STM32 Security Workshop

03 SBSFU presentation

Agenda

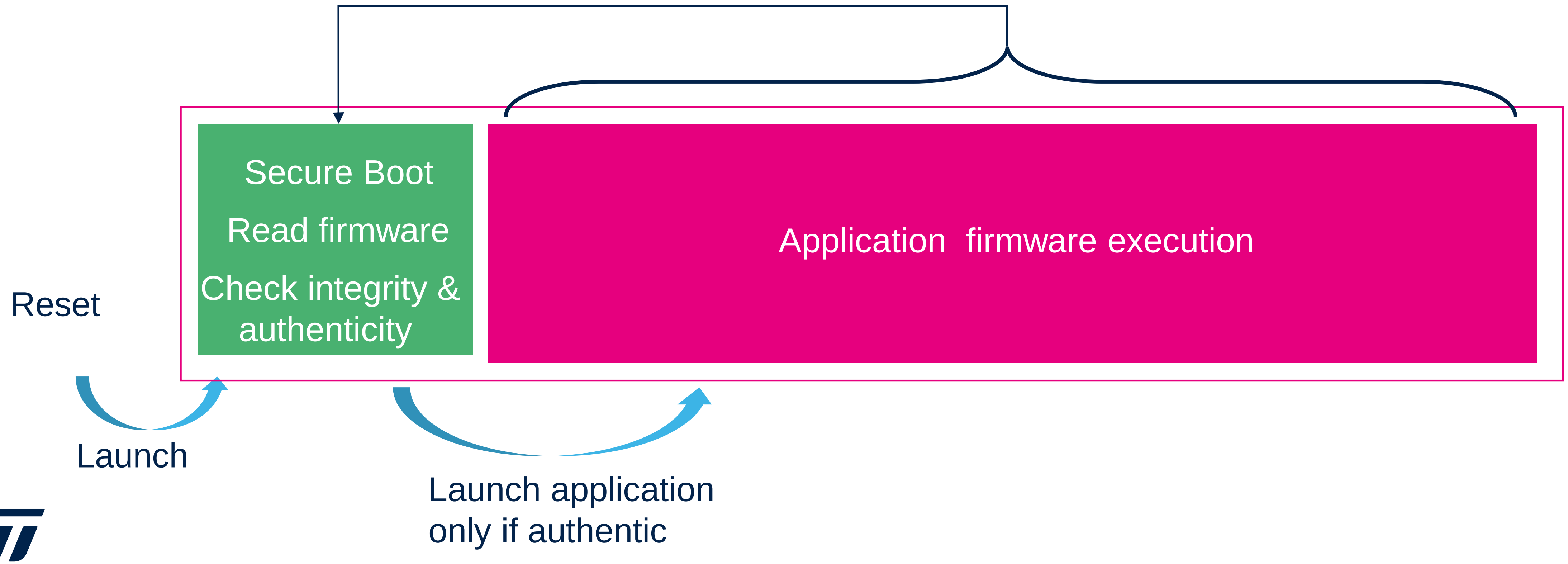
life.augmented

Agenda

life.augmented

Basic principle of a secure boot

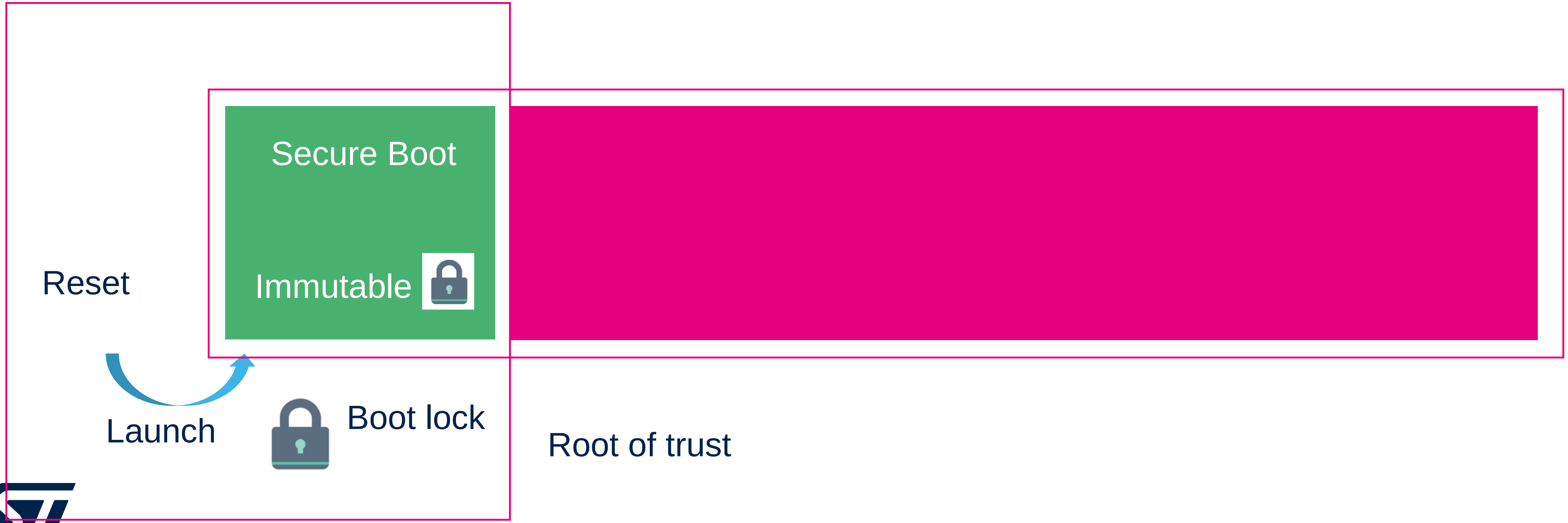
- A secure boot is a code executed after reset that verifies the application firmware is genuine before launching it or not!



What makes a secure boot secure ?

Secure Boot is the only possible code executed after reset

Secure Boot code is immutable (cannot be modified in any manner)



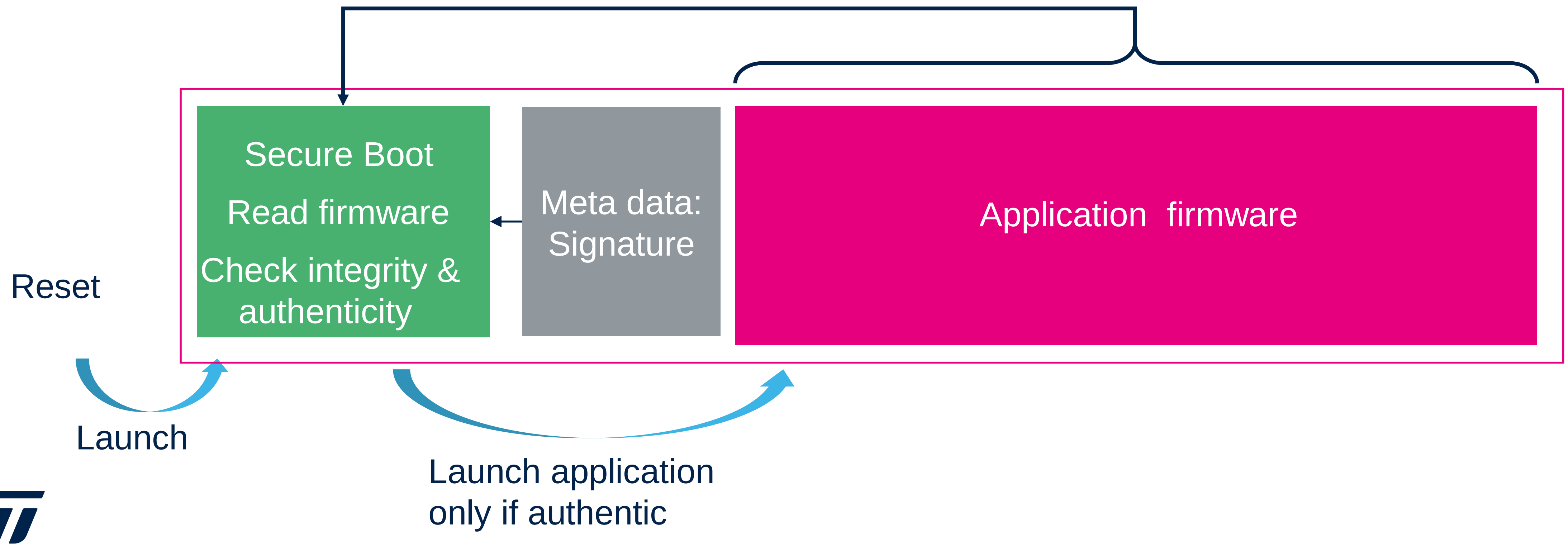
How integrity & Authenticity are checked ?

