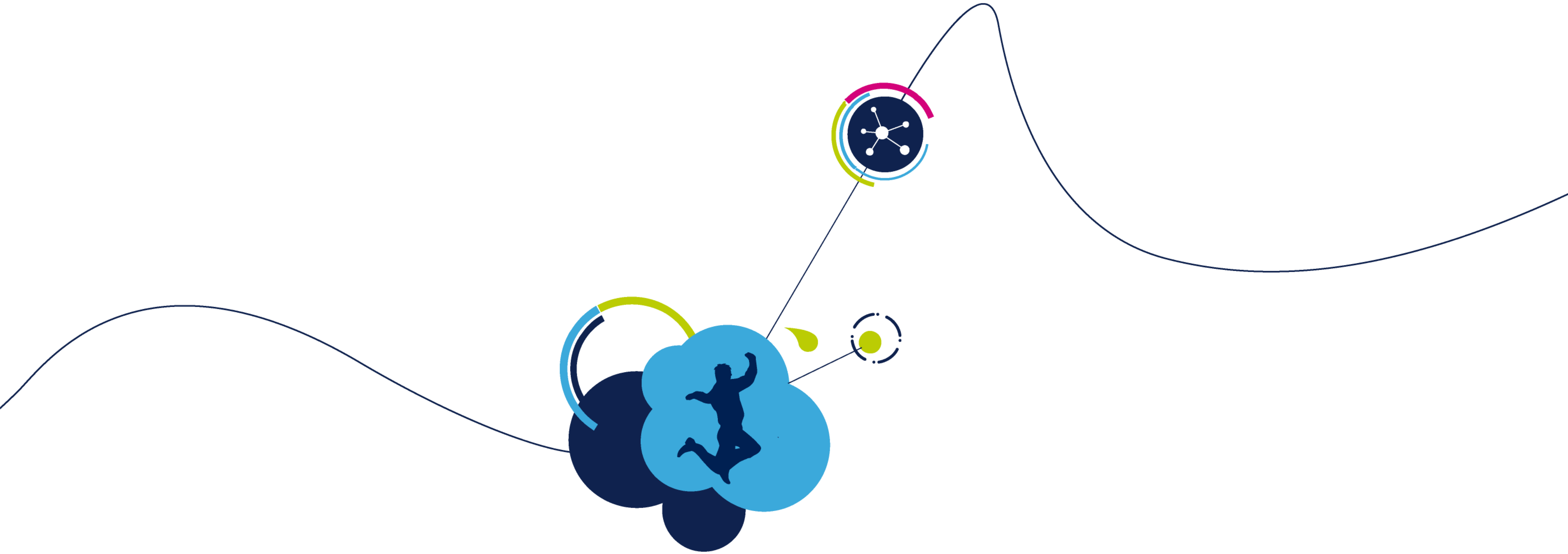
0x423350	Lot number (ASCII encoded: “B3P”)
0x0E	Wafer number (8-bit unsigned number)

Address offset: 0x08 : 0x20313558 Lot number (ASCII encoded: ” 15X”)

All LOTIDs are ASCII code of values ranging from {0,1,2,3,4,5,6,7,8,9} /Capital letters {A,B....Z}/ space so if we convert to Hex format they will be bytes in these ranges { 0x20 .. 0x39 } to { 0x41 .. 0x5A}

-> a value between 0x20 to 0x5A could be coded on 6 bits

We can save 14bits.



Write protection

STM32 Memory Features

FLASH Write Protection (**WRP**)

- The Flash memory write protection mechanism is designed to prevent unwanted write access to defined areas in Flash memory
- The write-protected area is defined on a per-sector basis.
- It’s configured thanks option byte.
    - STM32L4/G0/G4/WB/L5 : the WRP area is defined by “start” and “end” addresses
    - All others, one user option bits per sector.
- The embedded Flash memory write-protection user option bits can be modified without any restriction when the RDP level is set to level 0 or level 1.

STM32 Memory Features

RAM Write Protection

- RAM could also be write protected but this is only available on :
  - STM32L4/STM32L5 SRAM2 is write protectable with page granularity if 1Kbyte
  - STM32G4 CCM SRAM is write protectable
- It’s configured written in a system register (SYSCFG_SWPR1/2) and it can be removed/cleared by a system reset only
- for the STM32L5, if TZEN =1 only secure access could access the SYSCFG_SWPR1/2 register

STM32 Memory Features security tips

FLASH Write Protection (**WRP**)

- TIPS : WRP flash protection + RDP level2 is equivalent to ROM. This guaranty immutability of the code which is a key requirement for a secure boot.

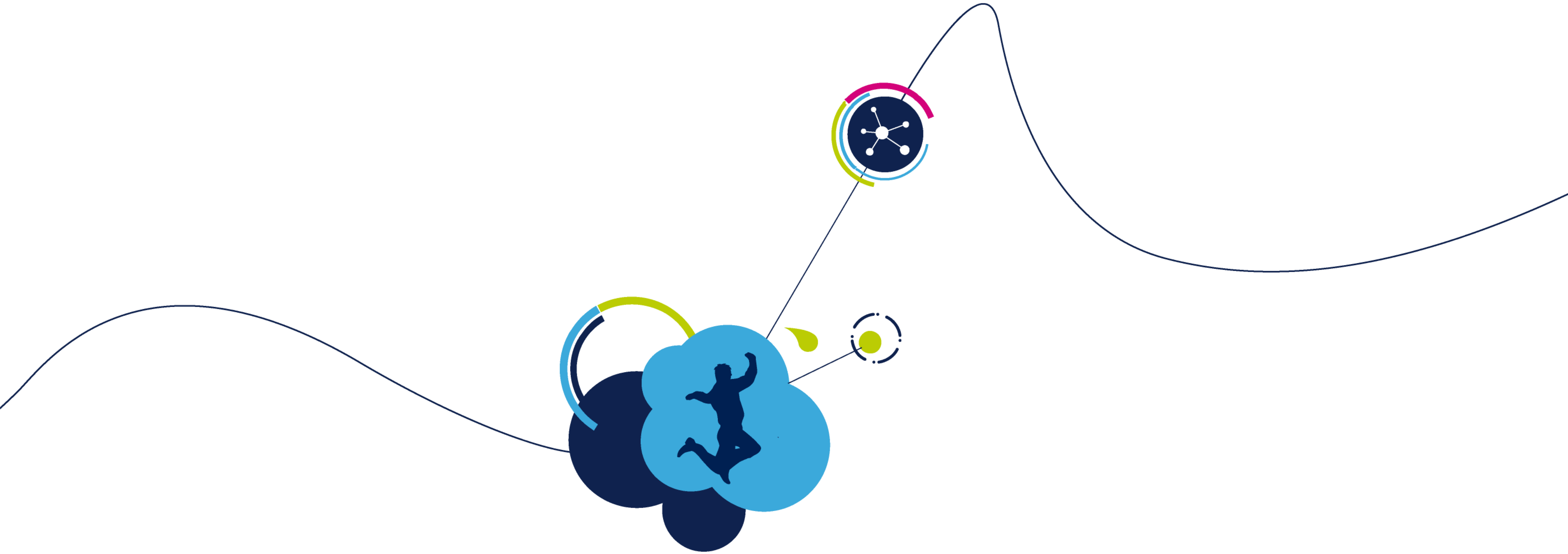
Security Features by STM32 Series

STM32 Series	Security Features																
	96-Bit Unique ID	FLASH WRP	FLASH PCROP	FLASH RDP	Unique entry point	Secure mem/ HDP	MPU	Firewall	Trustzone	OTFDEC	Tamper	TRNG	CRYPT AES	HASH	PKA	Cryptolib	Arm Cortex®
STM32 **F0**	●	●		●							●					●	**M0**
STM32 **F1**	●	●					●				●					●	**M3**
STM32 **F2**	●	●		●			●				●	●	◐	◐		●	**M3**
STM32 **F3**	●	●		●			●				●					●	**M4**
STM32 **F4**	●	●	◐	●			●				●	●	◐	◐		●	**M4**
STM32 **F7**	●	●	◐	●			●				●	●	◐	◐		●	**M7**
STM32 **L0**	●	●	●	●			●	●			●		◐			●	**M0+**
STM32 **L1**	●	●	●	●			●				●		◐			●	**M3**
STM32 **L4**	●	●	●	●			●	●			●	●	◐	◐		●	**M4**
STM32 **L5**	●	●		●	●	●	●		●	●	●	●	●	●	●	●	**M33**
STM32 **H7**	●	●	●	●	◐	◐	●				●	●	◐	◐		●	**M7/M4**
STM32 **G0**	●	●	●	●	◐	◐	●				●	●	◐			●	**M0+**
STM32 **G4**	●	●	●	●	◐	◐	●				●	●	◐			●	**M4**
STM32 **WB**	●	●	●	●			●				●	●	●		●	●	**M4/M0+**

● Available on all devices

◐ Depends on device part number

life.augmented

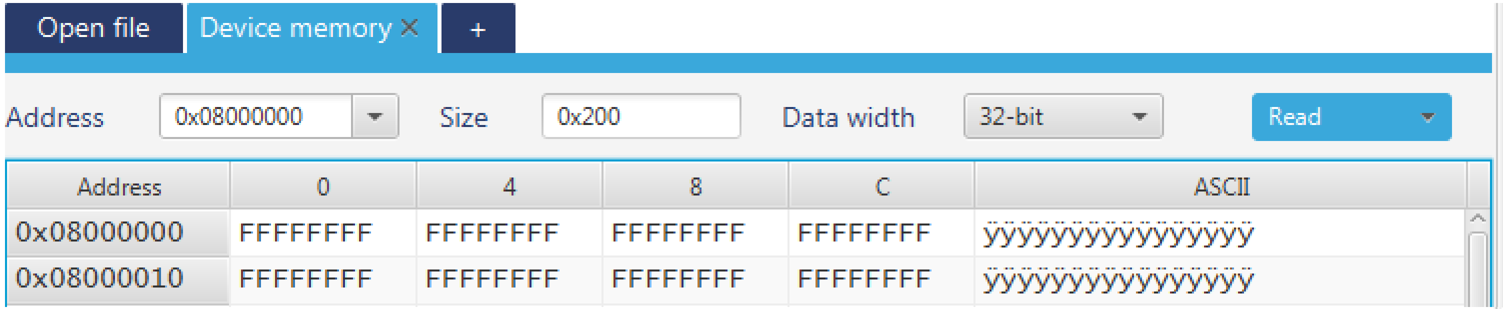


WRP Hands-on

Hands-on

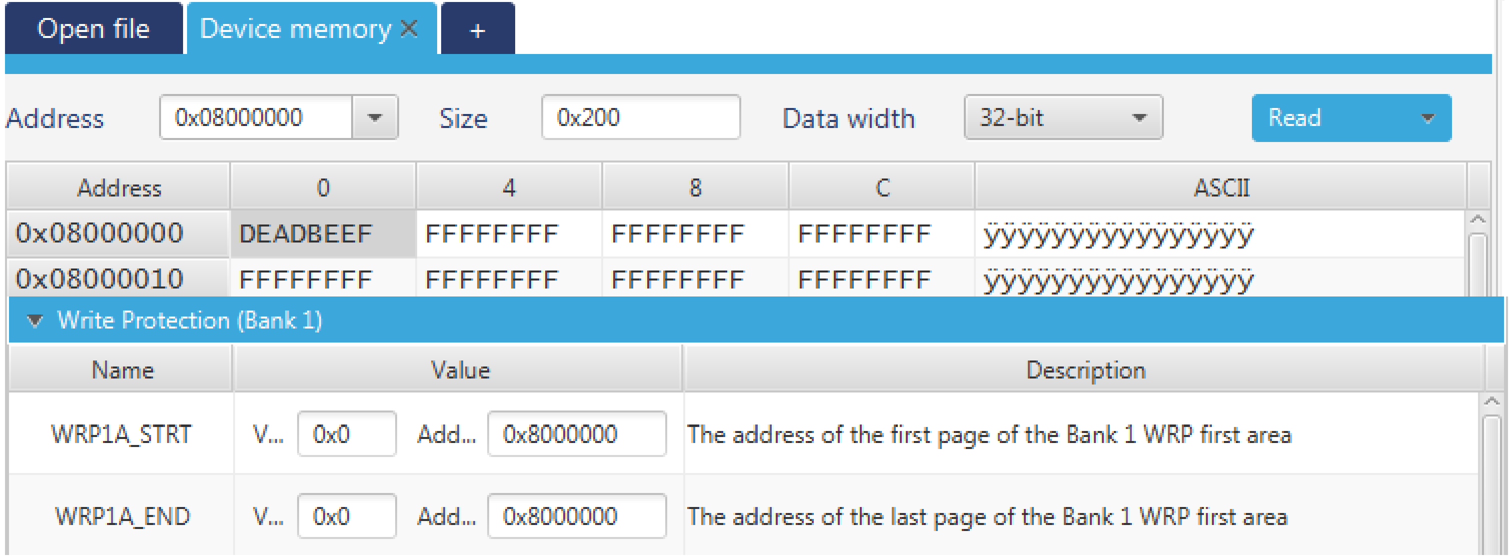
- Purpose :
  - Check the flash WRP protection with CubeProgrammer
  - Check the SRAM2 WRP protection thanks CubeIDE

Hands-on



Address 0x08000000 Size 0x200 Data width 32-bit

Address	0	4	8	C	ASCII
0x08000000	FFFFFFFF	FFFFFFFF	FFFFFFFF	FFFFFFFF	ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
0x08000010	FFFFFFFF	FFFFFFFF	FFFFFFFF	FFFFFFFF	ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ



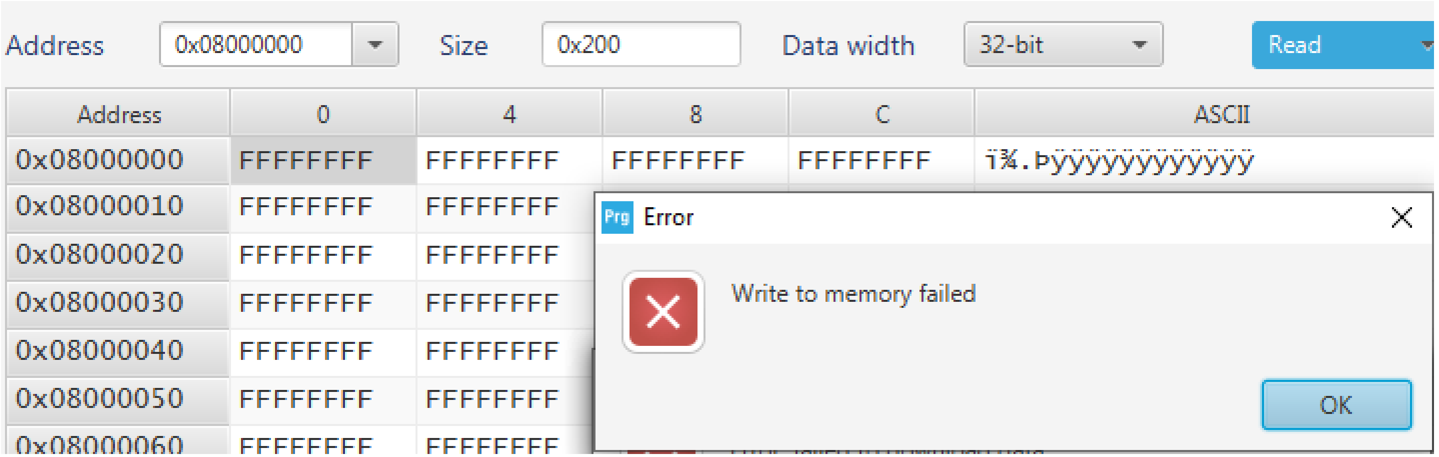
Address 0x08000000 Size 0x200 Data width 32-bit

Address	0	4	8	C	ASCII
0x08000000	DEADBEEF	FFFFFFFF	FFFFFFFF	FFFFFFFF	ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
0x08000010	FFFFFFFF	FFFFFFFF	FFFFFFFF	FFFFFFFF	ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ

Write Protection (Bank 1)

Name	Value	Description
WRP1A_STRT	V... 0x0 Add... 0x8000000	The address of the first page of the Bank 1 WRP first area
WRP1A_END	V... 0x0 Add... 0x8000000	The address of the last page of the Bank 1 WRP first area

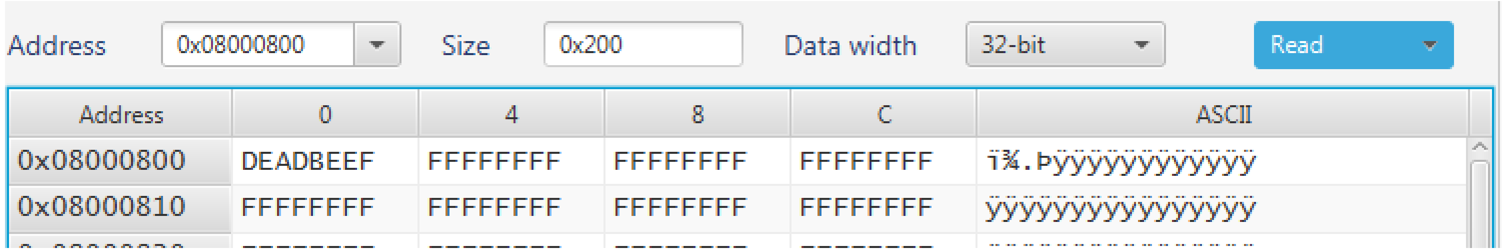
Hands-on



Address 0x08000000 Size 0x200 Data width 32-bit

Address	0	4	8	C	ASCII
0x08000000	FFFFFFFF	FFFFFFFF	FFFFFFFF	FFFFFFFF	ï¾.Þÿÿÿÿÿÿÿÿÿÿÿÿ
0x08000010	FFFFFFFF	FFFFFFFF			
0x08000020	FFFFFFFF	FFFFFFFF			
0x08000030	FFFFFFFF	FFFFFFFF			
0x08000040	FFFFFFFF	FFFFFFFF			
0x08000050	FFFFFFFF	FFFFFFFF			
0x08000060	FFFFFFFF	FFFFFFFF			

Write to memory failed



Address 0x08000800 Size 0x200 Data width 32-bit

Address	0	4	8	C	ASCII
0x08000800	DEADBEEF	FFFFFFFF	FFFFFFFF	FFFFFFFF	ï¾.Þÿÿÿÿÿÿÿÿÿÿÿÿ
0x08000810	FFFFFFFF	FFFFFFFF	FFFFFFFF	FFFFFFFF	ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ

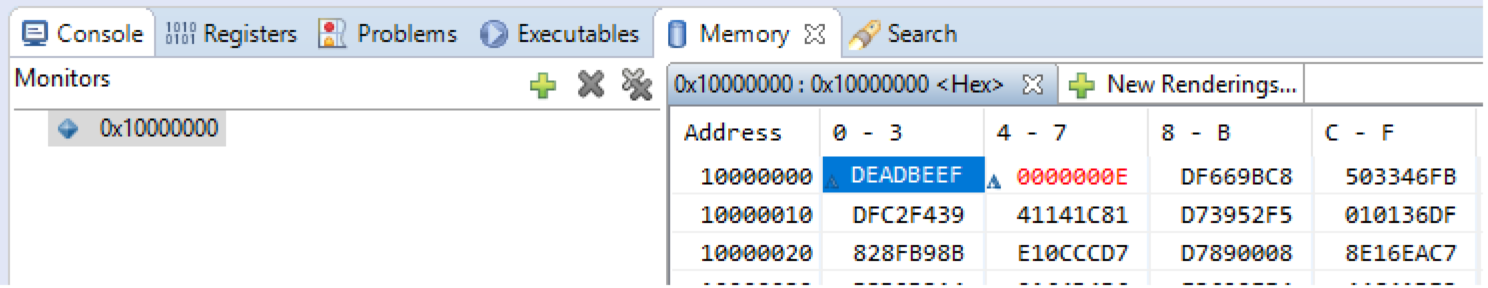
Hands-on

- Purpose :
  - Check the SRAM2 WRP protection thanks CubeIDE

Hands-on

- Check in the reference manual
  - SRAM2 address 0x10000000
- Start CubeIDE, generate a project for NucleoL476RG
- Add this code in the while(1) loop

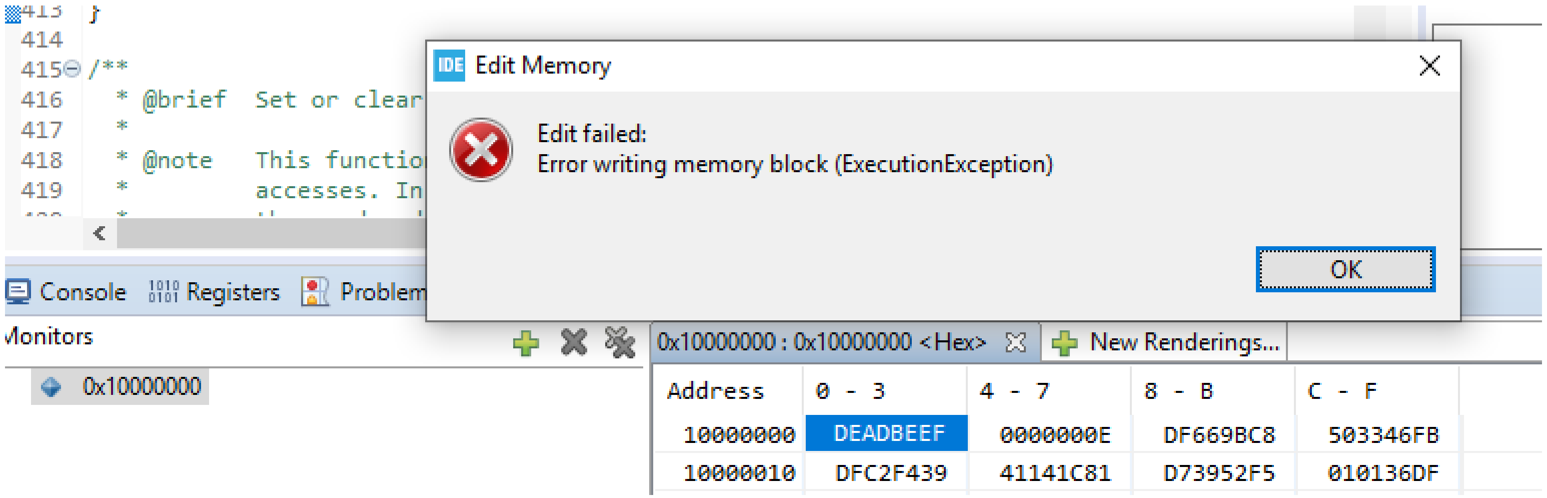
```
if (HAL_GPIO_ReadPin(B1_GPIO_Port, B1_Pin) == 0) {
SYSCFG->SWPR=0x1;
}
```



```
0x10000000

0x10000000 : 0x10000000 <Hex>

Address    0 - 3      4 - 7      8 - B      C - F
10000000   DEADBEEF   0000000E   DF669BC8   503346FB
10000010   DFC2F439   41141C81   D73952F5   010136DF
10000020   828FB98B   E10CCCD7   D7890008   8E16EAC7
```

Hands-on

- Push button, then try to write again in the SRAM2



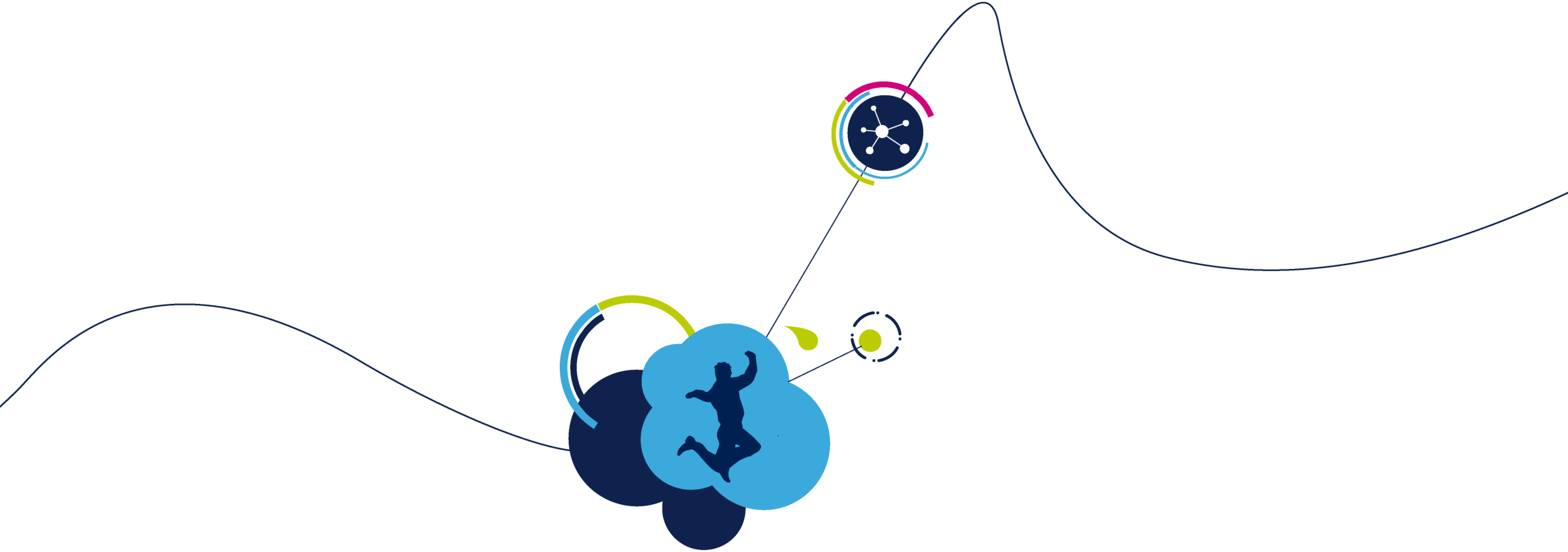
```
413 }
414
415 /**
416   * @brief  Set or clear
417   *
418   * @note   This functio
419   *         accesses. In
```

Edit Memory

Edit failed:
Error writing memory block (ExecutionException)

Address	0 - 3	4 - 7	8 - B	C - F
10000000	DEADBEEF	0000000E	DF669BC8	503346FB
10000010	DFC2F439	41141C81	D73952F5	010136DF

- Reset, then try to write again in the SRAM2



Proprietary Code Readout Protection

life.augmented

STM32 Memory Features

Proprietary Code ReadOut Protection **PCROP**

- This feature set a memory area in flash defined by user in execute only mode
- The CPU can only jump to a PCROP area but cannot read or write it.
- This region can’t be read via debugging link.
- Setting thanks option byte and configured:
    - by sectors on STM32F4, STM32F7, STM32L0, STM32L1
    - by area on STM32L4,STM32H7,STM32G0, STM32G4, STM32WB

STM32 Memory Features

Proprietary Code ReadOut Protection **PCROP**

- PCROP can only be removed when 2 conditions are covered
  - PCROP_RDP option bit is set to 1
  - Perform RDP1 to RDP0 regression

  => in that case a mass erase is performed and all PCROP regions removed

- If PCROP_RDP option is set to 0, the regression RDP1 to RDP0 will perform a mass erase without altering the PCROP regions setting and content.

PCROP software constraint

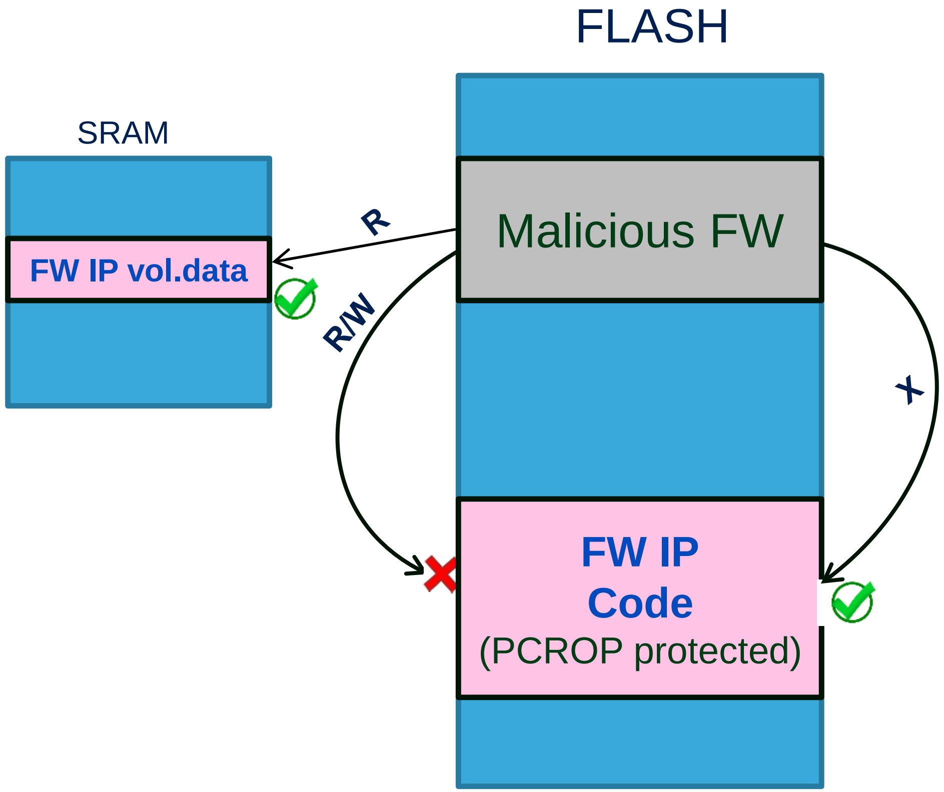
- The code in PCROP has to be compiled to remove all data access.
  - GCC option : No data reads in code memory
  - Keil: option : Execute-Only code
  - IAR option: No data reads in code memory

- Code running in PCROP must be put in a specific area in the scatter file to place this read-only code in the good area of the flash

STM32 Memory Features

Proprietary Code ReadOut Protection **PCROP**

- Benefits
  - FW IP protection
  - Mutual protection of FW IPs
  - Interesting feature in RDP level0
  - Code integrity
- Weakness
  - Volatile data (SRAM) not protected
  - Specific compilation required



STM32 Memory Features

Proprietary Code ReadOut Protection **PCROP**

- Tips : Possibility to hide some data in PCROP region
  Use assembly code which could generate a value without any data access. This could be use to protect cryptographic key values

```
MOVS   R1, #0x5f
LSLS    R1, R1, #24
MOVS   R5, #0x4d
LSLS    R5, R5, #16
ADD     R1, R1, R5
MOVS   R5, #0x45
LSLS    R5, R5, #8
ADD     R1, R1, R5
MOVS   R5, #0x4f
ADD     R1, R1, R5
```

Executing this code, you have 0x5f4d454f in R1 without any data access.

STM32 Memory Features

Proprietary Code ReadOut Protection **PCROP**

- STM32L0
  Mutual exclusion with WRP protection

- STM32F4
  Not available on STM32F405/407/415/417
  Mutual exclusion with WRP protection

- STM32F7

  Not available on STM32F74xxx/STM32F75xxx /STM32F76xxx/STM32F77xxx

Security Features by STM32 Series

STM32 Series	Security Features																
	96-Bit Unique ID	FLASH WRP	FLASH PCROP	FLASH RDP	Unique entry point	Secure mem/ HDP	MPU	Firewall	Trustzone	OTFDEC	Tamper	TRNG	CRYPT AES	HASH	PKA	Cryptolib	Arm Cortex®
STM32 **F0**	🟩	🟩		🟩							🟩					🟩	**M0**
STM32 **F1**	🟩	🟩					🟩				🟩					🟩	**M3**
STM32 **F2**	🟩	🟩		🟩			🟩				🟩	🟩	▨	▨		🟩	**M3**
STM32 **F3**	🟩	🟩		🟩			🟩				🟩					🟩	**M4**
STM32 **F4**	🟩	🟩	▨	🟩			🟩				🟩	🟩	▨	▨		🟩	**M4**
STM32 **F7**	🟩	🟩	▨	🟩			🟩				🟩	🟩	▨	▨		🟩	**M7**
STM32 **L0**	🟩	🟩	🟩	🟩			🟩	🟩			🟩		▨			🟩	**M0+**
STM32 **L1**	🟩	🟩	🟩	🟩			🟩				🟩		▨			🟩	**M3**
STM32 **L4**	🟩	🟩	🟩	🟩			🟩	🟩			🟩	🟩	▨	▨		🟩	**M4**
STM32 **L5**	🟩	🟩		🟩	🟩	🟩	🟩		🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	**M33**
STM32 **H7**	🟩	🟩	🟩	🟩	▨	▨	🟩				🟩	🟩	▨	▨		🟩	**M7/M4**
STM32 **G0**	🟩	🟩	🟩	🟩	▨	▨	🟩				🟩	🟩	▨			🟩	**M0+**
STM32 **G4**	🟩	🟩	🟩	🟩	▨	▨	🟩				🟩	🟩	▨			🟩	**M4**
STM32 **WB**	🟩	🟩	🟩	🟩			🟩				🟩	🟩	🟩		🟩	🟩	**M4/M0+**

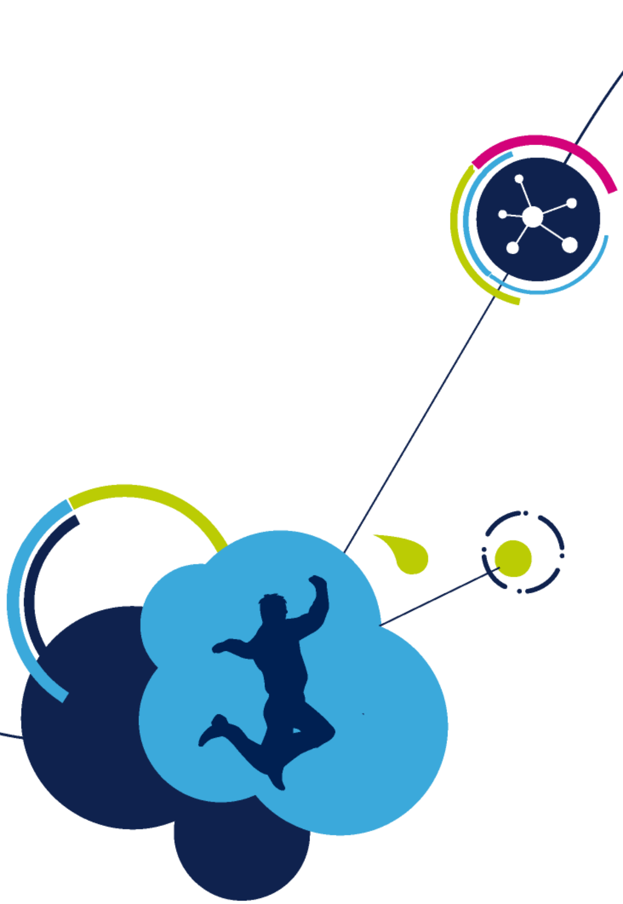
🟩 Available on all devices

▨ Depends on device part number

life.augmented

Reference

- X-CUBE-PCROP : practical usage of the PCROP protection feature on microcontrollers of the STM32F4, STM32F7 and STM32L4 Series
- AN4701:Proprietary code read-out protection on microcontrollers of the STM32F4 Series
- AN4758:Proprietary code read-out protection on STM32L4, STM32L4+ and STM32G4 Series microcontrollers
- AN4968:Proprietary code read out protection (PCROP) on STM32F72xxx and STM32F73xxx microcontrollers



PCROP Hands-on

Hands-on

- Purpose :
  - Hide a call to led blinking API in PCROP region

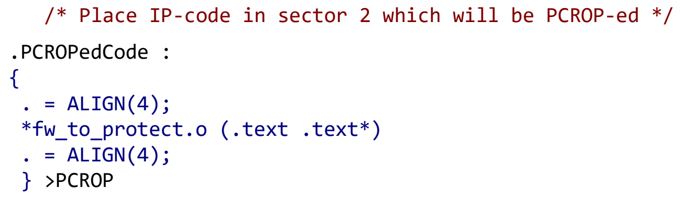
Hands-on

- Step 1 : create a fw_to_protect.c

```
#include "main.h"
uint32_t cpt = 0;

void  toggle_led(void);

void  toggle_led(void)
{
                        cpt++;
                         HAL_GPIO_TogglePin(LD2_GPIO_Port,LD2_Pin);
}
```

- Step 2 : modify the ld file

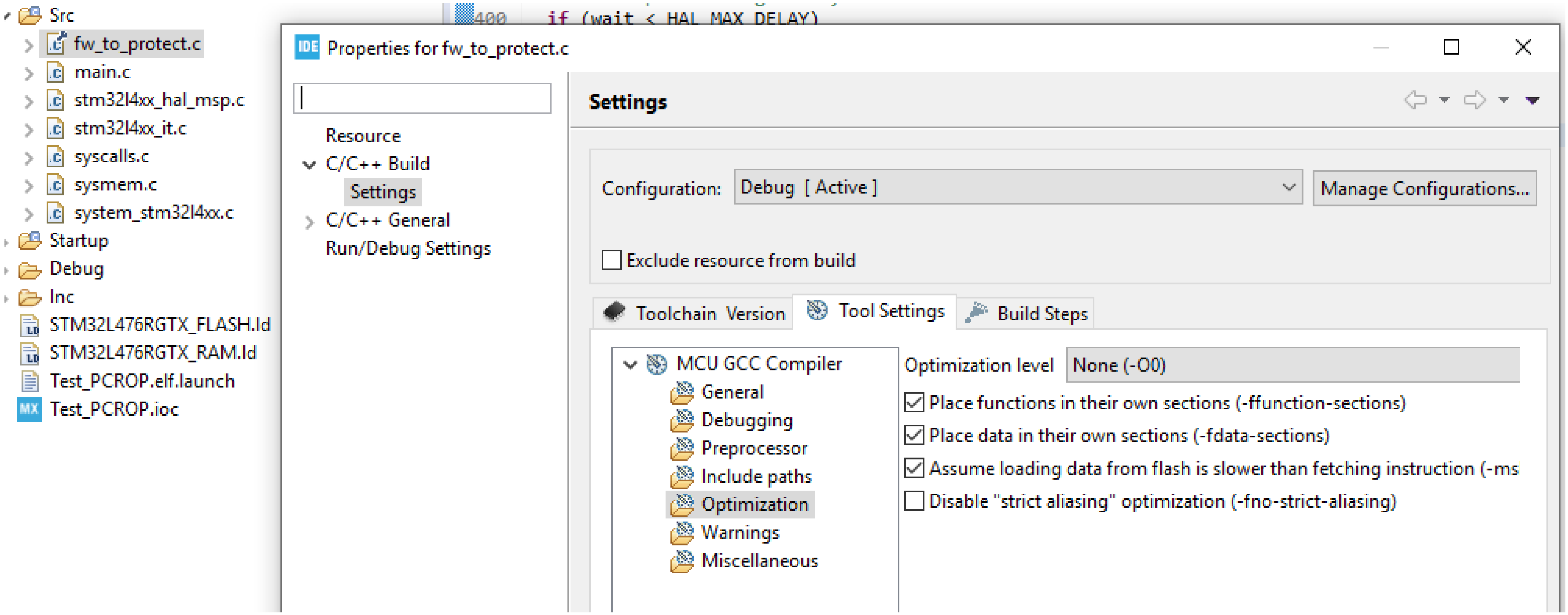
```
FLASH (rx): ORIGIN = 0x8000000,LENGTH = 16K
PCROP (x) : ORIGIN = 0x08008000, LENGTH = 16K
```



```
   /* Place IP-code in sector 2 which will be PCROP-ed */
.PCROPedCode :
{
 . = ALIGN(4);
 *fw_to_protect.o (.text .text*)
 . = ALIGN(4);
 } >PCROP
```

Hands-on

- Step 3 : modify compilation option



Src
- fw_to_protect.c
- main.c
- stm32l4xx_hal_msp.c
- stm32l4xx_it.c
- syscalls.c
- sysmem.c
- system_stm32l4xx.c

Startup
Debug
Inc
STM32L476RGTX_FLASH.ld
STM32L476RGTX_RAM.ld
Test_PCROP.elf.launch
Test_PCROP.ioc

```
400    if (wait < HAL_MAX_DELAY)
```

Properties for fw_to_protect.c

Resource
C/C++ Build
- Settings

C/C++ General
Run/Debug Settings

Settings

Configuration: Debug [Active]   Manage Configurations...

