life.augmented

LoRaWAN Security

T.O.M.A.S

Introduction

- Security at the heart of LoRaWAN specification
- Secure boot and secure FW update
- Secure FW install at untrusted manufacturer
- Secure key provisioning
- SW and HW features embedded on STM32WL

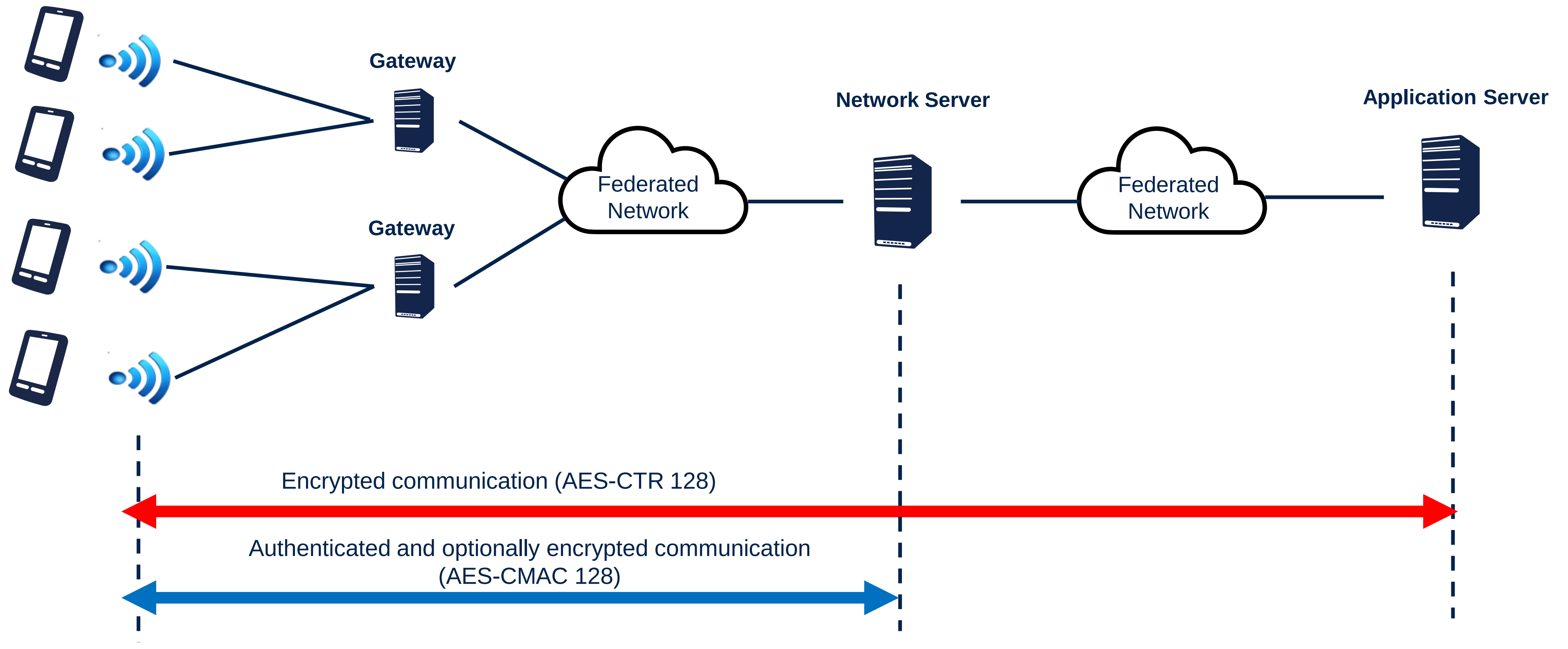
Agenda

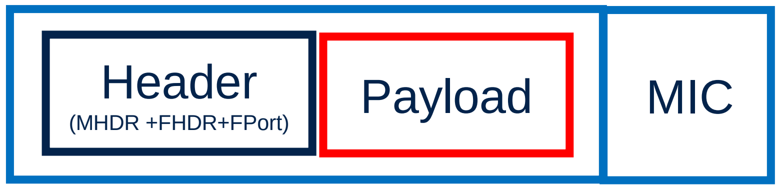
life.augmented

LoRaWAN 1.0 Security reminder

life.augmented

LoRaWAN protocol


Gateway
Gateway
Federated Network
Network Server
Federated Network
Application Server
Encrypted communication (AES-CTR 128)
Authenticated and optionally encrypted communication (AES-CMAC 128)