- Integrity is checked thanks to hash
  - Secure boot reads the whole application firmware to generate a hash
  - This hash needs to be compared to a reference
- Authenticity is checked thanks to a signature
  - Usually, signature is computed on a hash value
  - Signature is generated thank to a private key
  - Signature is checked thanks to associated public key

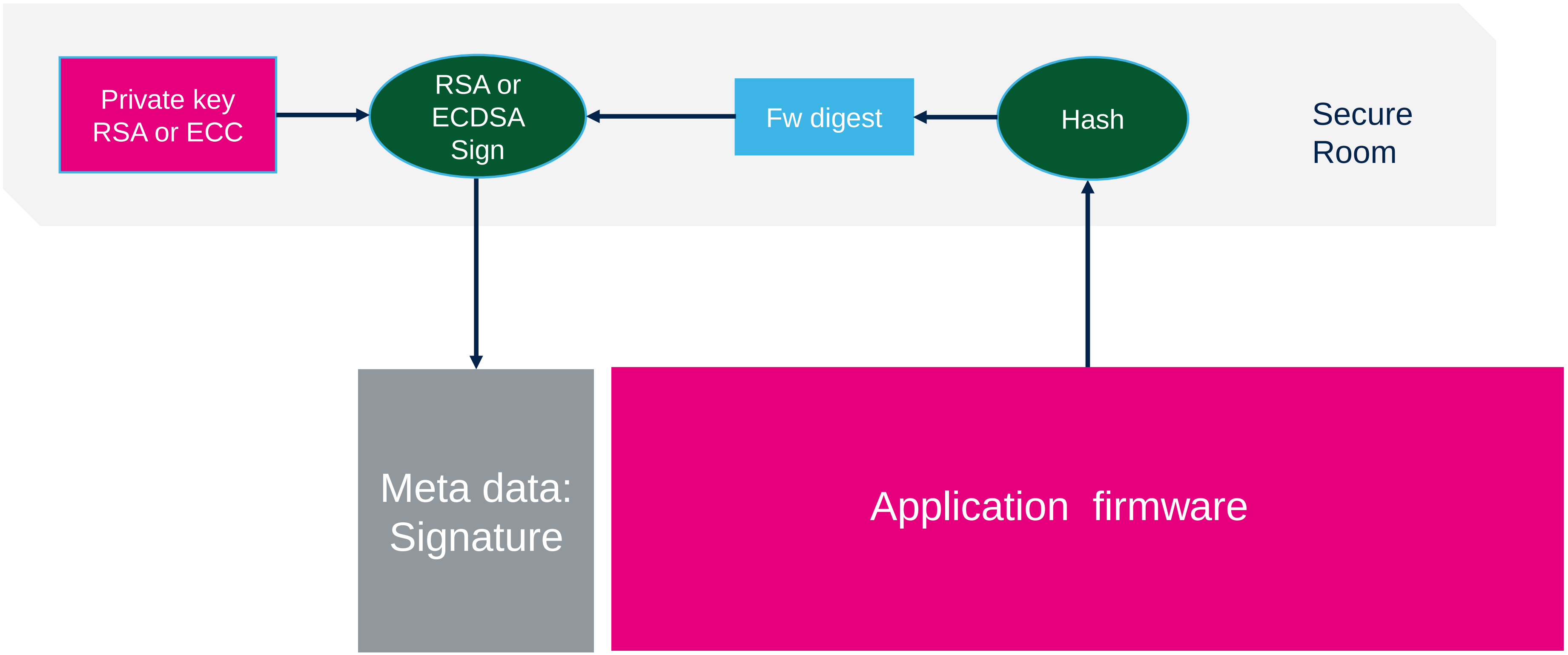
- Hash value and signature value need to be provided with the firmware !
- They are stored in a container called either meta data or header
- No need to encrypt the meta data thanks to signature mechanism.

Meta data

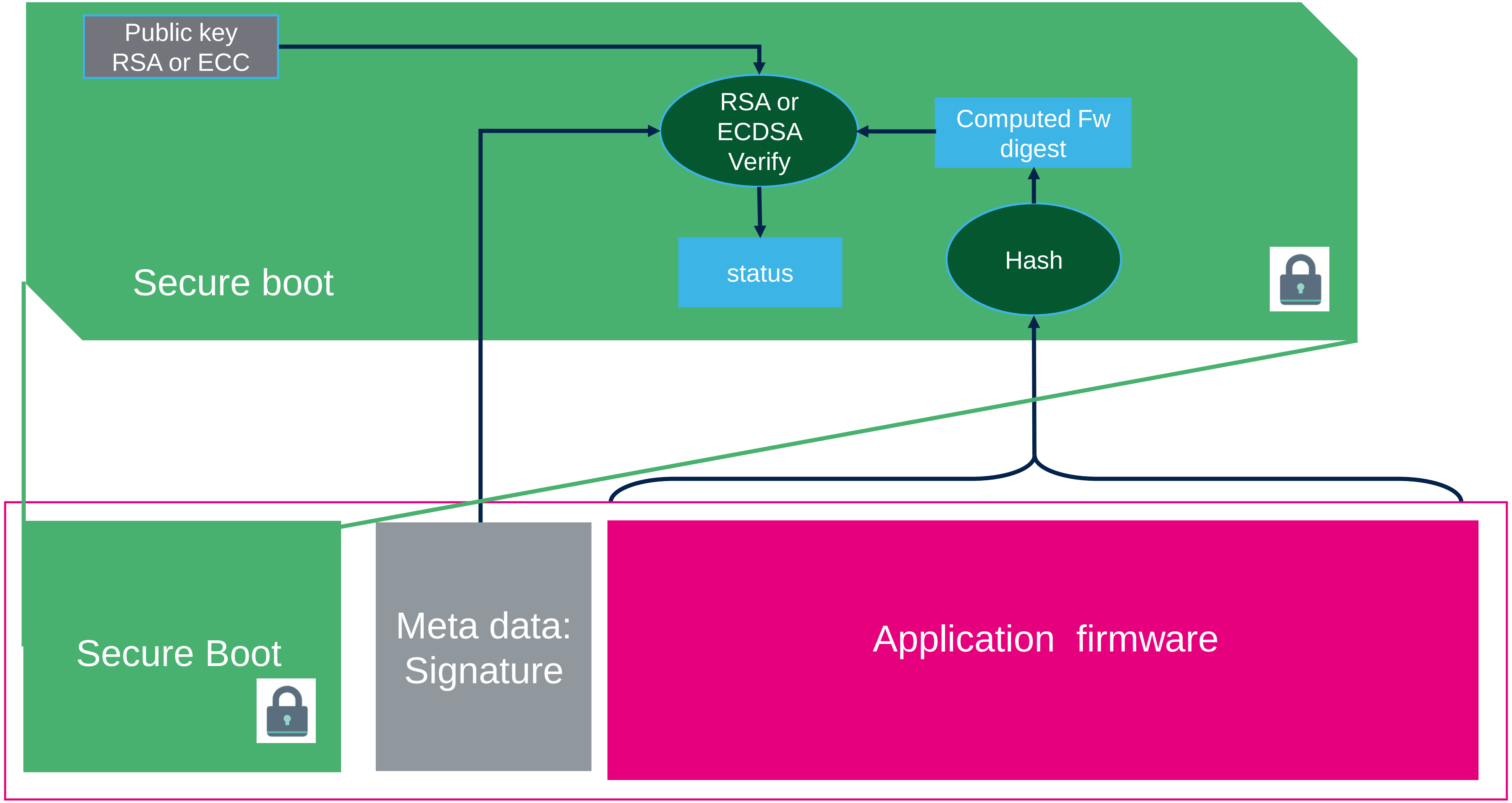
- Meta data is located besides the application firmware
- It provides necessary information to the secure boot to perform the checking



How meta data is built ?



How secure boot verifies the signature ?



Agenda

- 1 Secure boot principle
- 2 Secure firmware update principle
- 3 SBSFU Package
- 4 SBSFU and protection
- 5 SBSFU cryptography
- 6 SBSFU Architecture
- 7 SBSFU Advanced features
- 8 Conclusion

From secure boot to secure firmware update

- We have defined what is a secure boot
- How it checks the application firmware thanks to the meta data
- Now let’s see how we can update this application firmware in a secure way, that is secure firmware update

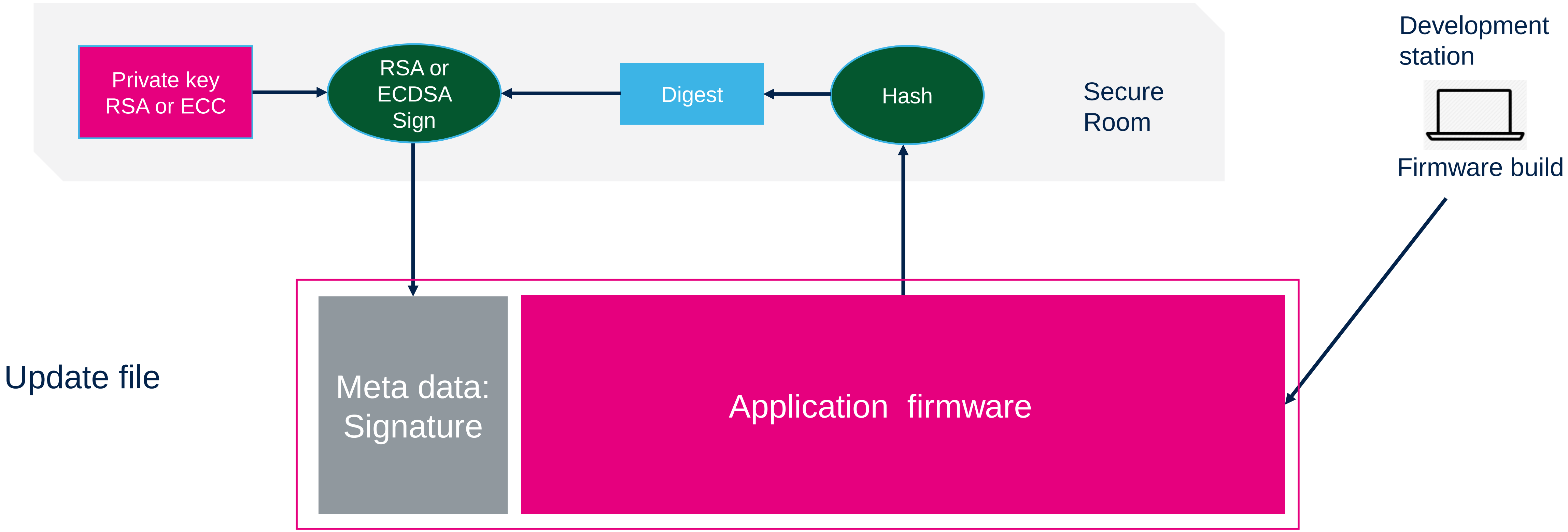
life.augmented

Secure firmware update principle

- An update is transmitted to the target
- This update contains an application firmware and associated meta data.
- With the meta data, the secure boot can check the application firmware integrity and authenticity
- If check is ok, the secure boot installs the new firmware

life.augmented

First step : create an update file



Second step : transmit update file to target

- 2 possible ways:
  - 1- File sent remotely. Case of IOT with connection to a server
  - 2- File transmitted thanks to a local connection (UART, USB, SDCard, I2C, SPI)
- In all cases the update file needs to be written on target flash with a loader
- The loader can be
  - 1- part of the secure boot
  - 2- a standalone application
  - 3- part of the application firmware