□ Exclude resource from build

Toolchain Version | Tool Settings | Build Steps

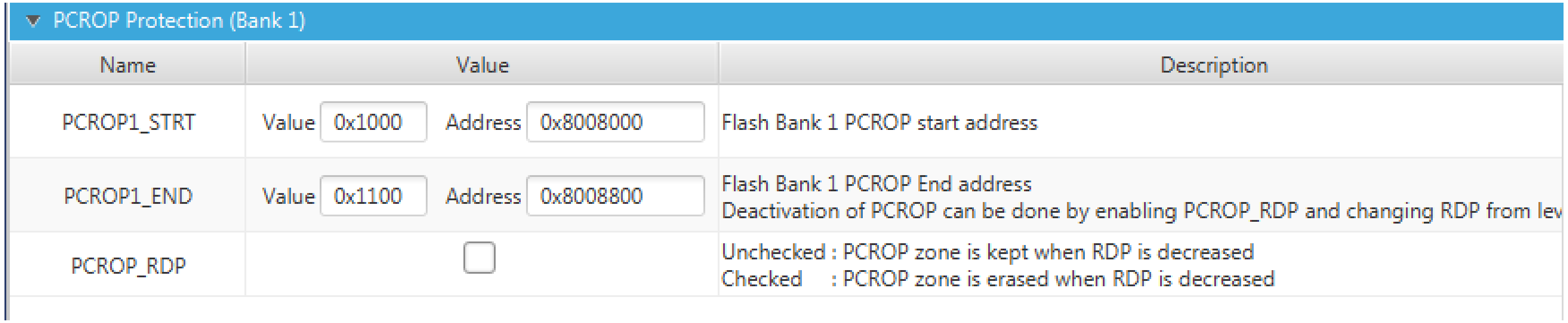
MCU GCC Compiler
- General
- Debugging
- Preprocessor
- Include paths
- Optimization
- Warnings
- Miscellaneous

Optimization level: None (-O0)

☑ Place functions in their own sections (-ffunction-sections)
☑ Place data in their own sections (-fdata-sections)
☑ Assume loading data from flash is slower than fetching instruction (-ms
□ Disable "strict aliasing" optimization (-fno-strict-aliasing)

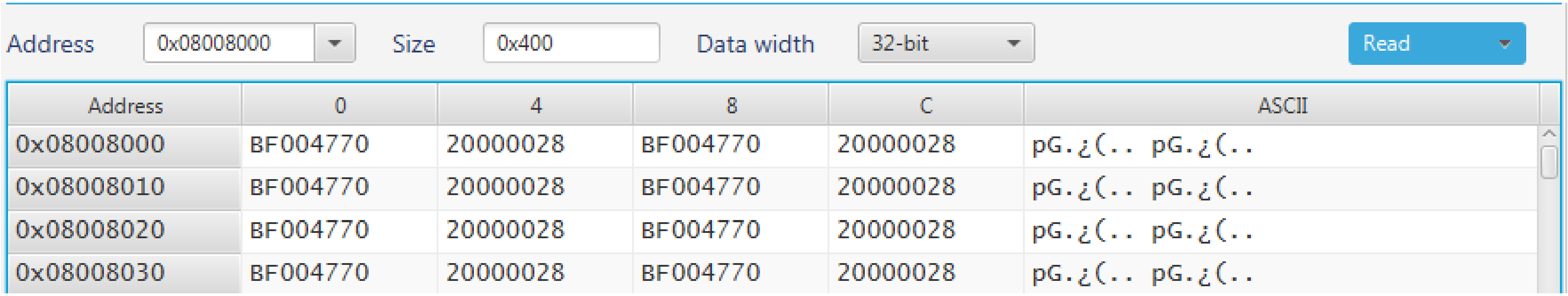
Hands-on

- Step 3 : Activate PCROP



▼ PCROP Protection (Bank 1)

Name	Value	Description
PCROP1_STRT	Value 0x1000 Address 0x8008000	Flash Bank 1 PCROP start address
PCROP1_END	Value 0x1100 Address 0x8008800	Flash Bank 1 PCROP End address Deactivation of PCROP can be done by enabling PCROP_RDP and changing RDP from lev
PCROP_RDP	□	Unchecked : PCROP zone is kept when RDP is decreased Checked : PCROP zone is erased when RDP is decreased

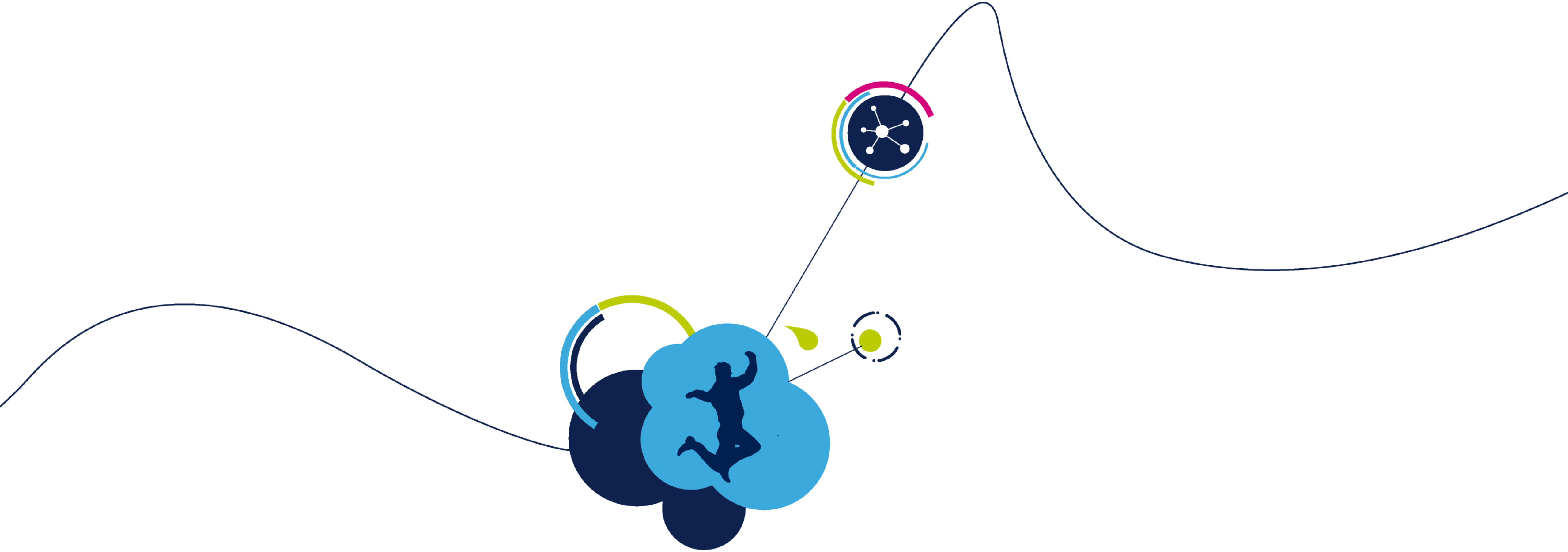


Address 0x08008000 Size 0x400 Data width 32-bit

Address	0	4	8	C	ASCII
0x08008000	BF004770	20000028	BF004770	20000028	pG.¿(.. pG.¿(..
0x08008010	BF004770	20000028	BF004770	20000028	pG.¿(.. pG.¿(..
0x08008020	BF004770	20000028	BF004770	20000028	pG.¿(.. pG.¿(..
0x08008030	BF004770	20000028	BF004770	20000028	pG.¿(.. pG.¿(..

Hands-on

- Step 4 : check protection
- Step 5: remove PCROP with STM32CubeProgrammer



Readout Protection

RDP protection

- 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

- RDP applies to all STM32 series for:
  - main Flash memory
  - option bytes modification (level 2 only)

RDP protection

- Depending on the STM32 series, additional protections may be available including:
  - backup registers for real-time clock (RTC)
    - STM32F0 / STM32F3 / STM32L4 / STM32L5 / STM32G0 /STM32G4 /STM32WB
  - backup SRAM
    - STM32F2 / STM32F4 / STM32F7 / STM32H7
  - SRAM2
    - STM32L4 / STM32L5 / STM32WB
  - CCM-SRAM (Core Couple Memory)
    - STM32G4

RDP protection

- Most of STM32 have 3 levels of Readout protection :
  - Level 0 : Value 0xAA
  - Level 1 : Value 0xBB and all values except 0xAA and 0xCC
  - Level 2 : Value 0xCC

- On STM32F1, only level 1 is available

- On STM32L5, there is a level 0.5 associated with Trustzone (will be covered with TrustZone)

Readout protection (1/2)

- Readout protection **Level 0** (no protection, factory default)
  - R/W/Erase possible on Flash memory, SRAM* and Backup registers*.
  - Option bytes change possible

Open device

Readout protection (1/2)

- Readout protection **Level 1**

  If boot mode = user Flash and no debugger access is detected then

  - R/W/Erase possible on Flash memory, SRAM* and Backup registers*.
  - Option bytes change possible

  If boot mode ≠ user Flash or debugger access is detected then

  - R/W/Erase to Flash memory, SRAM* (family dependant), and Backup registers are blocked (hard fault generated).
  - Option bytes change possible.

Memory protected

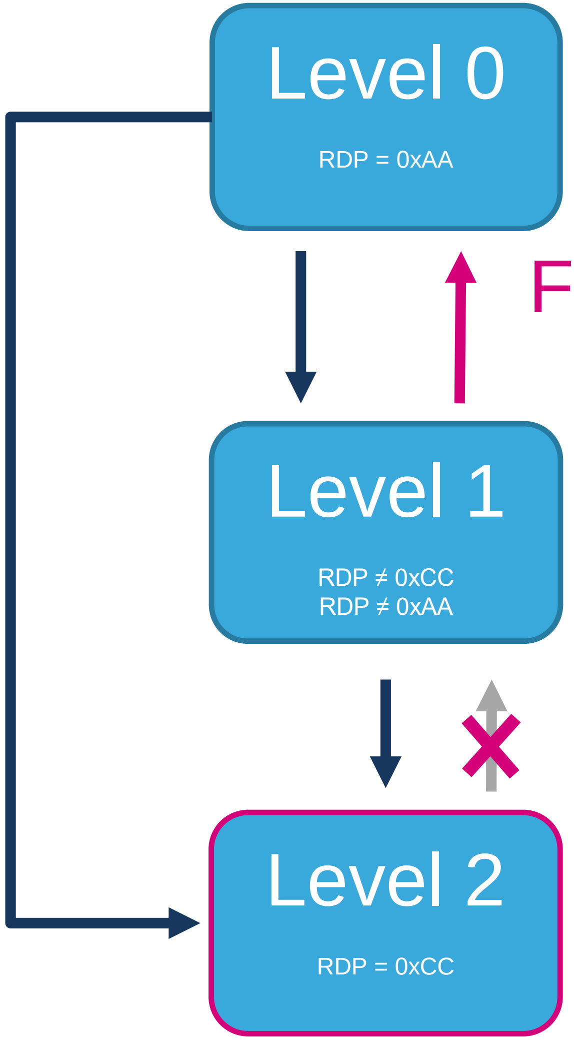
Flash / Backup* / Sram*

Readout protection (2/2)

- Readout protection Level 2
  - All protections provided by Level 1 are active.
  - Boot only from User Flash memory
  - No JTAG
  - Option bytes can no longer be changed
  - No RDP regression possible

Locked device

RDP transition



Flash memory
Mass Erase

Permanent state
Option byte changes no longer possible

RDP transition tips

- On activation of the RDP, a full power cycle is needed (or a transition from low power standby to run state for some device)
  This should be check in the reference manual.

RDP Level 1 Tips

- RDP1 : you can connect with debugger and see :
  - SRAM
  - Peripheral registers

- If software is reading the content of the flash to perform a CRC check or a hash, it could be possible, depending on implementation that all the firmware content would be visible somewhere in SRAM.

- If the software disables the SWD/JTAG GPIOs, you can still connect under reset and read the peripheral registers and SRAM content.

RDP level1 Tips

- To avoid the such issue, the software should introduce various delays in the code execution.

- The code used could be written so that only CPU registers are used to perform a sensitive operation.

RDP summary

RDP
level 0

- OPEN DEVICE

RdP
level 1

- Flash / Backup* / Sram* memories are secured
- All the rest is not

RdP
Level2

- Option bytes are frozen
- FW update remain possible
- High constraints:
  - No RMA (return mode analysis)

Security Features by STM32 Series

STM32 Series	Security Features															
	96-Bit Unique ID	FLASH WRP	FLASH PCROP	FLASH RDP	Unique entry point	Secure mem/ HDP	MPU	Firewall	Trustzone	OTFDEC	Tamper	TRNG	CRYPT AES	HASH	PKA	Cryptolib	Arm Cortex®
STM32 **F0**	🟩	🟩		🟩							🟩					🟩	**M0**
STM32 **F1**	🟩	🟩					🟩				🟩					🟩	**M3**
STM32 **F2**	🟩	🟩		🟩			🟩				🟩	🟩	▨	▨		🟩	**M3**
STM32 **F3**	🟩	🟩		🟩			🟩				🟩					🟩	**M4**
STM32 **F4**	🟩	🟩	▨	🟩			🟩				🟩	🟩	▨	▨		🟩	**M4**
STM32 **F7**	🟩	🟩	▨	🟩			🟩				🟩	🟩	▨	▨		🟩	**M7**
STM32 **L0**	🟩	🟩	🟩	🟩			🟩	🟩			🟩		▨			🟩	**M0+**
STM32 **L1**	🟩	🟩	🟩	🟩			🟩				🟩		▨			🟩	**M3**
STM32 **L4**	🟩	🟩	🟩	🟩			🟩	🟩			🟩	🟩	▨	▨		🟩	**M4**
STM32 **L5**	🟩	🟩		🟩	🟩	🟩	🟩		🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	**M33**
STM32 **H7**	🟩	🟩	🟩	🟩	▨	▨	🟩				🟩	🟩	▨	▨		🟩	**M7/M4**
STM32 **G0**	🟩	🟩	🟩	🟩	▨	▨	🟩				🟩	🟩	▨			🟩	**M0+**
STM32 **G4**	🟩	🟩	🟩	🟩	▨	▨	🟩				🟩	🟩	▨			🟩	**M4**
STM32 **WB**	🟩	🟩	🟩	🟩			🟩				🟩	🟩	🟩		🟩	🟩	**M4/M0+**

🟩 Available on all devices

▨ Depends on device part number

RDP Hands-on

Hands-on

- Purpose : test the RDP protection

Step 1 : create and load a blinking project

Step 2 : do a transition to RDP 1 and check the status

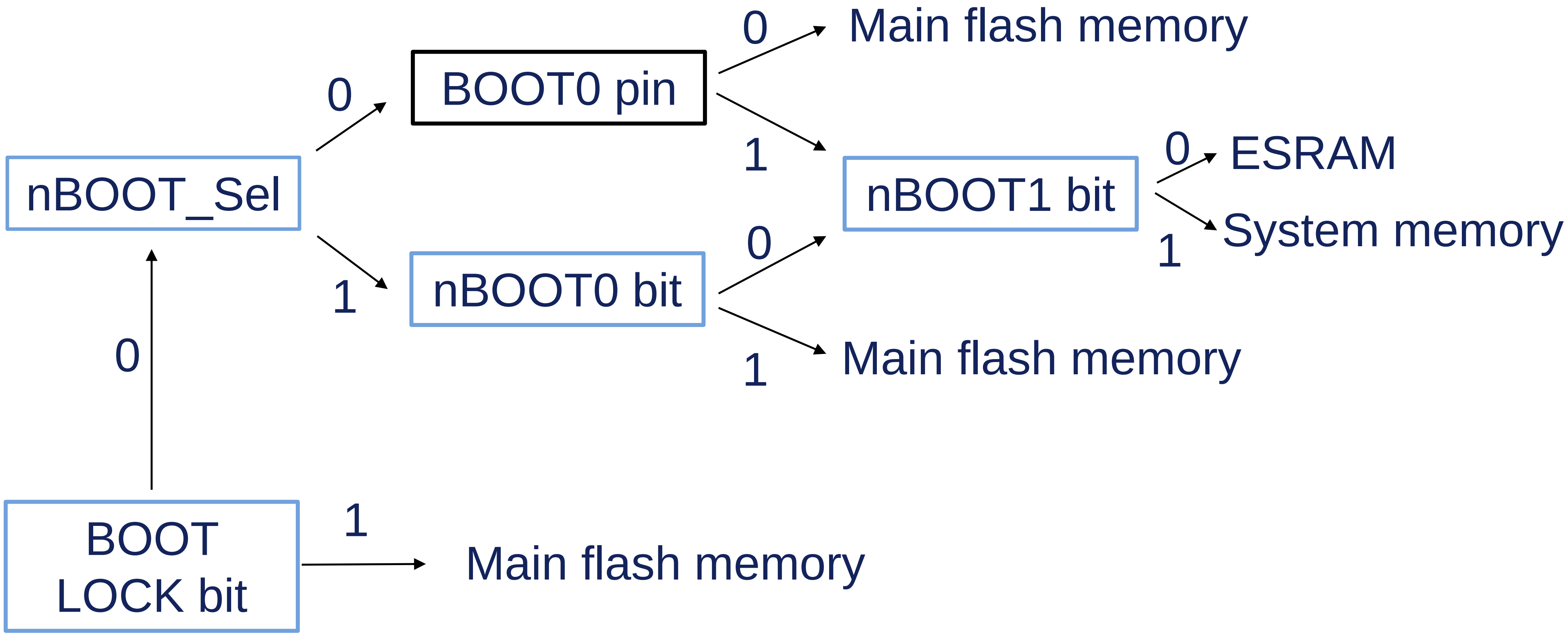
Step 3 : do a RDP regression to level 0

Unique boot entry

Boot configuration

- Reminder all STM32 could boot from :
  - User Flash memory (0x08000000)
  - Embedded bootloader
  - SRAM
- Using RDP2 restrict to boot from User Flash memory
- Additional feature on
  - STM32 G0, STM32 G4, STM32L5 : mechanism to insure boot from User flash memory even in RDP1
  - STM32H7 : mechanism link with secure memory / RSS

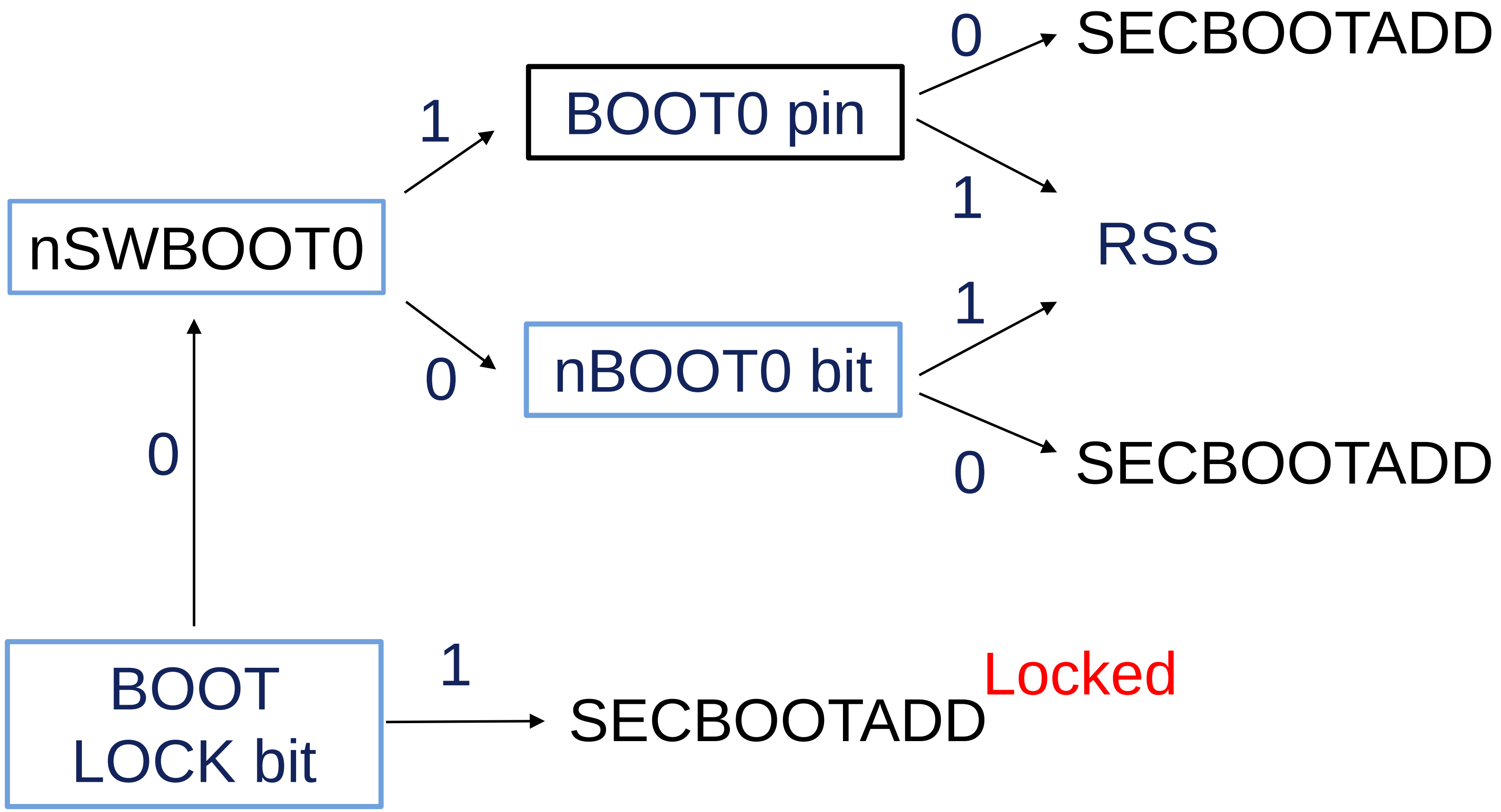
Boot configuration G0/G4



Boot lock STM32G0/G4

- Purpose: lock boot on user flash whatever boot configuration
- If BOOT_LOCK=1 device boots on user flash whatever the state of nBootX
- BOOT_LOCK can change from 0 => 1 in RDP0 and RDP1
- BOOT_LOCK can change from 1 => 0 **only** in RDP0
- So, in RDP1, with BOOT_LOCK set, the only way to go back is
  - Make RDP1 => RDP0 regression : mass erase will be performed
  - Change BOOT_LOCK value once in RDP0

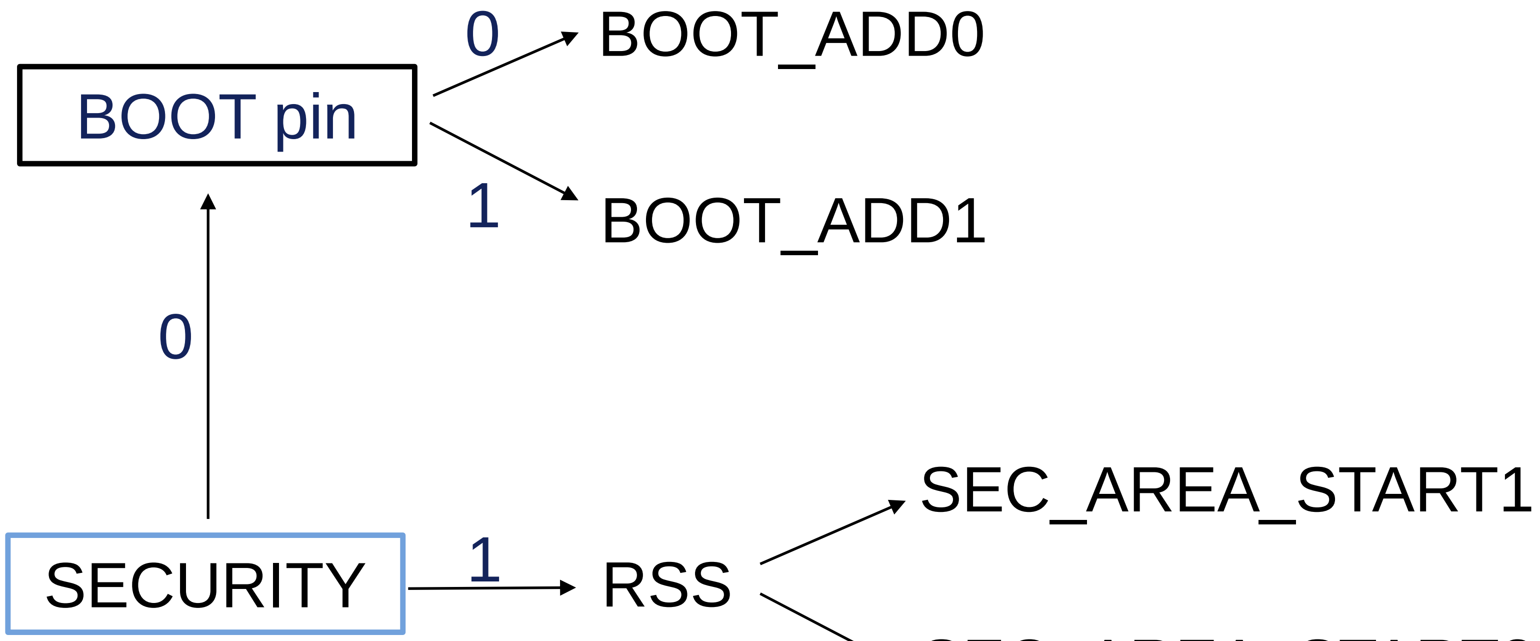
Boot configuration L5 (TZ activated)



Boot lock STM32L5

- Purpose: lock boot on user flash whatever boot configuration
- If BOOT_LOCK=1 device boots:
  - SECBOOTADD / option byte can't be modified
  - Device boots on SECBOOTADD
- BOOT_LOCK can change from 0 => 1 in RDP0 and RDP1
- BOOT_LOCK can change from 1 => 0 **only** in RDP0
- So, in RDP1, with BOOT_LOCK set, the only way to go back is
  - Make RDP1 => RDP0 regression : mass erase will be performed
  - Change BOOT_LOCK value once in RDP0

Boot configuration H7



STM32H7 boot in secure user memory

- Sum up: when security is activated and secure memory is define, STM32 will always boot on this secure memory.
- The only way to modify this is to do a RDP1-> RDP0 which imply a flash mass erase.
- Details description would be address in the secure user memory part

Security Features by STM32 Series

STM32 Series	Security Features																
	96-Bit Unique ID	FLASH WRP	FLASH PCROP	FLASH RDP	Unique entry point	Secure mem/ HDP	MPU	Firewall	Trustzone	OTFDEC	Tamper	TRNG	CRYPT AES	HASH	PKA	Cryptolib	Arm Cortex®
STM32 **F0**	🟩	🟩		🟩							🟩					🟩	**M0**
STM32 **F1**	🟩	🟩					🟩				🟩					🟩	**M3**
STM32 **F2**	🟩	🟩		🟩			🟩				🟩	🟩	▨	▨		🟩	**M3**
STM32 **F3**	🟩	🟩		🟩			🟩				🟩					🟩	**M4**
STM32 **F4**	🟩	🟩	▨	🟩			🟩				🟩	🟩	▨	▨		🟩	**M4**
STM32 **F7**	🟩	🟩	▨	🟩			🟩				🟩	🟩	▨	▨		🟩	**M7**
STM32 **L0**	🟩	🟩	🟩	🟩			🟩	🟩			🟩		▨			🟩	**M0+**
STM32 **L1**	🟩	🟩	🟩	🟩			🟩				🟩		▨			🟩	**M3**
STM32 **L4**	🟩	🟩	🟩	🟩			🟩	🟩			🟩	🟩	▨	▨		🟩	**M4**
STM32 **L5**	🟩	🟩		🟩	🟩	🟩	🟩		🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	**M33**
STM32 **H7**	🟩	🟩	🟩	🟩	▨	▨	🟩				🟩	🟩	▨	▨		🟩	**M7/M4**
STM32 **G0**	🟩	🟩	🟩	🟩	▨	▨	🟩				🟩	🟩	▨			🟩	**M0+**
STM32 **G4**	🟩	🟩	🟩	🟩	▨	▨	🟩				🟩	🟩	▨			🟩	**M4**
STM32 **WB**	🟩	🟩	🟩	🟩			🟩				🟩	🟩	🟩		🟩	🟩	**M4/M0+**

🟩 Available on all devices

▨ Depends on device part number

life.augmented

Hands-on

- Why no hands-on ?
  It’s closely link with other security.
  It should be done carefully there is risk to brick your board…

Secure user memory

Secure user memory

- Purpose : code, data and execution isolation of secure user firmware

- Principle : a user flash region executed at boot time, whatever the boot mode configuration. Once executed, the secure user part is closed and cannot be accessed anymore by any mean until the next boot.

- Configured thanks option bytes

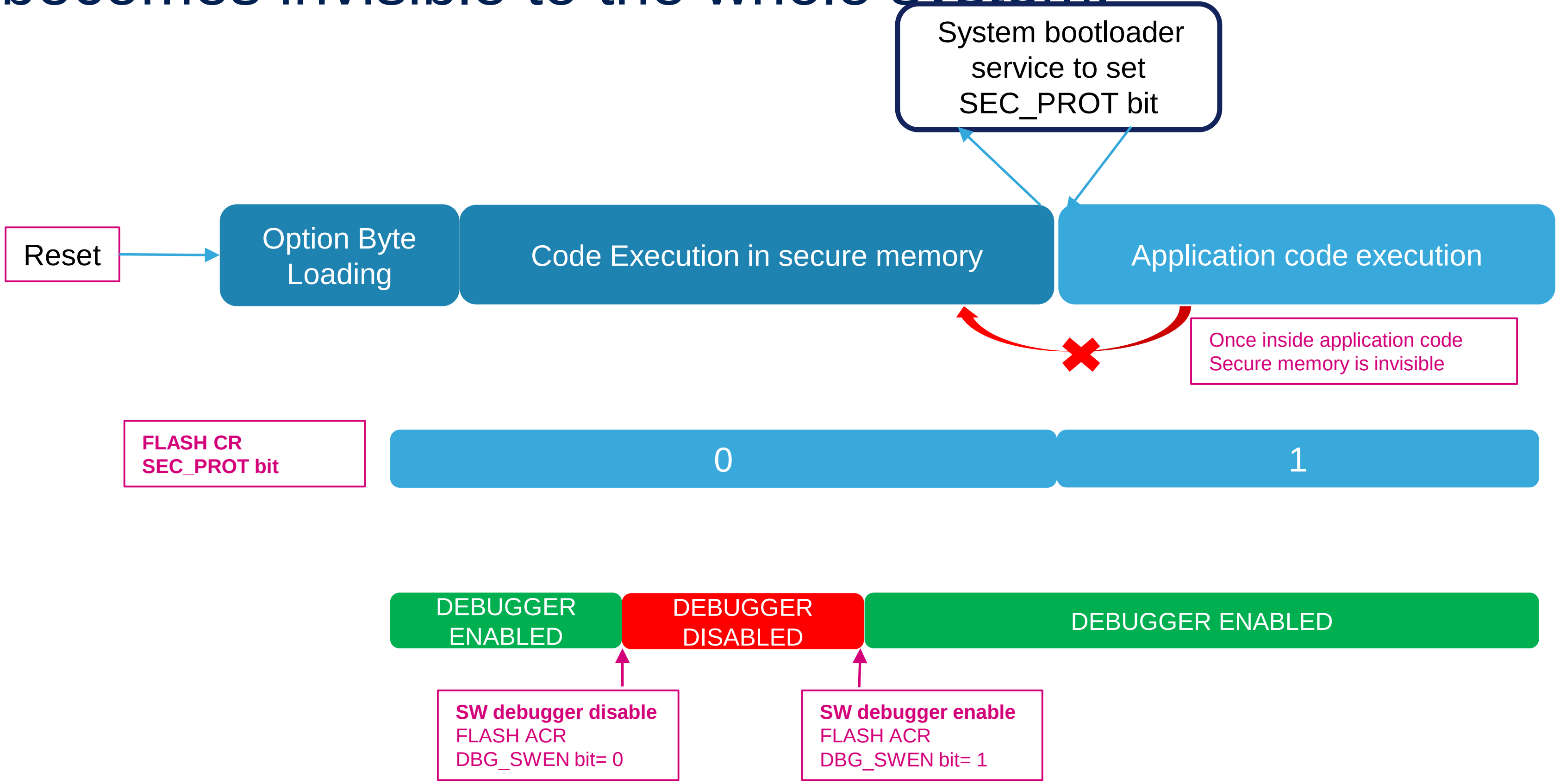
- Supported on
  - STM32G0/G4
  - STM32H7
  - STM32L5 (mechanism called HDP)

Secure memory on STM32G0/G4

- Secure memory area is located at beginning of the flash and executed after the reset
- Secure memory configuration : option byte SEC_SIZE (and SEC_SIZE2) could be only modify in RDP0
- It has a flash memory page granularity (2KB/4KB)
- This secure memory can be locked at application execution time,then secure memory is no more visible
- Debugger can be disabled when executing sensitive code
  - Useful in RDP1 when executing authentication, decryption

Secure memory on STM32G0/G4

- The secure memory lock could be done thanks to a service in system bootloader
  Reason is that as soon as the SEC_PROT bit is set to 1, the secure memory becomes invisible to the whole system.