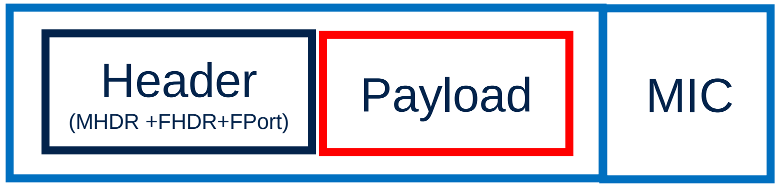


life.augmented

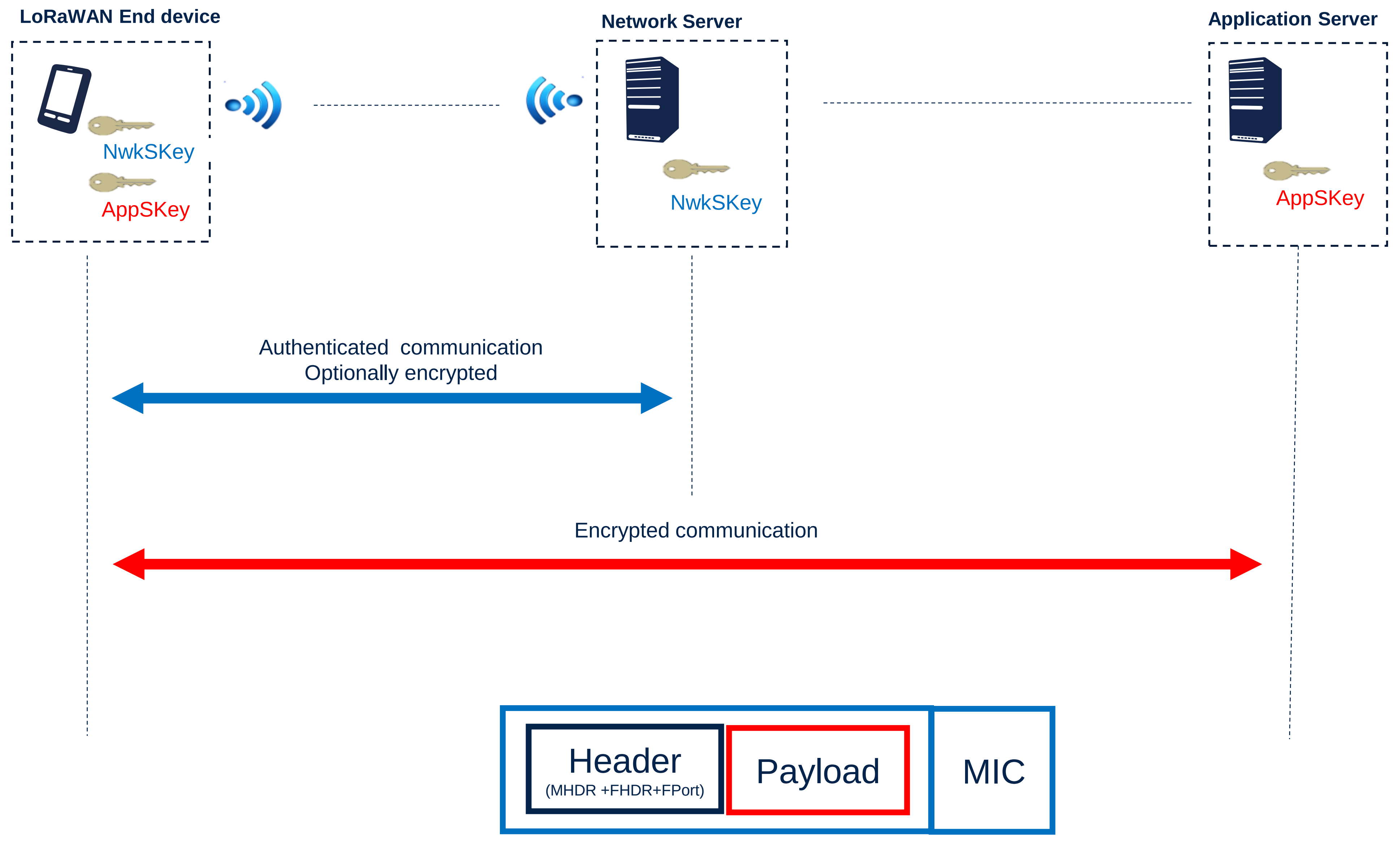
Protocole

- AppSKey: (128 bits AES)
  - unique key per device
  - use to encrypt payload using AES-CTR algo
- NwkSKey: (128 bits AES)
  - unique key per device
  - use to generate Message Integrity Code using AES-CMAC algo

- Activation procedures
  - Activation by personalization
  - Over-the-Air Activation