Third step : installation of the update

- Secure boot can now access the update file
- Secure boot will check integrity and authenticity thanks to meta data
- If check is ok the update file can be installed

life.augmented

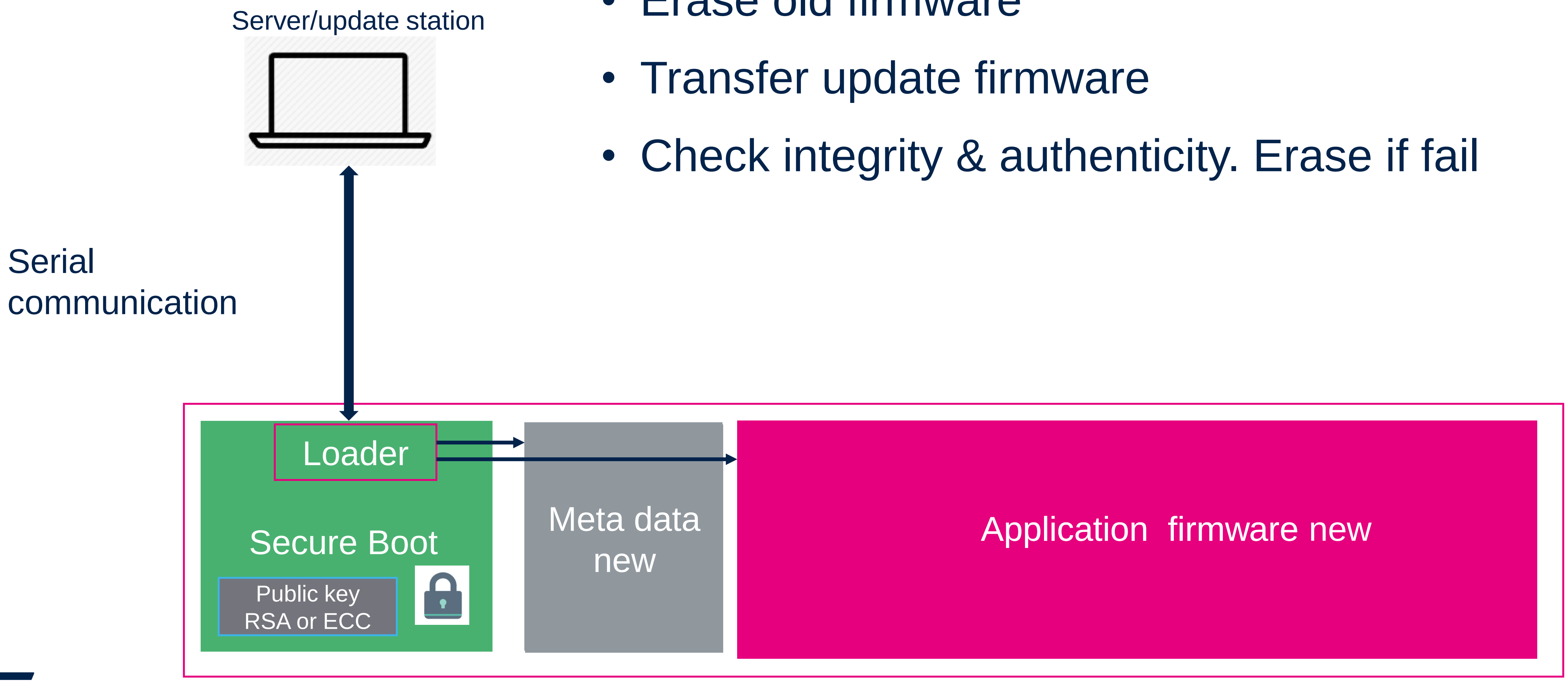
How new firmware can replace old one

- Local transfer
  - Can be handled from secure boot, standalone application or even application itself
  - If loader is located in secure boot or standalone application, it can first erase application and then load the new one
- Remote transfer
  - IOT common case : application transfers new firmware from remote server.
  - New firmware needs to be stored locally while current firmware is still running
  - If application has to manage the update, 2 slots are required:
    - a slot storing the active firmware
    - A slot storing the downloaded firmware

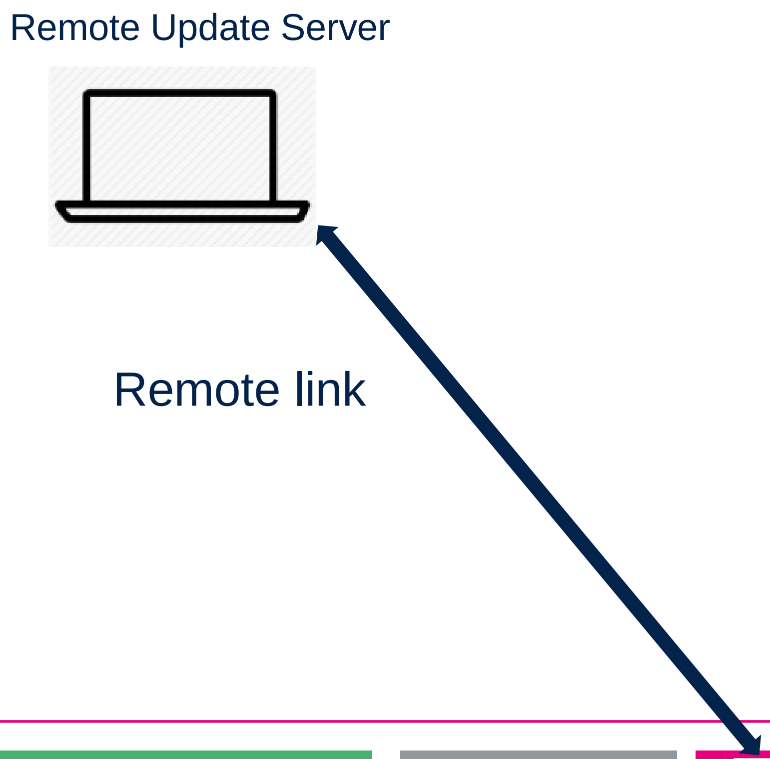
life.augmented

Local update example

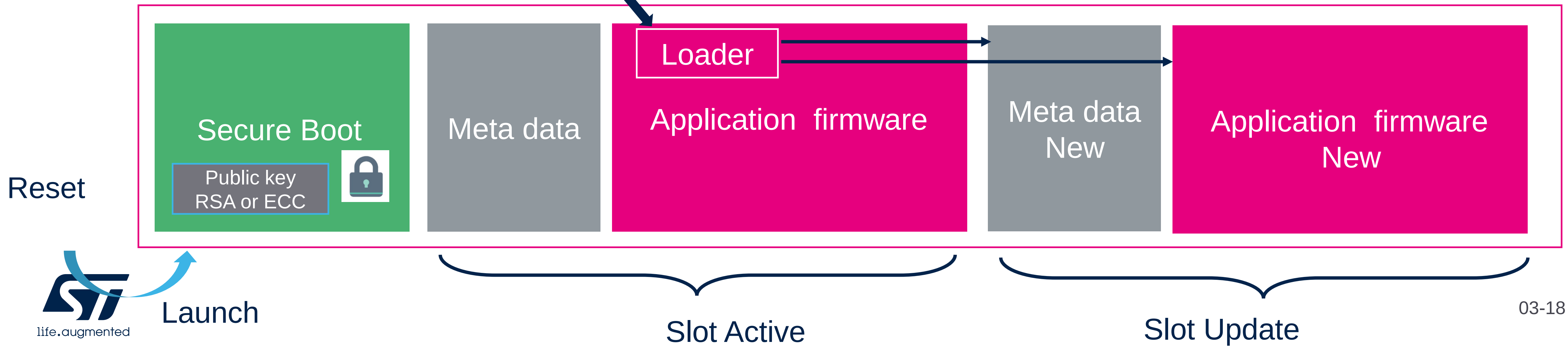
- Erase old firmware
- Transfer update firmware
- Check integrity & authenticity. Erase if fail



Remote update example



- Transfer update firmware
- Reset : secure boot detects new fw
- Secure boot checks new firmware
- Secure boot install new firmware (swap)