System bootloader service to set SEC_PROT bit

Reset

Option Byte Loading

Code Execution in secure memory

Application code execution

Once inside application code Secure memory is invisible

FLASH CR SEC_PROT bit

0

1

DEBUGGER ENABLED

DEBUGGER DISABLED

DEBUGGER ENABLED

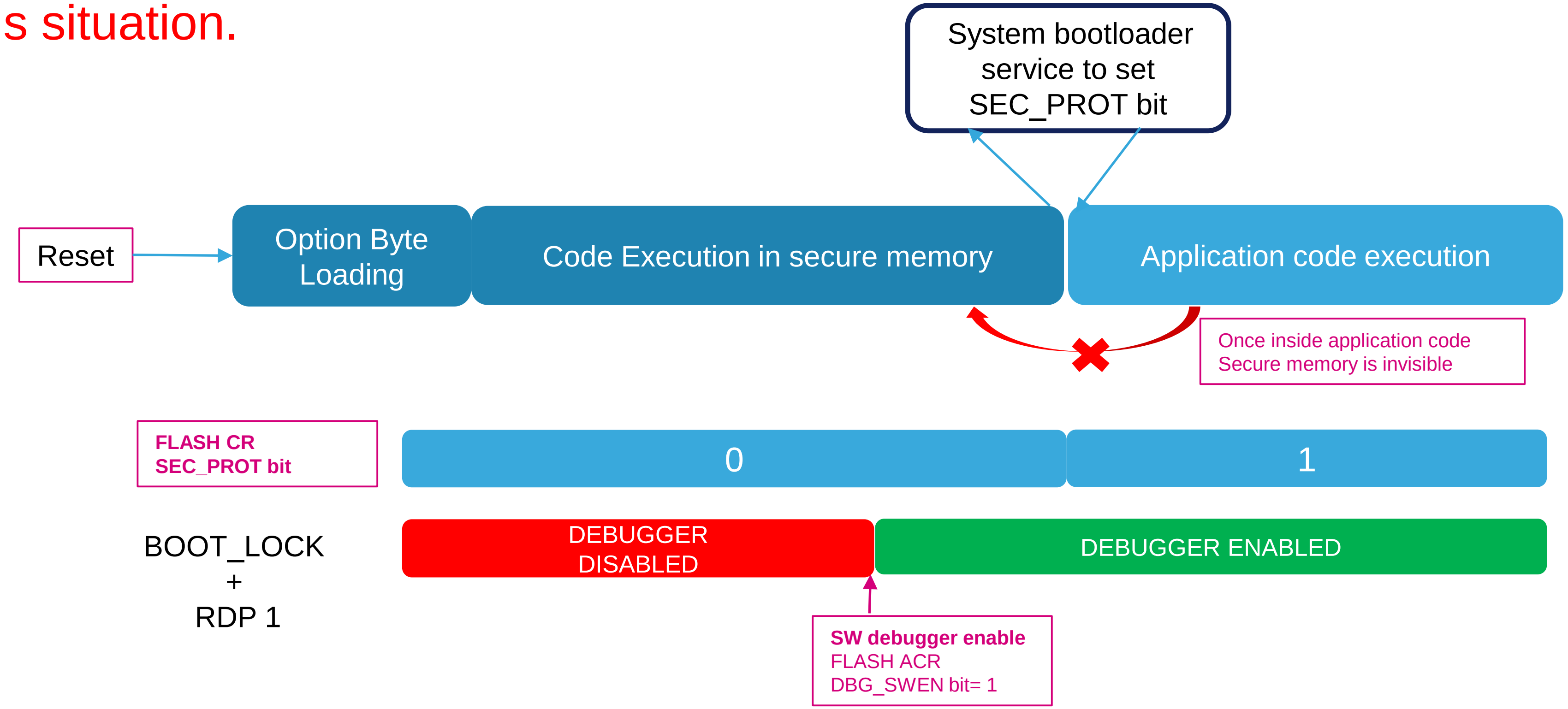
SW debugger disable
FLASH ACR
DBG_SWEN bit= 0

SW debugger enable
FLASH ACR
DBG_SWEN bit= 1

Secure memory on STM32G0

- CAUTION : If BOOT_LOCK is set in association with RDP Level 1, the debug capabilities of the device are stopped and the reset value of the DBG_SWEN bit of the FLASH_AC Register becomes zero.

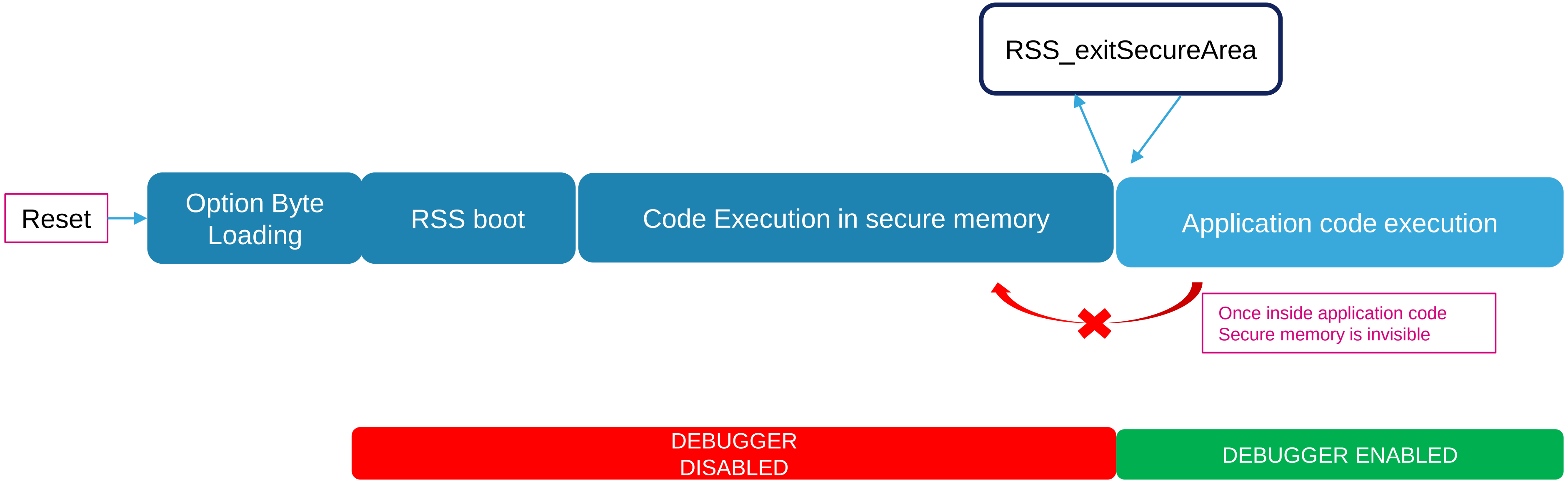
- If DBG_SWEN bit is not set by the application code after reset, there is no way to recover from this situation.


System bootloader service to set SEC_PROT bit
Reset
Option Byte Loading
Code Execution in secure memory
Application code execution
Once inside application code Secure memory is invisible
FLASH CR SEC_PROT bit
0
1
BOOT_LOCK + RDP 1
DEBUGGER DISABLED
DEBUGGER ENABLED
SW debugger enable FLASH ACR DBG_SWEN bit= 1

Secure memory on STM32H7

- Secure memory configuration could be done only in Secure Access mode (SECURITY option bit is set to 1)
- Secure memory area is executed after the reset whatever boot mode
- One user secure memory area per bank
- It has a flash memory page granularity (256 bytes)
- The secure memory lock is done thanks to a service in RSS
- Secure user memory can never be accessed by debugger (or Cortex M4)

Secure memory on STM32H7



Once inside application code
Secure memory is invisible

Secure memory on STM32H7

- In Secure access mode, to configure secure user memory you must :
  - call RSS_resetAndInitializeSecureAreas (Root Secure Service) which will set option byte :

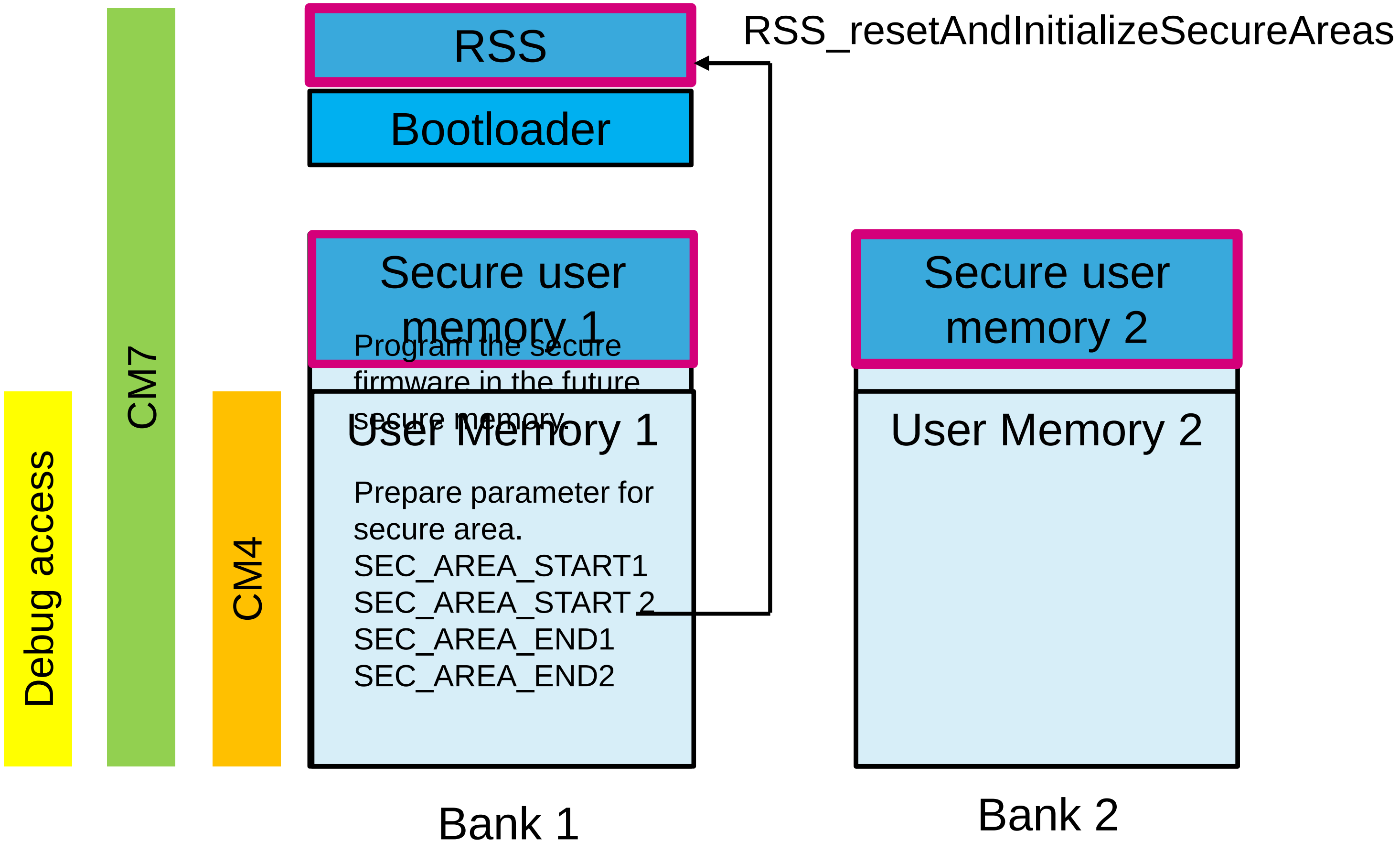
    SEC_AREA_START1/SEC_AREA_START 2

    SEC_AREA_END1/SEC_AREA_END2.

  Need some code execution to achieve configuration

- The secure user memory configuration could be removed on RDP level regression or flash bank erase.
  CAUTION : if you erase the secure memory area and keep this option bytes configured….No possibility to recover

Secure memory activation on STM32H7



Program the secure firmware in the future secure memory

Prepare parameter for secure area.
SEC_AREA_START1
SEC_AREA_START 2
SEC_AREA_END1
SEC_AREA_END2

▨ Secure Access mode only

Secure memory on STM32L5

- Secure hide protection (HDP) : once closed this part cannot be accessed anymore by any mean until the next boot.

- HDP is located at the start of the Flash watermark-based secure area.

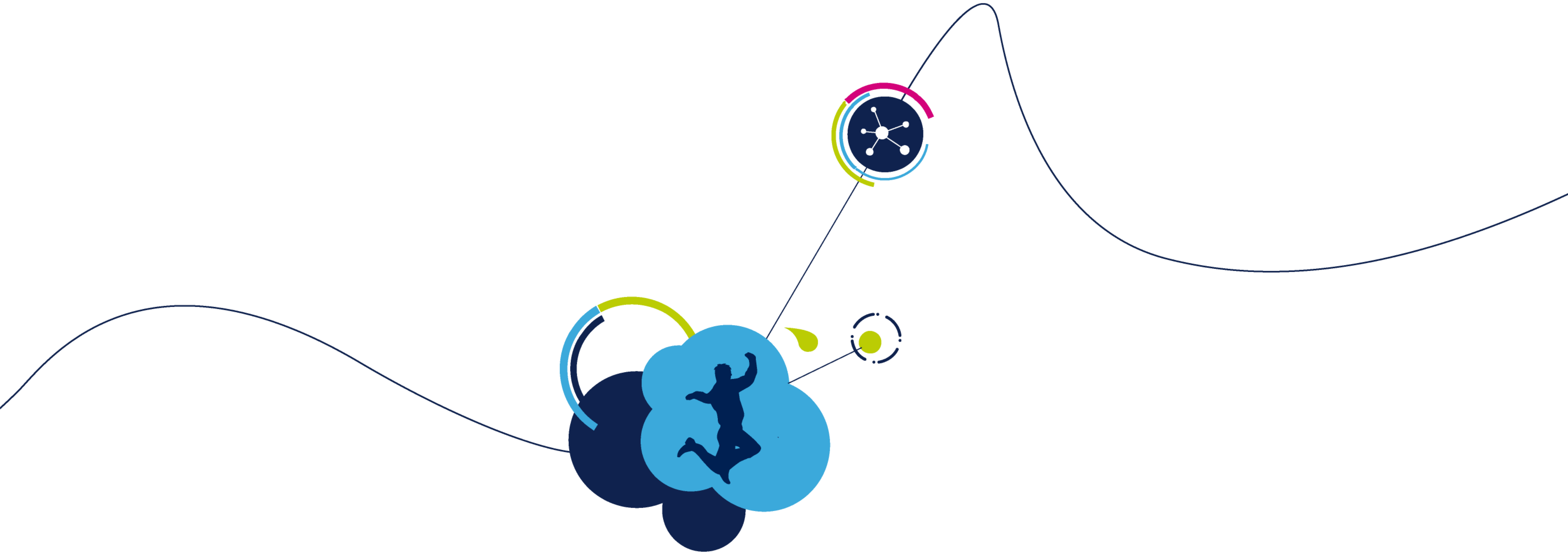
- Configured thanks 2 option bytes:
  - HDPxEN : HDP activation
  - HDP1_PEND / HDP2_PEND : end page offset regarding SECWM1_PSTRT/SECWM2_PSTRT

Security Features by STM32 Series

STM32 Series	Security Features															
	96-Bit Unique ID	FLASH WRP	FLASH PCROP	FLASH RDP	Unique entry point	Secure mem/ HDP	MPU	Firewall	Trustzone	OTFDEC	Tamper	TRNG	CRYPT AES	HASH	PKA	Cryptolib	Arm Cortex®
STM32 **F0**	🟩	🟩		🟩							🟩					🟩	**M0**
STM32 **F1**	🟩	🟩					🟩				🟩					🟩	**M3**
STM32 **F2**	🟩	🟩		🟩			🟩				🟩	🟩	▨	▨		🟩	**M3**
STM32 **F3**	🟩	🟩		🟩			🟩				🟩					🟩	**M4**
STM32 **F4**	🟩	🟩	▨	🟩			🟩				🟩	🟩	▨	▨		🟩	**M4**
STM32 **F7**	🟩	🟩	▨	🟩			🟩				🟩	🟩	▨	▨		🟩	**M7**
STM32 **L0**	🟩	🟩	🟩	🟩			🟩	🟩			🟩		▨			🟩	**M0+**
STM32 **L1**	🟩	🟩	🟩	🟩			🟩				🟩		▨			🟩	**M3**
STM32 **L4**	🟩	🟩	🟩	🟩			🟩	🟩			🟩	🟩	▨	▨		🟩	**M4**
STM32 **L5**	🟩	🟩		🟩	🟩	🟩	🟩		🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	**M33**
STM32 **H7**	🟩	🟩	🟩	🟩	▨	▨	🟩				🟩	🟩	▨	▨		🟩	**M7/M4**
STM32 **G0**	🟩	🟩	🟩	🟩	▨	▨	🟩				🟩	🟩	▨			🟩	**M0+**
STM32 **G4**	🟩	🟩	🟩	🟩	▨	▨	🟩				🟩	🟩	▨			🟩	**M4**
STM32 **WB**	🟩	🟩	🟩	🟩			🟩				🟩	🟩	🟩		🟩	🟩	**M4/M0+**

🟩 Available on all devices

▨ Depends on device part number



Secure memory Hands-on

life.augmented

Hands-on

- Purpose : test the secure memory mechanism without using the system bootloader service

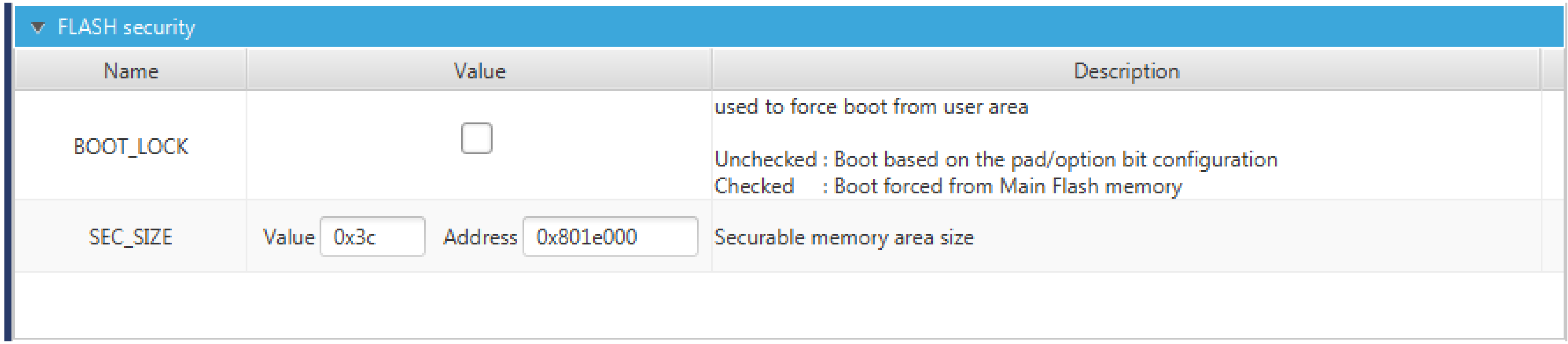
- Description : put a toggle led function in the secure memory area and call it regularly from the unsecure memory.
  On push button, lock the secure memory.

life.augmented

Hands-On

- Step 1 : create the code and put the blinking led in secure mem

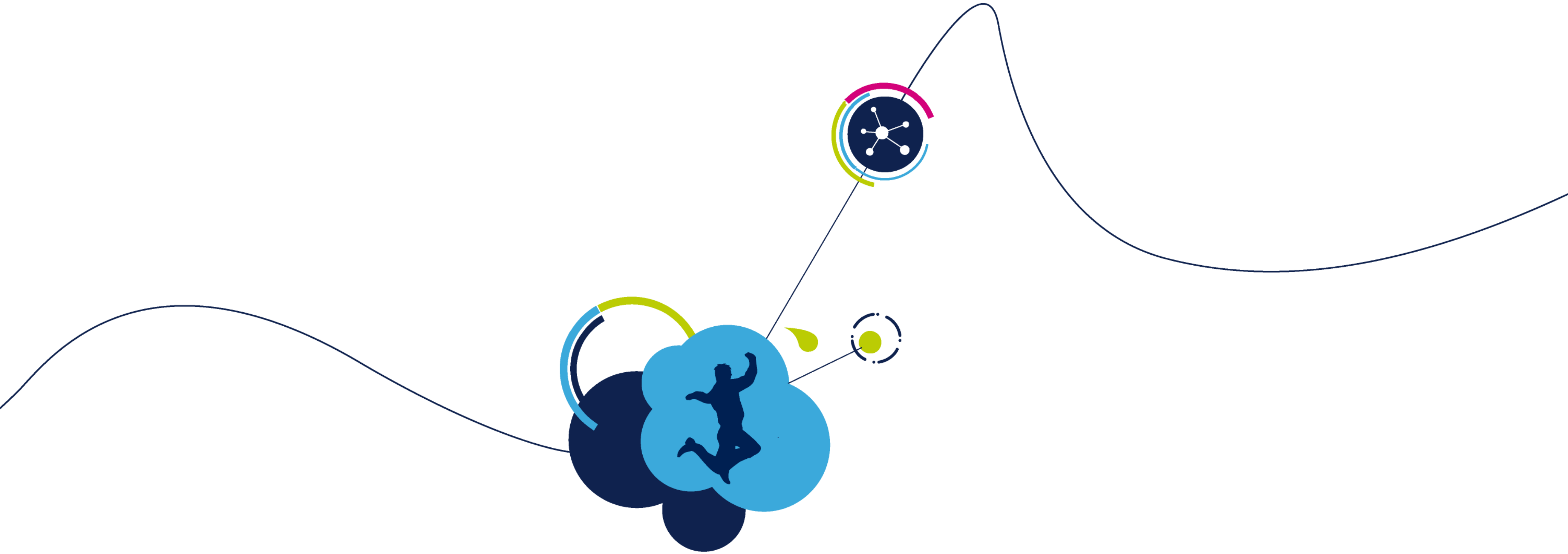
- Step 2 : configure the sec size



FLASH security

Name	Value	Description
BOOT_LOCK	□	used to force boot from user area Unchecked : Boot based on the pad/option bit configuration Checked : Boot forced from Main Flash memory
SEC_SIZE	Value 0x3c Address 0x801e000	Securable memory area size

- Step 3 : check the status after pushing the button



Secure memory Hands-on

Hands-on

- Purpose : test the secure memory mechanism with the bootloader service

- Description :
  Create a relocated standalone toggling led binary.
  Create a binary in the secure memory which will close the secure mem and launch the toggling led binary thank a system bootloader service

Hands-On

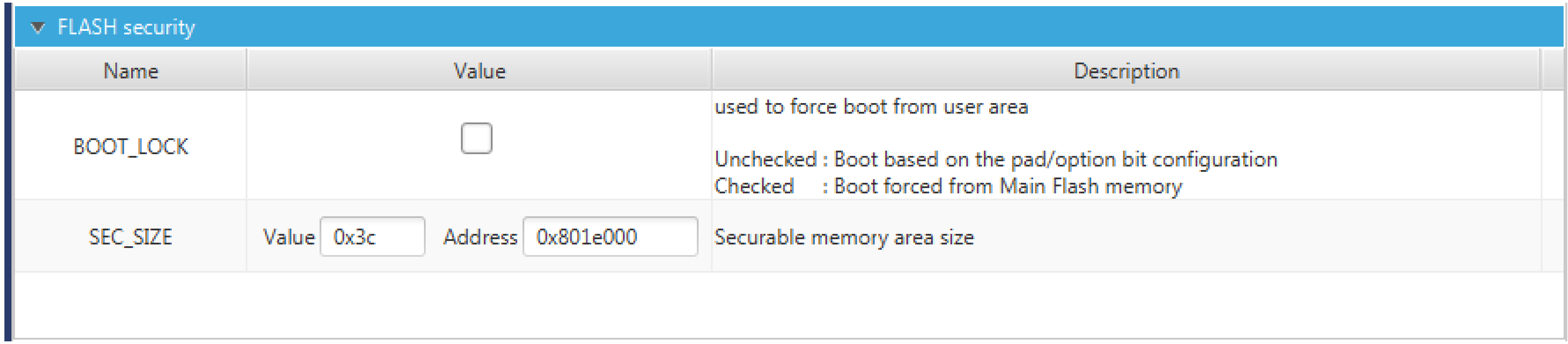
- Step 1 : have a binary with blinking led with vector table relocation at address 0x801E000
- Step 2 : modify the code to call RSS services

```
#define BL_EXIT_STICKY 0x1FFF6800
#define MAGIC_NUMBER   0x08192A3C
#define APPLICATION_ADDRESS 0x801E000

//R0: BL_EXIT_STICKY vector table address
//R1: magic number
//R2: application address
typedef  void (*pFunction)(uint32_t a, uint32_t b,
uint32_t c);

pFunction JumpToApplication;
uint32_t JumpAddress;
```

```
 /* Jump to user application */
      JumpAddress = *(__IO uint32_t*)
(BL_EXIT_STICKY + 4);
      JumpToApplication = (pFunction)
JumpAddress;

JumpToApplication(JumpAddress,MAGIC_NUMBER,
APPLICATION_ADDRESS);
```

life.augmented

- Step 3 : configure the sec size



FLASH security

Name	Value	Description
BOOT_LOCK	□	used to force boot from user area Unchecked : Boot based on the pad/option bit configuration Checked : Boot forced from Main Flash memory
SEC_SIZE	Value 0x3c Address 0x801e000	Securable memory area size

- Step 3 : check the status after pushing the button

Memory Protection Unit (MPU)

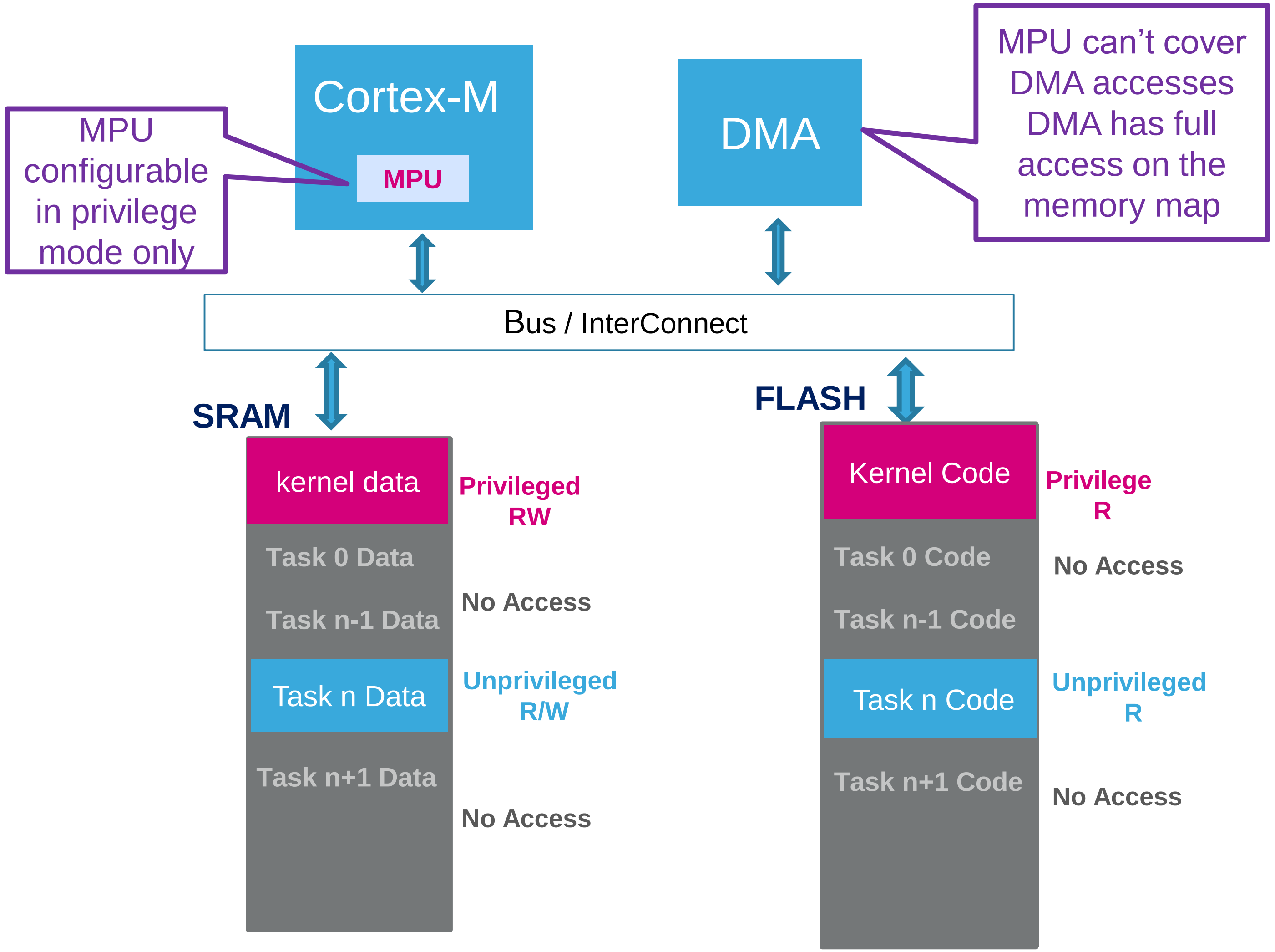
Memory Protection Unit

- 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.
- MPU attributes are only set for CPU access.
- 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

Memory Protection Unit

To control and restrict which subsystem can access what memory region and how (Read, Write, eXecute)

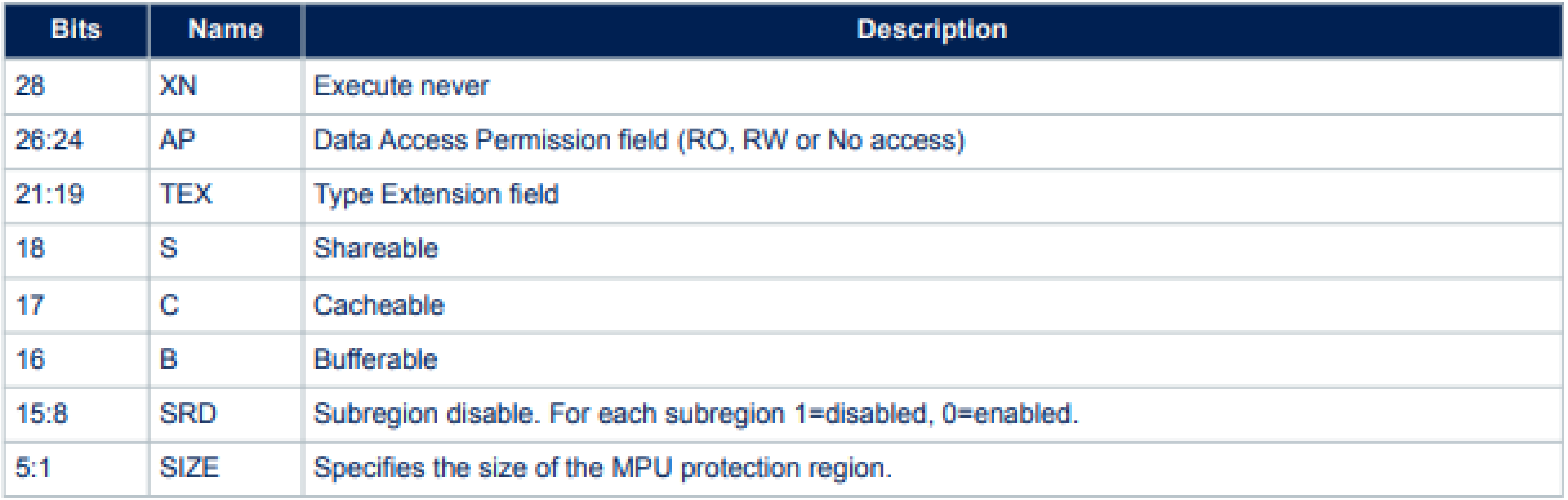
- Preventing user applications from corrupting data used by the operating system
- Separating data between processing tasks by blocking tasks from accessing others' data
- Detecting unexpected memory accesses (for example, stack corruption)
- In addition, the MPU can also be used to define memory access characteristics such as caching and buffering behaviors for different regions.
- up to 16 regions
- Illegal access → exception fault.



life.augmented

Memory Protection Unit

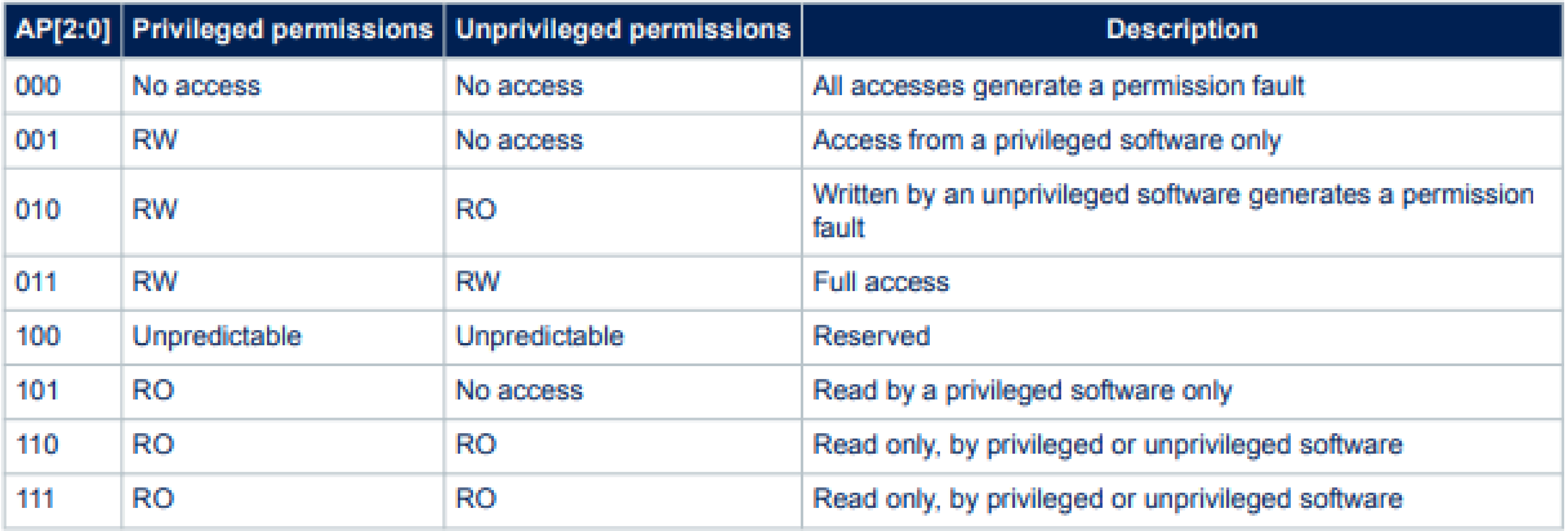
- Memory attribute :



Bits	Name	Description
28	XN	Execute never
26:24	AP	Data Access Permission field (RO, RW or No access)
21:19	TEX	Type Extension field
18	S	Shareable
17	C	Cacheable
16	B	Bufferable
15:8	SRD	Subregion disable. For each subregion 1=disabled, 0=enabled.
5:1	SIZE	Specifies the size of the MPU protection region.

Memory protection Unit a bit more …

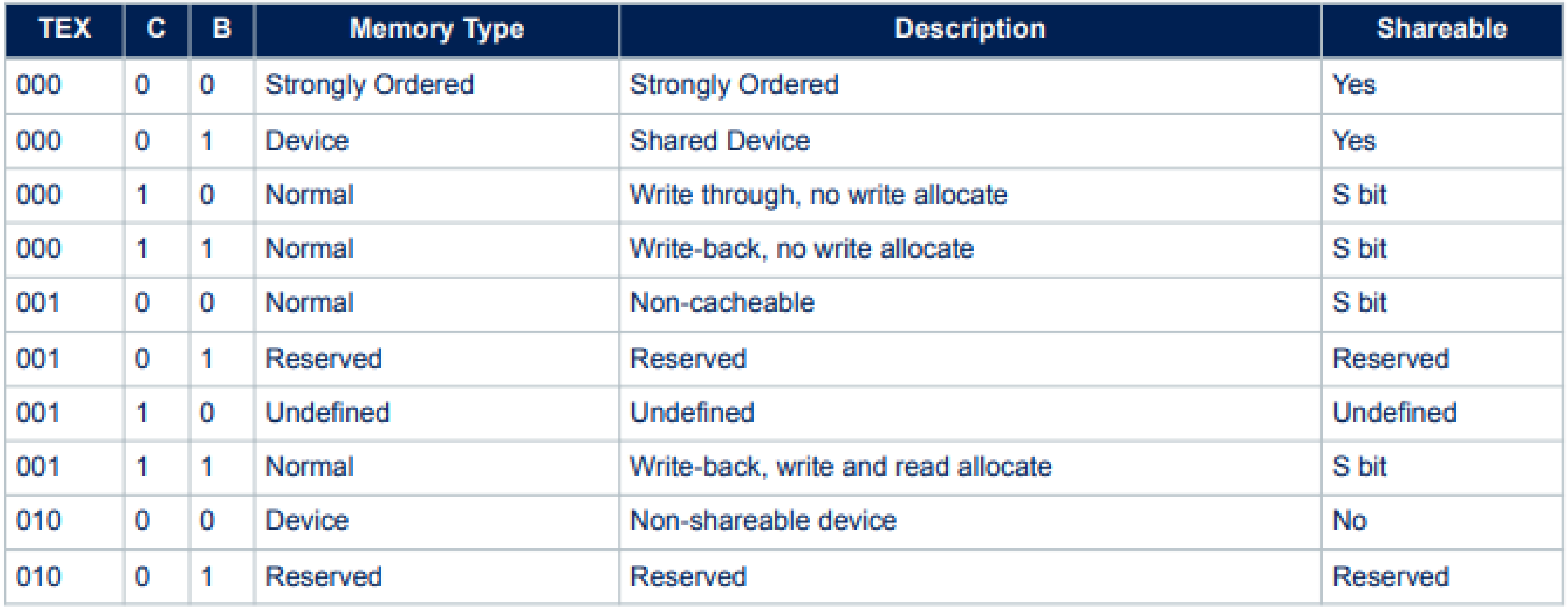
- Memory access permissions



AP[2:0]	Privileged permissions	Unprivileged permissions	Description
000	No access	No access	All accesses generate a permission fault
001	RW	No access	Access from a privileged software only
010	RW	RO	Written by an unprivileged software generates a permission fault
011	RW	RW	Full access
100	Unpredictable	Unpredictable	Reserved
101	RO	No access	Read by a privileged software only
110	RO	RO	Read only, by privileged or unprivileged software
111	RO	RO	Read only, by privileged or unprivileged software

Memory Protection Unit

- Cache properties and shareability:



TEX	C	B	Memory Type	Description	Shareable
000	0	0	Strongly Ordered	Strongly Ordered	Yes
000	0	1	Device	Shared Device	Yes
000	1	0	Normal	Write through, no write allocate	S bit
000	1	1	Normal	Write-back, no write allocate	S bit
001	0	0	Normal	Non-cacheable	S bit
001	0	1	Reserved	Reserved	Reserved
001	1	0	Undefined	Undefined	Undefined
001	1	1	Normal	Write-back, write and read allocate	S bit
010	0	0	Device	Non-shareable device	No
010	0	1	Reserved	Reserved	Reserved

Memory Protection Unit

- **Write through with no write allocate**: on hits it writes to the cache and the main memory, on misses it updates the block in the main memory not bringing that block to the cache.