Activation by personalization

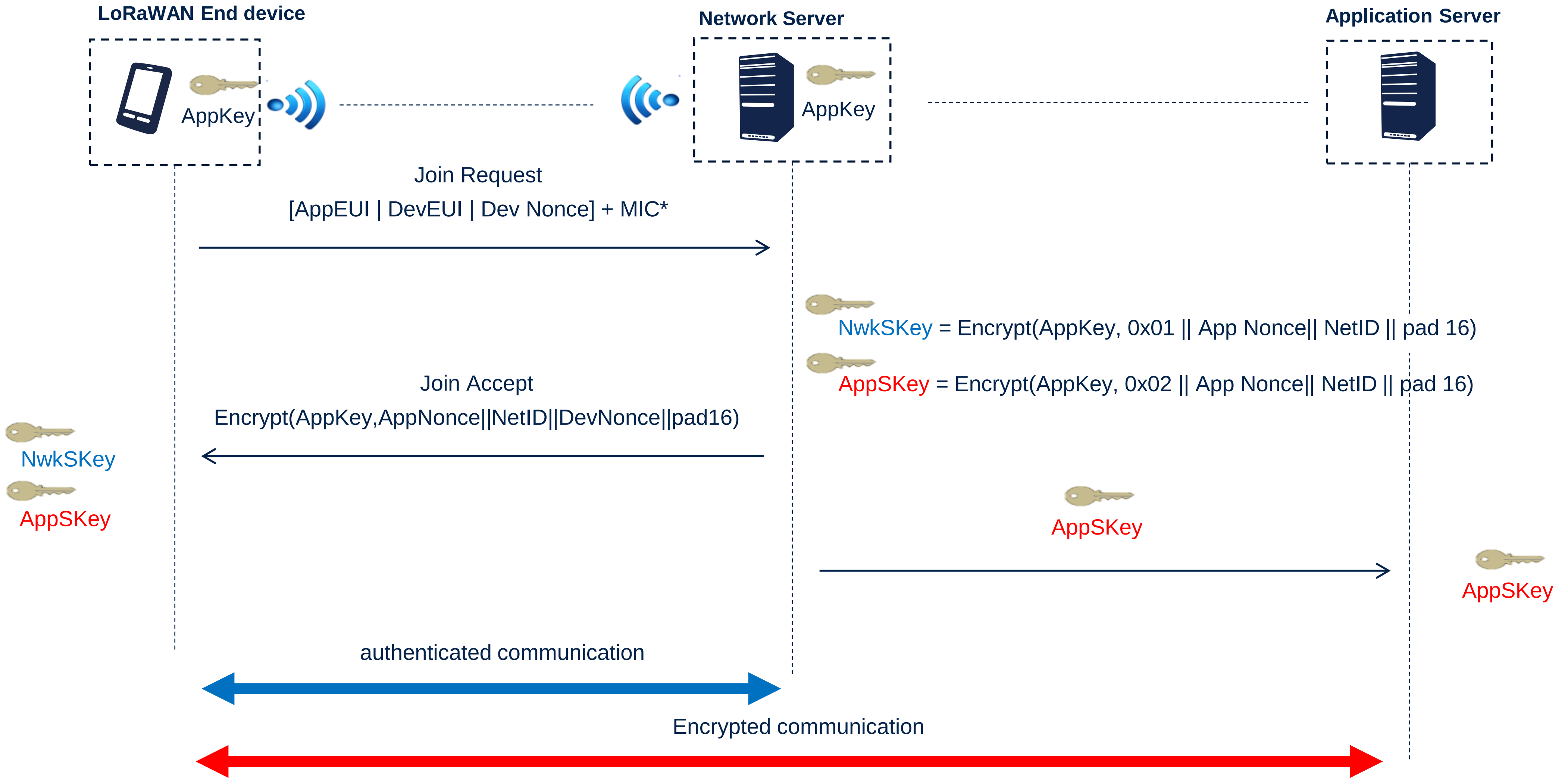


Activation by personalization

- Crypto algo used AES-CMAC / AES-CTR
- Each device have :
  - AppSKey: (128 bits AES) : unique key
  - NwkSKey: (128 bits AES) : unique key
  - DevAddr (32 bits) identifies the end-device within the current network. It is allocated be by Network server

life.augmented

Over-the-Air Activation (a.k.a. Join Procedure)


LoRaWAN End device
AppKey
Network Server
AppKey
Application Server
Join Request
[AppEUI | DevEUI | Dev Nonce] + MIC*
NwkSKey = Encrypt(AppKey, 0x01 || App Nonce|| NetID || pad 16)
AppSKey = Encrypt(AppKey, 0x02 || App Nonce|| NetID || pad 16)
Join Accept
Encrypt(AppKey,AppNonce||NetID||DevNonce||pad16)
NwkSKey
AppSKey
AppSKey
AppSKey
authenticated communication
Encrypted communication

Over The Air Activation

- Each end-device is provisioned with:
  - AppKey (128 bits AES) : unique key
  - DevEUI (64 bits) a globally unique identifier (EUI-64-based)

- AppEUI/JoinEUI (64 bits) : unique identifier which identifies the server that stores
  - AppKey
  - Association between End device and Application server

- Crypto algo used AES-CMAC / AES-CTR

life.augmented

LoRaWAN System security

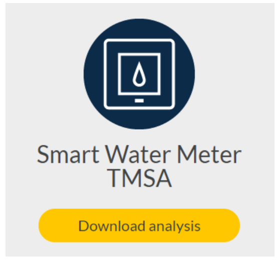
- Main assets, you need to protect :
  - Keys
  - Payload information
  - …
- Threats :
  - Inner attack : a malicious code which a hacker able to inject in your system
  - Outer attack:
    - Hacker spy the communication remotly
    - Hacker have a physical access to the board
- Counter measure :

  All the SW/HW mechanism of STM32WL

life.augmented

TMSA : Thread Model Security Analysis

- https://developer.arm.com/architectures/security-architectures/platform-security-architecture


Smart Water Meter TMSA

- Assets to protect :
  - Firmware
  - Meter ID
  - Logs
  - Credentials
  - Measurements
  - Firmware certificate
- Threats :
  - Tamper
  - Escalation Privilege
  - DoS
  - Impersonation
  - Men In The Middle
  - Repudiation
- Mitigation :
  - Tamper detection
  - SecureBoot
  - Secure firmware update
  - Isolation
  - AntiRollback

life.augmented

Thank you

life.augmented

LoRaWAN Device Security overview

life.augmented

What make a LoRaWAN end device secure ?

- Ensure the firmware that is running on the device is genuine.

  This will be addressed by Secure Boot

- Ensure the capability to update firmware in a secure way.

  This will be addressed by Secure Firmware Update

- Ensure the firmware and secrets are properly protected against attack.

  This will be addressed by Secure MCU Framework (combination of STM32 Security HW IP and Software good practice)

Secure boot is needed

- Secure boot is an immutable code always executed after the reset
- This piece of code
  - Checks platform security
  - Ensures about the authenticity/integrity of the firmware installed and launches it
- It will rely on cryptography mechanism associated with keys

Secure firmware update is needed

- The firmware update can be received over the air (FUOTA).

In this case due to RF bandwidth, a partial update of the firmware is a needed functionality.