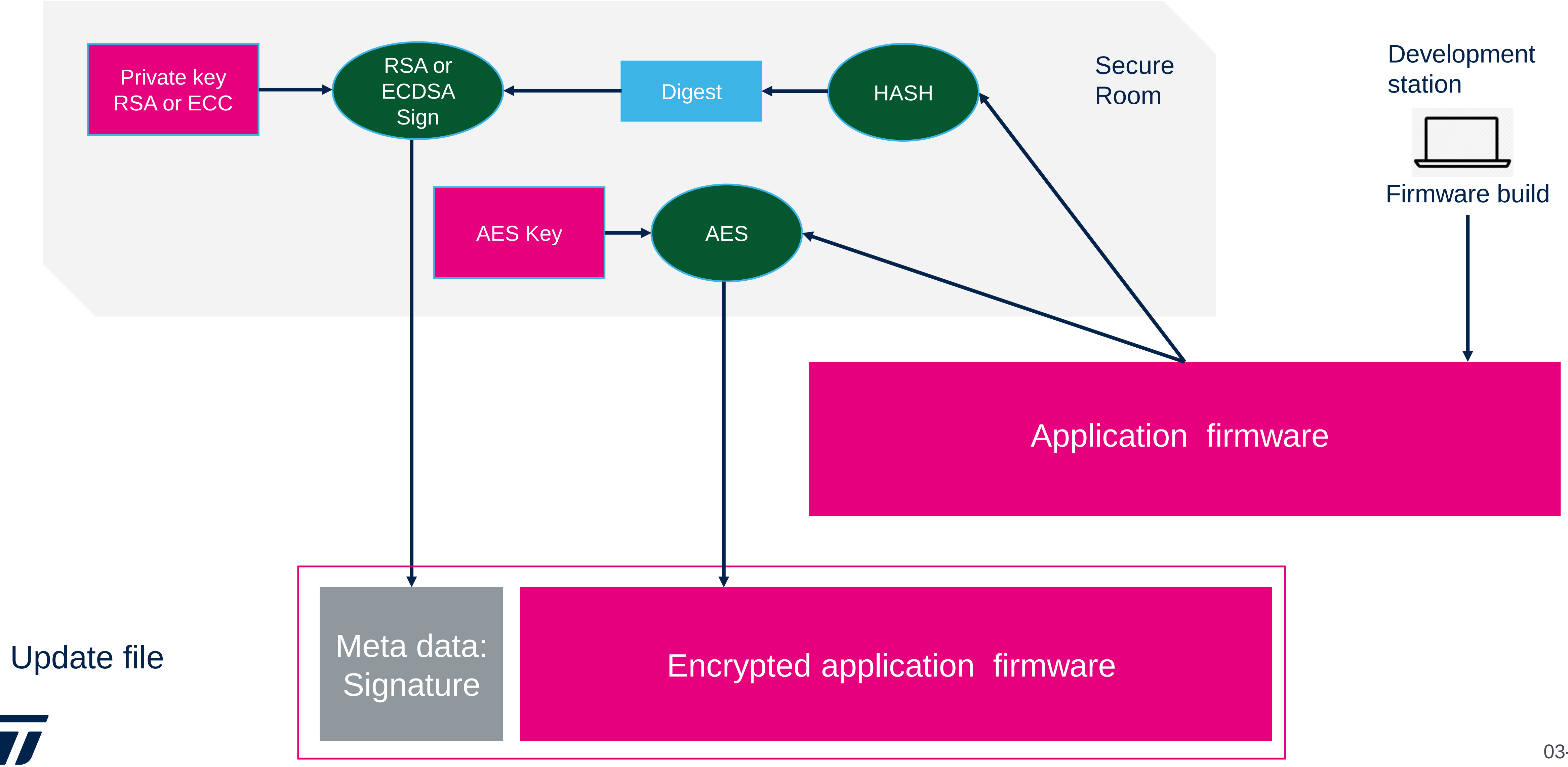


Update firmware encryption

- Firmware is most of the time a valuable asset
- To protect this asset during update, the encryption is needed
- This encryption is performed with a symmetric key
- This symmetric key is used to
  - Encrypt the firmware when generating the update file
  - Decrypt the firmware when installing the update file
- Same symmetric key is required both in secure room AND in secure boot

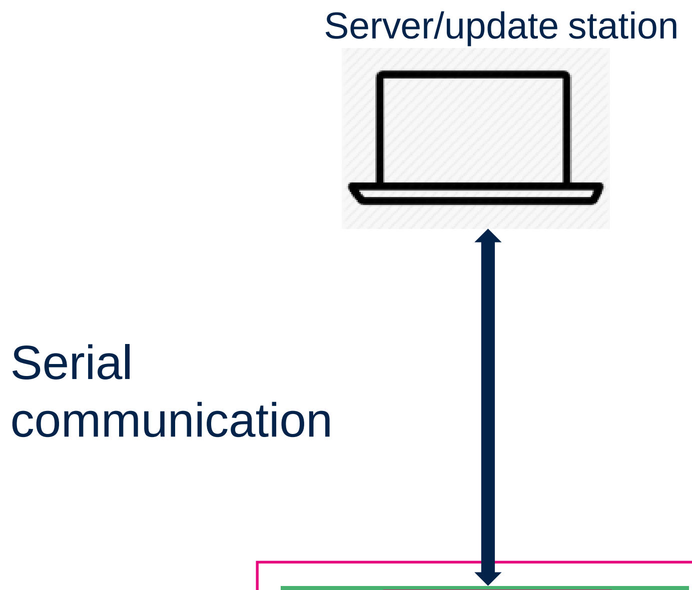
life.augmented

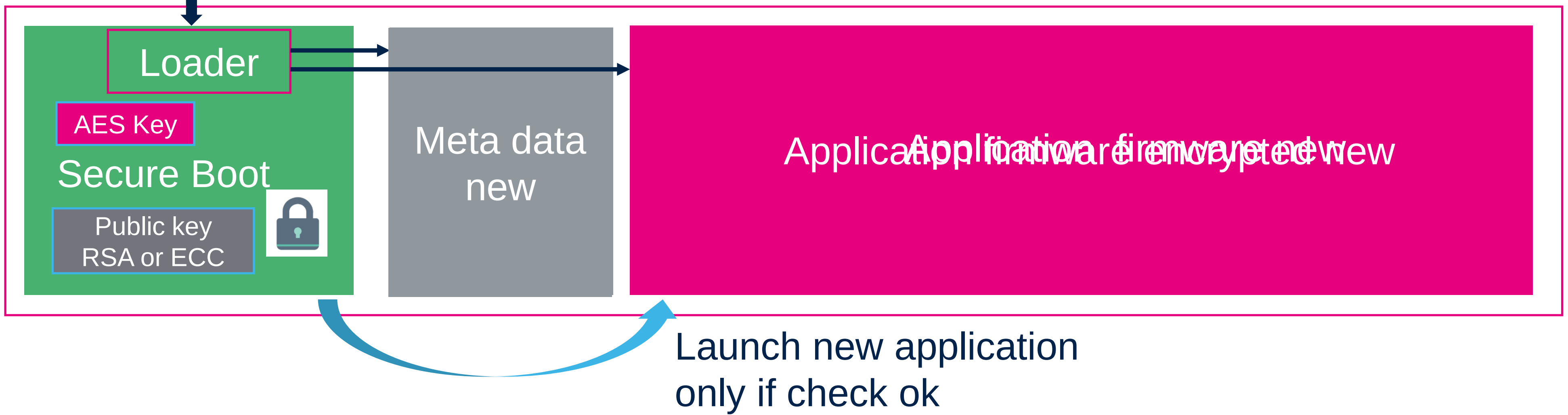
Create an encrypted update file



Local update with encryption example

- Erase old firmware
- Transfer update firmware
- Decrypt new firmware (could be done during transfer)
- Check integrity & authenticity. Erase if fail
- Launch application if OK





Keys Management

- As we could see we need
  - An asymmetric key pair to generate and check the signature
  - A symmetric key to encrypt and decrypt the firmware
- The asymmetric key pair can be generated with a tool like openssl
  - The private key is used to sign the firmware. It has to be kept in a safe place
  - The public key is used to check signature. It is provisioned on the device staying immutable
- The symmetric key can be generated with a random
  - The key is used to encrypt and decrypt the firmware
  - The key needs to be kept in a safe place
  - The key is provisioned on the device

Key protection

- The keys require different protections
- The keys used in secure room (private key and symmetric key) must be stored in a safe place by the OEM
- The public key is provisioned with the secure boot
  - The knowledge of the key is not a problem: it is public
  - But, as this key is used to check the firmware signature, it must remain immutable
- The symmetric key is also provisioned with the secure boot
  - The key must remain secret !
  - So, it needs to be protected inside the secure boot
  - The protection against read access must use an isolation mechanism

Conclusion

- We have seen the basic principles used in a secure boot and secure firmware update
- As you can see this involves many concepts that require some development
- Also, STM32 family implements different security mechanisms

- To address this complexity, ST proposes the X-CUBE-SBSFU package

Agenda

1 Secure boot principle

2 Secure firmware update principle

3 SBSFU Package

4 SBSFU and protection

5 SBSFU cryptography

6 SBSFU Architecture

7 SBSFU Advanced features

8 Conclusion

life.augmented

What is X-CUBE-SBSFU ?

- This package is an implementation of a secure boot and secure firmware update
- It addresses nearly all the STM32 families (except F0 to F3)
- It takes into account the STM32 family diversity
- SBSFU is provided to help ST customers developing their own SBSFU adapted to their level of requirement

Important notice

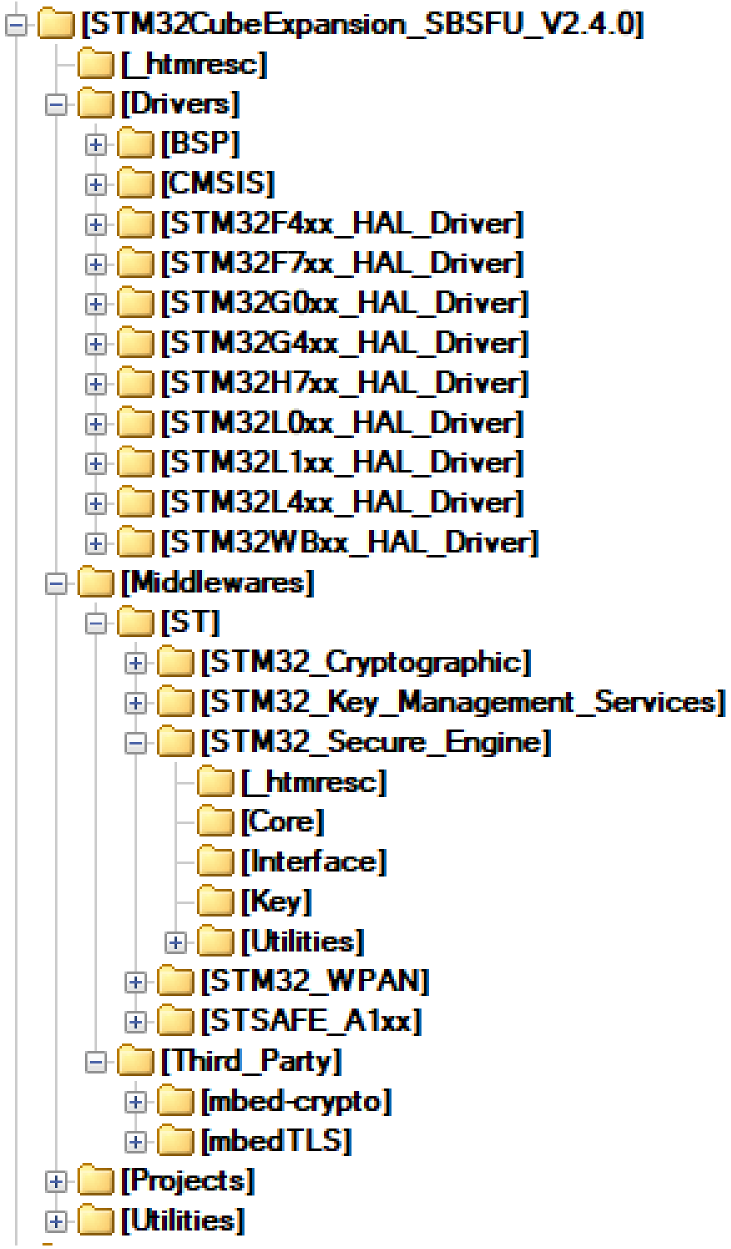
- X-CUBE-SBSFU is provided as reference code to demonstrate state-of-the-art usage of the STM32 security protection mechanisms. It is a starting point for OEMs to develop their own Secure Boot and Secure Firmware Update applications as a function of their product security requirement levels.