- **Write-back with no write allocate**: on hits it writes to the cache setting dirty bit for the block, the main memory is not updated. On misses it updates the block in the main memory not bringing that block to the cache.

- **Write-back with write and read allocate**: on hits it writes to the cache setting dirty bit for the block, the main memory is not updated. On misses it updates the block in the main memory and brings the block to the cache.

Memory Protection Unit Tips

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

- Cortex-M7 speculative prefetch : possible impact on the memories or devices which are sensitive to multiple accesses.
  In order to protect normal memories from a speculative prefetch it is recommended to change memory attributes from normal to a strongly ordered or to device memory thanks to the MPU.

Security Features by STM32 Series

STM32 Series	Security Features															
	96-Bit Unique ID	FLASH WRP	FLASH PCROP	FLASH RDP	Unique entry point	Secure mem/ HDP	MPU	Firewall	Trustzone	OTFDEC	Tamper	TRNG	CRYPT AES	HASH	PKA	Cryptolib	Arm Cortex®
STM32 **F0**	🟩	🟩		🟩							🟩					🟩	**M0**
STM32 **F1**	🟩	🟩					🟩				🟩					🟩	**M3**
STM32 **F2**	🟩	🟩		🟩			🟩				🟩	🟩	▨	▨		🟩	**M3**
STM32 **F3**	🟩	🟩		🟩			🟩				🟩					🟩	**M4**
STM32 **F4**	🟩	🟩	▨	🟩			🟩				🟩	🟩	▨	▨		🟩	**M4**
STM32 **F7**	🟩	🟩	▨	🟩			🟩				🟩	🟩	▨	▨		🟩	**M7**
STM32 **L0**	🟩	🟩	🟩	🟩			🟩	🟩			🟩		▨			🟩	**M0+**
STM32 **L1**	🟩	🟩	🟩	🟩			🟩				🟩		▨			🟩	**M3**
STM32 **L4**	🟩	🟩	🟩	🟩			🟩	🟩			🟩	🟩	▨	▨		🟩	**M4**
STM32 **L5**	🟩	🟩		🟩	🟩	🟩	🟩		🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	**M33**
STM32 **H7**	🟩	🟩	🟩	🟩	▨	▨	🟩				🟩	🟩	▨	▨		🟩	**M7/M4**
STM32 **G0**	🟩	🟩	🟩	🟩	▨	▨	🟩				🟩	🟩	▨			🟩	**M0+**
STM32 **G4**	🟩	🟩	🟩	🟩	▨	▨	🟩				🟩	🟩	▨			🟩	**M4**
STM32 **WB**	🟩	🟩	🟩	🟩			🟩				🟩	🟩	🟩		🟩	🟩	**M4/M0+**

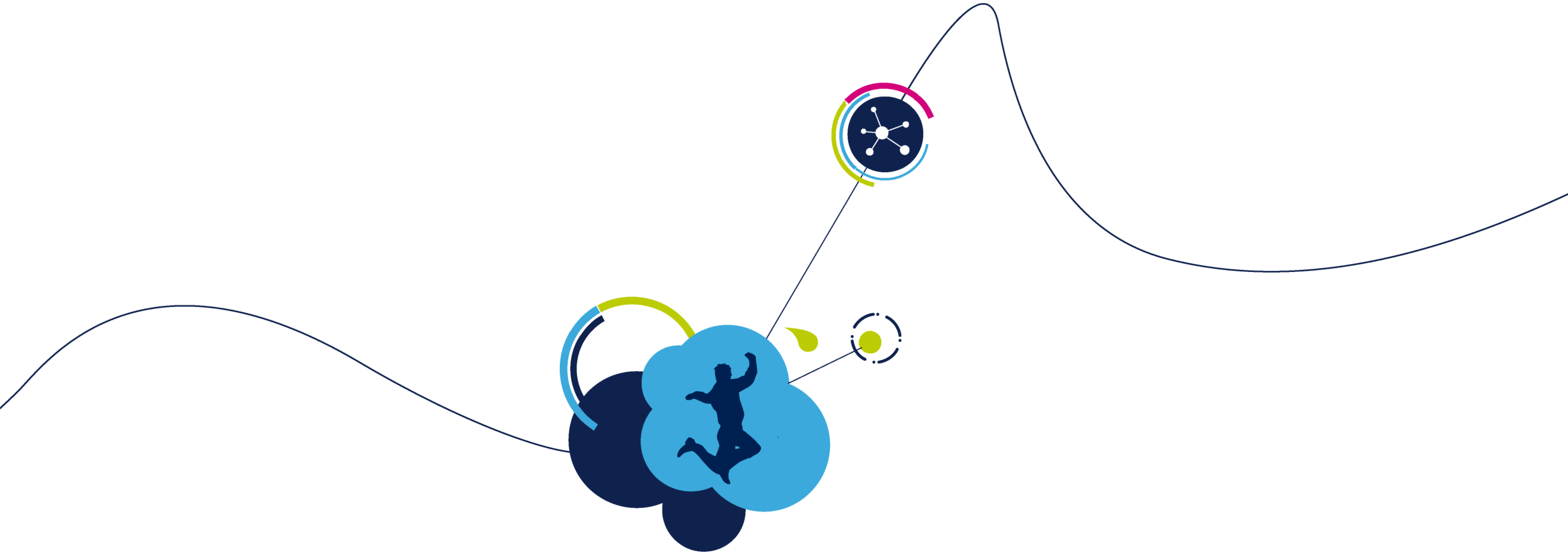
🟩 Available on all devices

▨ Depends on device part number

life.augmented

Reference

- AN4838 : Managing memory protection unit (MPU) in STM32 MCUs
- PM0253 : STM32F7 Series and STM32H7 Series Cortex®-M7 processor programming manual
- PM0056 : STM32F10xxx/20xxx/21xxx/L1xxxx Cortex®-M3 programming manual
- PM0223 : STM32L0 Series and STM32G0 Series Cortex®-M0+ programming manual
- PM0214 : STM32 Cortex®-M4 programming manual



MPU Hands-on

Hands-on

- Purpose : protect a data array thanks MPU then access this data with DMA.

- Step 1 : start from HAL example: STM32Cube_FW_G0_V1.3.0\Projects\NUCLEO-G071RB\Examples\CORTEX\CORTEXM_MPU

- Step 2 : add a dma memory to memory to get acces to protected data.

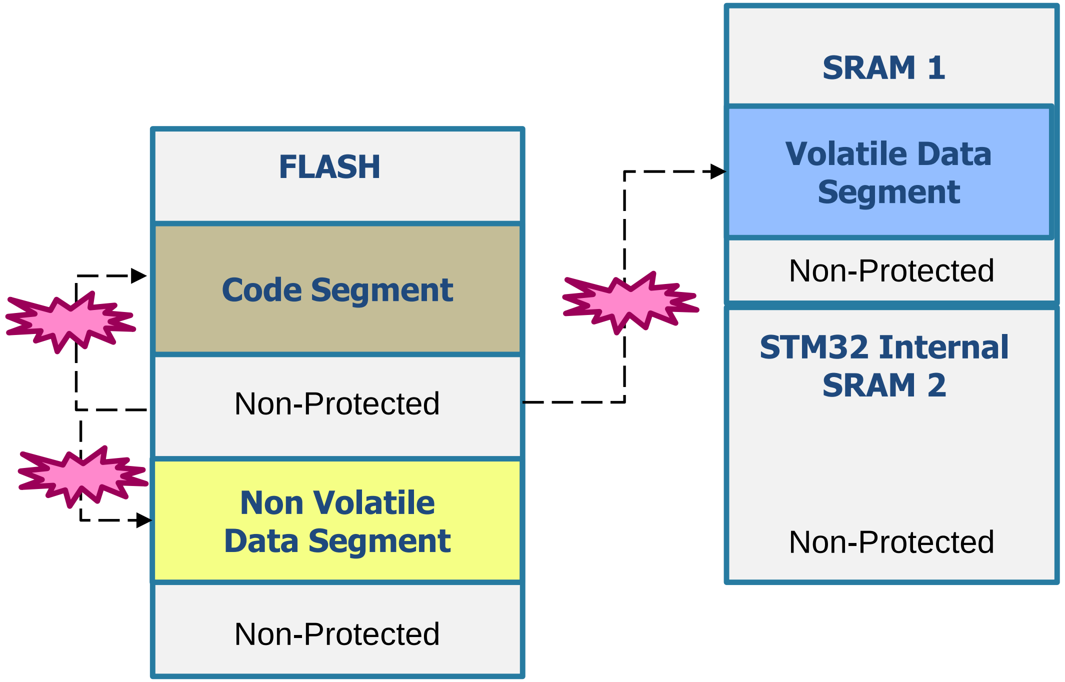
FIREWALL

life.augmented

Firewall

- Firewall : it’s is an hardware IP, which allow to create a security enclave (Flash/RAM) with a unique entry point
- This protection is dynamically managed at runtime.

Security Feature : Firewall



FLASH

Code Segment

Non-Protected

Non Volatile Data Segment

Non-Protected

SRAM 1

Volatile Data Segment

Non-Protected

STM32 Internal SRAM 2

Non-Protected

- The Firewall monitors accesses to trusted areas, referred to as segments:
  - 🟫 **Code Segment** (FLASH)
    - **Instruction fetches** or **data read accesses** can only occur when Firewall is open
  - 🟨 **Non Volatile Data Segment** (FLASH)
    - Usually contains **sensitive constants** (e.g. cryptographic keys)
  - 🟦 **Volatile Data Segment** (SRAM 1)
    - Contains potentially variable data used by the protected code.
    - **Data can only be accessed during the protected execution state**

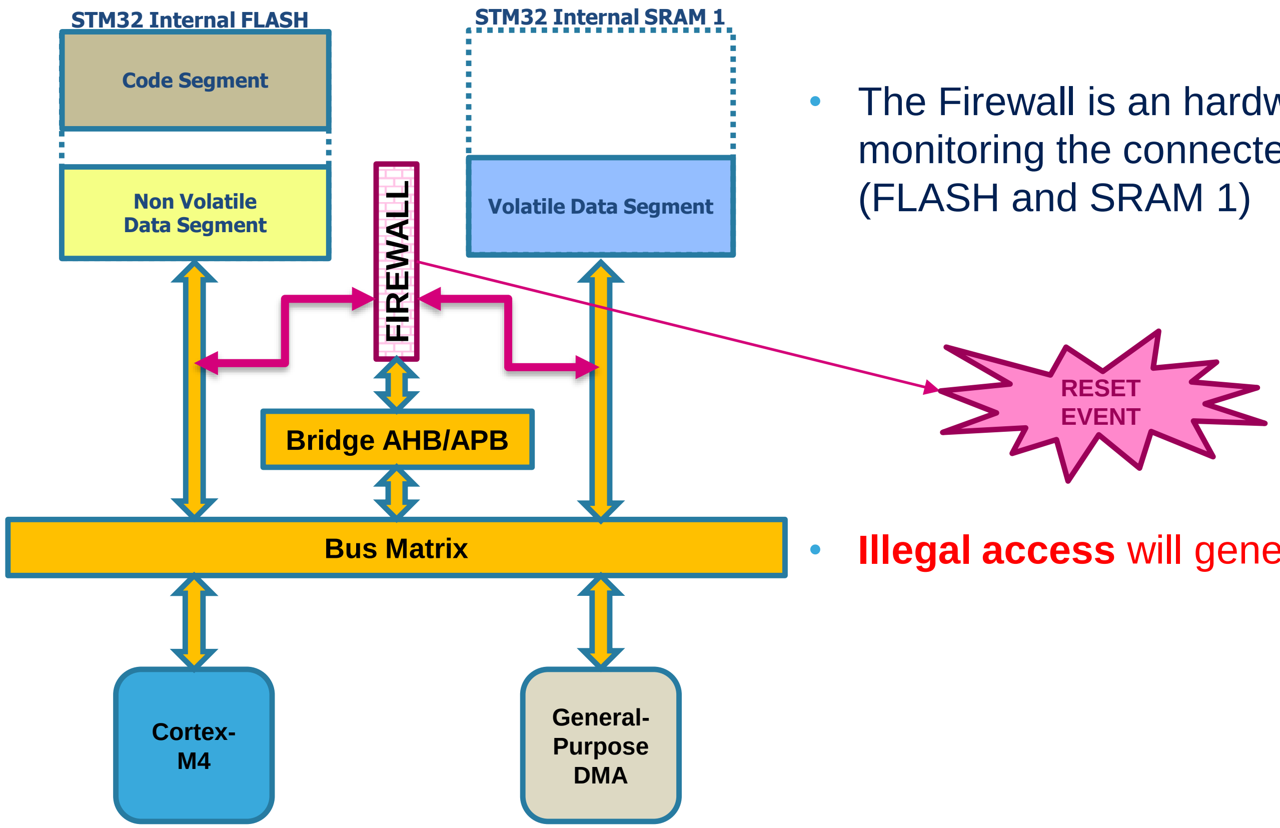
RESET EVENT

Illegal accesses based on the Segment properties and current Firewall State will cause a **RESET**

Firewall configuration

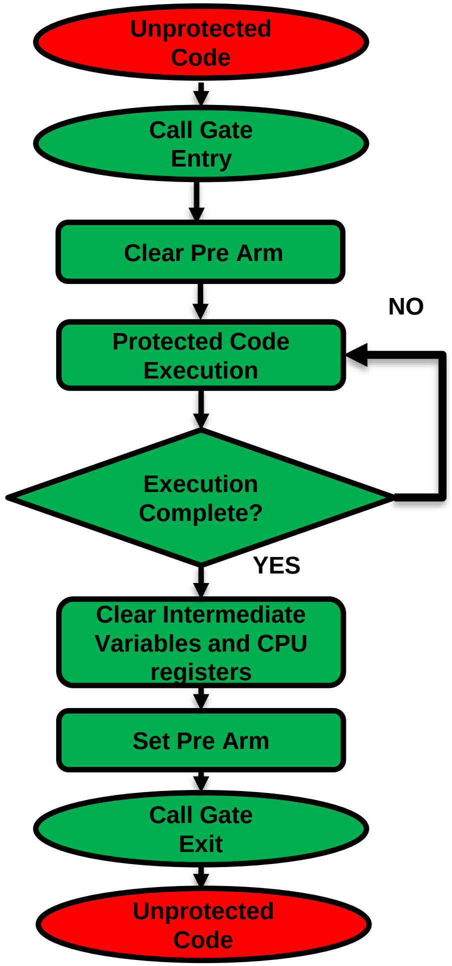
- Code segment /Non Volatile data segment
  - 256-byte granularity
  - size of the segment expressed in bytes but is a multiple of 256 bytes
- Volatile data segment
  - 64-byte granularity
  - size of the segment expressed in bytes but is a multiple of 64 bytes
  - Additional attribute : shared/not shared executable/not executable

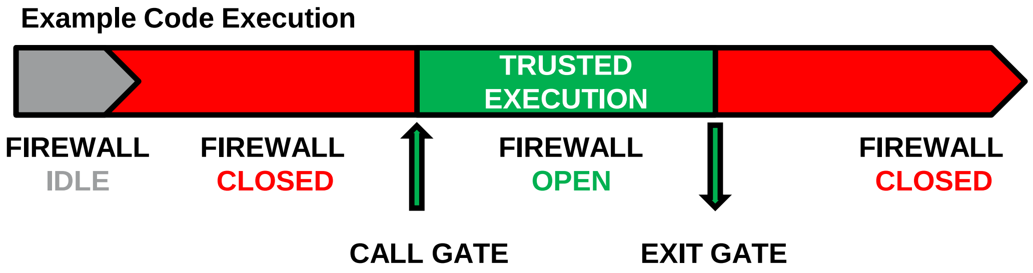
Firewall Architecture

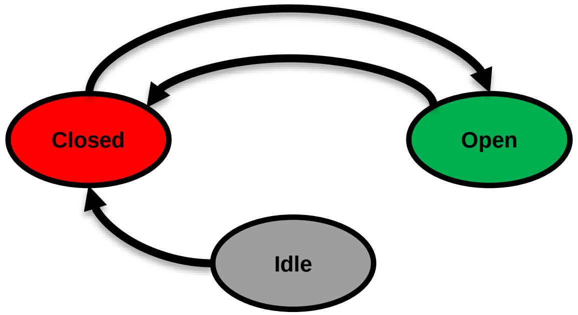


- The Firewall is an hardware peripheral monitoring the connected memories (FLASH and SRAM 1)
- **Illegal access** will generate a **RESET**

Firewall : Run Time Protection







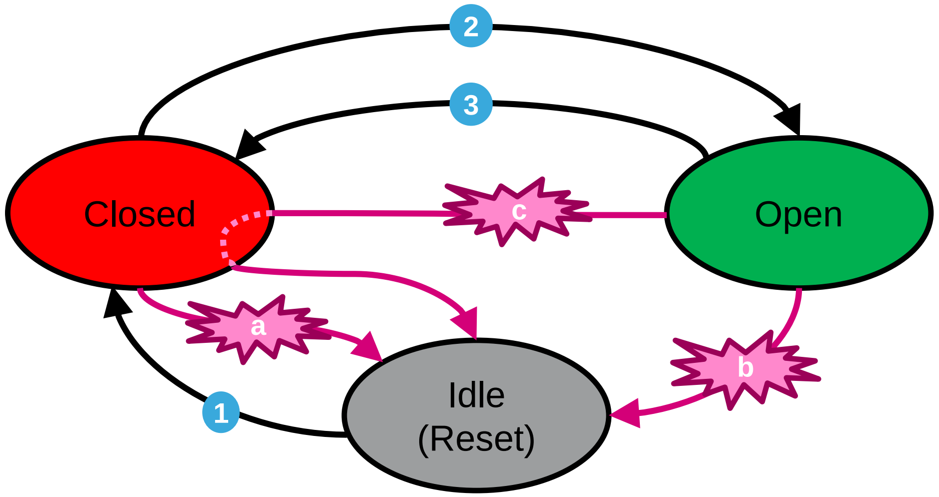
Firewall PreArm bit

- FPA: Firewall prearm

  0 : Any code executed outside the protected segment when the Firewall is opened will generate a system reset.

  1 : any code executed outside the protected segment will close the Firewall.

Firewall : States



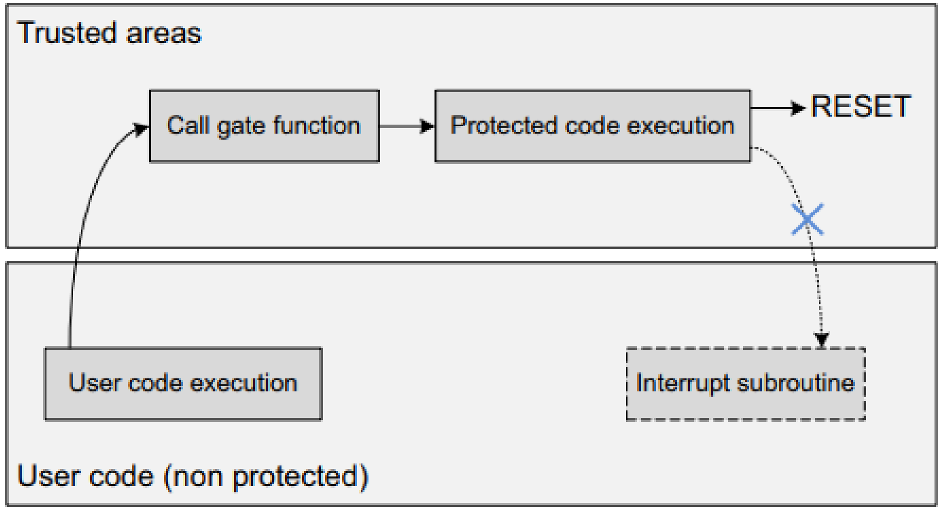
- Correct transactions
  - 1 Configuration and Enable
    - Not reconfigurable at run time
  - 2 Enter the Call Gate
    - Following proper sequence for entering
  - 3 Leave the protected area
    - Following proper sequence for exiting

- Exceptions (result in **RESET**):
  - a Illegal Access to Protected Segments
  - Instruction Fetch outside Protected Segment :
    - b Reset immediately
    - c Firewall is Closed and after code execution system is reset (e.g. return from interrupt …)

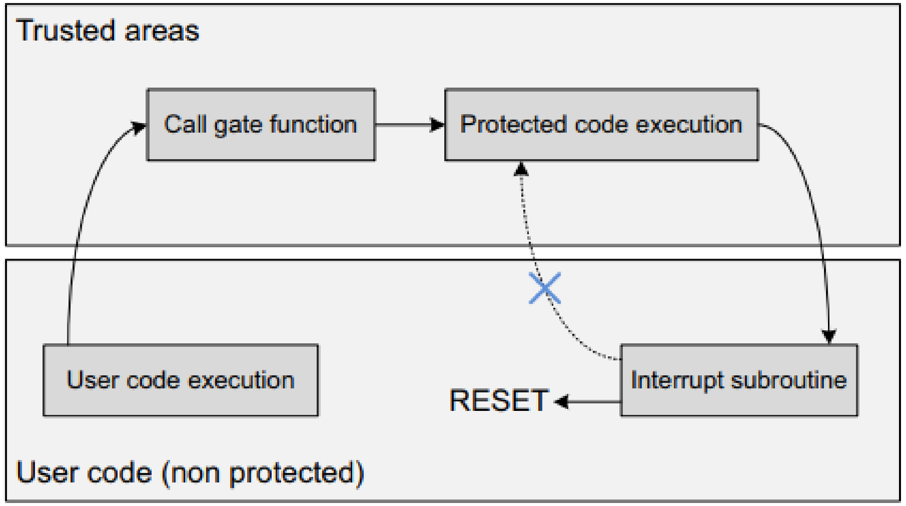
Firewall and interrupt

- When the FIREWALL is OPEN, no interrupt must take place during the execution of the protected code.

FIREWALL is OPEN with FPA = 0



FIREWALL is OPEN with FPA = 1



Firewall : Access Properties

PROTECTED SEGMENTS		Firewall State		
		Closed	Open	Idle (deactivated)
Code Segment (FLASH)		**R/W/X illegal**, except the call gate function execution	**R/X allowed** **W illegal**	**R/W/X Allowed**
Non Volatile Data Segment (FLASH)		**R/W/X illegal**	**R/W allowed** **X illegal**	
Volatile Data Segment (SRAM1)	**Not** Shared **Not** Executable			
	Not Shared Executable (X)	**R/W/X illegal**, except the call gate function execution	**R/W/X allowed**	
	Shared **Not** Executable	**R/W/X allowed**	**R/W/X allowed**	
	Shared Executable (X)			

Access:

R: **R**ead
W: **W**rite
X: E**X**ecute

Firewall tips

- The code protected by the Firewall must not be interruptible. It is up to the user code to disable any interrupt source before executing the code protected by the Firewall.
- To open the Firewall, the code currently executed must jump to the 2nd word of the “call gate” and execute the code from this point.

Security Features by STM32 Series

STM32 Series	Security Features																
	96-Bit Unique ID	FLASH WRP	FLASH PCROP	FLASH RDP	Unique entry point	Secure mem/ HDP	MPU	Firewall	Trustzone	OTFDEC	Tamper	TRNG	CRYPT AES	HASH	PKA	Cryptolib	Arm Cortex®
STM32 **F0**	🟩	🟩		🟩							🟩					🟩	**M0**
STM32 **F1**	🟩	🟩					🟩				🟩					🟩	**M3**
STM32 **F2**	🟩	🟩		🟩			🟩				🟩	🟩	▨	▨		🟩	**M3**
STM32 **F3**	🟩	🟩		🟩			🟩				🟩					🟩	**M4**
STM32 **F4**	🟩	🟩	▨	🟩			🟩				🟩	🟩	▨	▨		🟩	**M4**
STM32 **F7**	🟩	🟩	▨	🟩			🟩				🟩	🟩	▨	▨		🟩	**M7**
STM32 **L0**	🟩	🟩	🟩	🟩			🟩	🟩			🟩		▨			🟩	**M0+**
STM32 **L1**	🟩	🟩	🟩	🟩			🟩				🟩		▨			🟩	**M3**
STM32 **L4**	🟩	🟩	🟩	🟩			🟩	🟩			🟩	🟩	▨	▨		🟩	**M4**
STM32 **L5**	🟩	🟩		🟩	🟩	🟩	🟩		🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	**M33**
STM32 **H7**	🟩	🟩	🟩	🟩	▨	▨	🟩				🟩	🟩	▨	▨		🟩	**M7/M4**
STM32 **G0**	🟩	🟩	🟩	🟩	▨	▨	🟩				🟩	🟩	▨			🟩	**M0+**
STM32 **G4**	🟩	🟩	🟩	🟩	▨	▨	🟩				🟩	🟩	▨			🟩	**M4**
STM32 **WB**	🟩	🟩	🟩	🟩			🟩				🟩	🟩	🟩		🟩	🟩	**M4/M0+**

🟩 Available on all devices

▨ Depends on device part number

life.augmented

Reference

- AN4730 : Using the FIREWALL embedded in STM32L0/L4/L4+ Series MCUs for secure access to sensitive parts of code and data

Firewall Hands-on

Hands-on

- Purpose : isolate a led blinking function behind a firewall
- Step 1 : create a basic project with STM32CubeIDE and integrate the firewall HAL
- Step 2 : modify the linker script to define our security enclave

```
 FLASH (rx)                  : ORIGIN = 0x8000000,        LENGTH = 1008K
SecureFlash  (rx)            : ORIGIN = 0x80fc004,        LENGTH = 16K
SramDataProtected (xrw)      : ORIGIN = 0x2000FE00,       LENGTH = 512

 .mysection :
{    . = ALIGN(4);
     *(.mysection*)
     . = ALIGN(4);
  } > SecureFlash

.memory_vdata_protected_data :
  {
    *(.protected_data)
    *(.protected_data*)
} > SramDataProtected
```

Hands-on

- Step 3 :configure the firewall

```
/* No protected code segment (length set to 0) */
    fw_init.CodeSegmentStartAddress      = 0x80fc000;
    fw_init.CodeSegmentLength            = 0x2000;
    /* No protected non-volatile data segment */
    fw_init.NonVDataSegmentStartAddress = 0x80fe000;
    fw_init.NonVDataSegmentLength       = 0x200;
    /* Protected volatile data segment (in SRAM1 memory) start address and length */
    fw_init.VDataSegmentStartAddress    = 0x2000FE00;
    fw_init.VDataSegmentLength          = 512;
```

- Step 4 : create the secure code and call it thanks callgate.

```
void __attribute__((__section__(".mysection")))  protected_fonction()
{
                __HAL_FIREWALL_PREARM_DISABLE();
                                ToggleLed();
                __HAL_FIREWALL_PREARM_ENABLE();
}
```

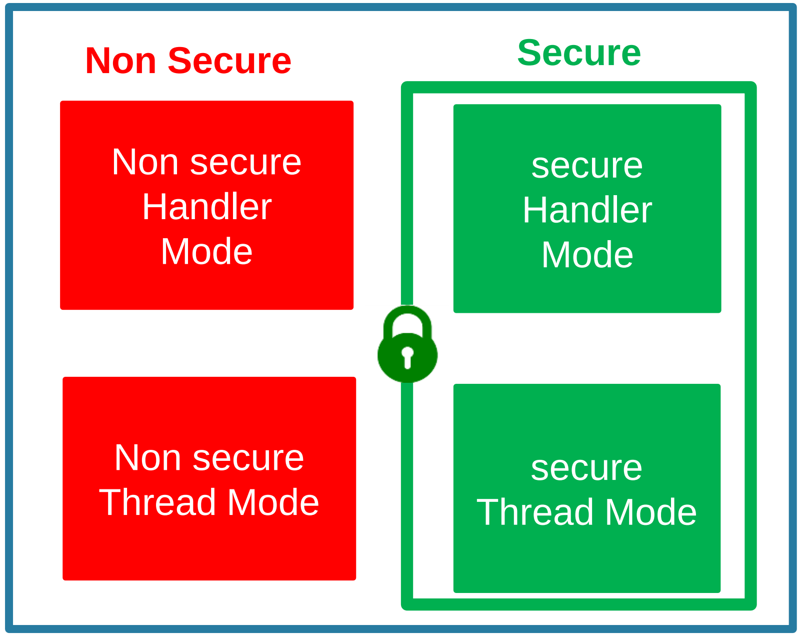
TrustZone

TrustZone Concept

- TrustZone feature is optional on ARM-V8M core.
- Code running in Secure state can access both secure and non secure information
- When enabled, the system by default starts up in secure state. When not enabled, the system is always in non secure state
- The division of secure and non secure worlds is memory map and the transitions takes place automatically without the need of secure monitor exception handler, thus optimizing switching overhead.

Additional state in ARMv8-M

- Addition of an extra processor state → secure / non secure
- System starts in secure state
- Split of secure and non-secure memory map done in SAU (Secure attribution unit)



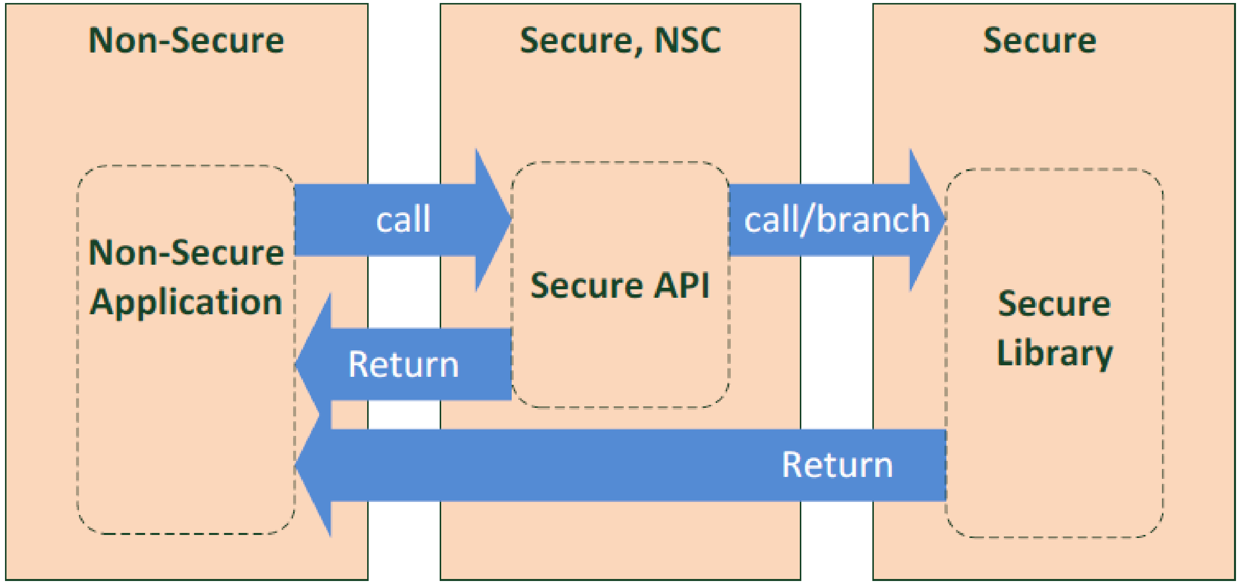
ARMv8-M

Features

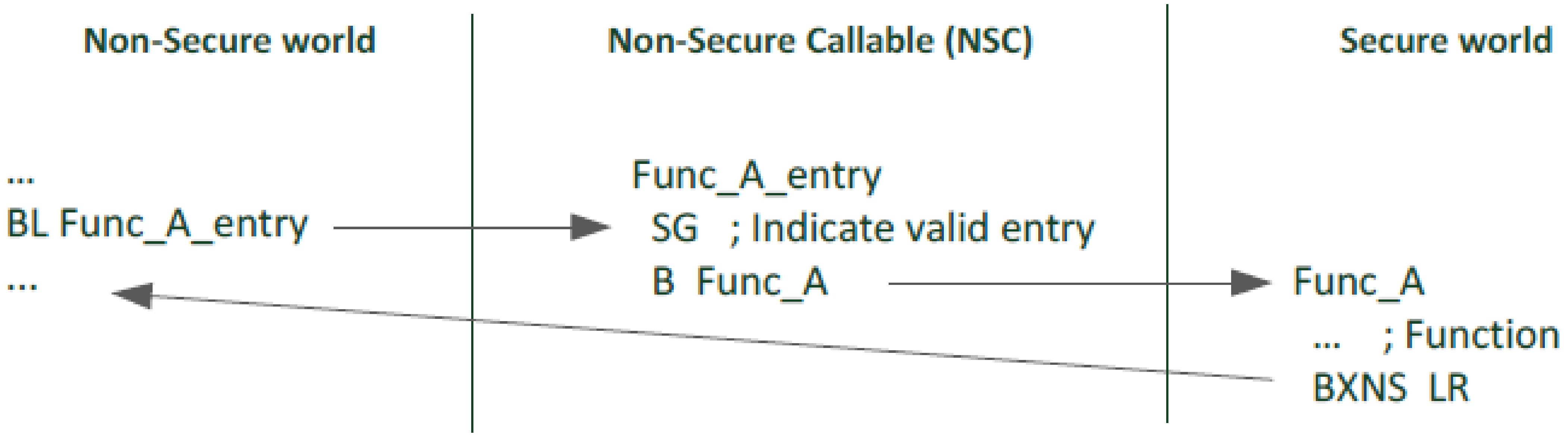
- Four stacks and four stack pointer registers
- Hardware stack-limit checking
- The memory space is partitioned into secure and non-secure spaces using programmable MPU-like Security Attribution Unit (SAU), or fixed/external security configuration
- Exception handling hardware automatically saves and then clears the secure register states when switching to the non-secure exception state
- Non-secure entry to secure code restricted to secure code locations containing a Secure Gateway (SG) instruction and tagged as a non-secure callable (NSC) region.