- The authenticity and integrity should be checked before installation
- Again, it will rely on cryptography mechanism associated with keys

Key management

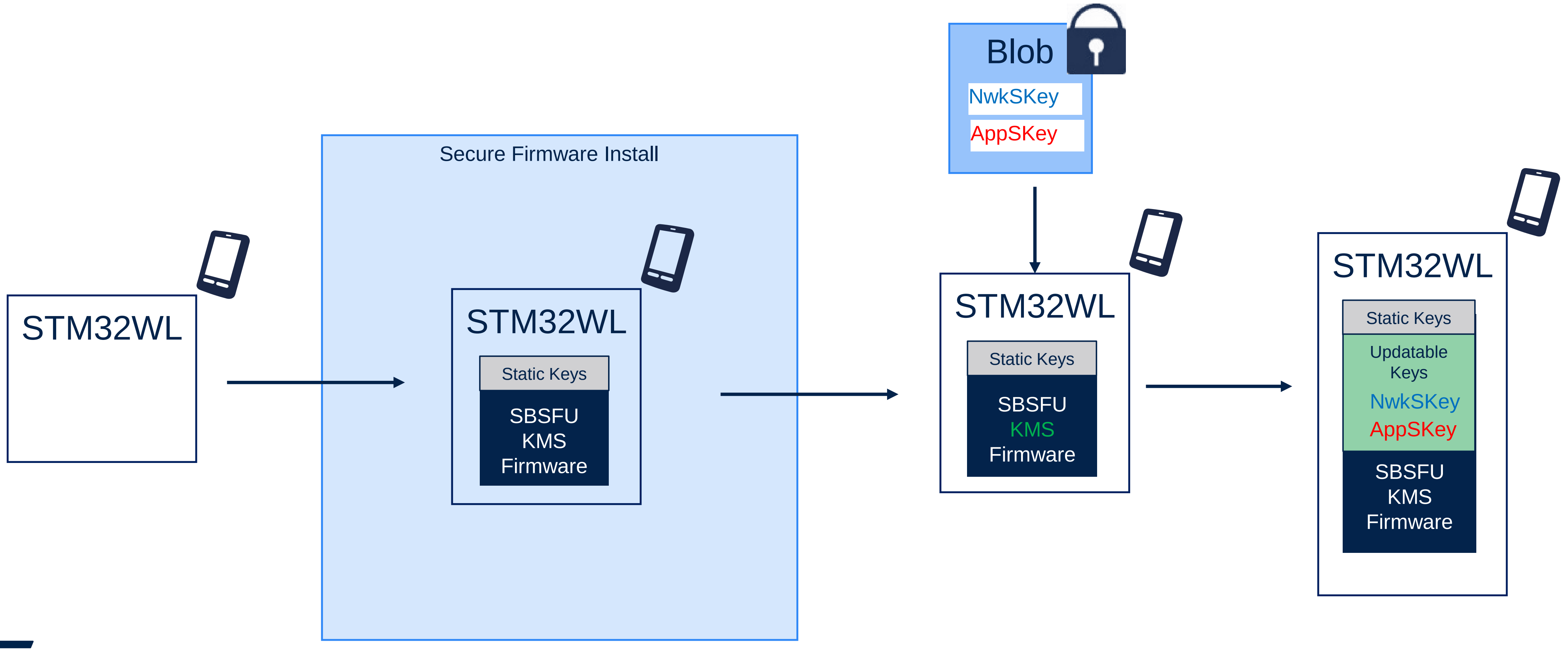
- Secure boot / Secure firmware update relies on static embedded keys
  - How to provision those keys ?

    SFI or secure flash environment allow to flash first firmware
- LoRaWAN security (AppSKey, NwkSKey or AppKey) can rely on updatable keys
  - How to provision those keys ?

    Key Management Services and the import service
- How to minimize exposure of key value?

    Key Management Services API

Key provisioning example



Thank you

life.augmented

STM32WL HW and SW Security mechanism

life.augmented

STM32WLEx\STM32WL5x Common Security mechanism

- Boot lock
- Readout Protection (RDP)
  - Level 0: no readout protection
  - Level 1: memory readout protection
  - Level 2: chip readout protection
- Tamper detection

Address outer attack

- Proprietary code Read Out Protection (PcROP)
  - 2 configurable areas of Flash memory
- Write protection (WRP)
  - 2 configurable areas per Flash memory

Address outer/Inner attack

- Memory protection unit (MPU)

Address Inner attack

Those mechanism allow you to achieve a certain level of security….

STM32WLEx (single core)

- ST delivers a code example of Secure Boot and Secure Firmware Update which relies on those security mechanism
- Secure Boot and Secure Firmware Update
  - ensure firmware authenticity and integrity
  - update firmware in a secure way
  - RDP +WRP (not activated by default)

  This is a first level of security which allow to protect against any attack via JTAG