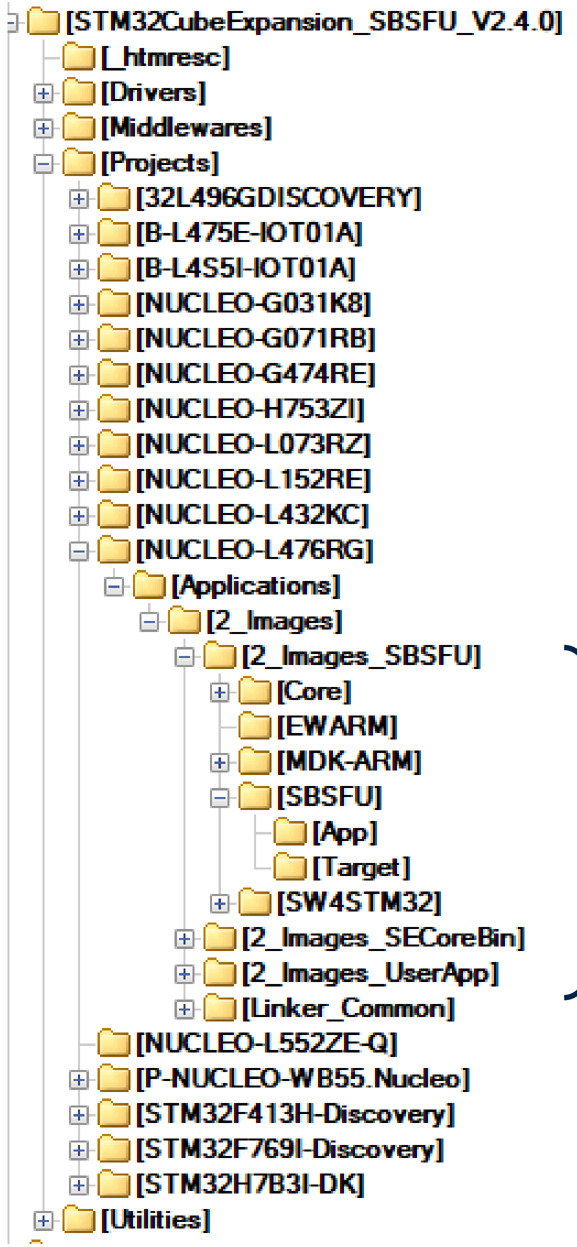
SBSFU in a Nutshell

- X-CUBE-SBSFU is a standalone package containing full functional secure boot and secure firmware update examples for each supported STM32 family: F4, F7, G0, G4, H7, L0, L1, L4, L4+STSAFE, WB
- Current version is STM32CubeExpansion_SBSFU_V2.4.0
- 34 examples available
  - 1 image (10) : Use of 1 SLOT
  - 2 images (13) : Use of 2 SLOTS
  - 2 images OSC (6) : Use opensource crypto instead of X-Cube-Cryptolib
  - 2 images STSAFE (1): Use of STSafe to store the certificate chain
  - 2 images KMS (2) : Key Management System : key store
  - 2 images ExtFlash (2) : Use of external memory to store firmware
- Documentation :
  - Introduction to STM32 microcontrollers security (AN5156)
  - Getting started with the X-CUBE-SBSFU STM32Cube Expansion Package (UM 2262)
  - Integration guide for the X-CUBE-SBSFU STM32Cube Expansion Package (AN5056)

life.augmented

Package content





Everything needed in one single package !

This is the example we will work on in this workshop

life.augmented

Agenda

life.augmented

How SBSFU addresses security requirements ?

- SBSFU support all protection mechanisms available in STM32!
- Confidentiality of the firmware
- Unique entry point
- Immutability of the secure boot code
- Isolation mechanisms to protect against inner attacks

Confidentiality of the firmware

- The main point is to prevent external access to firmware
- This is achieved thanks to RDP level 2: No more jtag access is possible
- RDP Level 2 is setup using option bytes
- Update is still possible using the SBSFU
- SBSFU ensures at each boot that RDP configuration is setup as expected

life.augmented

Unique entry point

- Ensure that after a reset, the platform always boots at the same address
- This feature is ensured thanks to RDP Level 2
- The RDP Level 2 deactivates the ability to boot on system bootloader or RAM

Code Immutability

- Ensure that secure boot code and authentication key cannot be modified in any manner
- This feature is ensured thanks to the combination of WRP and RDP Level 2
- WRP (Write protection) is setup thanks to option bytes
- WRP can be setup to protect a selection of flash sectors
- The RDP level 2 makes option bytes no more changeable.

RDP usage

- RDP Level 2 brings the confidentiality + bootlock + immutability (with WRP) for all STM32.
- On recent STM32 families RDP Level 1 can be used with some limitations
- This concerns G0, G4 and H7 implementing secure memory
- The combination of RDP1 and secure memory brings the impossibility to connect with debugger and the inability to change the content of secure memory
  - Confidentiality is ensured thanks to jtag deactivation
  - Unique boot entry ensured thanks to BOOT_LOCK on G0 and G4
  - Immutability ensured thanks to secure memory

Cryptography integration and isolation

- Cryptography used to ensure integrity and authenticity of the application
- This feature is ensured thanks to
  - X-CUBE-Cryptolib libraries
  - Opensource mbedTLS crypto
- Isolation of symmetric key
  - On STM32L4, L0 protected thanks to PCROP and firewall
  - On STM32F4, F7, L1 protected thanks to MPU privileged
  - On STM32H7, G0, G4 protected thanks to PCROP and MPU privileged during SBSFU execution and thanks to secure memory during application execution
  - On STM32WB protected thanks to isolated M0 core

Additional security mechanisms

- Tamper
- Watchdog
- Disable debug GPIOs
- Disable DMA clocks to enforce MPU protection

life.augmented

SBSFU security mechanisms handling

- All these mechanisms are handled by SBSFU
- At each boot, SBSFU ensures
  - All static protections (Option bytes configuration) are well setup
  - All dynamic protections (MPU, Firewall, Watchdog, tamper) are setup according to expected configuration
- From v2.2.0 version a control flow mechanism was introduced
- This mechanism is a counter measure against power glitch attack
- If an instruction is bypassed because of glitch, for instance to avoid a static protection check or a dynamic protection activation, this will be detected.

Conclusion

- Proper handling security mechanisms is one of the most important things
- SBSFU provides a framework to easily setup all the mechanisms needed
- SBSFU ensures all mechanisms are properly setup