Software flow between memory types

- 3 different attribute in SAU (security attribution unit) and IDAU (implementation defined attribution unit)
  - Secure
  - Non-Secure Callable (NSC)
  - Non-Secure



Non-secure program calling a secure function



Non-Secure world

```
...
BL Func_A_entry
...
```

Non-Secure Callable (NSC)

```
Func_A_entry
  SG   ; Indicate valid entry
  B  Func_A
```

Secure world

```
Func_A
 ...    ; Function
 BXNS  LR
```

Non-secure program calling a secure function

- IDAU define Secure-NSC, Non-Secure and exempted regions / granularity = 64 MB
- SAU = 8 regions, used to overwrite IDAU in order to set secure areas and confirm Non-sec ones
- As showed below, the code area as defined in the memory map (final) requires 3 SAU regions

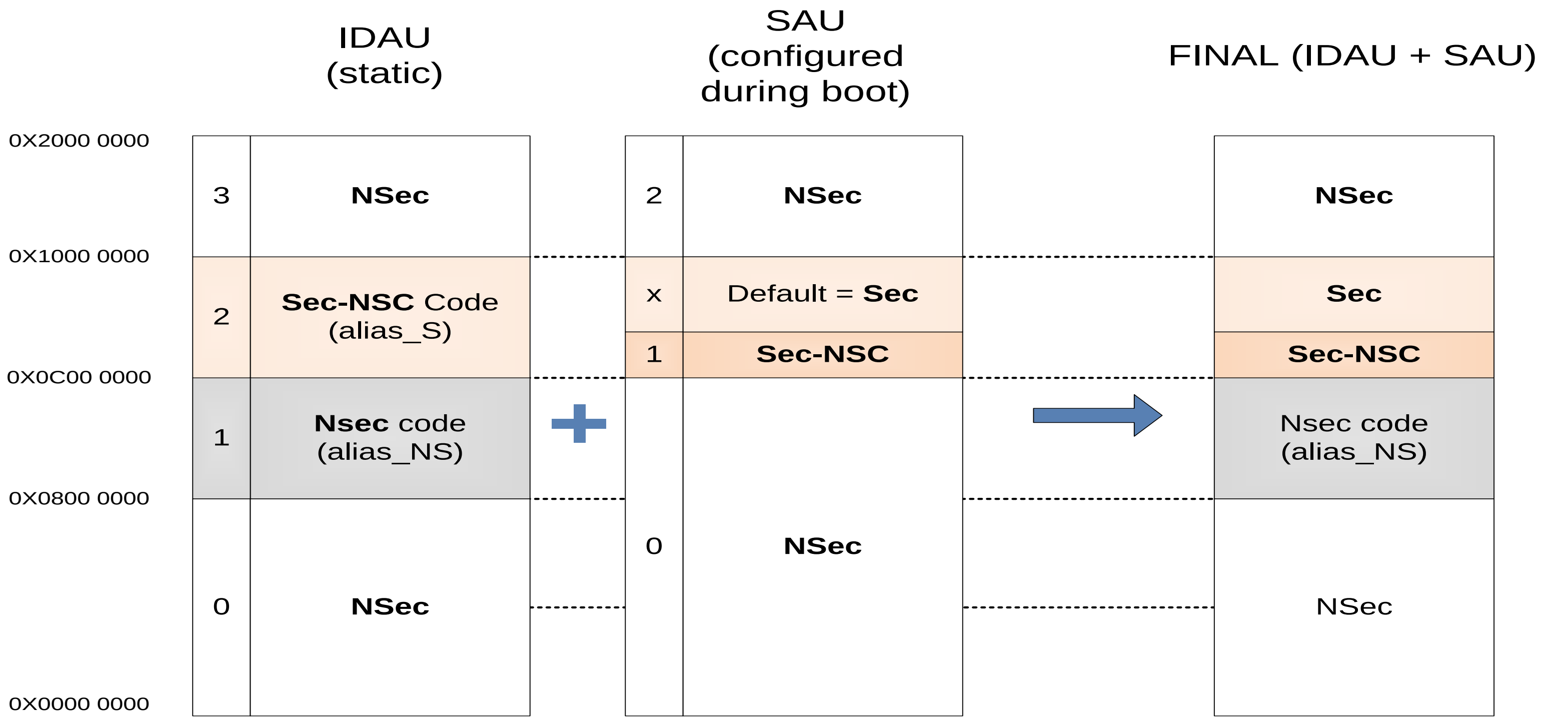
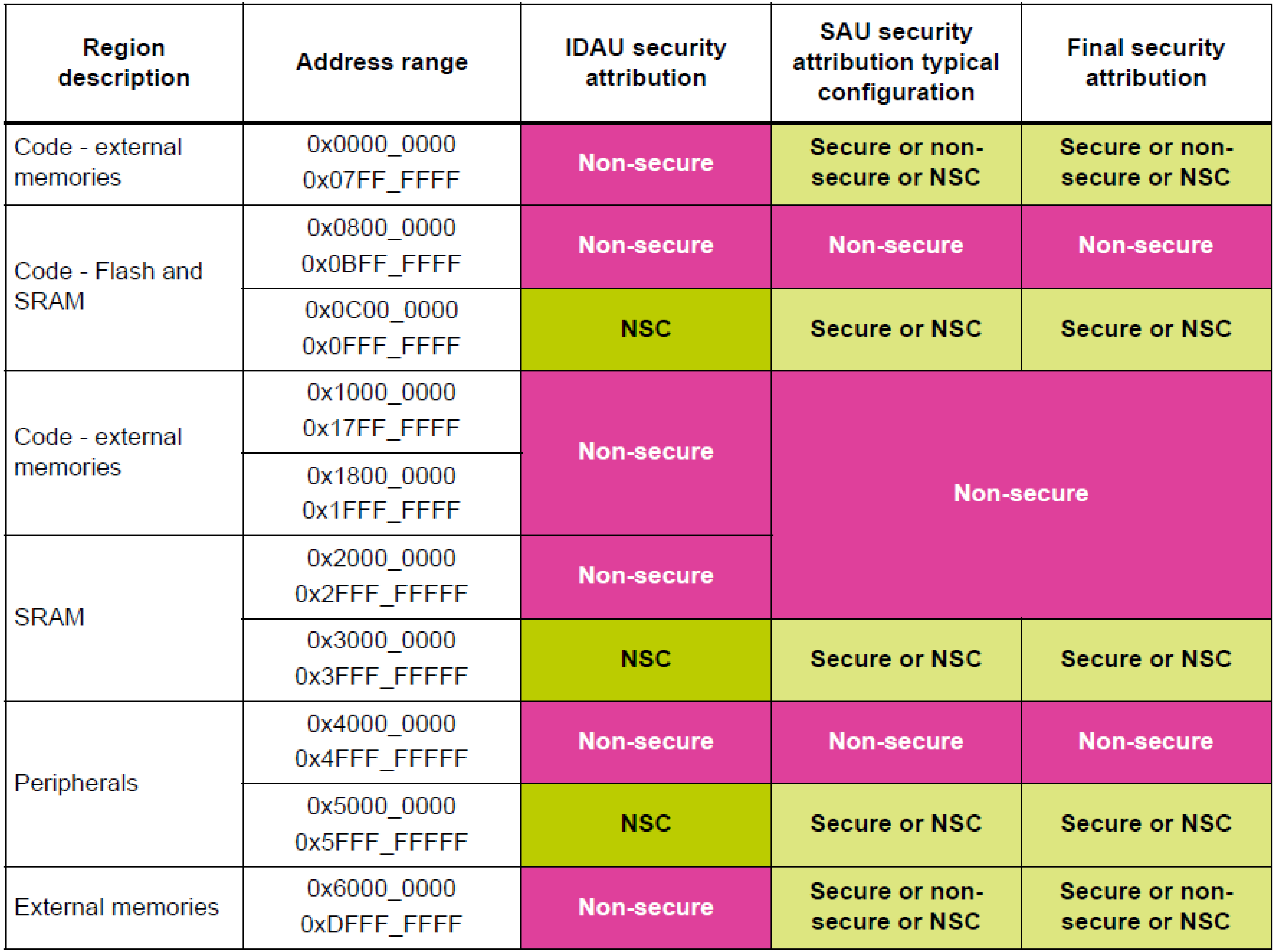
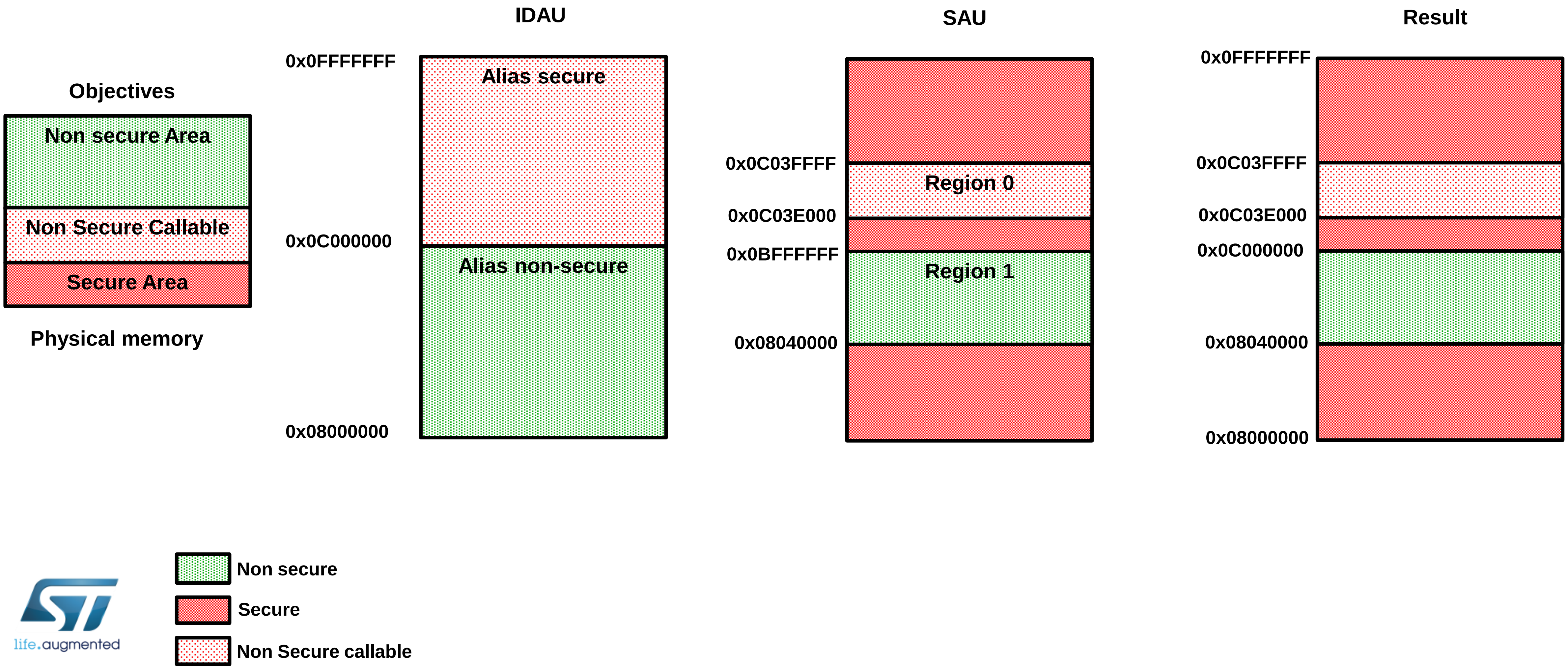


Table 1. Example of memory map security attribution vs SAU configuration regions[(1)] [(2)]



Region description	Address range	IDAU security attribution	SAU security attribution typical configuration	Final security attribution
Code - external memories	0x0000_0000 0x07FF_FFFF	Non-secure	Secure or non-secure or NSC	Secure or non-secure or NSC
Code - Flash and SRAM	0x0800_0000 0x0BFF_FFFF	Non-secure	Non-secure	Non-secure
	0x0C00_0000 0x0FFF_FFFF	NSC	Secure or NSC	Secure or NSC
Code - external memories	0x1000_0000 0x17FF_FFFF	Non-secure	Non-secure	
	0x1800_0000 0x1FFF_FFFF			
SRAM	0x2000_0000 0x2FFF_FFFFF	Non-secure		
	0x3000_0000 0x3FFF_FFFFF	NSC	Secure or NSC	Secure or NSC
Peripherals	0x4000_0000 0x4FFF_FFFFF	Non-secure	Non-secure	Non-secure
	0x5000_0000 0x5FFF_FFFFF	NSC	Secure or NSC	Secure or NSC
External memories	0x6000_0000 0xDFFF_FFFF	Non-secure	Secure or non-secure or NSC	Secure or non-secure or NSC

Example with Flash and SRAM



Objectives

Non secure Area

Non Secure Callable

Secure Area

Physical memory

IDAU

0x0FFFFFFF

Alias secure

0x0C000000

Alias non-secure

0x08000000

SAU

0x0C03FFFF

Region 0

0x0C03E000

0x0BFFFFFF

Region 1

0x08040000

Result

0x0FFFFFFF

0x0C03FFFF

0x0C03E000

0x0C000000

0x08040000

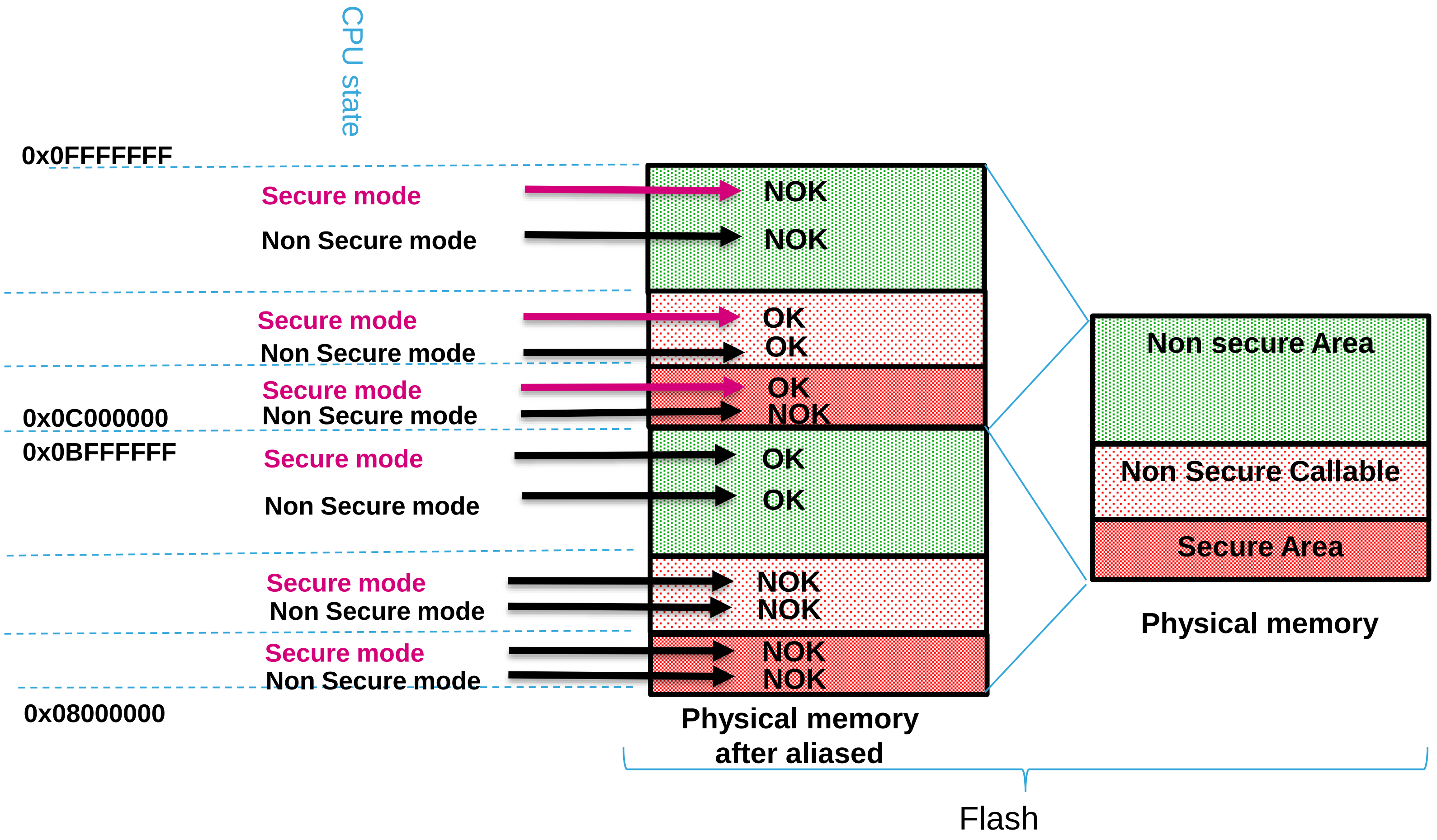
0x08000000

🟩 Non secure

🟥 Secure

▨ Non Secure callable

Access rule



CPU state

0x0FFFFFFF

Secure mode → NOK

Non Secure mode → NOK

Secure mode → OK

Non Secure mode → OK

Secure mode → OK

Non Secure mode → NOK

0x0C000000

0x0BFFFFFF

Secure mode → OK

Non Secure mode → OK

Secure mode → NOK

Non Secure mode → NOK

Secure mode → NOK

Non Secure mode → NOK

0x08000000

Physical memory after aliased

Non secure Area

Non Secure Callable

Secure Area

Physical memory

Flash

🟩 Non secure

🟥 Secure

▨ Non Secure callable

Trustzone activation

- One option byte to activate : TZEN
- SAU configuration to be done at boot time
- Additional flash security features are available:
  - Secure watermark-based user options bytes defining secure areas and HDP areas.
  - Secure or non-secure block-based areas can be configured on-the-fly after reset
  - An additional RDP protection: RDP level 0.5.

Flash TrustZone security

- Secure watermark-based area protection: part of the Flash memory can be protected against non-secure read and write access

- 2 areas could be configured thanks option bytes
  - SECWM1_PSTRT/SECWM1_PEND
  - SECWM2_PSTRT/SECWM2_PEND

- Those options byte could be only modified by a secure code if HDPxACCDIS bit is cleared.
  When it is set, options bytes are locked and can not be modified until next system reset.

Flash TrustZone security

- Secure hide protection (HDP) : once closed (set HDPxACCDIS bit in FLASH_SECHDPCR register), this part cannot be accessed anymore by any mean until the next boot.

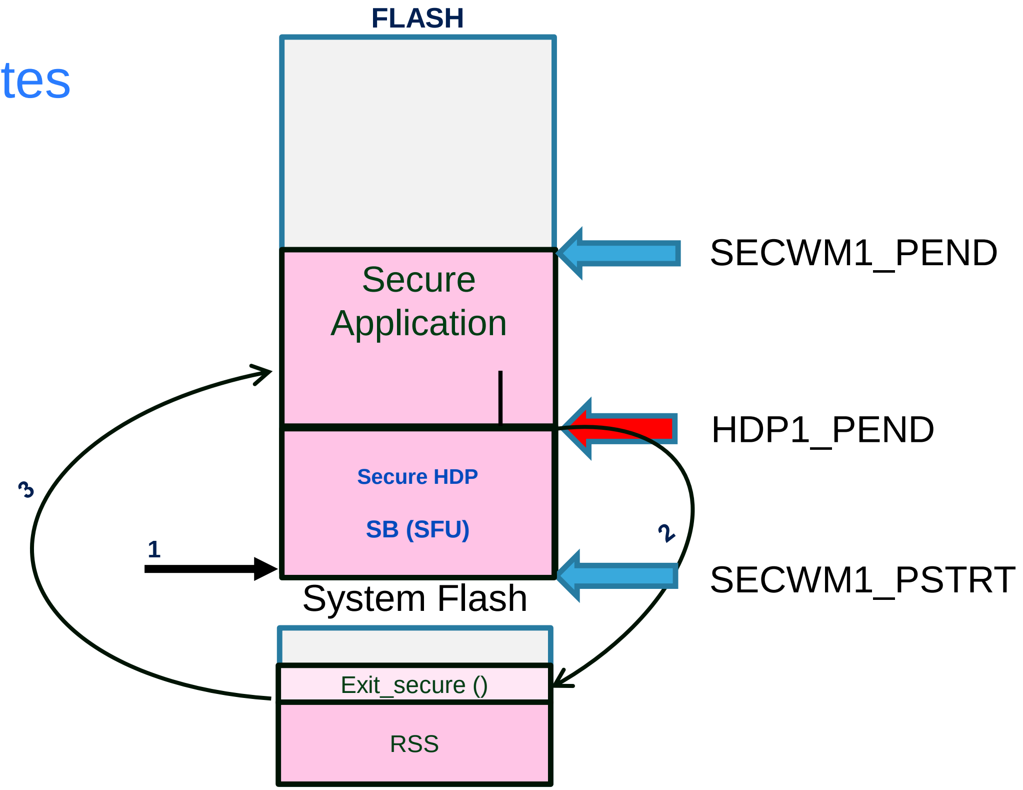
- HDP is located at the start of the Flash watermark-based secure area.

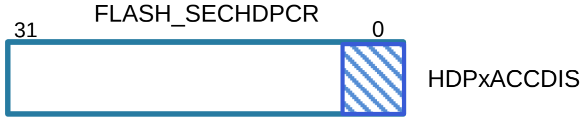
- Configured thanks 2 option bytes:
  - HDPxEN : HD activation
  - HDP1_PEND / HDP2_PEND : end page offset regarding SECWM1_PSTRT/SECWM2_PSTRT

- Those options byte could be only modified if HDPxACCDIS bit is cleared. When it is set, options bytes are locked and can not be modified until next system reset.

WATERMARK and HDP

- Secure watermark area
  - Start and End addresses defined in secure option bytes
- Secure Hide protection area
  - Start @ same as Secure area one
  - End @ defined in secure option bytes



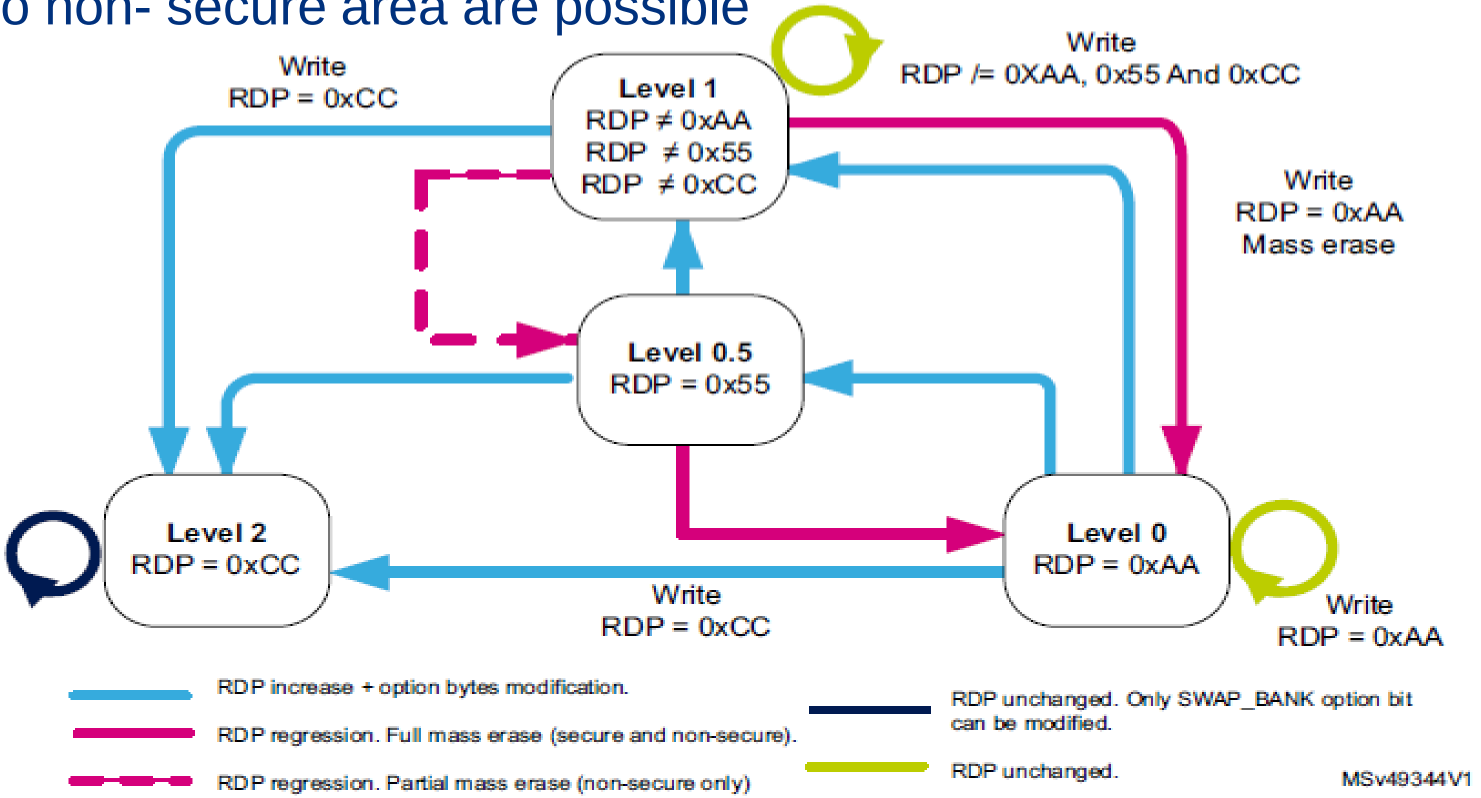


Flash TrustZone security

- Secure block-based area (SECBB) protection : any page can be programmed on-the-fly as secure or non-secure mode.
- Purpose : create secure area dynamically
- Configured thanks : FLASH_SECBB1Rx/ FLASH_SECBB2Rx registers (can only be access from secure mode)
- Caution : switching a page or memory block from secure to non-secure does not erase the content.

Flash TrustZone security

- RDP 0.5 : non-secure debug only
  - The debug access to secure area is prohibited.
  - Debug access to non- secure area are possible


Level 1
RDP ≠ 0xAA
RDP ≠ 0x55
RDP ≠ 0xCC
Level 0.5
RDP = 0x55
Level 2
RDP = 0xCC
Level 0
RDP = 0xAA
Write
RDP = 0xCC
Write
RDP /= 0XAA, 0x55 And 0xCC
Write
RDP = 0xAA
Mass erase
Write
RDP = 0xCC
Write
RDP = 0xAA
RDP increase + option bytes modification.
RDP regression. Full mass erase (secure and non-secure).
RDP regression. Partial mass erase (non-secure only)
RDP unchanged. Only SWAP_BANK option bit can be modified.
RDP unchanged.
MSv49344V1

Trustzone deactivation

- Deactivation of TZEN (from 1 to 0) is only possible when the RDP is changing from level 1 to level 0.

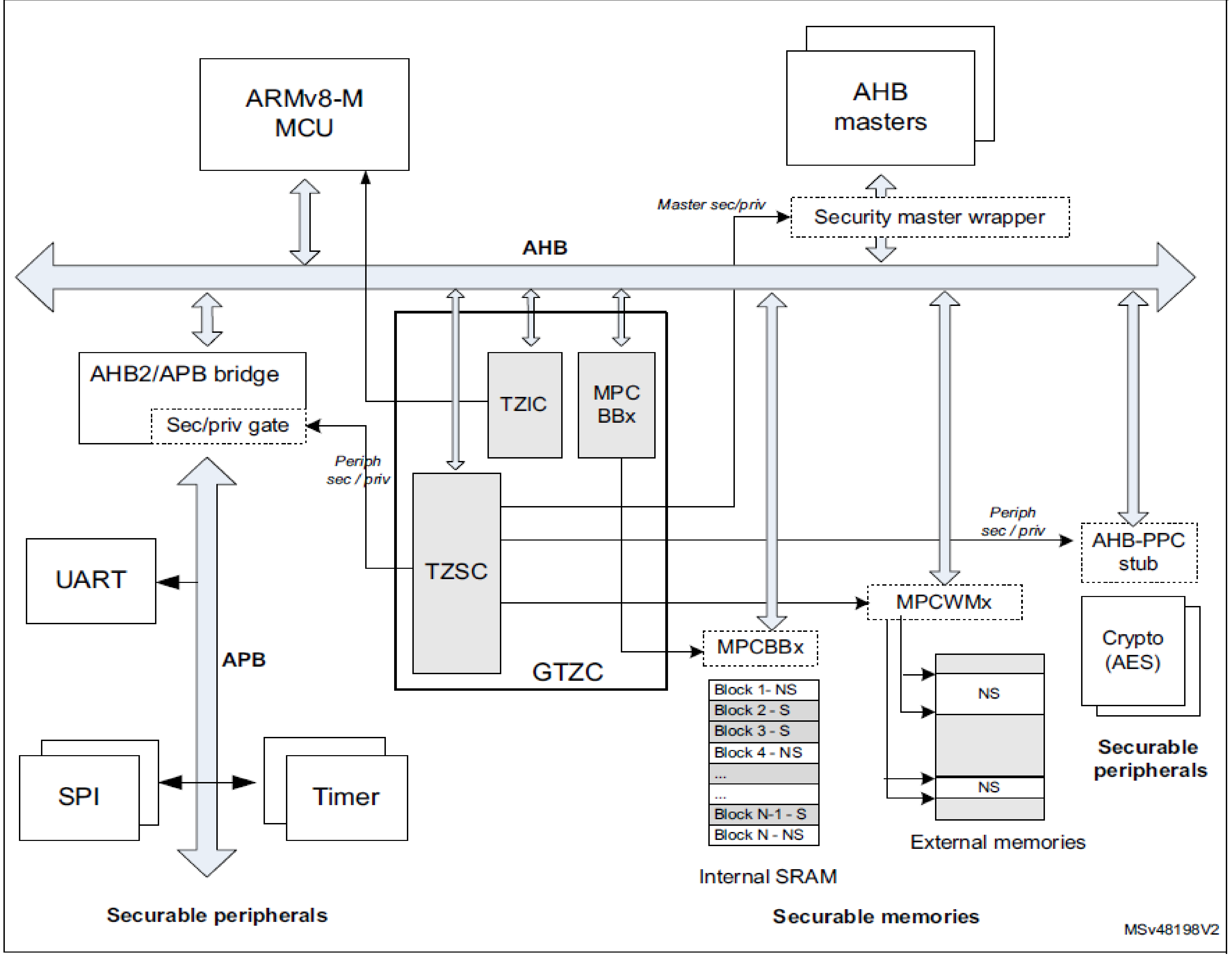
- When the TrustZone is deactivated after option bytes loading, the following security features are deactivated:
  - Watermark-based secure area
  - Block-based secure area
  - RDP level 0.5
  - Secure interrupt
  - All secure registers are RAZ/WI. (Read-as-zero, writes ignored)

GTZC

TZSC / MPC-BB / TZIC

Global TrustZone® controller (GTZC)

- TZSC: TrustZone® security controller :allows configuring the security attribute of:
  - Peripherals can be configured as secure or non-secure
  - External memories region: through watermark memory protection controller (watermark memory protection controller MPCWMx, x = 1,2,3)
- MPCBB: block-based memory protection controller allows configuring the security attribute of the SRAM1 and SRAM2 blocks
- TZIC: TrustZone illegal access controller :
  - gathers all illegal access events in the system and generates a secure interrupt towards NVIC.



ARMv8-M MCU

AHB masters

Master sec/priv

Security master wrapper

AHB

AHB2/APB bridge

Sec/priv gate

TZIC

MPC BBx

Periph sec / priv

TZSC

GTZC

UART

APB

SPI

Timer

Periph sec / priv

AHB-PPC stub

MPCWMx

MPCBBx

Crypto (AES)

Block 1- NS
Block 2 - S
Block 3 - S
Block 4 - NS
...
...
Block N-1 - S
Block N - NS

NS

NS

Internal SRAM

External memories

Securable peripherals

Securable peripherals

Securable memories

MSv48198V2

TrustZone peripheral classification

- When the TrustZone security is active, a peripheral can be either securable or TrustZone-aware:
  - Securable: its security attribute is configured by GTZC/TZSC controller.
  - TrustZone-aware: its security attribute is configured using some peripheral secure registers.

  For example the GPIO is a TrustZone-aware peripheral and the security attribute is configured through GPIOx_SECCFGR secure register.

TrustZone peripheral classification

- TrustZone-aware peripherals :

Bus	Peripherals
AHB2	GPIOA..GPIOH
AHB1	MPCBB2 MPCBB1 MPCWM2 MPCWM1 TZIC TZSC EXTIT Flash memory RCC DMAMUX1 DMA2 DMA1
AHB2	OTFDEC
APB2	SYSCFG
APB1	PWR RTC

- The remaining peripherals are Securable.

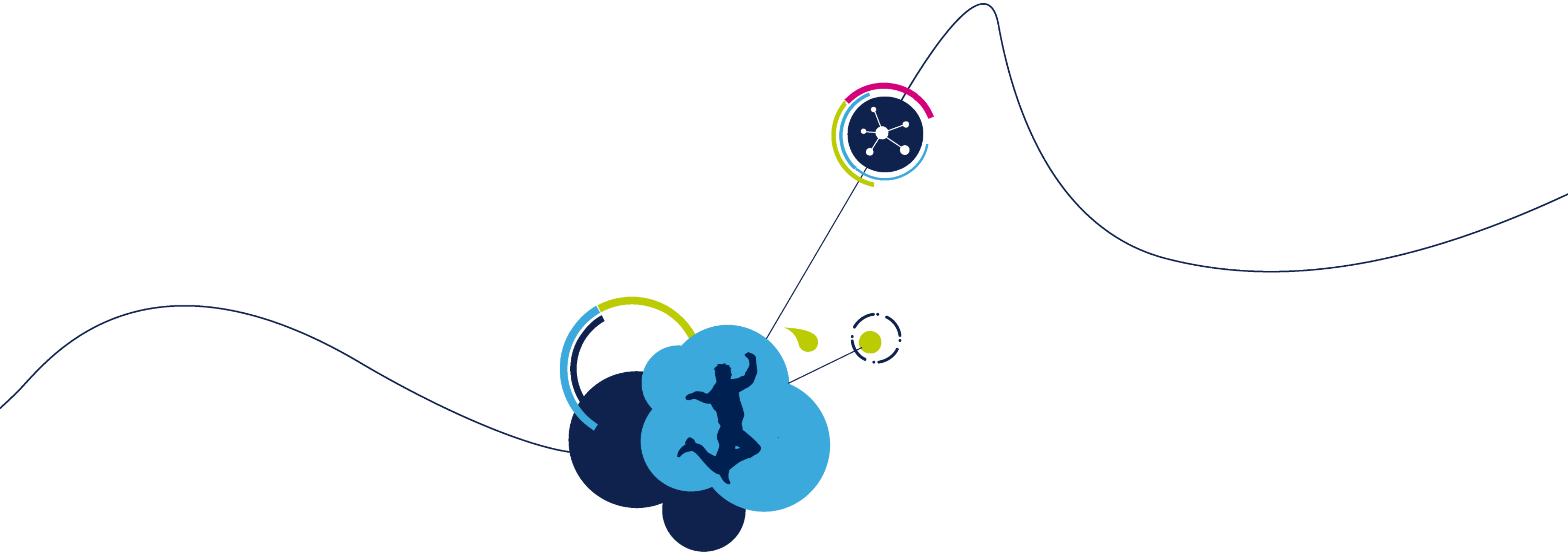
Security Features by STM32 Series

STM32 Series	Security Features																
	96-Bit Unique ID	FLASH WRP	FLASH PCROP	FLASH RDP	Unique entry point	Secure mem/ HDP	MPU	Firewall	Trustzone	OTFDEC	Tamper	TRNG	CRYPT AES	HASH	PKA	Cryptolib	Arm Cortex®
STM32 **F0**	🟩	🟩		🟩							🟩					🟩	**M0**
STM32 **F1**	🟩	🟩					🟩				🟩					🟩	**M3**
STM32 **F2**	🟩	🟩		🟩			🟩				🟩	🟩	▨	▨		🟩	**M3**
STM32 **F3**	🟩	🟩		🟩			🟩				🟩					🟩	**M4**
STM32 **F4**	🟩	🟩	▨	🟩			🟩				🟩	🟩	▨	▨		🟩	**M4**
STM32 **F7**	🟩	🟩	▨	🟩			🟩				🟩	🟩	▨	▨		🟩	**M7**
STM32 **L0**	🟩	🟩	🟩	🟩			🟩	🟩			🟩		▨			🟩	**M0+**
STM32 **L1**	🟩	🟩	🟩	🟩			🟩				🟩		▨			🟩	**M3**
STM32 **L4**	🟩	🟩	🟩	🟩			🟩	🟩			🟩	🟩	▨	▨		🟩	**M4**
STM32 **L5**	🟩	🟩		🟩	🟩	🟩	🟩		🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	**M33**
STM32 **H7**	🟩	🟩	🟩	🟩	▨	▨	🟩				🟩	🟩	▨	▨		🟩	**M7/M4**
STM32 **G0**	🟩	🟩	🟩	🟩	▨	▨	🟩				🟩	🟩	▨			🟩	**M0+**
STM32 **G4**	🟩	🟩	🟩	🟩	▨	▨	🟩				🟩	🟩	▨			🟩	**M4**
STM32 **WB**	🟩	🟩	🟩	🟩			🟩				🟩	🟩	🟩		🟩	🟩	**M4/M0+**

🟩	Available on all devices
▨	Depends on device part number

Reference

- AN5347 : STM32L5 Series TrustZone® features
- RM0438 : STM32L552xx and STM32L562xx advanced Arm®-based 32-bit MCUs



TrustZone Hands-on

life.augmented

Hands-on

Purpose : create a basic example of secure/non secure appli

Step 1: activate TrustZone on our target

Step 2 : create a basic TrustZone project and assign the GPIO of the led to the secure word

Step 3 : in the non-secure world, on a push button event call a NSC procedure to toggle the LED.