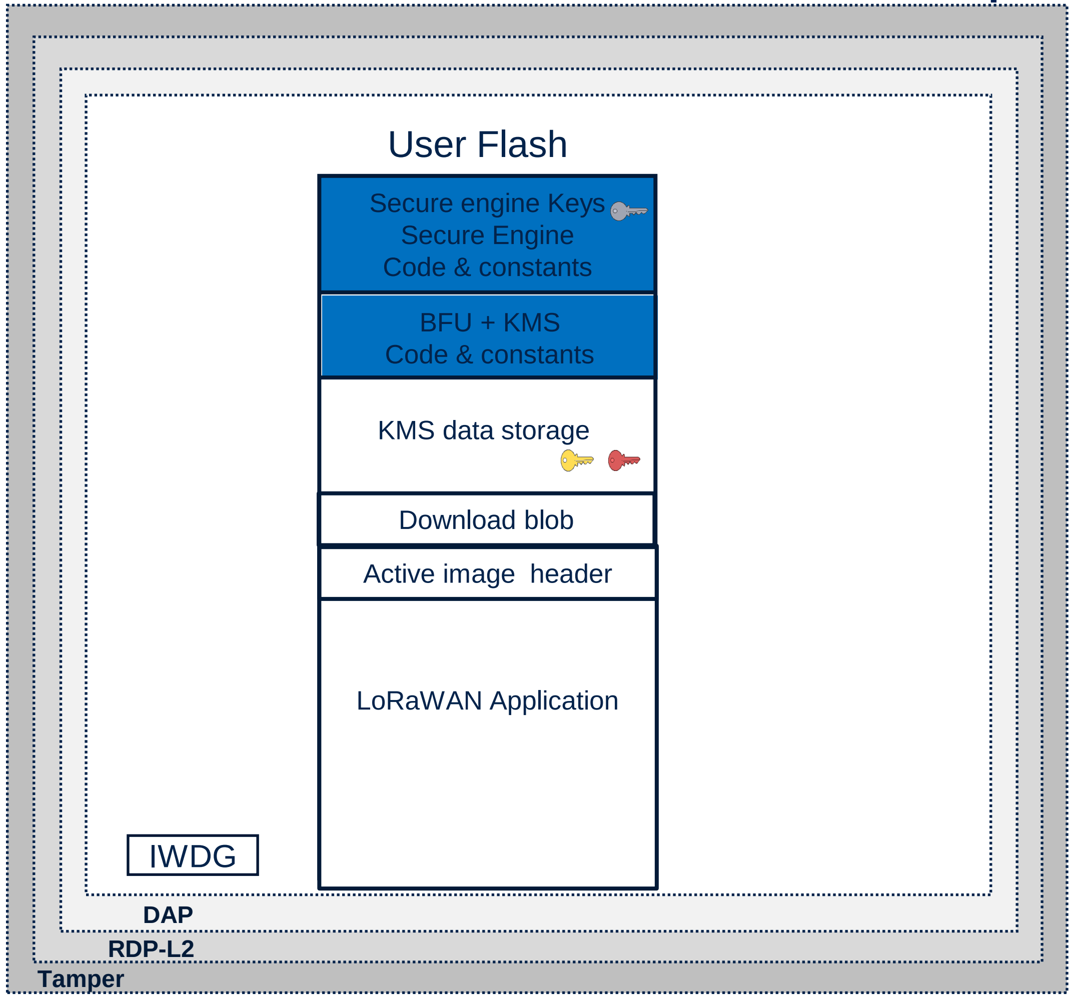
  Beware : as we don’t have isolation in our example, it means the surface of attack include all the internal flash (Appli+RF stack).

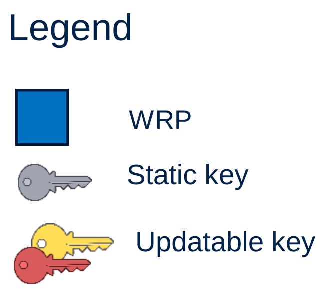
STM32Cube_FW_WL\Projects\NUCLEO-WL55JC\Applications\

- BFU_1_Image
  Secure boot and secure firmware update single slot with local loader and KMS

STM32WLEx (single core)

- ST Boot and Local Firmware Update and KMS :





- Bootlock
- RDP level 2 : device closed/OB freeze
- WRP : immutability of booting code
- Tamper : detection of tamper
- IWDG : unexpected behavior

life.augmented

STM32WL5x dual core additional mechanism

- Isolation Core M0+ mechanism (Memory/Peripheral)
- Privileged protection (Memory/Peripheral)
- Secure Memory HDP