life.augmented

Agenda

1 Secure boot principle

2 Secure firmware update principle

3 SBSFU Package

4 SBSFU and protection

5 SBSFU cryptography

6 SBSFU Architecture

7 SBSFU Advanced features

8 Conclusion

life.augmented

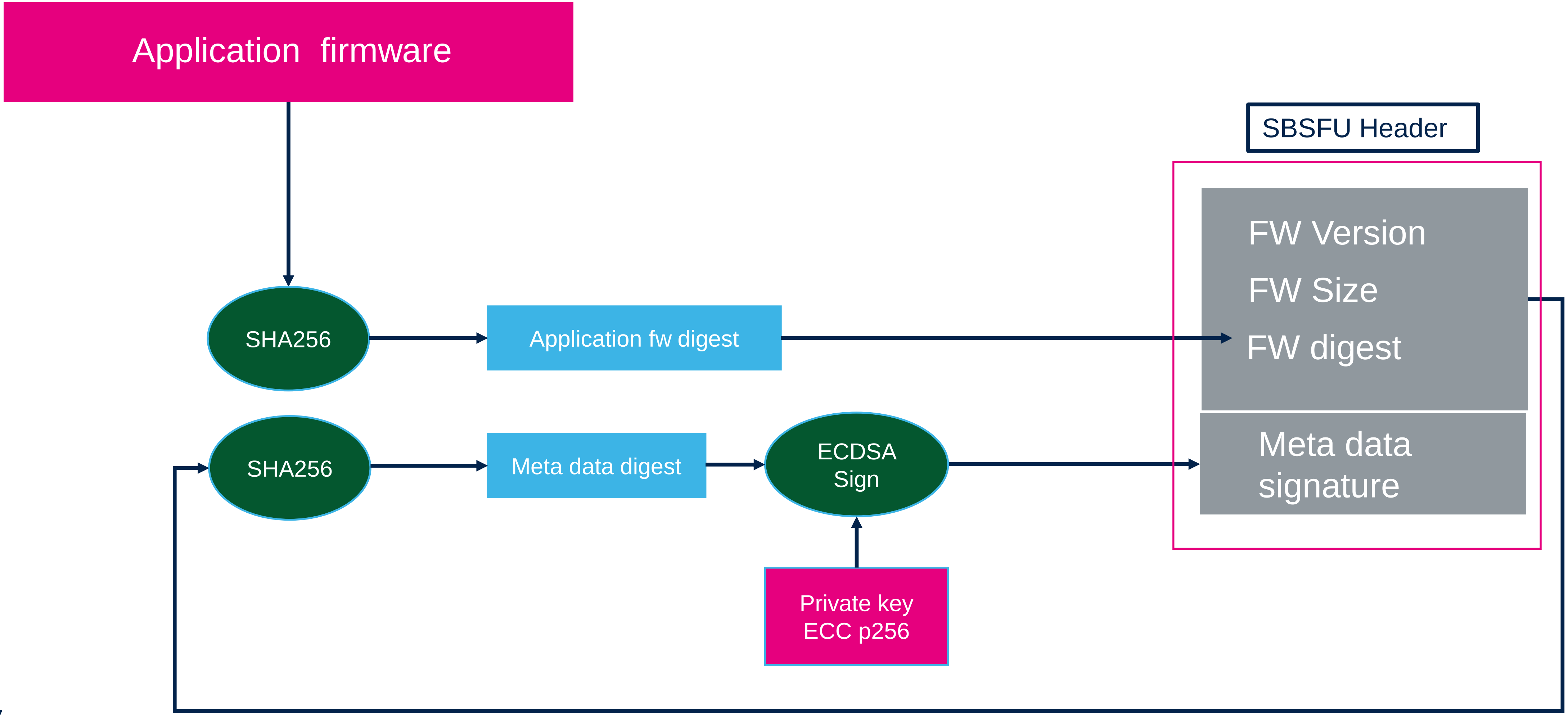
How SBSFU uses cryptography ?

- After security mechanisms are properly setup, the cryptography operations can be securely executed
- SBSFU uses following crypto algorithms :
  - Integrity: SHA256
  - Authentication: Elliptic curve ECDSA with p256 curve
  - Confidentiality: AES CBC

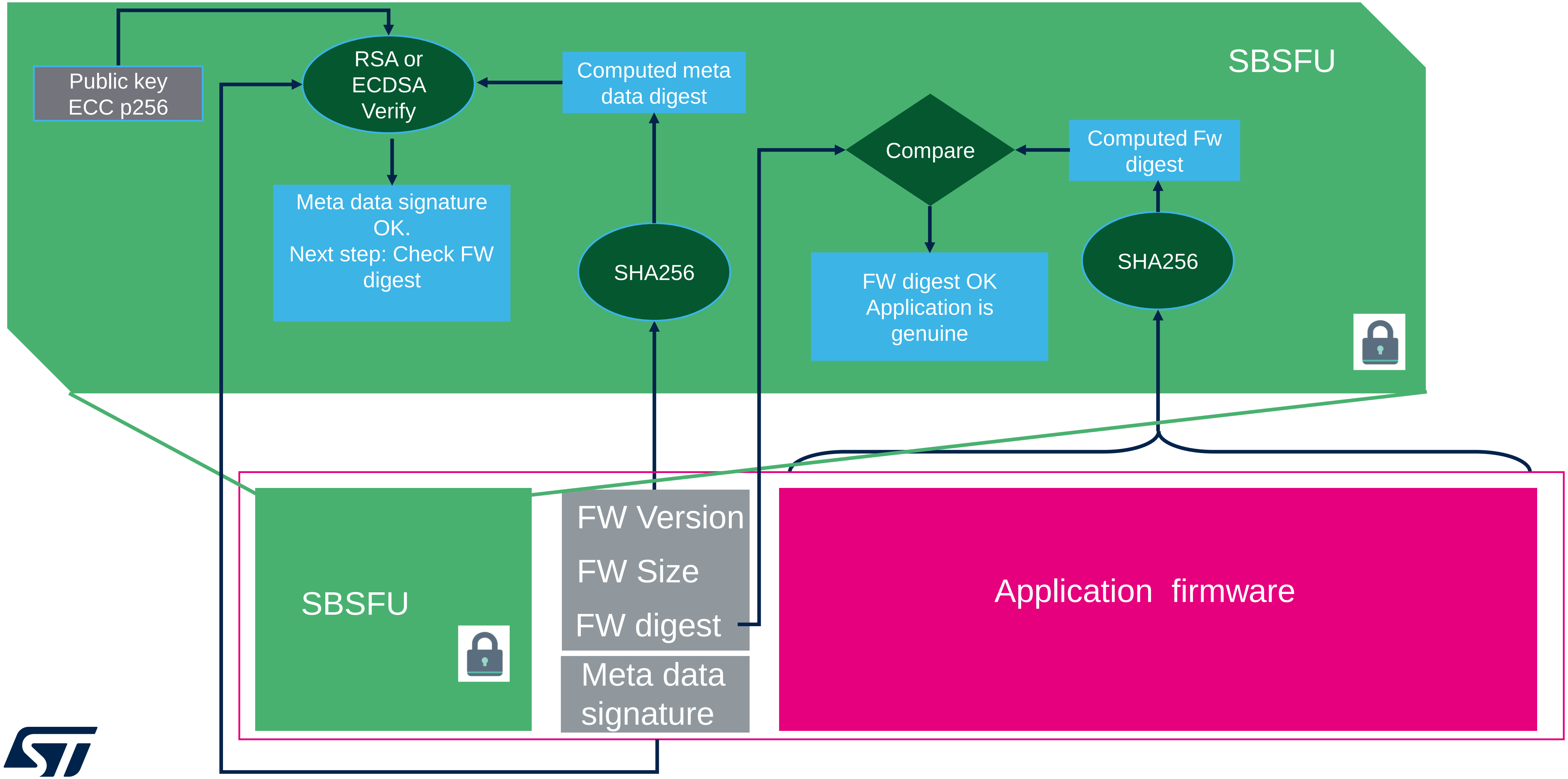
SBSFU authentication process

- SBSFU implements 2 steps authentication
  - 1st step: Authenticate the metadata
  - 2nd step: Once metadata content can be trusted, the SBSFU simply checks firmware digest