Step 4 : deactivate TrustZone on our target

OTFDEC

On-the-fly decryption

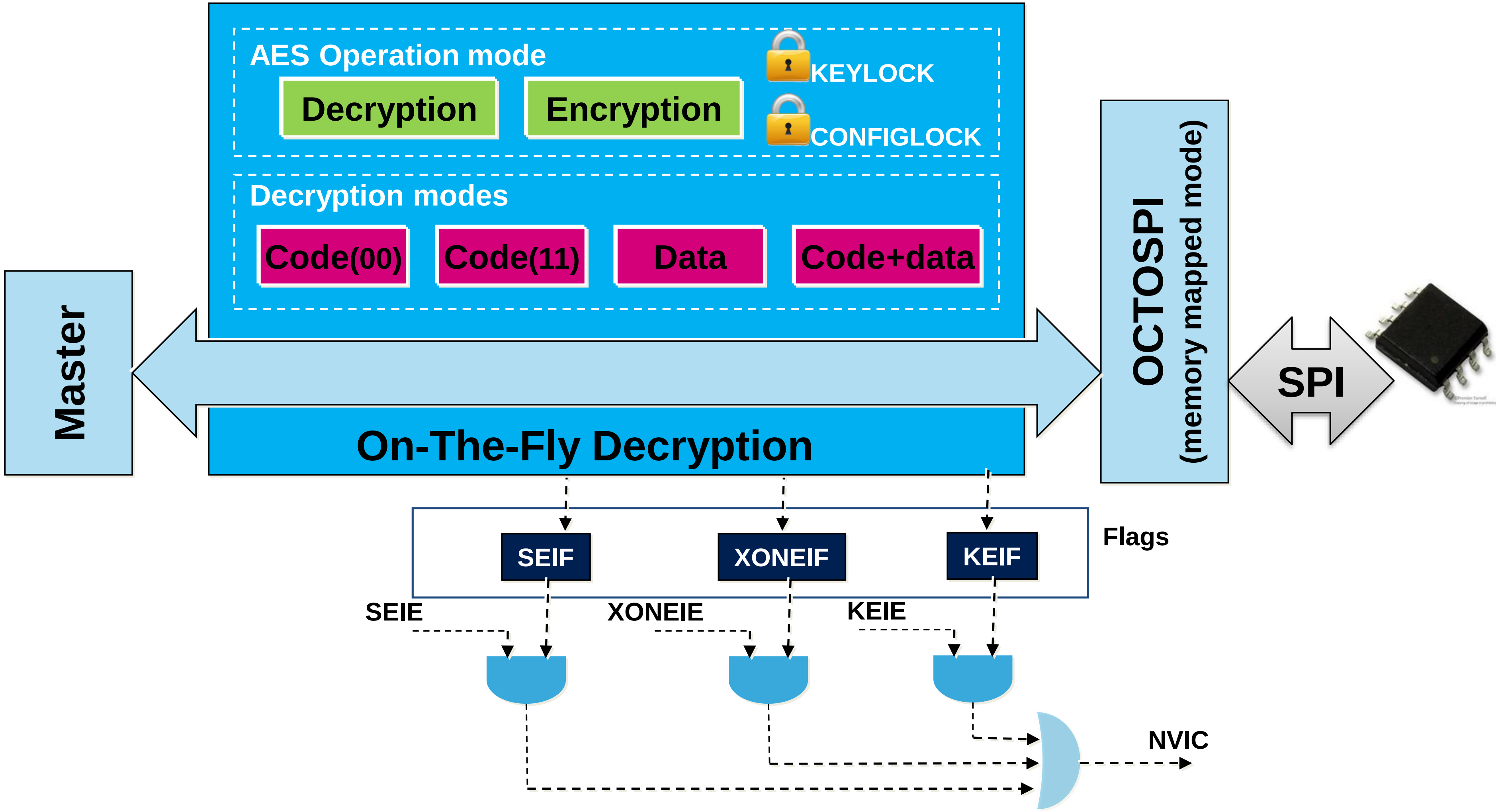
- Purpose : encrypt and decrypt with low latency code or data stored within external Flash (external OctoSPI memories used in Memory-mapped mode)
- AES in counter mode, with a 128-bit key to achieve the lowest possible latency
- Four regions (granularity 4096 bytes) can be define with:
  - 128 bits secret key
  - two bytes firmware version
  - eight bytes application-defined nonce
- Secure only programming if TrustZone security is enabled.

On-the-fly decryption

- Encryption keys confidentiality and integrity protection :
  - Write-only registers, with software locking mechanism
  - Availability of 8-bit CRC as public key information

- 4 decryption operation mode :
  - 00: instruction fetch only
  - 01: data read access only
  - 10: code or data accesses
  - 11: instruction fetch only with enhanced encryption

OTFDEC Block Diagram



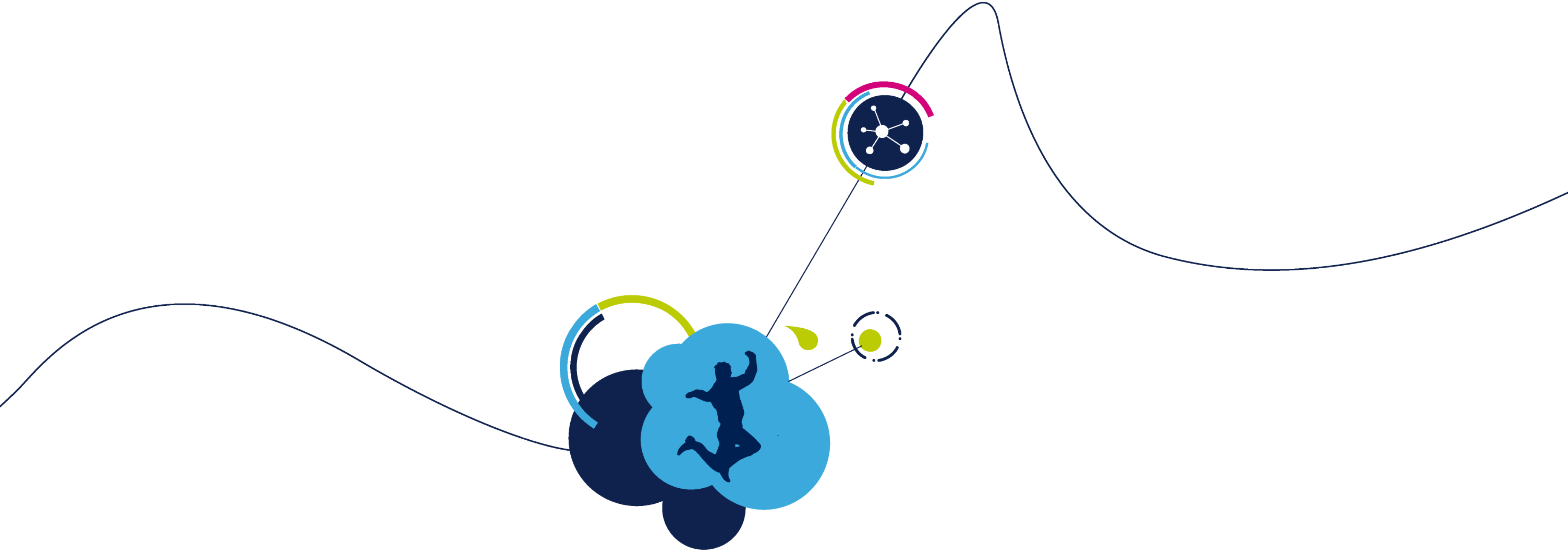
Security Features by STM32 Series

STM32 Series	Security Features																
	96-Bit Unique ID	FLASH WRP	FLASH PCROP	FLASH RDP	Unique entry point	Secure mem/ HDP	MPU	Firewall	Trustzone	OTFDEC	Tamper	TRNG	CRYPT AES	HASH	PKA	Cryptolib	Arm Cortex®
STM32 **F0**	🟩	🟩		🟩							🟩					🟩	**M0**
STM32 **F1**	🟩	🟩					🟩				🟩					🟩	**M3**
STM32 **F2**	🟩	🟩		🟩			🟩				🟩	🟩	▨	▨		🟩	**M3**
STM32 **F3**	🟩	🟩		🟩			🟩				🟩					🟩	**M4**
STM32 **F4**	🟩	🟩	▨	🟩			🟩				🟩	🟩	▨	▨		🟩	**M4**
STM32 **F7**	🟩	🟩	▨	🟩			🟩				🟩	🟩	▨	▨		🟩	**M7**
STM32 **L0**	🟩	🟩	🟩	🟩			🟩	🟩			🟩		▨			🟩	**M0+**
STM32 **L1**	🟩	🟩	🟩	🟩			🟩				🟩		▨			🟩	**M3**
STM32 **L4**	🟩	🟩	🟩	🟩			🟩	🟩			🟩	🟩	▨	▨		🟩	**M4**
STM32 **L5**	🟩	🟩		🟩	🟩	🟩	🟩		🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	**M33**
STM32 **H7**	🟩	🟩	🟩	🟩	▨	▨	🟩				🟩	🟩	▨	▨		🟩	**M7/M4**
STM32 **G0**	🟩	🟩	🟩	🟩	▨	▨	🟩				🟩	🟩	▨			🟩	**M0+**
STM32 **G4**	🟩	🟩	🟩	🟩	▨	▨	🟩				🟩	🟩	▨			🟩	**M4**
STM32 **WB**	🟩	🟩	🟩	🟩			🟩				🟩	🟩	🟩		🟩	🟩	**M4/M0+**

🟩 Available on all devices

▨ Depends on device part number

life.augmented



Tamper

Tamper protection

- Purpose : detect physical tampering in a secure application and to automatically erase sensitive data in case of intrusion.

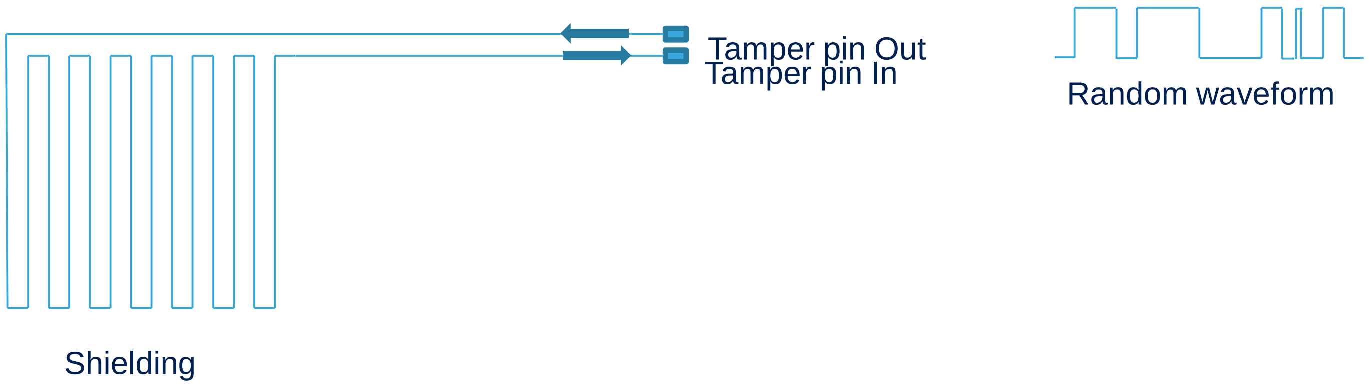
- Part of the RTC domain (except on L5) / available in VBAT mode

- On detection
  - erase of the backup register
  - prohibit access the backup SRAM or erase it (STM32L5)
  - can generate a timestamp event

- Available on all family

L5 active tamper

- Passive tamper detection just checks a static level
  - if an attack manages to short the tamper input pin to the inactive state, then there is no tamper event detection.
- The Active Tampering feature increases the security level by auto checking that the tamper pins are not externally opened or shorted
- This is achieve by generation of random patterns on inputs/outputs .



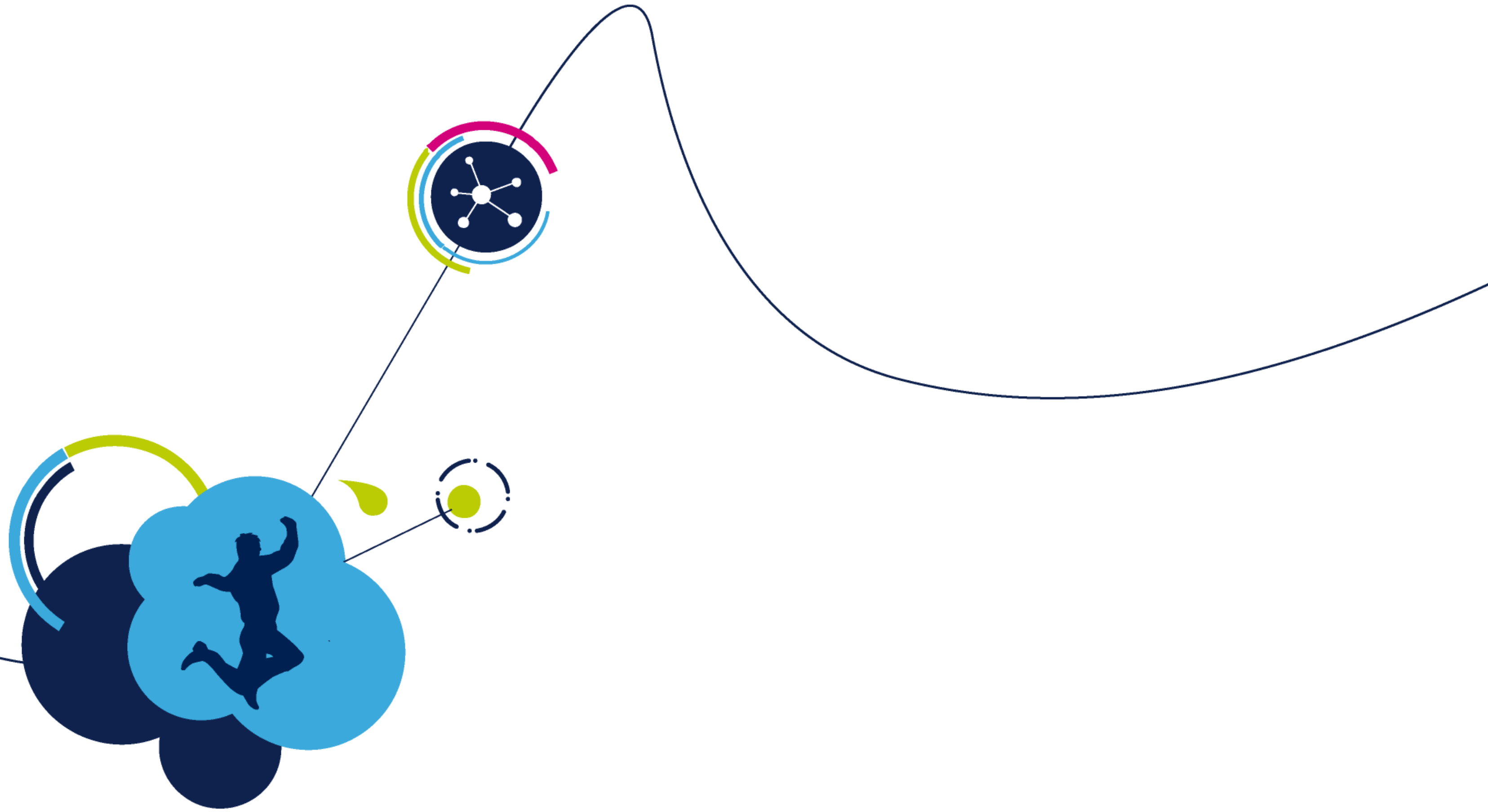
Security Features by STM32 Series

STM32 Series	Security Features																
	96-Bit Unique ID	FLASH WRP	FLASH PCROP	FLASH RDP	Unique entry point	Secure mem/ HDP	MPU	Firewall	Trustzone	OTFDEC	Tamper	TRNG	CRYPT AES	HASH	PKA	Cryptolib	Arm Cortex®
STM32 **F0**	🟩	🟩		🟩							🟩					🟩	**M0**
STM32 **F1**	🟩	🟩					🟩				🟩					🟩	**M3**
STM32 **F2**	🟩	🟩		🟩			🟩				🟩	🟩	▨	▨		🟩	**M3**
STM32 **F3**	🟩	🟩		🟩			🟩				🟩					🟩	**M4**
STM32 **F4**	🟩	🟩	▨	🟩			🟩				🟩	🟩	▨	▨		🟩	**M4**
STM32 **F7**	🟩	🟩	▨	🟩			🟩				🟩	🟩	▨	▨		🟩	**M7**
STM32 **L0**	🟩	🟩	🟩	🟩			🟩	🟩			🟩		▨			🟩	**M0+**
STM32 **L1**	🟩	🟩	🟩	🟩			🟩				🟩		▨			🟩	**M3**
STM32 **L4**	🟩	🟩	🟩	🟩			🟩	🟩			🟩	🟩	▨	▨		🟩	**M4**
STM32 **L5**	🟩	🟩		🟩	🟩	🟩	🟩		🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	**M33**
STM32 **H7**	🟩	🟩	🟩	🟩	▨	▨	🟩				🟩	🟩	▨	▨		🟩	**M7/M4**
STM32 **G0**	🟩	🟩	🟩	🟩	▨	▨	🟩				🟩	🟩	▨			🟩	**M0+**
STM32 **G4**	🟩	🟩	🟩	🟩	▨	▨	🟩				🟩	🟩	▨			🟩	**M4**
STM32 **WB**	🟩	🟩	🟩	🟩			🟩				🟩	🟩	🟩		🟩	🟩	**M4/M0+**

🟩 Available on all devices

▨ Depends on device part number

life.augmented



Tamper Hands-on

Hands-on

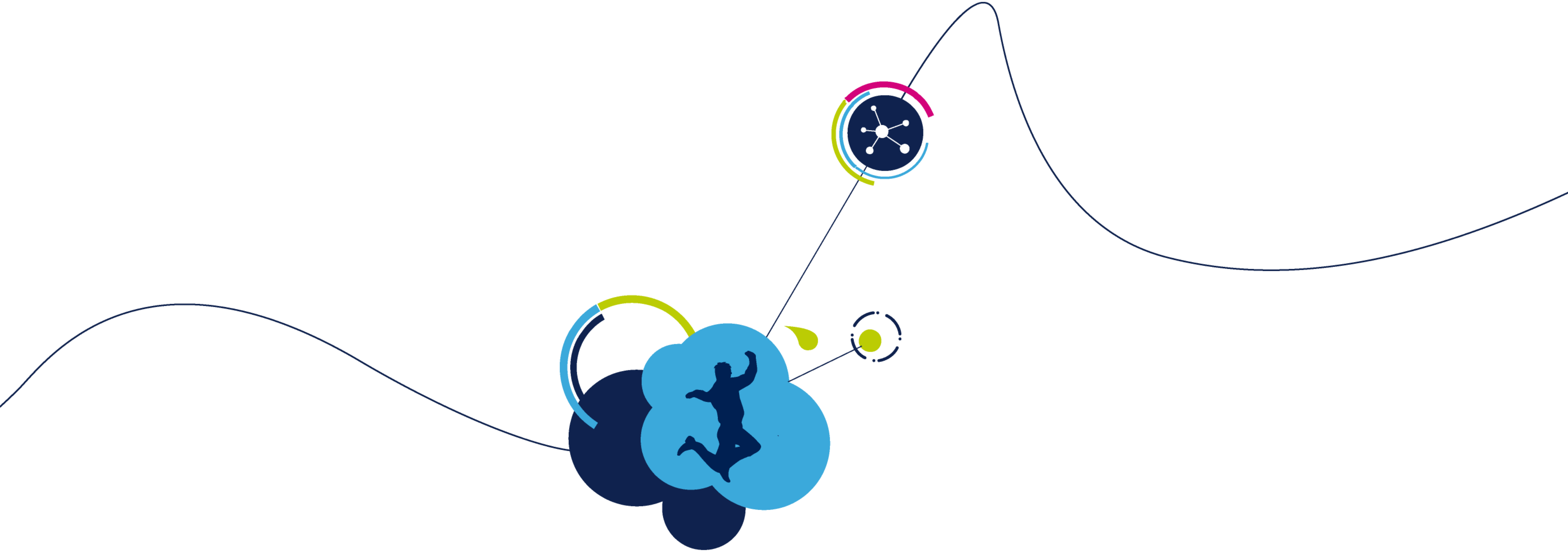
- Purpose : check the anti-tamper mechanism

- Step 1 : CubeIDE with Nucleo STM32L476RG and create a code which configure tamper and put some data in the backup register

```
HAL_RTCEx_BKUPWrite(&hrtc,RTC_BKP_DR0,0xdeadbee0);
HAL_RTCEx_BKUPWrite(&hrtc,RTC_BKP_DR1,0xdeadbee1);
```

- Step 2 : simulate a tamper event thanks pushbutton and check the backup register content

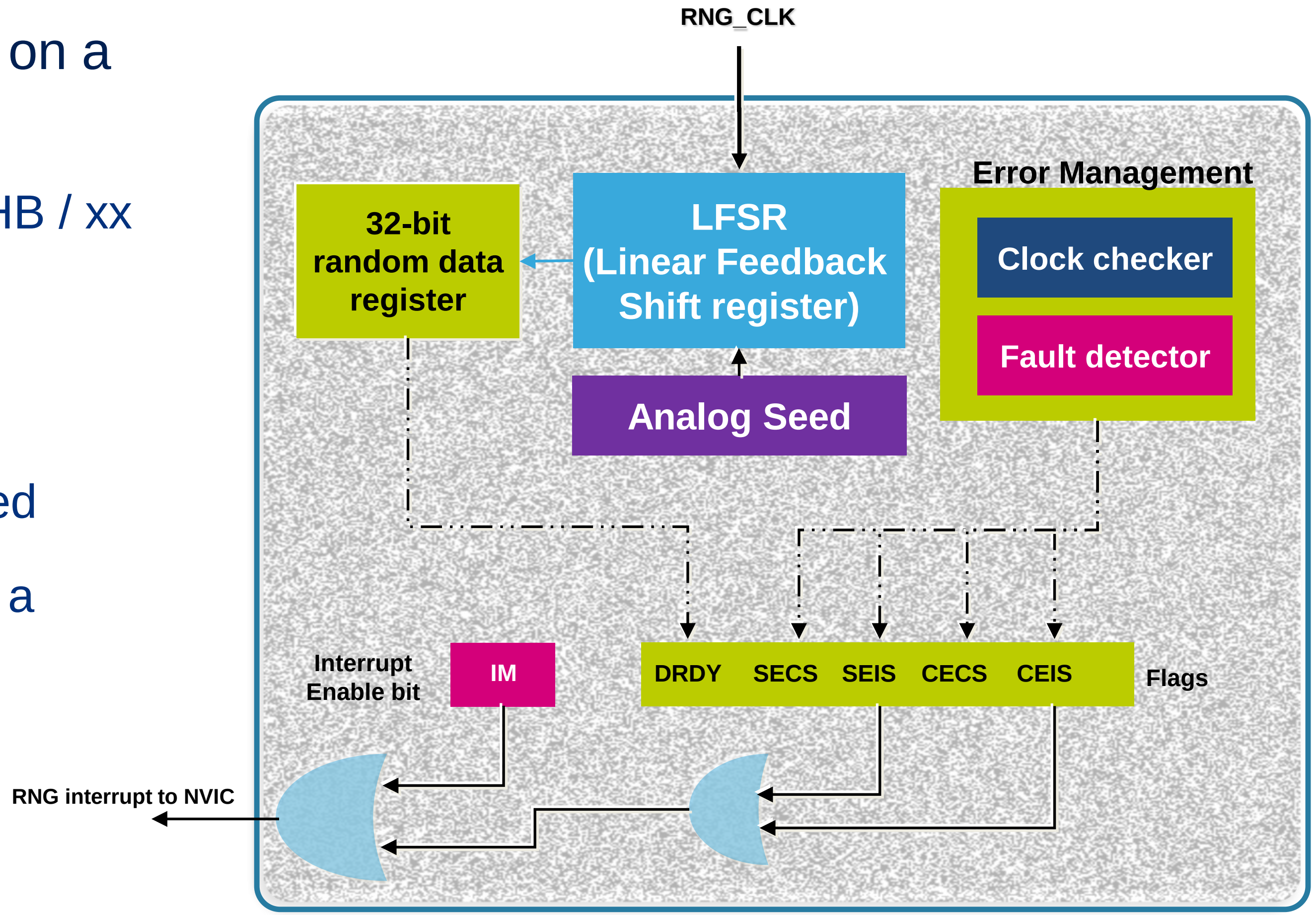
Crypto HW resources



TRNG

True Random Number Generator

- 32-bit Random Number Generator based on a noise source
  - Generated at an average frequency of AHB / xx
- Three flags:
  - Valid random data is ready
  - An abnormal sequence occurs on the seed
  - A frequency error is detected when using a PLL48 RNG clock source
- One interrupt
  - To indicate an error (an abnormal seed sequence or a frequency error)



Security Features by STM32 Series

STM32 Series	Security Features																
	96-Bit Unique ID	FLASH WRP	FLASH PCROP	FLASH RDP	Unique entry point	Secure mem/ HDP	MPU	Firewall	Trustzone	OTFDEC	Tamper	TRNG	CRYPT AES	HASH	PKA	Cryptolib	Arm Cortex®
STM32 **F0**	🟩	🟩		🟩							🟩					🟩	**M0**
STM32 **F1**	🟩	🟩					🟩				🟩					🟩	**M3**
STM32 **F2**	🟩	🟩		🟩			🟩				🟩	🟩	▨	▨		🟩	**M3**
STM32 **F3**	🟩	🟩		🟩			🟩				🟩					🟩	**M4**
STM32 **F4**	🟩	🟩	▨	🟩			🟩				🟩	🟩	▨	▨		🟩	**M4**
STM32 **F7**	🟩	🟩	▨	🟩			🟩				🟩	🟩	▨	▨		🟩	**M7**
STM32 **L0**	🟩	🟩	🟩	🟩			🟩	🟩			🟩	▨	▨			🟩	**M0+**
STM32 **L1**	🟩	🟩	🟩	🟩			🟩				🟩		▨			🟩	**M3**
STM32 **L4**	🟩	🟩	🟩	🟩			🟩	🟩			🟩	🟩	▨	▨		🟩	**M4**
STM32 **L5**	🟩	🟩		🟩	🟩	🟩	🟩		🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	**M33**
STM32 **H7**	🟩	🟩	🟩	🟩	▨	▨	🟩				🟩	🟩	▨	▨		🟩	**M7/M4**
STM32 **G0**	🟩	🟩	🟩	🟩	▨	▨	🟩				🟩	🟩	▨			🟩	**M0+**
STM32 **G4**	🟩	🟩	🟩	🟩	▨	▨	🟩				🟩	🟩	▨			🟩	**M4**
STM32 **WB**	🟩	🟩	🟩	🟩			🟩				🟩	🟩	🟩		🟩	🟩	**M4/M0+**

🟩 Available on all devices

▨ Depends on device part number

life.augmented

Reference

- **AN4230**: STM32 microcontroller random number generation validation using the NIST statistical test suite

AES / CRYPT

AES

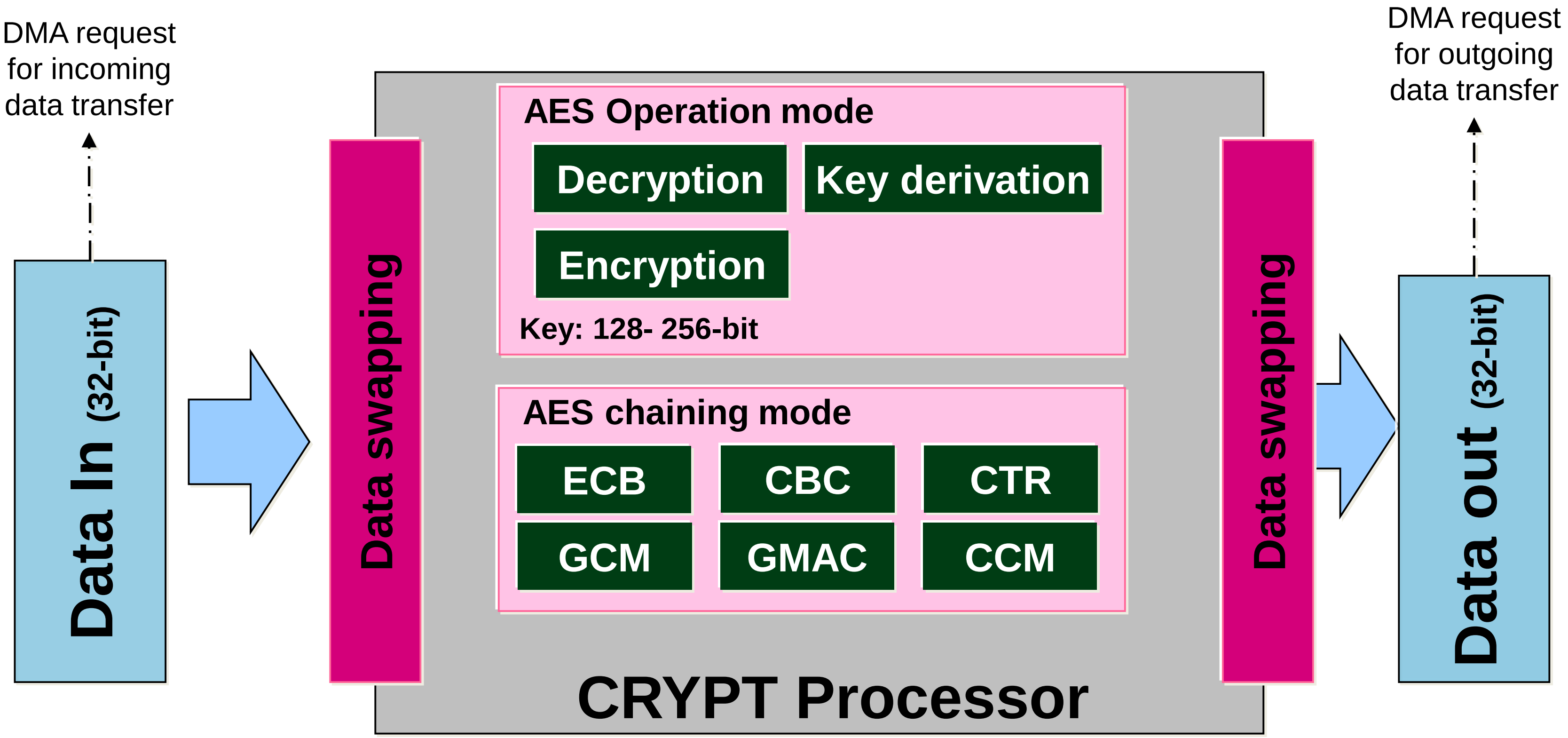
- Algorithms supported :
  - AES chaining modes ECB, CBC, CTR, GCM, GMAC, CCM for key sizes of 128 or 256 bits

- DMA transfers for incoming and outgoing data (two DMA channels are required).

- Available on :
  - STM32F423
  - STM32F730-732-733
  - STM32L021-041-062-063-081-082-083
  - STM32L162
  - STM32L422-442-443-462-485-486-4a6-4s5-4s7-4s9
  - STM32L562
  - STM32G041-081
  - STM32G441-483-484
  - STM32WB50-W55

AES Block Diagram

- NIST FIPS compliant



What do we call a key derivation ?

- For an ECB or CBC decryption, a key for the first round of decryption must be derived from the key of the last round of encryption. This is why a complete key schedule of encryption is required before performing the decryption.

Figure 74. ECB encryption and decryption principle

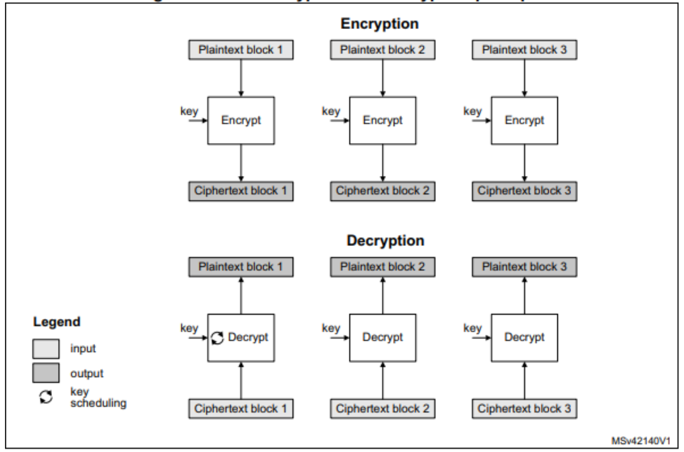
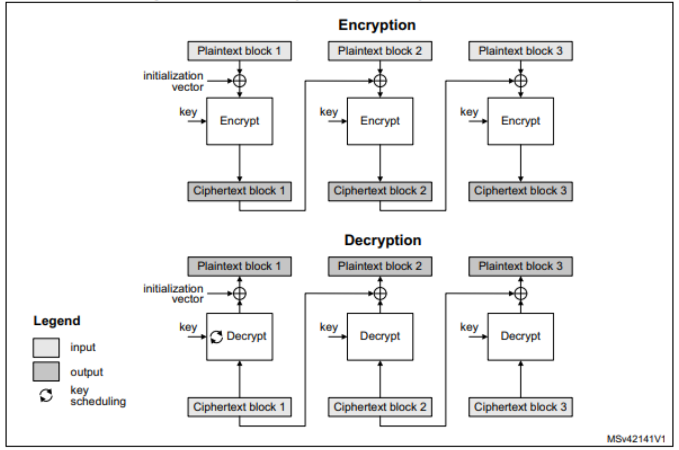


Figure 75. CBC encryption and decryption principle



- This key preparation is not required for AES decryption in modes other than ECB or CBC.

What about padding?