$\}$ Address Inner attack

- Debug Core M0+ access

$\}$ Address outer attack

- Bootlock Core M0+ mechanism

$\}$ Root of trust

life.augmented

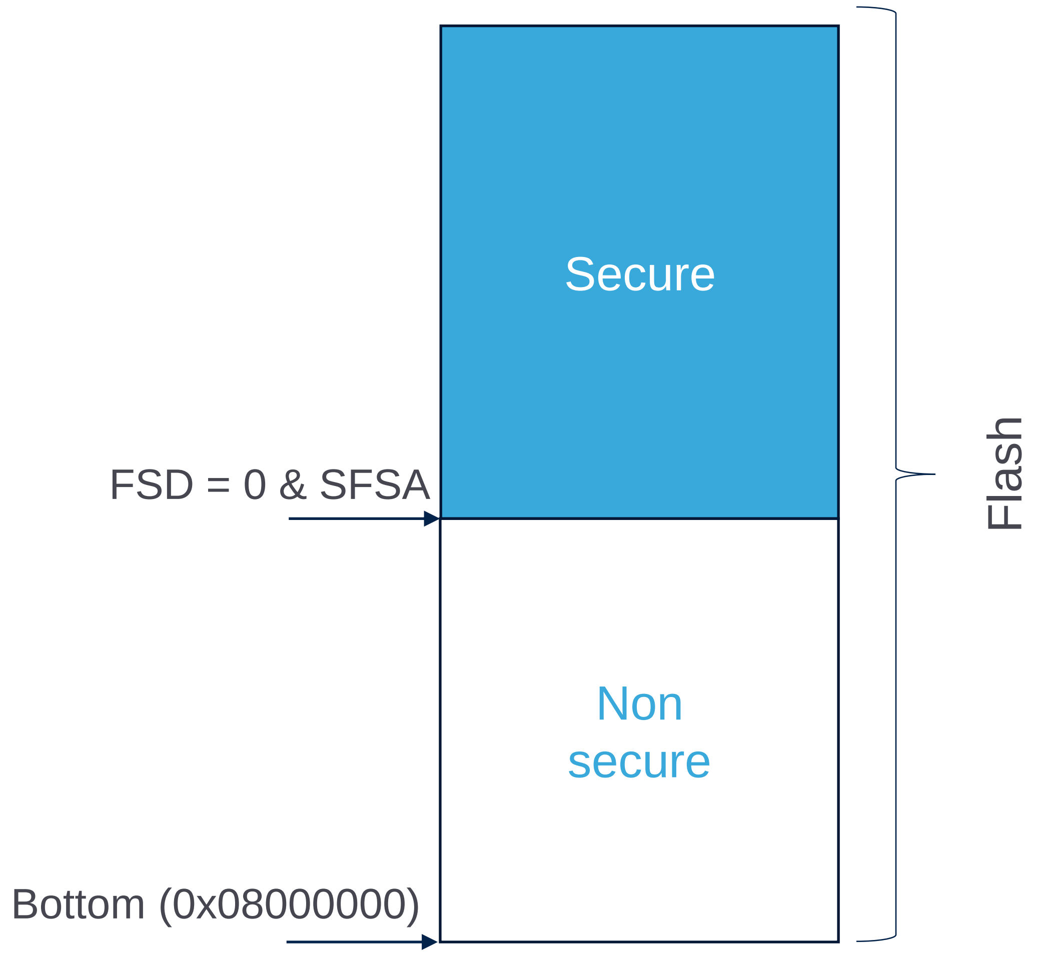
STM32WL5xxx additional mechanism

- Dual core brings new hardware security mechanism which allow to achieve an upper level of security :
  - Cortex M0+ security features
    - Secure Flash memory, SRAM1 and SRAM2 areas exclusively accessible by the Cortex-M0+.
    - Secure peripherals
      - Exclusive access to SUBGHZSPI, AES, PKA, and TRNG by the Cortex-M0+.
      - Exclusive Cortex-M0+ access to secured DMA channels.
    - Resource protection based on privilege
      - Memories and peripherals may be protected by privilege.
    - Debug security
      - Secure memory areas and peripherals not accessible through debug port.
  - Bootlock for the Cortex M0+/Cortex M4

life.augmented

Cortex M0+ security Flash isolation

- Security enable (**FSD** aka Flash Secure Disable) : Global enable of the Cortex-M0+ security
- Flash memory security
  - Secure Flash Start Address (**SFSA**)



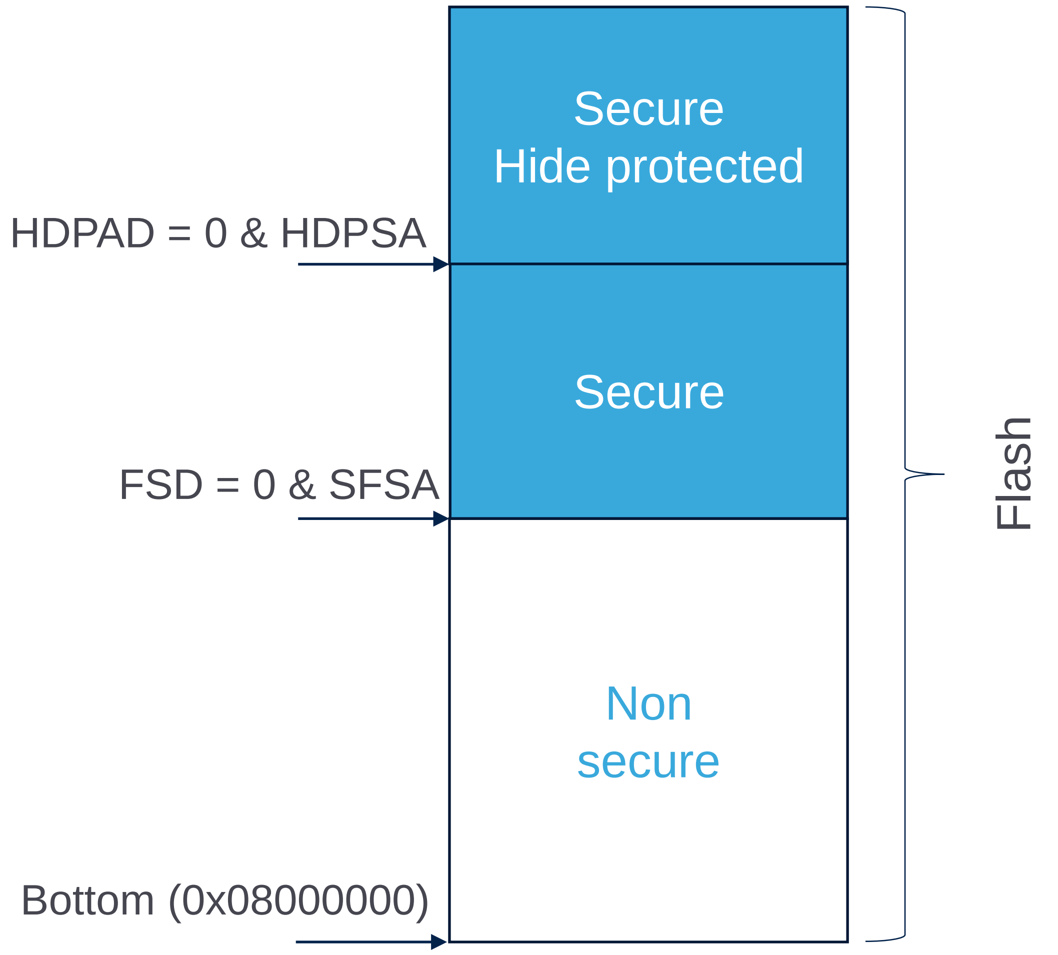
🟦 Exclusive access for Cortex-M0+

☐ Accessible by both Cortex-M4 and Cortex-M0+

Cortex M0+ security Flash isolation

- Flash memory security
  - Hide protection disable (**HDPAD**) / Hide protection Start Address (**HDPSA**)

Reminder: Hide protection allows to make memory disappear from the system after its use