SBSFU meta data



SBSFU integrity and authenticity check



life.augmented

Agenda

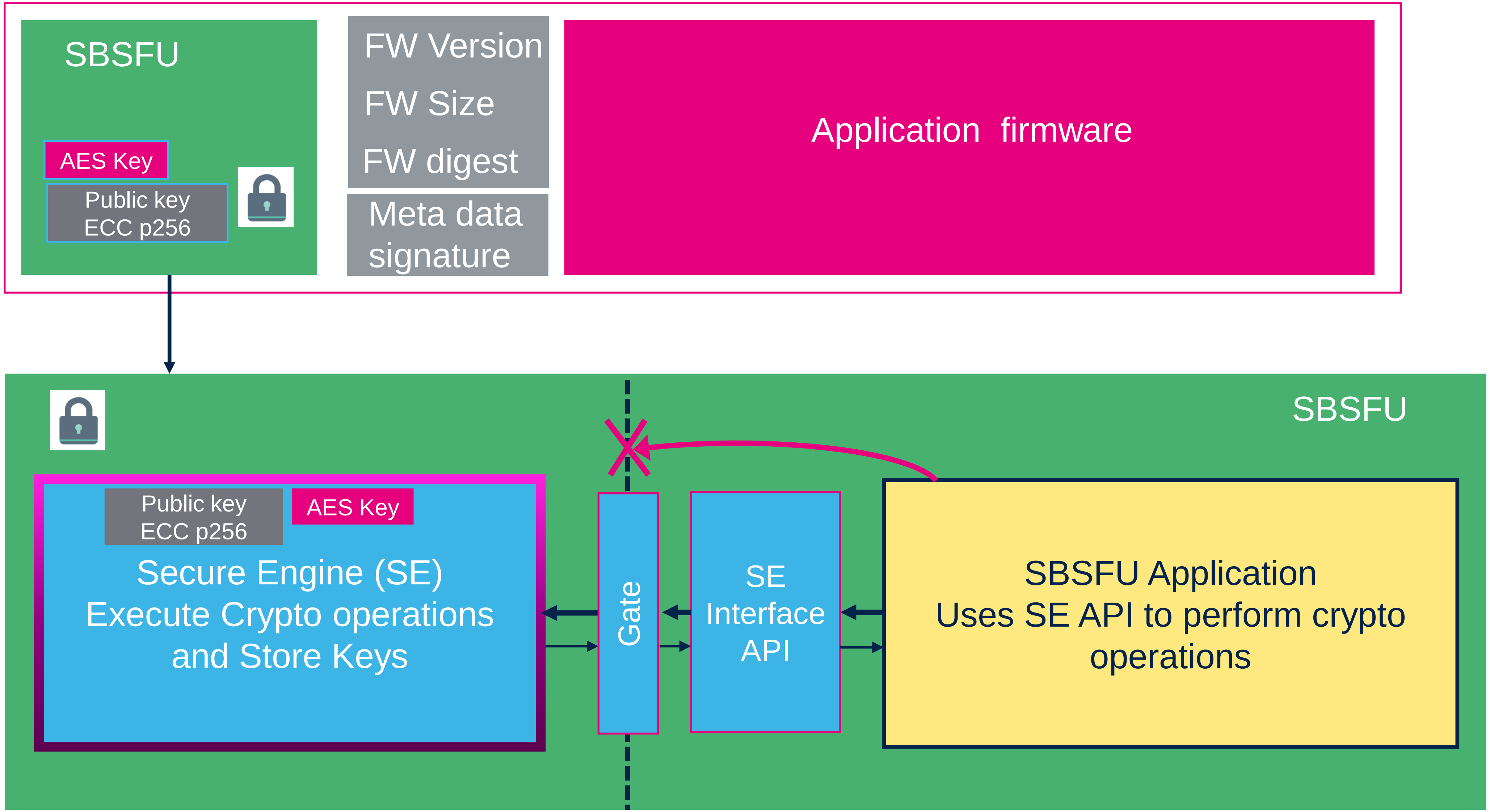
life.augmented

SBSFU architecture overview

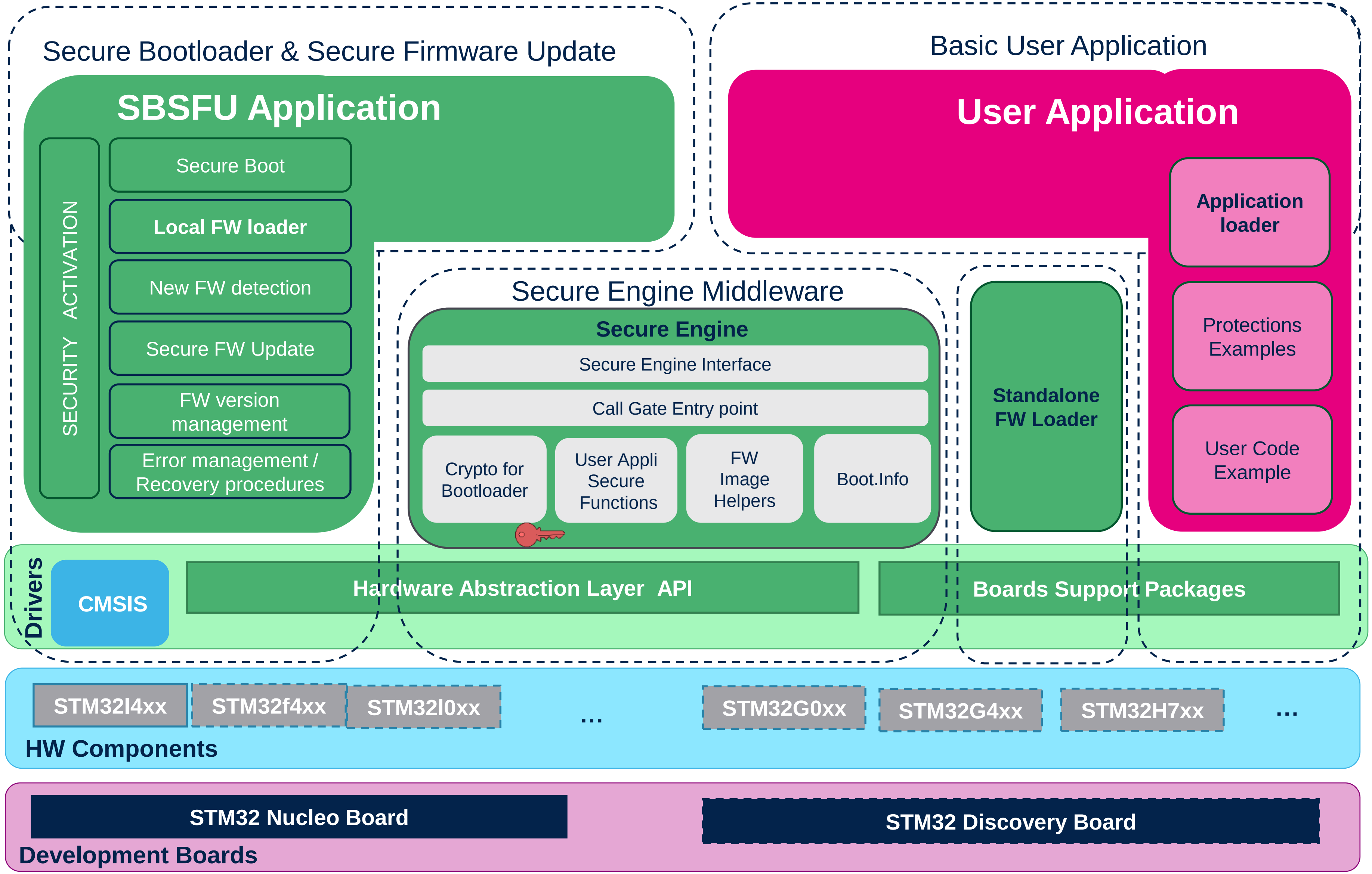
- SBSFU is split in 2 components
  - Secure engine
  - SBSFU application
- Secure engine is a framework to handle critical functions in an isolated environment.
- Secure engine is composed of
  - An API that defines the services provided in secure engine
  - A Core that implements these services

life.augmented

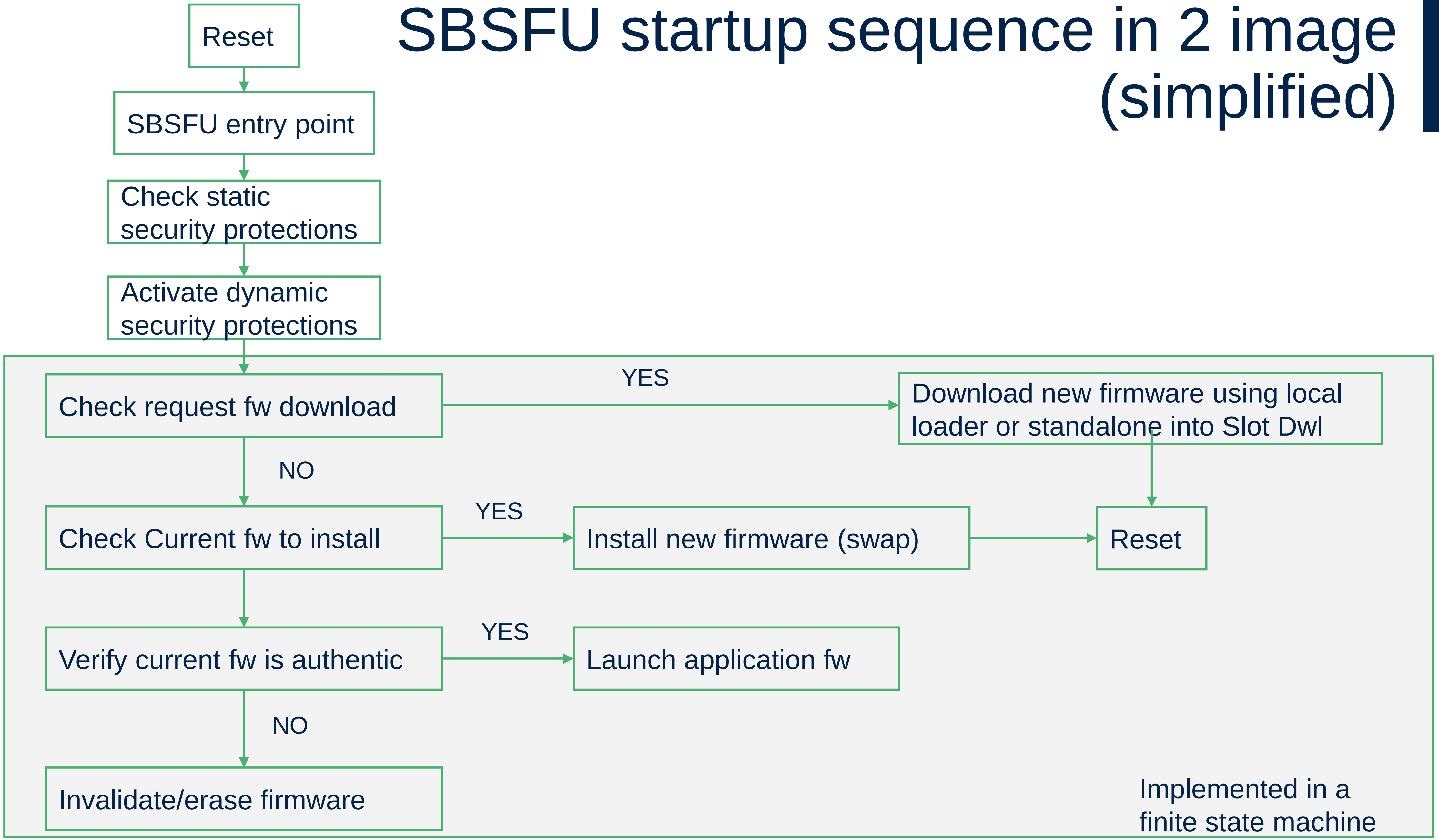
SBSFU Application and secure engine



Package Architecture Overview



SBSFU startup sequence in 2 image (simplified)