- AES peripheral does not implement automatic data padding operation

Performance

Extract from STM32G0x1 RM

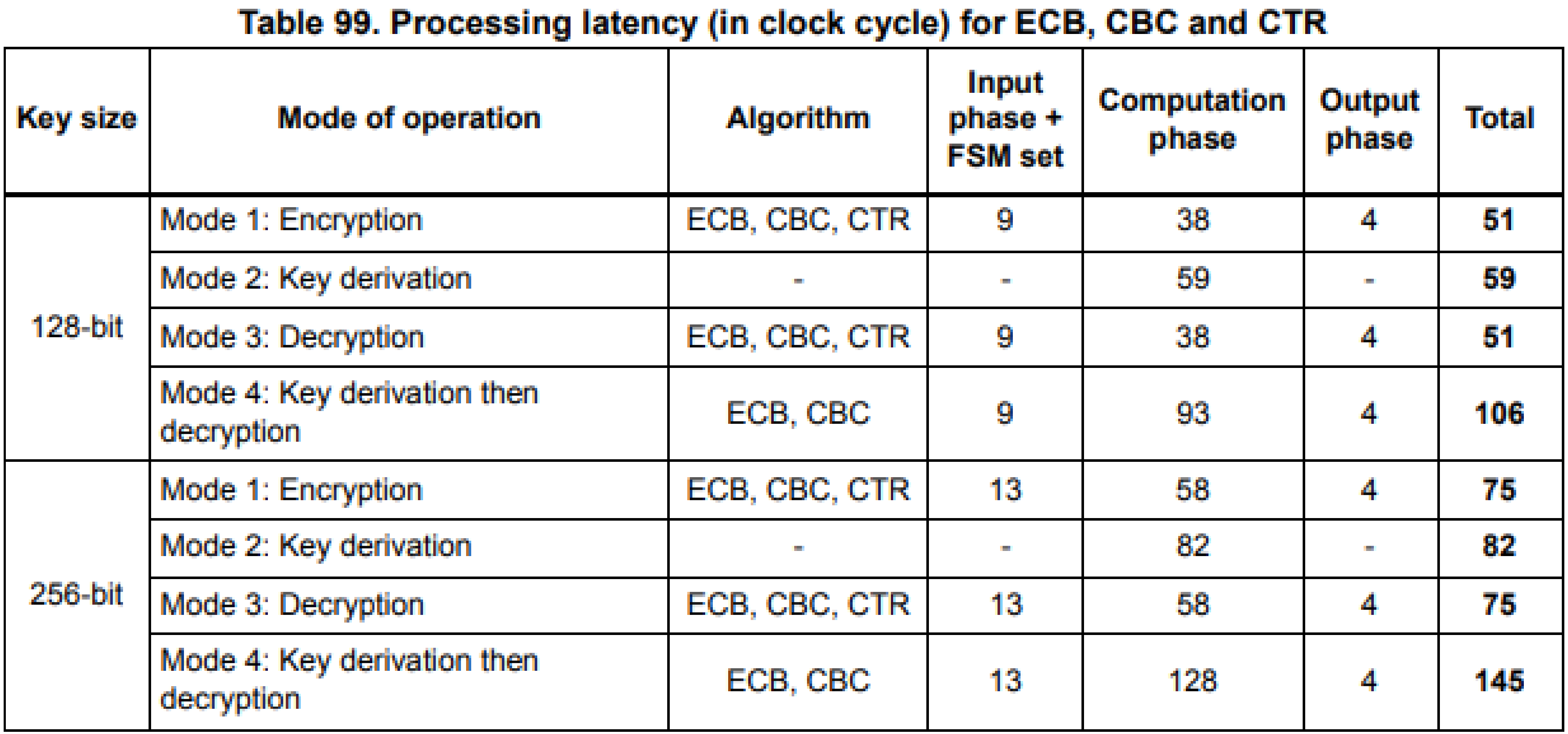


Table 99. Processing latency (in clock cycle) for ECB, CBC and CTR

Key size	Mode of operation	Algorithm	Input phase + FSM set	Computation phase	Output phase	Total
128-bit	Mode 1: Encryption	ECB, CBC, CTR	9	38	4	**51**
	Mode 2: Key derivation	-	-	59	-	**59**
	Mode 3: Decryption	ECB, CBC, CTR	9	38	4	**51**
	Mode 4: Key derivation then decryption	ECB, CBC	9	93	4	**106**
256-bit	Mode 1: Encryption	ECB, CBC, CTR	13	58	4	**75**
	Mode 2: Key derivation	-	-	82	-	**82**
	Mode 3: Decryption	ECB, CBC, CTR	13	58	4	**75**
	Mode 4: Key derivation then decryption	ECB, CBC	13	128	4	**145**

Performance

Extract from STM32G0x1 RM

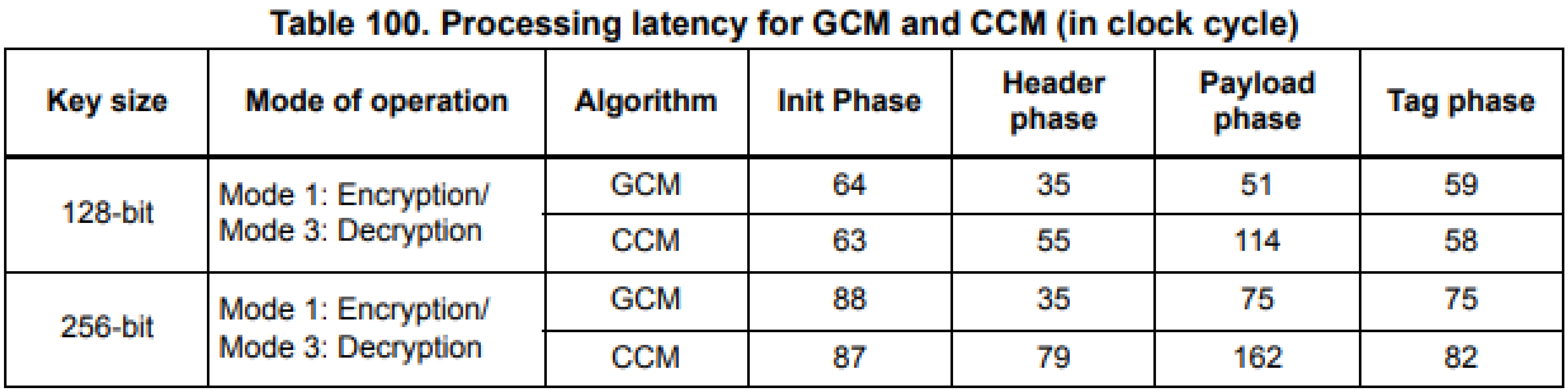


Table 100. Processing latency for GCM and CCM (in clock cycle)

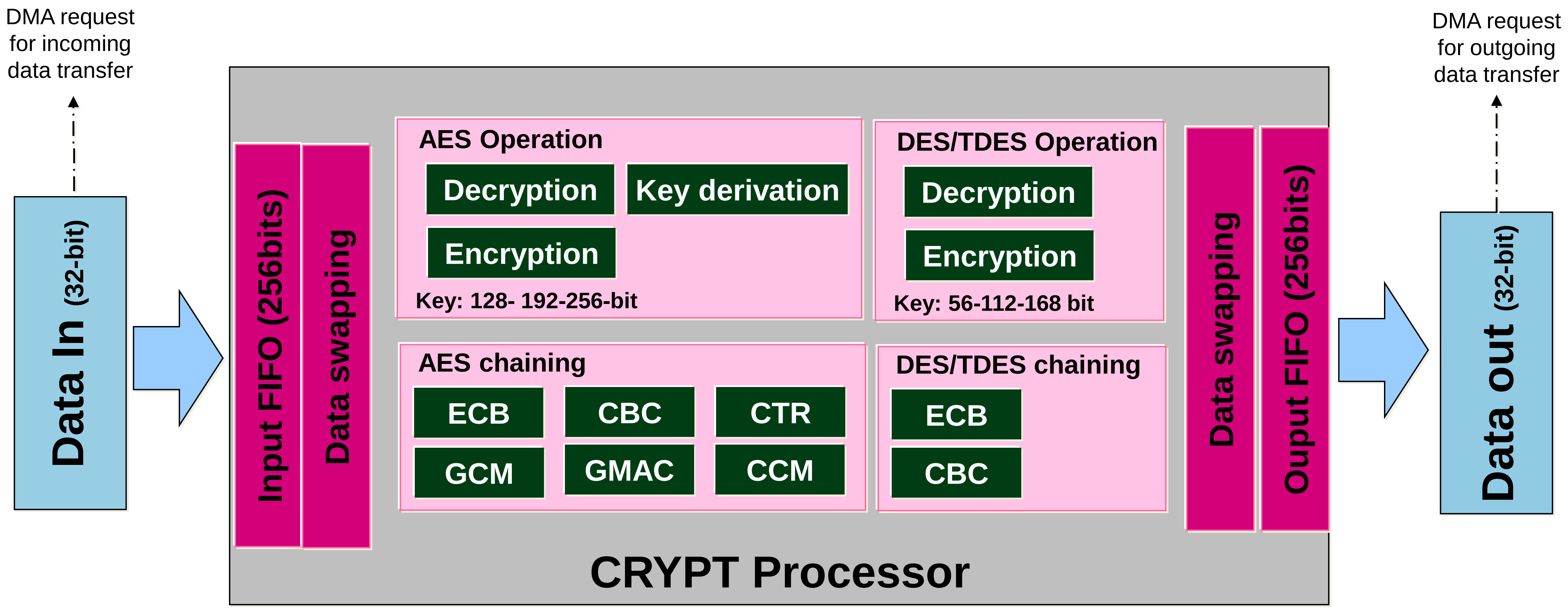
Key size	Mode of operation	Algorithm	Init Phase	Header phase	Payload phase	Tag phase
128-bit	Mode 1: Encryption/ Mode 3: Decryption	GCM	64	35	51	59
		CCM	63	55	114	58
256-bit	Mode 1: Encryption/ Mode 3: Decryption	GCM	88	35	75	75
		CCM	87	79	162	82

CRYPT

- Hardware acceleration of
  - DES/TDES chaining modes ECB and CBC standard 56-bit keys with 8-bit parity per key
  - AES chaining modes ECB, CBC, CTR, GCM, GMAC, CCM for key sizes of 128, 192 or 256 bits
- DMA transfers for incoming and outgoing data (two DMA channels are required).
- Input and output FIFOs (each 8 words deep) for better performance.
- Available on :
  - STM32F215-217
  - STM32F415-417-437-439-479
  - STM32F750-756-777-779
  - STM32H750-753-755-757

CRYPT Block Diagram

- NIST FIPS compliant



Performance

Extract from STM32H742 RM

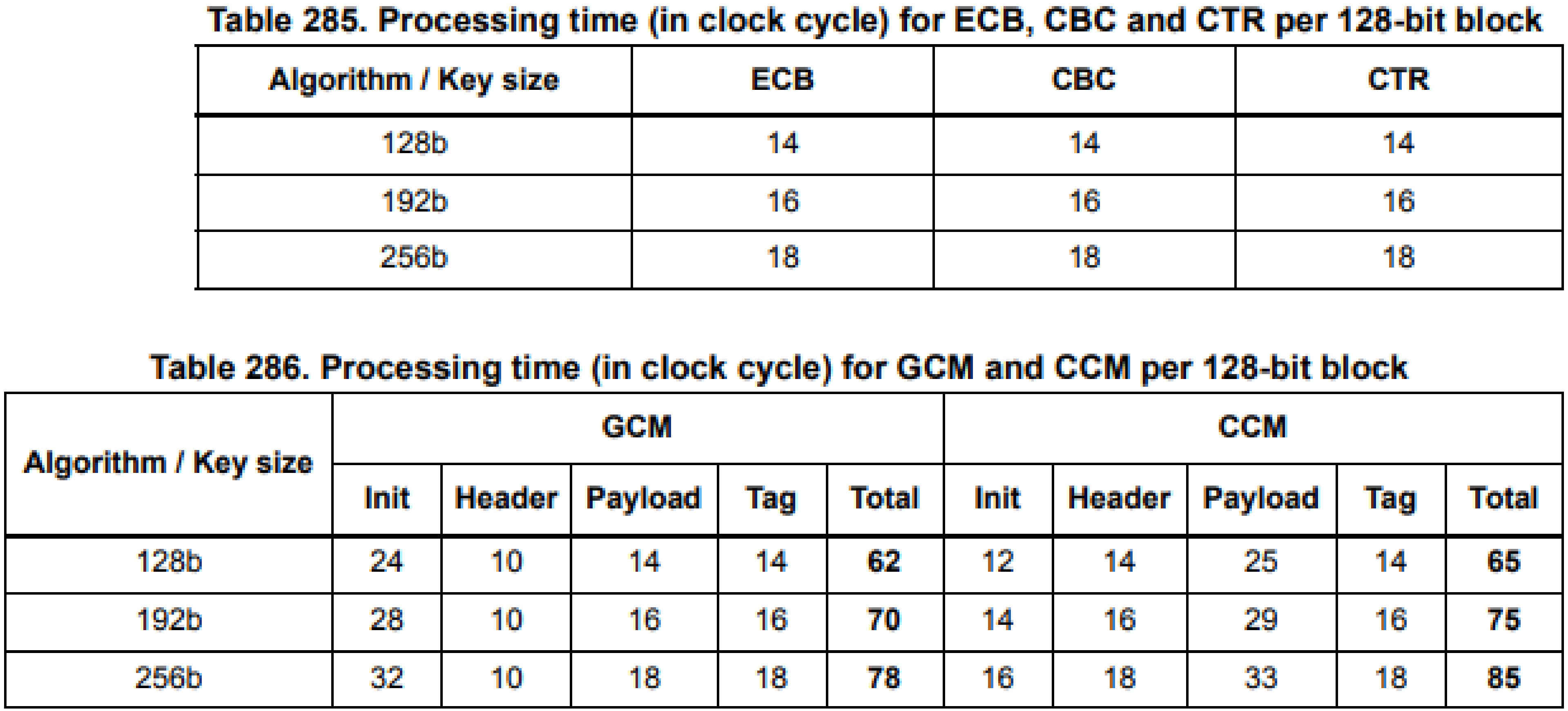


Table 285. Processing time (in clock cycle) for ECB, CBC and CTR per 128-bit block

Algorithm / Key size	ECB	CBC	CTR
128b	14	14	14
192b	16	16	16
256b	18	18	18

Table 286. Processing time (in clock cycle) for GCM and CCM per 128-bit block

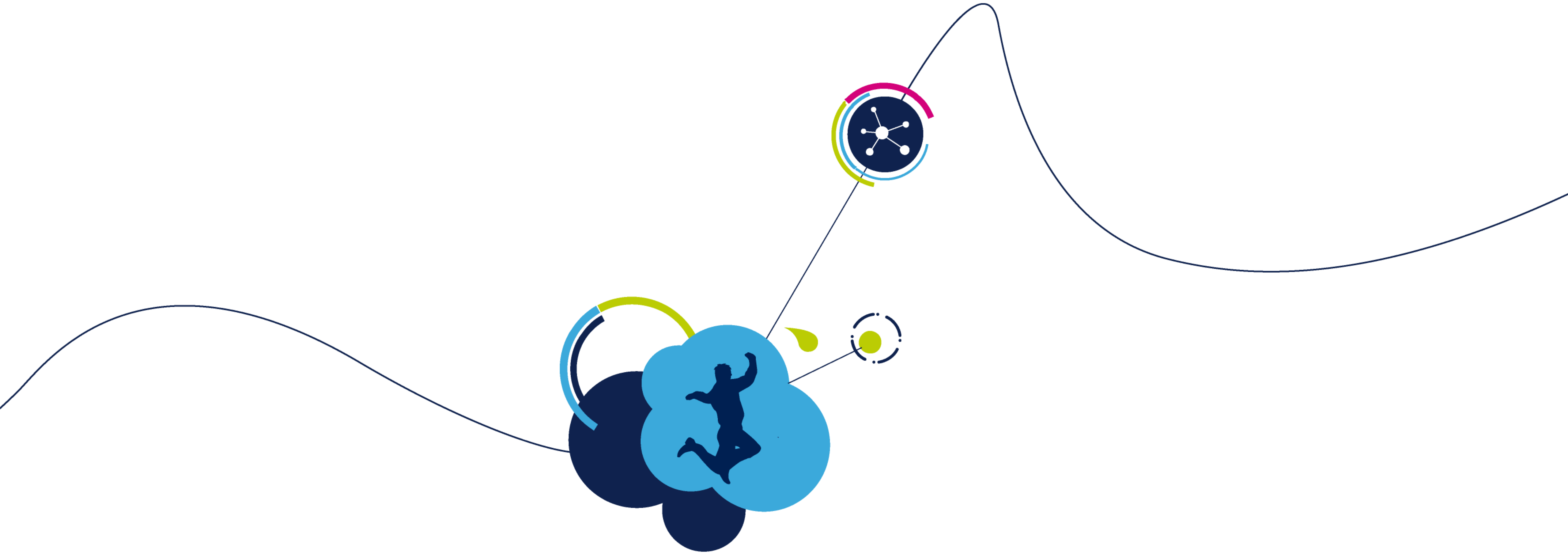
Algorithm / Key size	GCM					CCM				
	Init	Header	Payload	Tag	Total	Init	Header	Payload	Tag	Total
128b	24	10	14	14	**62**	12	14	25	14	**65**
192b	28	10	16	16	**70**	14	16	29	16	**75**
256b	32	10	18	18	**78**	16	18	33	18	**85**

Security Features by STM32 Series

STM32 Series	Security Features																
	96-Bit Unique ID	FLASH WRP	FLASH PCROP	FLASH RDP	Unique entry point	Secure mem/ HDP	MPU	Firewall	Trustzone	OTFDEC	Tamper	TRNG	CRYPT AES	HASH	PKA	Cryptolib	Arm Cortex®
STM32 **F0**	🟩	🟩		🟩							🟩					🟩	**M0**
STM32 **F1**	🟩	🟩					🟩				🟩					🟩	**M3**
STM32 **F2**	🟩	🟩		🟩			🟩				🟩	🟩	▨	▨		🟩	**M3**
STM32 **F3**	🟩	🟩		🟩			🟩				🟩					🟩	**M4**
STM32 **F4**	🟩	🟩	▨	🟩			🟩				🟩	🟩	▨	▨		🟩	**M4**
STM32 **F7**	🟩	🟩	▨	🟩			🟩				🟩	🟩	▨	▨		🟩	**M7**
STM32 **L0**	🟩	🟩	🟩	🟩			🟩	🟩			🟩	▨	▨			🟩	**M0+**
STM32 **L1**	🟩	🟩	🟩	🟩			🟩				🟩		▨			🟩	**M3**
STM32 **L4**	🟩	🟩	🟩	🟩			🟩	🟩			🟩	🟩	▨	▨		🟩	**M4**
STM32 **L5**	🟩	🟩		🟩	🟩	🟩	🟩		🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	**M33**
STM32 **H7**	🟩	🟩	🟩	🟩	▨	▨	🟩				🟩	🟩	▨	▨		🟩	**M7/M4**
STM32 **G0**	🟩	🟩	🟩	🟩	▨	▨	🟩				🟩	🟩	▨			🟩	**M0+**
STM32 **G4**	🟩	🟩	🟩	🟩	▨	▨	🟩				🟩	🟩	▨			🟩	**M4**
STM32 **WB**	🟩	🟩	🟩	🟩			🟩				🟩	🟩	🟩		🟩	🟩	**M4/M0+**

🟩 Available on all devices

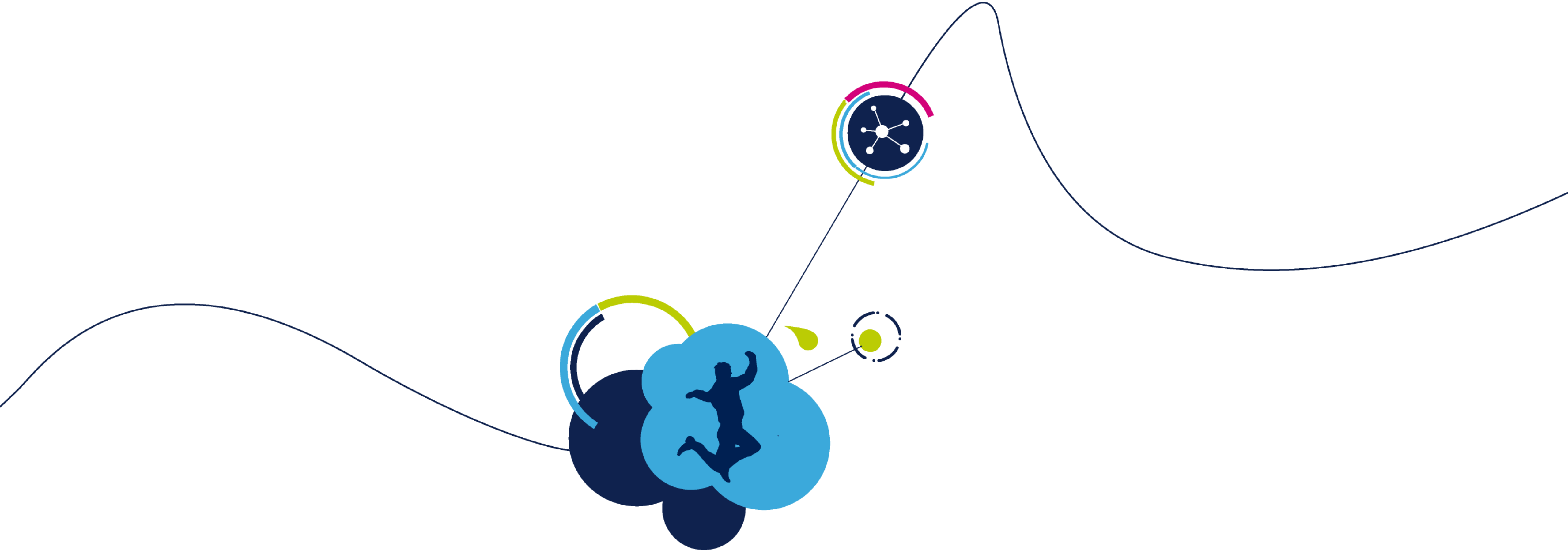
▨ Depends on device part number



AES Hands-on

Hands-on

- Create example of AES ECB encrypt/decrypt on a P-Nucleo-WB55
- We will select a NIST test vector from this document : https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-38a.pdf



PKA

Public Key Accelerator

- Computation of cryptographic public key primitives related to
    - RSA
    - Diffie-Hellmann
    - ECC (Elliptic Curve Cryptography) over GF(p) (Galois fields).

- PKA can be used to accelerate a number of public key cryptographic functions :
    - RSA encryption and decryption
    - RSA key finalization
    - CRT (Chinese Remainder Theorem)-RSA decryption
    - DSA and ECDSA signature generation and verification
    - DH and ECDH key agreement

PKA

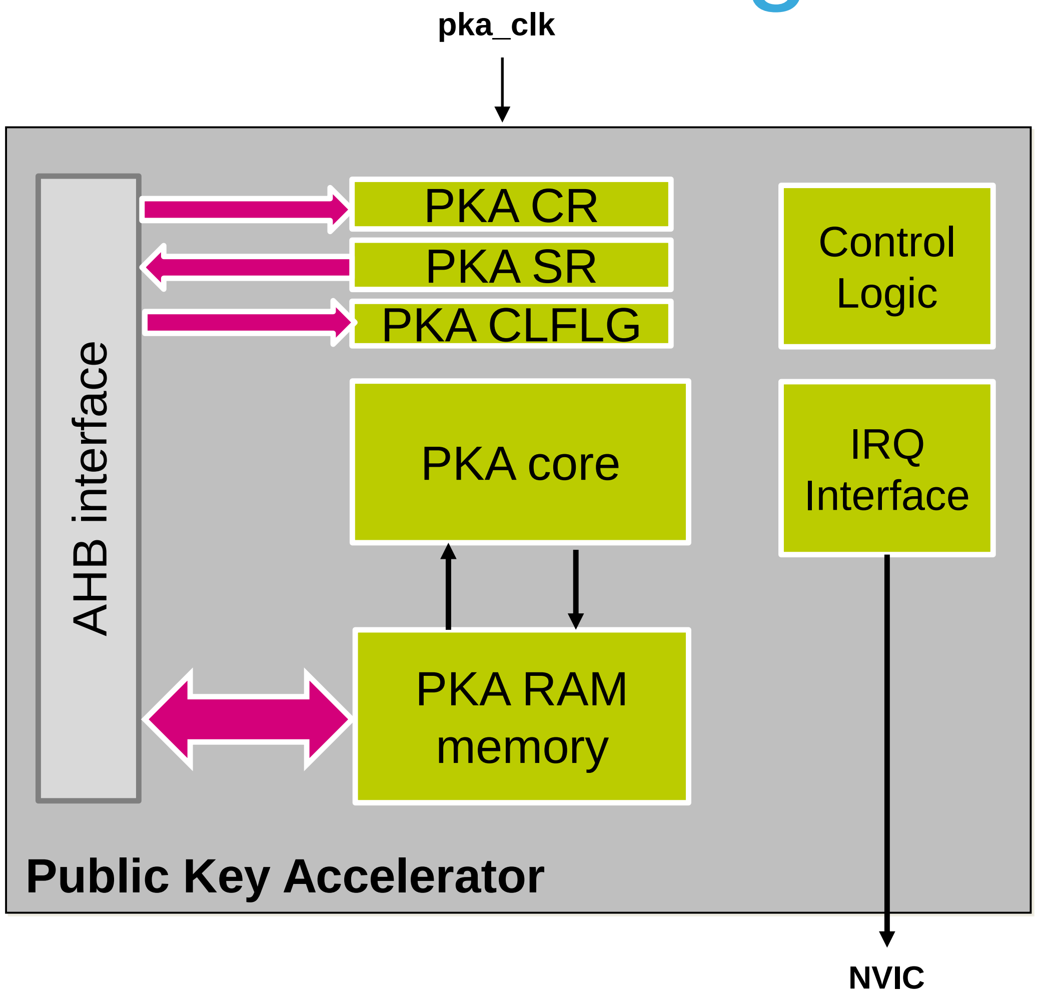
- Capability
  - RSA/DH – 3136 bits
  - ECC – 640 bits

- Interface :
  - registers
  - a memory of 3576 bytes (894 words of 32 bits) called PKA RAM. Access is done though the PKA AHB interface

- Available on :
  - STM32WB55
  - STM32L5

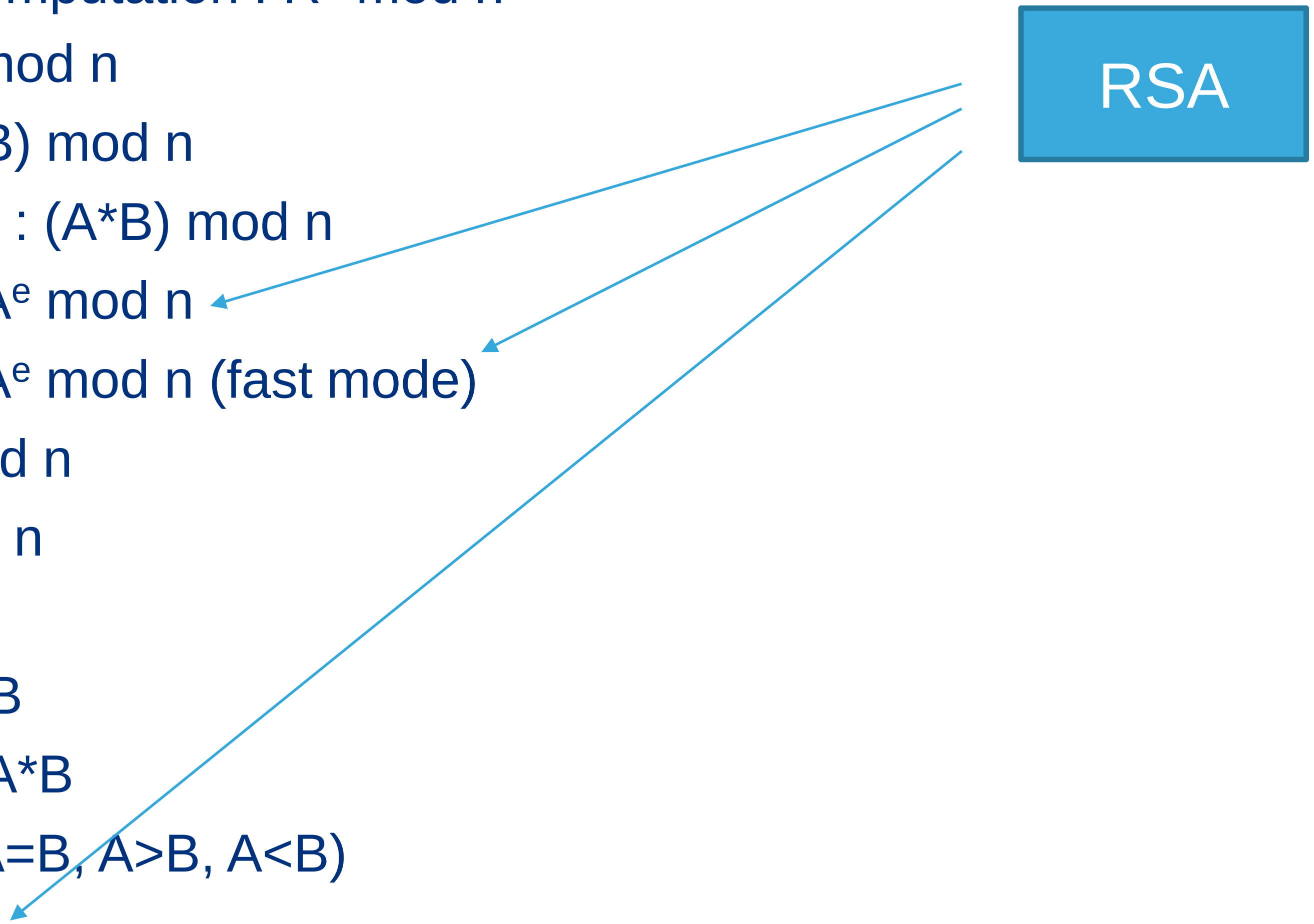
PKA Block Diagram

- PKA RAM : 3576 bytes (894 words of 32 bits)
- 3 individual maskable interrupt :
  - ADDRERR : access to unmapped address
  - RAMERR : PKA RAM access while PKA operation is in progress
  - PROCEND :PKA end of operation



PKA

- Provide arithmetic functions:
  - Montgomery parameter computation : R^2 mod n
  - Modular addition : (A+B) mod n
  - Modular subtraction : (A-B) mod n
  - Montgomery multiplication : (A*B) mod n
  - Modular exponentiation : A^e mod n
  - Modular exponentiation : A^e mod n (fast mode)
  - Modular inversion : A^{-1} mod n
  - Modular reduction : A mod n
  - Arithmetic addition : A+B
  - Arithmetic subtraction : A-B
  - Arithmetic multiplication : A*B
  - Arithmetic comparison : (A=B, A>B, A<B)
  - RSA CRT exponentiation



PKA

- prime field (Fp) elliptic curve functions:
  - Point on elliptic curve Fp check
  - ECC scalar multiplication kP
  - ECC scalar multiplication kP (fast mode)
  - ECDSA sign
  - ECDSA verification

ECDSA signature generation

ECDSA signature check

RSA key format

- RSA key pair is composed :
  - E: public exponent
  - N: modulus
  - D: private exponent
  - P: prime 1
  - Q: prime 2

Public Key

Private Key

Needed for key creation but not used for encryption or decryption… Anyway should be secret.

- Precomputed values to optimize decryption / signature with CRT
  - d_P= D mod (P -1) = exponent1
  - d_Q= D mod (Q -1) = exponent2
  - q_{inv}= D^{-1} mod P = coefficient

RSA creation

- First choose 2 prime number P and Q, compute N=Q*P

P =5, Q=11 so N=55

- Then chose E prime number which should have no prime factor common with (P-1)*(Q-1)

(P-1)*(Q-1) = (5-1) * (11-1) = 40 = 2*2*2*5 so E could be 7

- Public key will be : E = 7 and N = 55
- Now chose a number D which respect this rule : E*D mod ((P-1) * (Q-1)) = 1

7*D mod 40 = 1… D=23 is a good candidate.

- Private key will be : D = 23 and N = 55

How to encrypt a number M ?

$C = M^E$ modulo N

How to decrypt a number M ?

$M = C^D$ modulo N

RSA

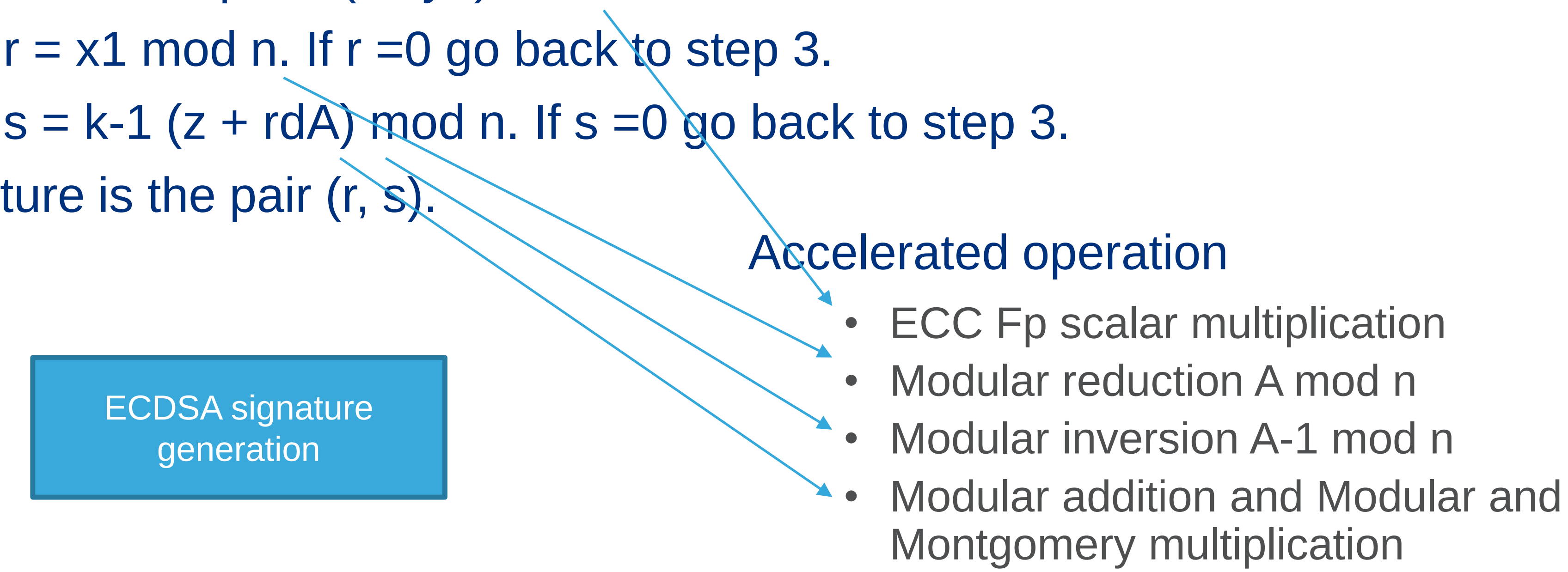
- RSA encryption /decryption :
  - $C = M^E$ modulo N / $M = C^D$ modulo N
    - Modular exponentiation A^e mod n
    - Modular exponentiation A^e mod n (fast mode)

- RSA decryption accelerated thanks CRT https://www.di-mgt.com.au/crt_rsa.html
  - $m1 = C^{dP}$ mod p and $m2 = C^{dQ}$ mod p
  - $h = q_{inv} .(m1 - m2)$ mod p
  - $M = m2 + h.q$

life.augmented

ECDSA signature generation

- Algorithm to sign a message m using a private key integer d
  - 1.Calculate e = HASH(m), where HASH is a cryptographic hash function.
  - 2. Let z be the Ln leftmost bits of e, where Ln is the bit length of the group order n.
  - 3. Select a cryptographically secure random integer k where 0 < k < n.
  - 4. Calculate the curve point (x1,y1) = k x G.
  - 5. Calculate r = x1 mod n. If r =0 go back to step 3.
  - 6. Calculate s = k-1 (z + rdA) mod n. If s =0 go back to step 3.
  - 7. The signature is the pair (r, s).



Accelerated operation

- ECC Fp scalar multiplication
- Modular reduction A mod n
- Modular inversion A-1 mod n
- Modular addition and Modular and Montgomery multiplication

ECDSA signature generation

PKA

- TIPS:
  - Take care about input argument in the PKA RAM
    - ROS (RSA Operand Size): data size is (rsa_size/32+1) words, with rsa_size equal to the chosen modulus length. For example, when computing RSA with an operand size of 1024 bits, ROS is equal to 33 words, or 1056 bits.
    - EOS (ECC Operand Size): data size is (ecc_size/32+1) words, with ecc_size equal to the chosen prime modulus length. For example, when computing ECC with an operand size of 192 bits, EOS is equal to 7 words, or 224 bits.
    - When elements are written as input in the memory of the PKA, an additional word with all bits equal to zero has to be added.
  - CAUTION : remember to clean PKA RAM when operation are finished.

Performance

Extract from STM32WB55 RM

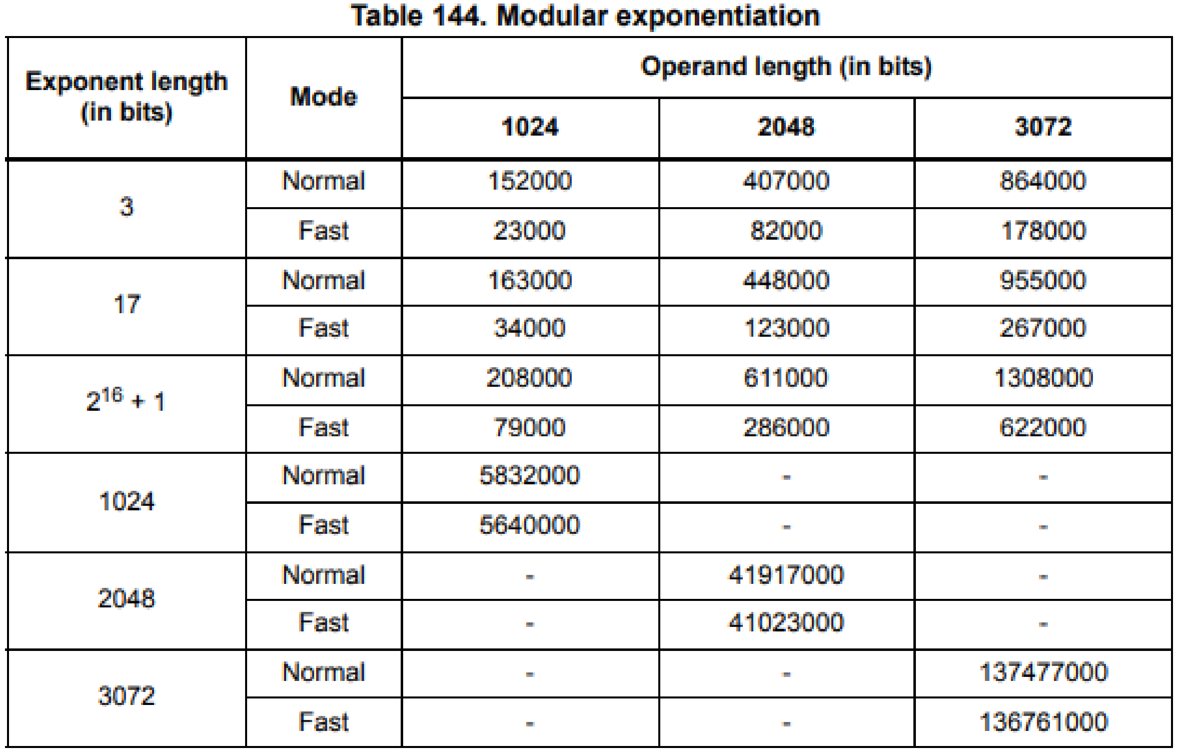


Table 144. Modular exponentiation

Exponent length (in bits)	Mode	Operand length (in bits)		
		1024	2048	3072
3	Normal	152000	407000	864000
	Fast	23000	82000	178000
17	Normal	163000	448000	955000
	Fast	34000	123000	267000
$2^{16} + 1$	Normal	208000	611000	1308000
	Fast	79000	286000	622000
1024	Normal	5832000	-	-
	Fast	5640000	-	-
2048	Normal	-	41917000	-
	Fast	-	41023000	-
3072	Normal	-	-	137477000
	Fast	-	-	136761000

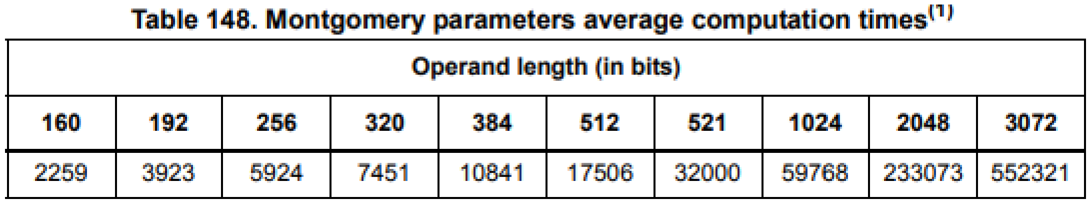


Table 148. Montgomery parameters average computation times[1]

Operand length (in bits)									
160	192	256	320	384	512	521	1024	2048	3072
2259	3923	5924	7451	10841	17506	32000	59768	233073	552321

Performance

Extract from STM32WB55 RM

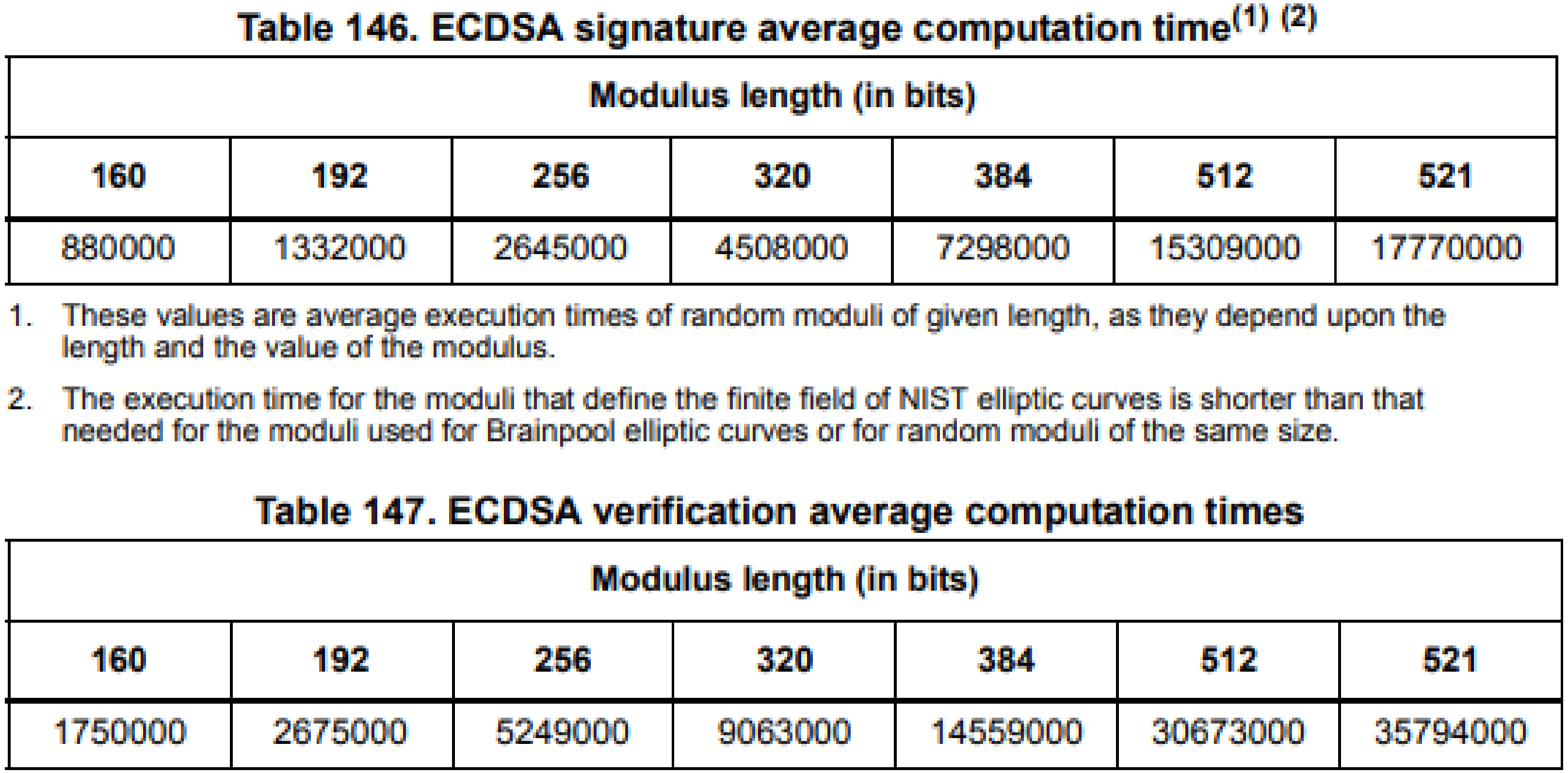


Table 146. ECDSA signature average computation time[1] [2]

Modulus length (in bits)						
160	**192**	**256**	**320**	**384**	**512**	**521**
880000	1332000	2645000	4508000	7298000	15309000	17770000

1. These values are average execution times of random moduli of given length, as they depend upon the length and the value of the modulus.