🟦 Exclusive access for Cortex-M0+

☐ Accessible by both Cortex-M4 and Cortex-M0+

Cortex M0+ security RAM isolation

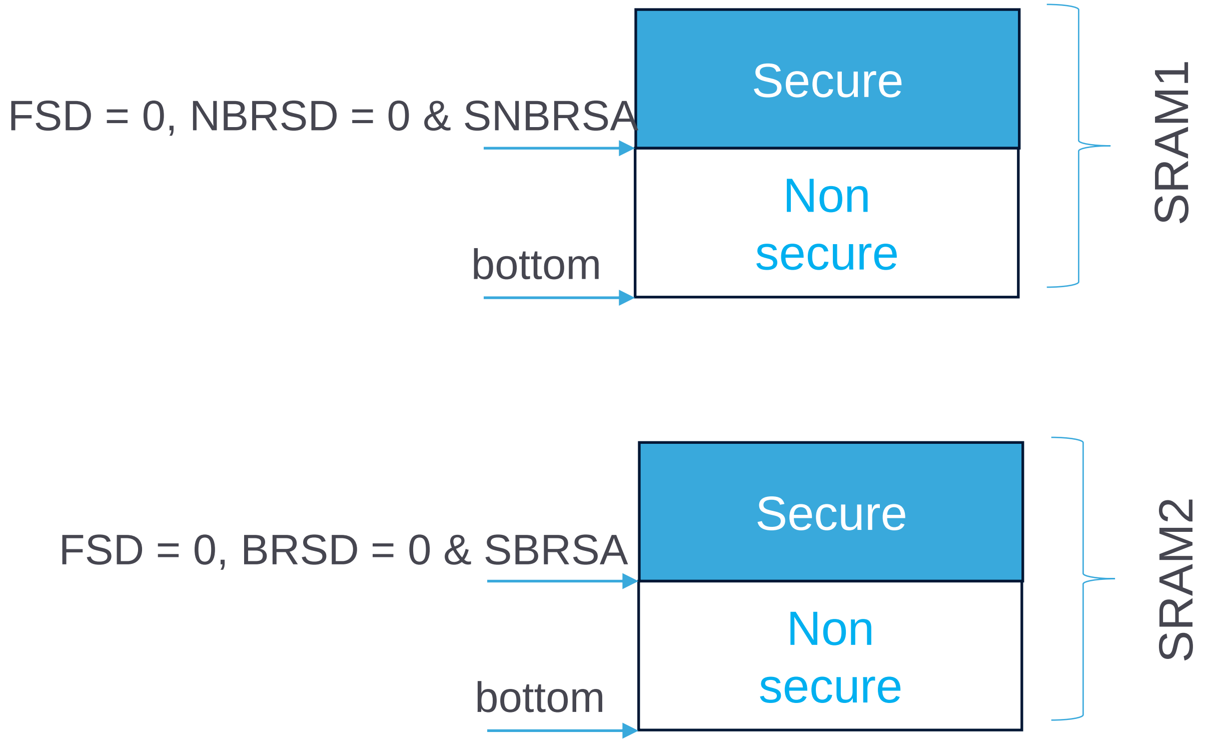
- Security enable (**FSD** aka Flash Secure Disable) : Global enable of the Cortex-M0+ security
- RAM security
  - RAM security enable

    NBRSD: non-backup SRAM1

    BRSD: backup SRAM2
  - Secure RAM Start Address

    SNBRSA: non-backup SRAM1

    SBRSA: backup SRAM2



🟦 Exclusive access for Cortex-M0+

☐ Accessible by both Cortex-M4 and Cortex-M0+

sub-GHz radio security

- Sub-GHz radio access handled by secure user options
- Controlled by **SUBGHZSPISD** option
  - Allows to control access to the sub-GHz radio, to be exclusively accessible by the secure Cortex-M0+.
  - Security setting is applied from reset.

Secure peripherals

- Peripheral security configured in the GTZC_TZSC.
  - Securable IP : AES, PKA and true RNG
  - Allows peripherals to be secured at run time.
  - Allows peripheral sharing on a need as basis between the secure CM0+ and the non-secure CM4.
  - Peripheral security is only available when Security is enabled in (FSD)
- DMA channel security configured in the DMA.
  - Allows DMA channels to be secured at run time.
  - Allows DMA channels sharing on a need as basis between the secure CM0+ and the non-secure CM4.

life.augmented

Privileged protection

- Privileged protection is handled by register bits in the GTZC_TZSC.
  - Allows privileged resources to be protected from unprivileged accesses.
- A single privileged watermark is available for each memory (FLASH/SRAM1/SRAM2)
- Privileged protection is available only on resources that feature security protection
  - Memories, sub-GHZ radio access, AES, PKA, true RNG, DMA channels.

life.augmented

Security illegal accesses

- Any illegal accesses to secure resources can be signaled to the secure Cortex-M0+ thanks GTZC_TZIC
- When enabled an illegal access will wakeup the Cortex-M0+ from any operating mode.
- It is up to the Cortex-M0+ firmware on what action to take.
- Illegal accesses information is available from:
  - Secure/privileged memory areas Flash, SRAM1, and SRAM2.
  - Secure/privileged peripherals DMA, DMAMUX, SUBGHZSPI, AES, PKA and true RNG.
  - Security and privilege control in GTZC and PWR.

life.augmented

Debug disable

- Debug access handled by secure user options
- Controlled by the Debug Disable Secure option (**DDS**)
  - Disable debug port access to the Cortex-M0+.
- Debug can be enabled and disabled in Secure and Non-secure modes.
  - Debug access control is independent from security
  - In secure mode debug access can only be changed by the secure Cortex-M0+ side.