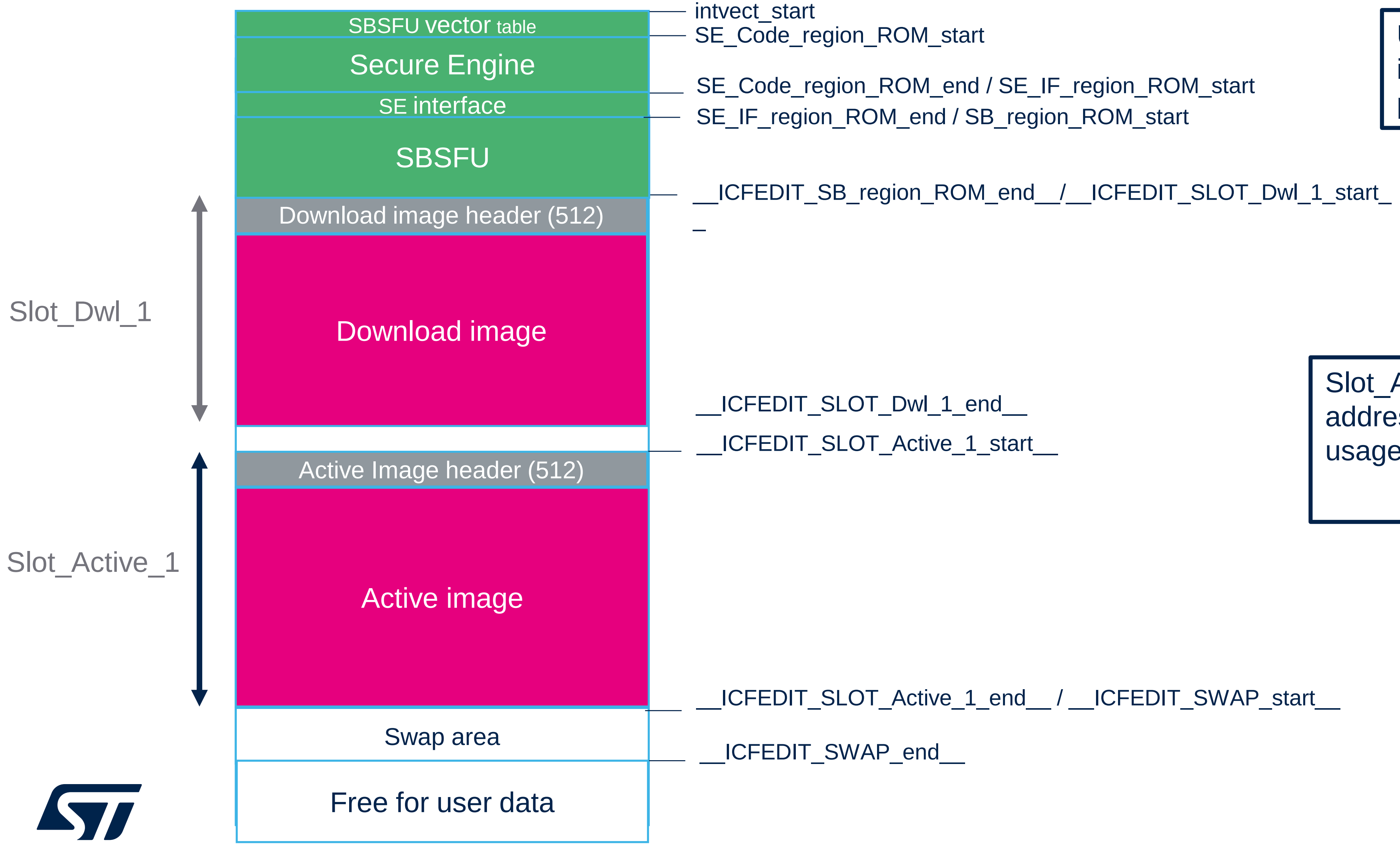
Checking static security

- STM32 implements static security settings through option bytes
- SBSFU provides an easy way to activate each protection through compile time option (#define)
- Just after booting, SBSFU ensures option bytes are in expected state
- 2 possible behaviors when an option byte is not in expected state
  - 1) Development mode: SBSFU programs the option byte and resets.
  - 2) Production mode: SBSFU provides a default behavior (reset) that customers can change. But no code used to program option bytes
- Static protections handled
  - RDP, WRP, PCROP, Secure Memory, Bootlock

Dynamic security setting

- This step is done just after static protection check
- Purpose is to setup all security related configurations such as
  - MPU
  - Firewall (for L4 and L0)
  - Watchdog
  - Tamper
  - Disable debug GPIOs
  - Disable DMA clocks
- Each protection can be enabled/disabled through compilation flag

SBSFU FLASH mapping on STM32L476

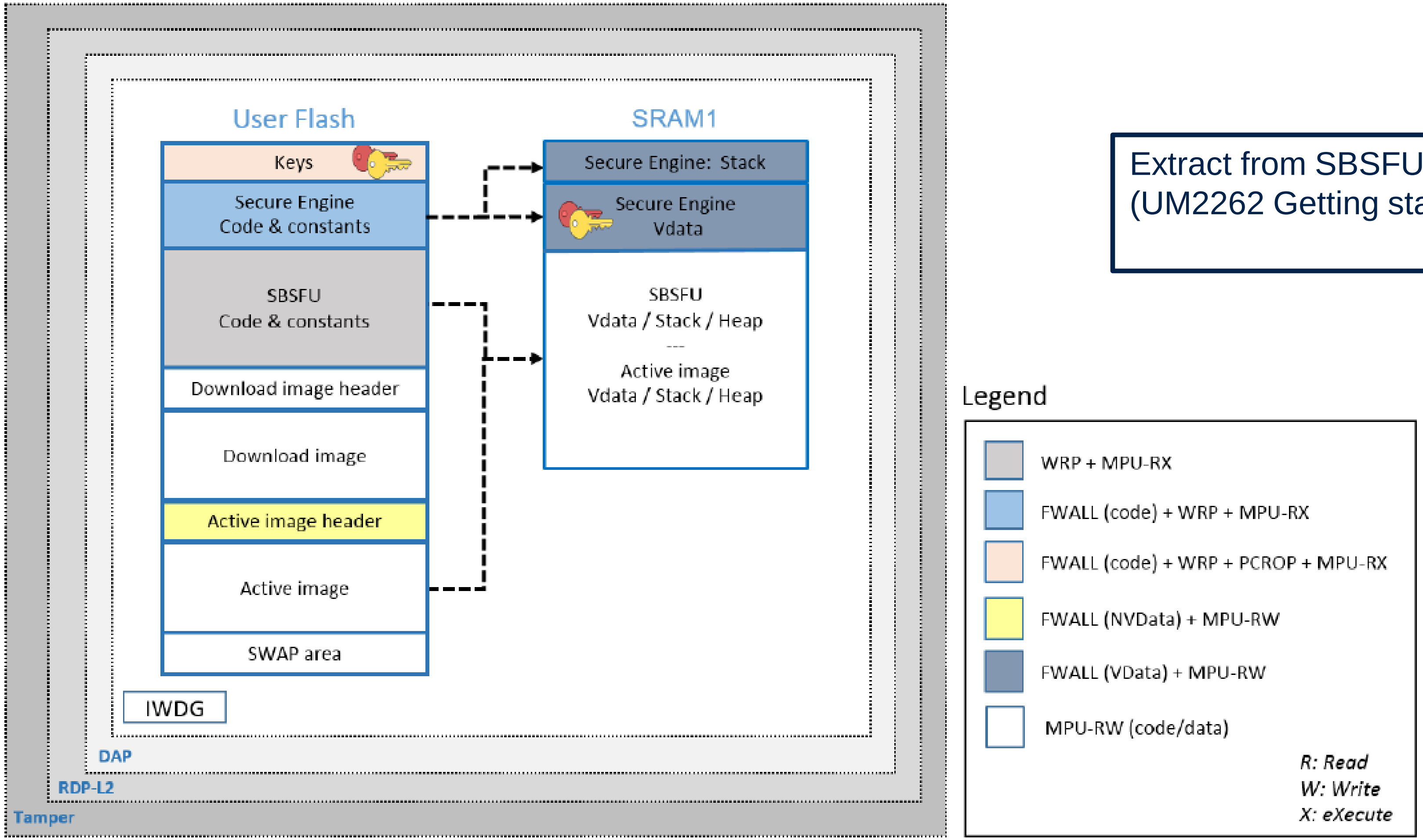


Use of symbols to gather information used by different projects in one place.

Slot_Active_1 after Slot_Dwl_1 to address a constraint specific to firewall usage on STM32L4.

life.augmented

SBSFU protection on STM32L4



Extract from SBSFU User manual
(UM2262 Getting started with the X-CUBE-SBSFU)

life.augmented

SBSFU figures

- Memory footprint
  - Flash : between 32 KB and 56 KB depending on features, debug, target and compiler used.
  - RAM : 4KB used by the secure engine
- Boot time
  - Boot time mainly depends on the crypto operations(about 90% of boot time)
  - This depends on the performance of addressed target and available hardware accelerator.
  - Order of magnitude is between 100ms and 1 second
- Update time
  - The update time depends on
    - Download interface speed
    - Flash writing speed
    - Note that single image update is much faster compared to dual image update

life.augmented

Agenda

life.augmented

SBSFU package advanced features

- Key Management Service
  - Added services in Secure Engine to manage static and dynamic key storage. New key can be provisioned with similar mechanism as firmware update
- Secure Element support
  - Example supporting the STSAFE-A110 used to validate authentication key.
- External flash support
  - Support of external flash accessible through QSPI
  - External flash used for Slot_Dwl
  - On STM32 supporting on the fly decoder (STM32H7B3) slot_active can also be in external flash
- Partial update
  - Ability to update only one part of the firmware to reduce update size.

life.augmented

Update options in 2 images

- Default legacy 2.3.0 behaviour
  - Usage of SWAP area but no rollback authorized
- No swap
  - Installation without decryption in place, without SWAP.
  - If interrupted, restart from last point
  - No support of partial update in this setup
- Self test by user application before complete validation
  - Swap application during install
  - New fresh application performs a self test (server connection for instance)
  - If test OK, application calls SE_APP_ValidateFw API. Firmware is validated
  - Is self test fails, application should force a reset. The SBSFU will then rollback to previous image

Multislot

- In order to manage multi core devices SBSFU can define up to 3 active slots
- Associated to these slots you can have up to 3 download slots
- It is possible to keep only 1 download slot for 3 active slots
- SBSFU check authenticity and integrity of each slot at startup

- SBSFU jumps to first valid slot in numerical order

Agenda

life.augmented

Conclusion

- X-CUBE-SBSFU package is supporting all recent STM32 families
- Only exception concerns the STM32 families based on CortexM33
  - The introduction of TZ and decision to support the ARM TFM framework lead to change the secure boot strategy on STM32L5.
  - For STM32L5, ST supports a X-CUBE-SBSFU equivalent solution based on MCU boot
- SBSFU code has been audited by external LAB (Brightsight)
- SBSFU provides many different configurations to fit with various setups
- SBSFU package is maintained and new features come regularly
- We support customer with their SBSFU implementation

Thank you

life.augmented