2. The execution time for the moduli that define the finite field of NIST elliptic curves is shorter than that needed for the moduli used for Brainpool elliptic curves or for random moduli of the same size.

Table 147. ECDSA verification average computation times

Modulus length (in bits)						
160	**192**	**256**	**320**	**384**	**512**	**521**
1750000	2675000	5249000	9063000	14559000	30673000	35794000

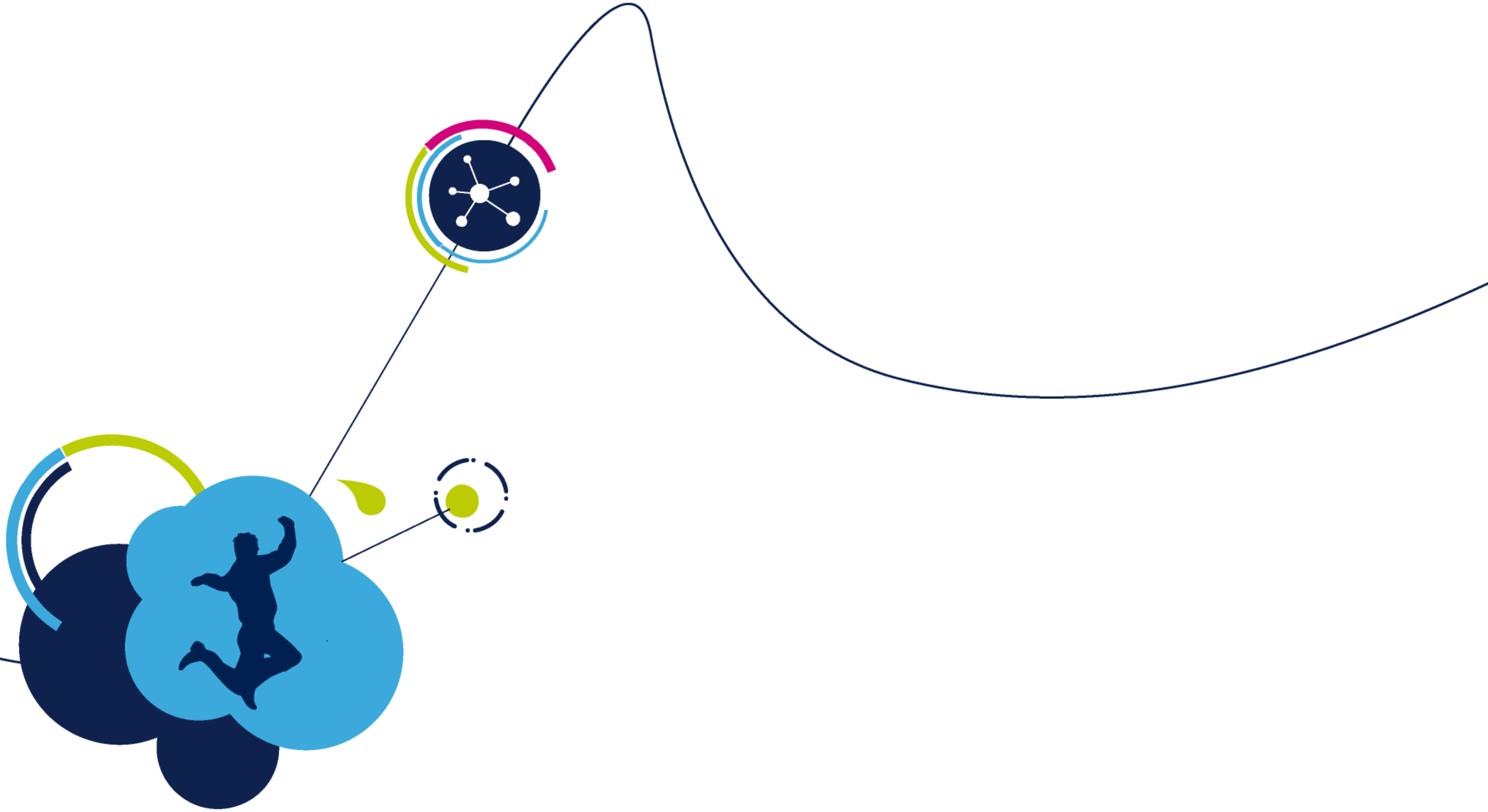
Security Features by STM32 Series

STM32 Series	Security Features																
	96-Bit Unique ID	FLASH WRP	FLASH PCROP	FLASH RDP	Unique entry point	Secure mem/ HDP	MPU	Firewall	Trustzone	OTFDEC	Tamper	TRNG	CRYPT AES	HASH	PKA	Cryptolib	Arm Cortex®
STM32 **F0**	●	●		●							●					●	**M0**
STM32 **F1**	●	●					●				●					●	**M3**
STM32 **F2**	●	●		●			●				●	●	◐	◐		●	**M3**
STM32 **F3**	●	●		●			●				●					●	**M4**
STM32 **F4**	●	●	◐	●			●				●	●	◐	◐		●	**M4**
STM32 **F7**	●	●	◐	●			●				●	●	◐	◐		●	**M7**
STM32 **L0**	●	●	●	●			●	●			●	◐	◐			●	**M0+**
STM32 **L1**	●	●	●	●			●				●		◐			●	**M3**
STM32 **L4**	●	●	●	●			●	●			●	●	◐	◐		●	**M4**
STM32 **L5**	●	●		●	●	●	●		●	●	●	●	●	●	●	●	**M33**
STM32 **H7**	●	●	●	●	◐	◐	●				●	●	◐	◐		●	**M7/M4**
STM32 **G0**	●	●	●	●	◐	◐	●				●	●	◐			●	**M0+**
STM32 **G4**	●	●	●	●	◐	◐	●				●	●	◐			●	**M4**
STM32 **WB**	●	●	●	●			●				●	●	●		●	●	**M4/M0+**

● Available on all devices

◐ Depends on device part number

life.augmented

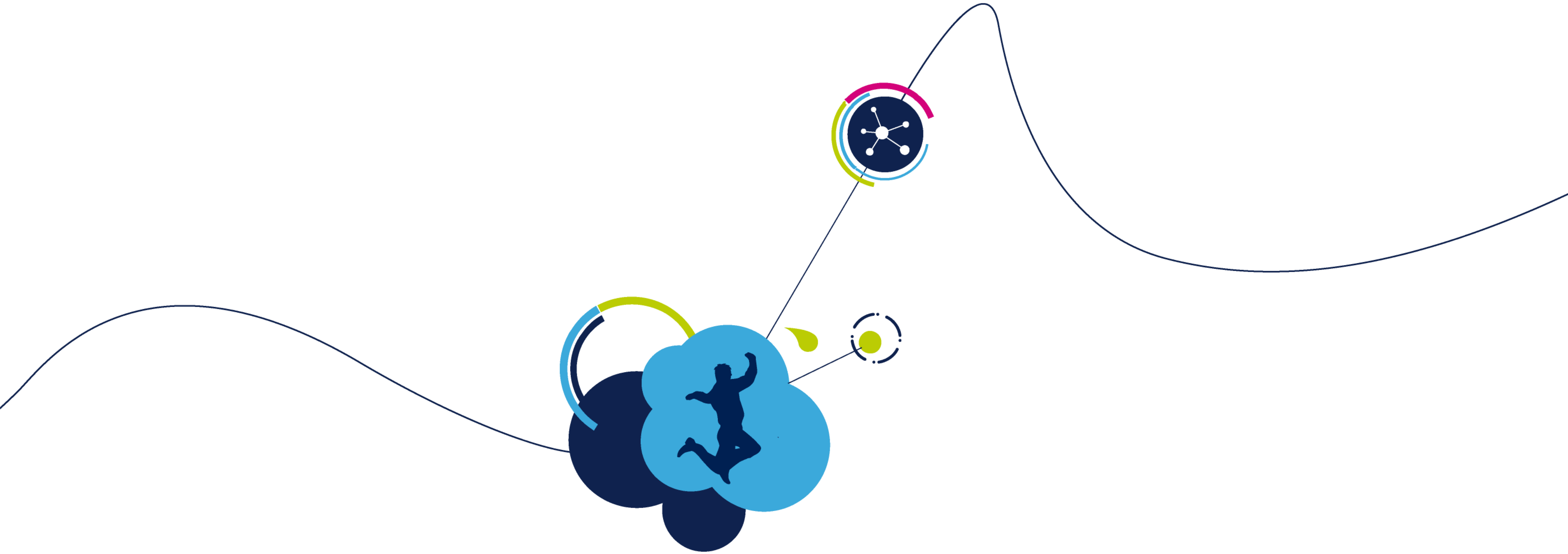


PKA Hands-on

life.augmented

Hands-on

- Generate a RSA encryption with openssl then do it on P-Nucleo-WB55



HASH

STM32 Hash Processor

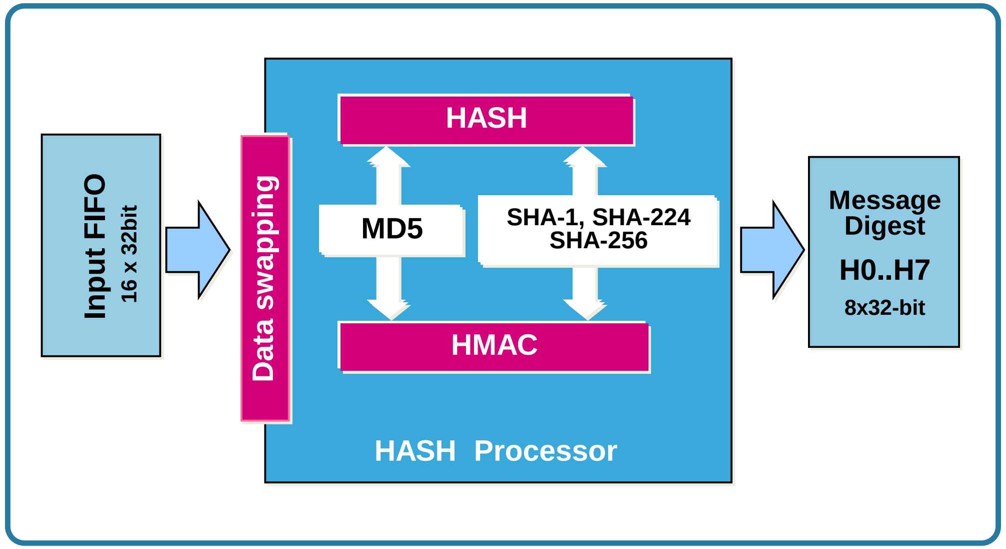
- compliant implementation of the secure hash algorithm
  - SHA-1, SHA-224, SHA-256 (FIPS PUB 180-4)
  - MD5 (IETF RFC 1321)
  - HMAC (IETF RFC 2104 and FIPS PUB 198-1)
- Automatic padding to complete the input bit string to fit digest minimum block size of 512 bits (accordingly to Federal Information Processing Standards PUB 180-1 and PUB 180-2)

STM32 Hash Processor

- 2 maskable interrupt :

  Digest calculation completion

  Data input buffer ready



- Available on :
  - STM32F214-217
  - STM32F415-417-437-439-479
  - STM32F750-756-777-778-779
  - STM32H750-753-755-757
  - STM32L4A6-4S5-4S7-4S9

STM32 Hash Processor

- Performance (extract from the STM32H750 RM)

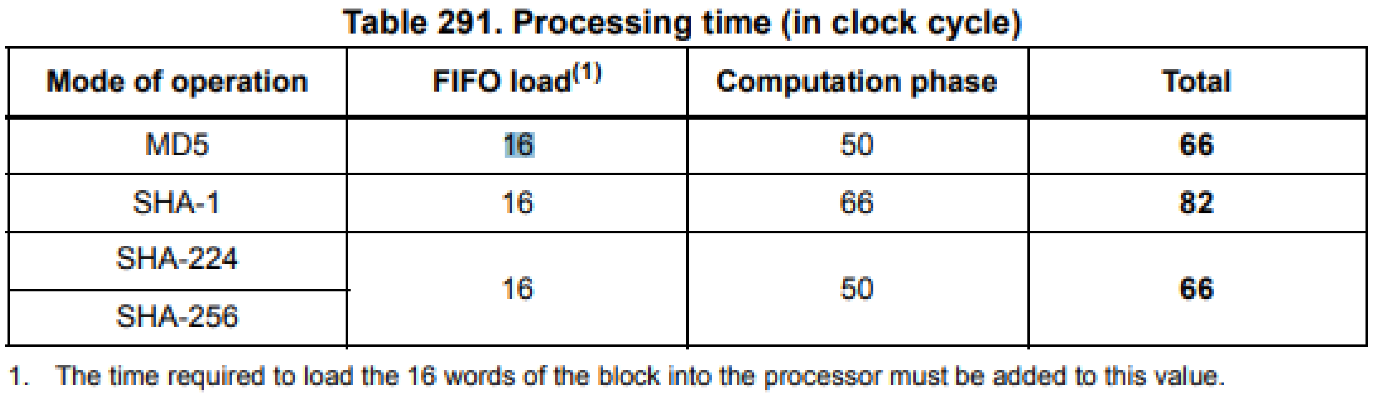


Table 291. Processing time (in clock cycle)

Mode of operation	FIFO load[1]	Computation phase	Total
MD5	16	50	**66**
SHA-1	16	66	**82**
SHA-224	16	50	**66**
SHA-256			

[1] The time required to load the 16 words of the block into the processor must be added to this value.

Security Features by STM32 Series

STM32 Series	Security Features																
	96-Bit Unique ID	FLASH WRP	FLASH PCROP	FLASH RDP	Unique entry point	Secure mem/ HDP	MPU	Firewall	Trustzone	OTFDEC	Tamper	TRNG	CRYPT AES	HASH	PKA	Cryptolib	Arm Cortex®
STM32 **F0**	🟩	🟩		🟩							🟩					🟩	**M0**
STM32 **F1**	🟩	🟩					🟩				🟩					🟩	**M3**
STM32 **F2**	🟩	🟩		🟩			🟩				🟩	🟩	▨	▨		🟩	**M3**
STM32 **F3**	🟩	🟩		🟩			🟩				🟩					🟩	**M4**
STM32 **F4**	🟩	🟩	▨	🟩			🟩				🟩	🟩	▨	▨		🟩	**M4**
STM32 **F7**	🟩	🟩	▨	🟩			🟩				🟩	🟩	▨	▨		🟩	**M7**
STM32 **L0**	🟩	🟩	🟩	🟩			🟩	🟩			🟩	▨	▨			🟩	**M0+**
STM32 **L1**	🟩	🟩	🟩	🟩			🟩				🟩		▨			🟩	**M3**
STM32 **L4**	🟩	🟩	🟩	🟩			🟩	🟩			🟩	🟩	▨	▨		🟩	**M4**
STM32 **L5**	🟩	🟩		🟩	🟩	🟩	🟩		🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	**M33**
STM32 **H7**	🟩	🟩	🟩	🟩	▨	▨	🟩				🟩	🟩	▨	▨		🟩	**M7/M4**
STM32 **G0**	🟩	🟩	🟩	🟩	▨	▨	🟩				🟩	🟩	▨			🟩	**M0+**
STM32 **G4**	🟩	🟩	🟩	🟩	▨	▨	🟩				🟩	🟩	▨			🟩	**M4**
STM32 **WB**	🟩	🟩	🟩	🟩			🟩				🟩	🟩	🟩		🟩	🟩	**M4/M0+**

🟩 Available on all devices

▨ Depends on device part number

life.augmented

Hands-on

- No nucleo or disco with hash…

CryptoLib on STM32

life.augmented

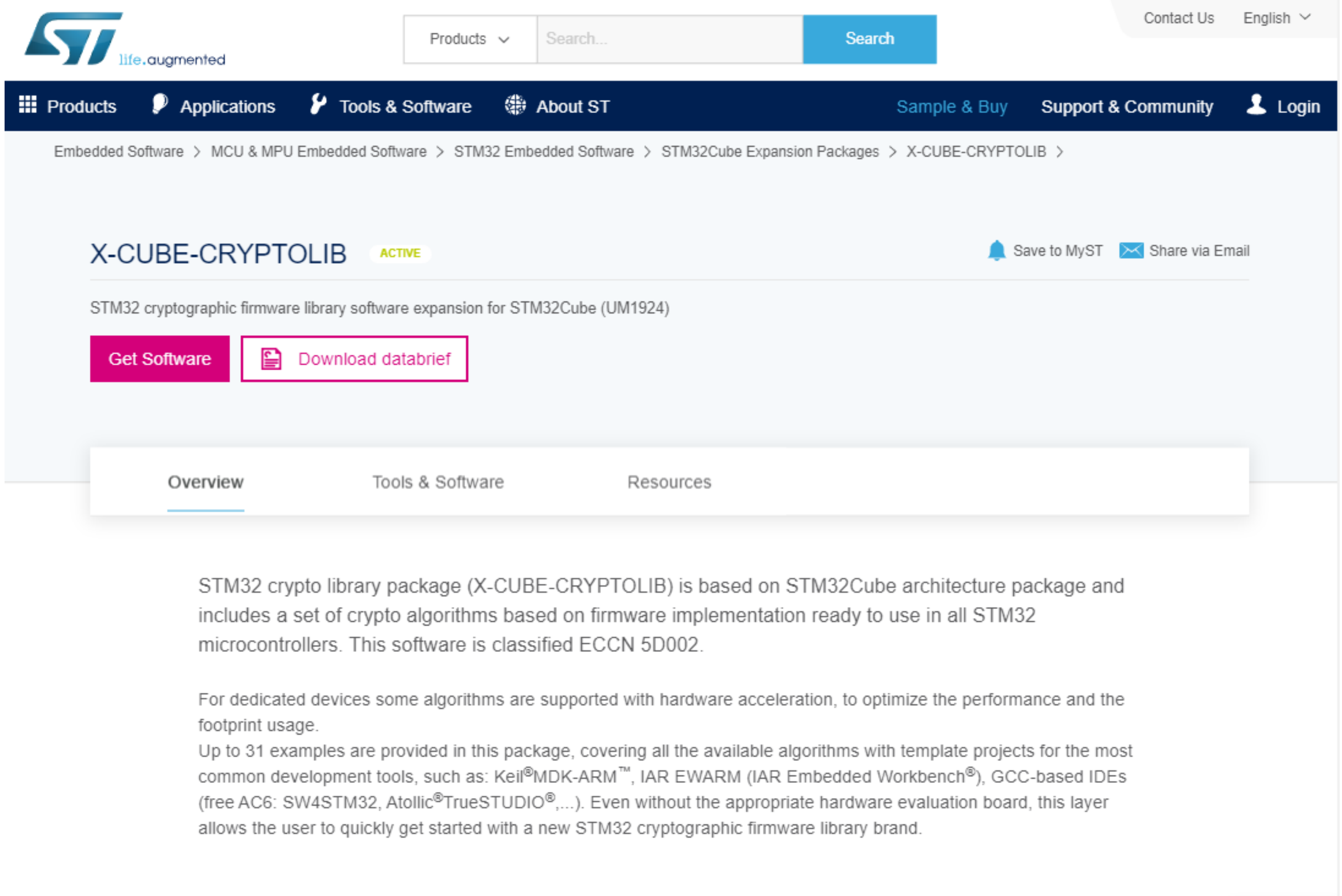
STM32 Crypto Library

CAVP FIPS Certified

- This unique library contains a software implementation of the cryptographic algorithms and also a hardware accelerators enhancement for some of them.
- Removes the burden of algorithm validation
- Allows OEMs to fasten their security certification process
- Includes all the major algorithms for encryption, hashing, message authentication, and digital signing

STM32 Crypto library

- https://www.st.com/en/embedded-software/x-cube-cryptolib.html



X-CUBE-CRYPTOLIB

STM32 cryptographic firmware library software expansion for STM32Cube (UM1924)

STM32 crypto library package (X-CUBE-CRYPTOLIB) is based on STM32Cube architecture package and includes a set of crypto algorithms based on firmware implementation ready to use in all STM32 microcontrollers. This software is classified ECCN 5D002.

For dedicated devices some algorithms are supported with hardware acceleration, to optimize the performance and the footprint usage.

Up to 31 examples are provided in this package, covering all the available algorithms with template projects for the most common development tools, such as: Keil®MDK-ARM™, IAR EWARM (IAR Embedded Workbench®), GCC-based IDEs (free AC6: SW4STM32, Atollic®TrueSTUDIO®,...). Even without the appropriate hardware evaluation board, this layer allows the user to quickly get started with a new STM32 cryptographic firmware library brand.

STM32 Crypto Library

Software, Hybrid / Hardware

- STM32 Firmware Crypto Library V3.1.0
  - All algorithms are based on firmware implementation without using any hardware acceleration
- STM32 Hardware Acceleration Crypto Library V3.1.0
  - Support the algorithms based on firmware implementation with hardware acceleration (Hybrid)
- The STM32 Crypto Libraries are distributed by ST as an object code library, accessed by the user application through an API

STM32 Crypto

- Hardware acceleration could rely on Random number generator (RNG), Crypto accelerator and AES Hardware Accelerator
- The library is compiled for Cortex® M0, M0+, M3, M4, and M7 cores. Note that the library is compiler-dependent (IAR, Keil®, GCC) and is compiled with two optimization levels (High size, High speed).

- STM32CryptographicV3.1.1_CMx_C_O where:
  - x: the CMx core class (CM0, CM0PLUS, CM3, CM4 or CM7)
  - C: compiler (IAR, Keil®, GCC)
  - O: specify the compiler optimization

  In case of GCC IDE:
    - <empty>: high size optimization
    - ot: high speed optimization

  In case of IAR IDE:
    - <empty>: high size optimization
    - ot: high speed optimization
    - nsc: the option No Size constraints is enabled

  In Case of Keil® IDE:
    - <empty>: high size optimization
    - ot: high speed optimization
    - slsm: the option Split Load and Store Multiple is enabled
    - o1elfspf: the option One ELF Section per Function is enabled

STM32 Crypto algorithm

- DES, TripleDES. Supported modes :
  - ECB
  - CBC
- AES-128, AES-192, AES-256 bits. Supported modes :
  - ECB
  - CBC
  - CTR
  - CFB
  - OFB
  - CCM
  - GCM
  - CMAC
  - KEY WRAP
  - XTS

STM32 Crypto algorithm

- HASH functions with HMAC support:
  - MD5
  - SHA-1
  - SHA-224
  - SHA-256
  - SHA-384
  - SHA-512
- ARC4
- CHACHA20-POLY1305
- Random engine based on DRBG-AES-128

STM32 Crypto algorithm

- RSA signature functions with PKCS#1v1.5
- RSA encryption/decryption functions with PKCS#1v1.5
- ECC (Elliptic Curve Cryptography):
    - Key generation
    - Scalar multiplication (the base for ECDH)
    - ECDSA
- ED25519
- Curve25519

STM32 Crypto

- TIPS:
  - The CRC peripheral is used by the STM32 crypto firmware library. When using Cryptolib APIs CRC shall be activated and insure CRC INIT register to 0xFFFFFFFF. Otherwise API results will not be valid.
  - FPU libraries version are available and located on AccHw_Crypto
  - Performance and memory footprint could be find documentation folder in the package
  - Please many tips available in the **UM1924**, FAQ section.

Security Features by STM32 Series

STM32 Series	Security Features																
	96-Bit Unique ID	FLASH WRP	FLASH PCROP	FLASH RDP	Unique entry point	Secure mem/ HDP	MPU	Firewall	Trustzone	OTFDEC	Tamper	TRNG	CRYPT AES	HASH	PKA	Cryptolib	Arm Cortex®
STM32 **F0**	🟩	🟩		🟩							🟩					🟩	**M0**
STM32 **F1**	🟩	🟩					🟩				🟩					🟩	**M3**
STM32 **F2**	🟩	🟩		🟩			🟩				🟩	🟩	▨	▨		🟩	**M3**
STM32 **F3**	🟩	🟩		🟩			🟩				🟩					🟩	**M4**
STM32 **F4**	🟩	🟩	▨	🟩			🟩				🟩	🟩	▨	▨		🟩	**M4**
STM32 **F7**	🟩	🟩	▨	🟩			🟩				🟩	🟩	▨	▨		🟩	**M7**
STM32 **L0**	🟩	🟩	🟩	🟩			🟩	🟩			🟩		▨			🟩	**M0+**
STM32 **L1**	🟩	🟩	🟩	🟩			🟩				🟩		▨			🟩	**M3**
STM32 **L4**	🟩	🟩	🟩	🟩			🟩	🟩			🟩	🟩	▨	▨		🟩	**M4**
STM32 **L5**	🟩	🟩		🟩	🟩	🟩	🟩		🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	**M33**
STM32 **H7**	🟩	🟩	🟩	🟩	▨	▨	🟩				🟩	🟩	▨	▨		🟩	**M7/M4**
STM32 **G0**	🟩	🟩	🟩	🟩	▨	▨	🟩				🟩	🟩	▨			🟩	**M0+**
STM32 **G4**	🟩	🟩	🟩	🟩	▨	▨	🟩				🟩	🟩	▨			🟩	**M4**
STM32 **WB**	🟩	🟩	🟩	🟩			🟩				🟩	🟩	🟩		🟩	🟩	**M4/M0+**

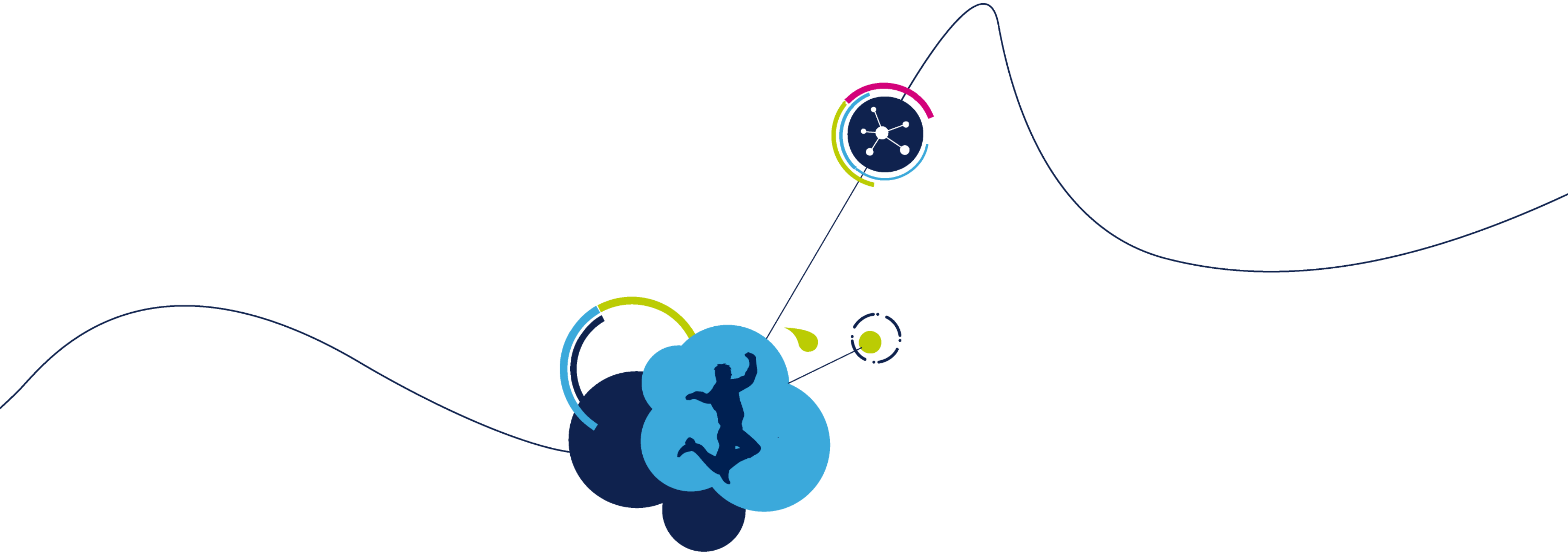
🟩 Available on all devices

▨ Depends on device part number

life.augmented

Reference

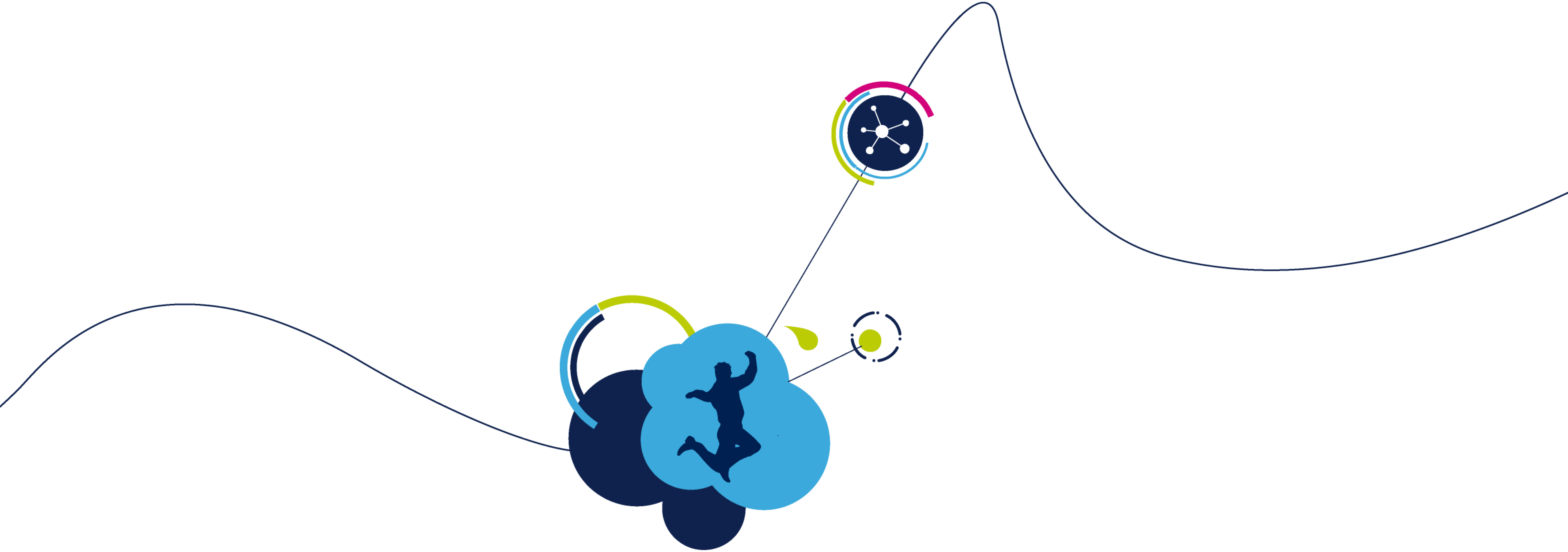
- **UM1924** STM32 crypto library
- Many code example inside the package.



Cryptolib Hands-on

Hands-on

- Launch an example from the package
- How to integrate cryptolib in a new project (cube IDE) ?
  - Create a project with Cube IDE
  - Add the library and the header file
  - Implement a basic code example



Conclusion

Conclusion

- We have describe and experience together all the security blocks available across STM32 families
  - Resources: STM32 Unique ID
  - Static Protection
    - WRP
    - PCROP
    - RDP
    - Unique entry boot
    - User Secure Mem / HDP
  - Dynamic protection :
    - MPU
    - FIREWALL
    - TrustZone
    - OTFDEC
    - Tamper
  - Crypto HW resources :
    - TRNG
    - CRYPT / AES
    - HASH
    - PKA
  - Crypto SW resources :
    - CryptoLib

STM32 isolation features comparison

Item	Pro’s	Con’s
Firewall	Code & data Protection Unique entry/exit point / dynamic	No interrupts allowed inside the protected area RDP2 required
MPU	Available on almost all STM32	**Impact on the User code** (unprivileged) Access control for CPU only
Secure mem HDP	Code & data isolation Area not accessible by user application	Static (not usable at runtime)
Trustzone	Code and data protection dynamic Enhanced Product Life Cycle Debug S/NS - IRQ handling (S ⇔ NS)	SW development model different

Security Features by STM32 Series

STM32 Series	Security Features																
	96-Bit Unique ID	FLASH WRP	FLASH PCROP	FLASH RDP	Unique entry point	Secure mem/ HDP	MPU	Firewall	Trustzone	OTFDEC	Tamper	TRNG	CRYPT AES	HASH	PKA	Cryptolib	Arm Cortex®
STM32 **F0**	🟩	🟩		🟩							🟩					🟩	**M0**
STM32 **F1**	🟩	🟩					🟩				🟩					🟩	**M3**
STM32 **F2**	🟩	🟩		🟩			🟩				🟩	🟩	▨	▨		🟩	**M3**
STM32 **F3**	🟩	🟩		🟩			🟩				🟩					🟩	**M4**
STM32 **F4**	🟩	🟩	▨	🟩			🟩				🟩	🟩	▨	▨		🟩	**M4**
STM32 **F7**	🟩	🟩	▨	🟩			🟩				🟩	🟩	▨	▨		🟩	**M7**
STM32 **L0**	🟩	🟩	🟩	🟩			🟩	🟩			🟩		▨			🟩	**M0+**
STM32 **L1**	🟩	🟩	🟩	🟩			🟩				🟩		▨			🟩	**M3**
STM32 **L4**	🟩	🟩	🟩	🟩			🟩	🟩			🟩	🟩	▨	▨		🟩	**M4**
STM32 **L5**	🟩	🟩		🟩	🟩	🟩	🟩		🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	**M33**
STM32 **H7**	🟩	🟩	🟩	🟩	▨	▨	🟩				🟩	🟩	▨	▨		🟩	**M7/M4**
STM32 **G0**	🟩	🟩	🟩	🟩	▨	▨	🟩				🟩	🟩	▨			🟩	**M0+**
STM32 **G4**	🟩	🟩	🟩	🟩	▨	▨	🟩				🟩	🟩	▨			🟩	**M4**
STM32 **WB**	🟩	🟩	🟩	🟩			🟩				🟩	🟩	🟩		🟩	🟩	**M4/M0+**

🟩 Available on all devices

▨ Depends on device part number

life.augmented

Next step

- The next step is to understand how to combine those security block to build your STM32 security with secure boot and secure firmware update.