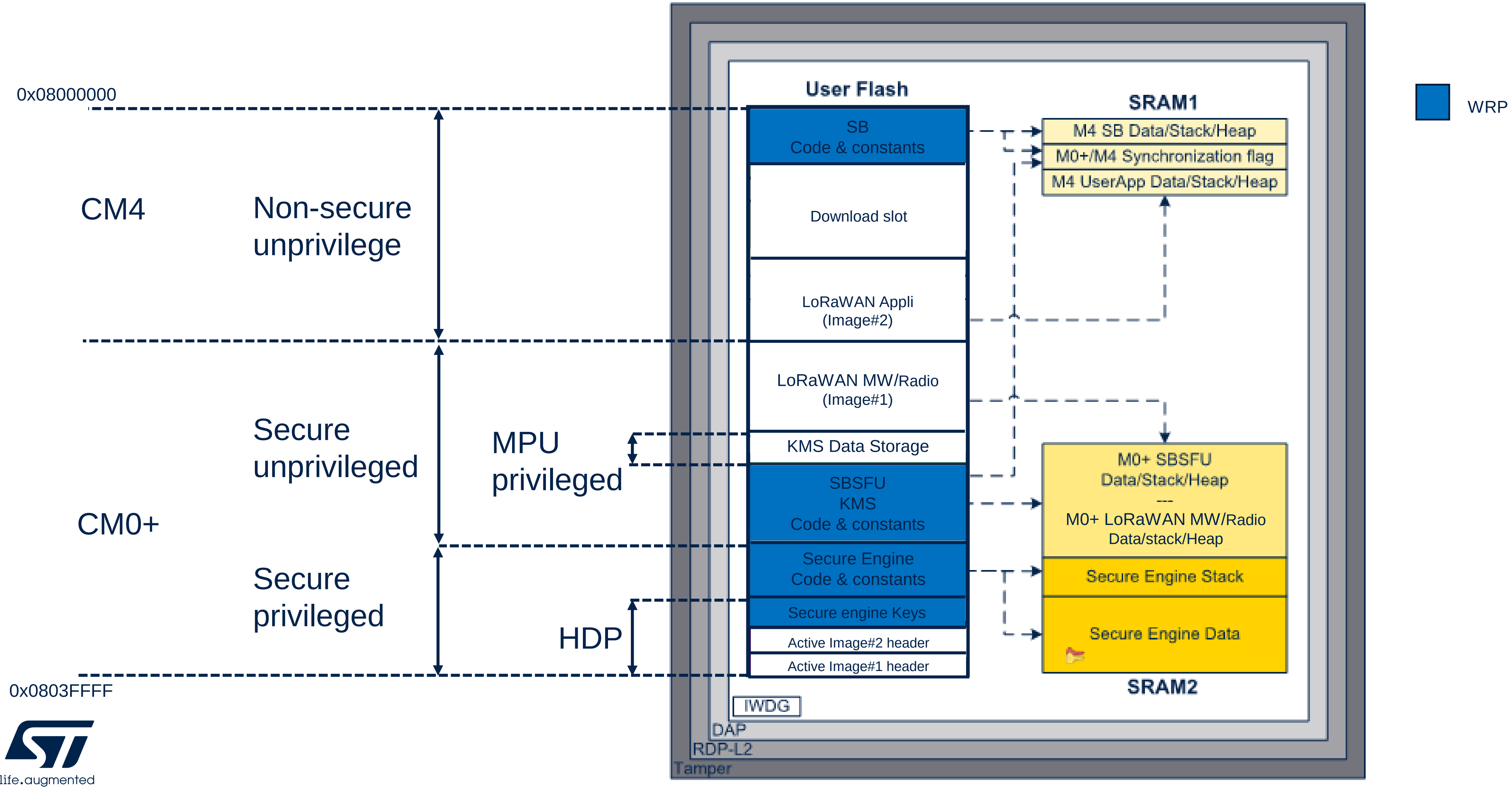
Cortex-M0+/M4 boot lock(chain of trust)

- The Cortex-M0+ boot reset vector is programmed in the Secure Boot Reset Vector (**SBRV**) option.
  - Word-aligned value.
- The Cortex-M0+ may boot from Flash memory or SRAM as selected by the Secure CPU2 option (**C2OPT**).
- CPU2 boot lock (**C2BOOT_LOCK**)
  - This bit allows to lock the boot mode of the Cortex-M0+.
  - SBRV and C2OPT can no longer be modified
- CPU1 boot lock (**BOOT_LOCK**)
  - System boot via BOOT0/BOOT1 from SRAM1 or bootloader or CPU2 SFI/RSS boot is no longer possible.

STM32WL5x

- As on the single core, ST delivers code examples of Secure Boot and Secure Firmware Update which rely on those new security mechanism and in such a way increase the security level of your system
- All the isolation mechanism available has been exploited :
  - HDP
  - MPU
  - Cortex M0+ isolation
- STM32Cube_FW_WL\Projects\NUCLEO-WL55JC\Applications\
  - SBSFU_2_Images_DualCore :
    SBSFU dual slot with local loader and KMS
  - LoRaWAN_FUOTA :
    SBSFU(OTA) with a LoRaWAN application integrated and KMS.

STM32WL5x : SBSFU(OTA) and LoRaWAN



STM32WLEx\STM32WL5x Security comparison

Protection	Type of attack	Mitigate	Protected resources	Integrity	Confidentiality	STM32WLEx HW	STM32WLEx SW	STM32WL5x HW	STM32WL5x SW
RDP	Outer	Avoid any attacks thanks to the debugging Prevent usage of embedded bootloader Prevent option byte modification	All the platform resources	x	x	y	a	y	y
WRP(+RDP)	Outer/Inner	Avoid modification of the secure boot and firmware update code	All the flash region protected (S-BSFU/Keys) Ensure immutability of code and keys.	x		y	a	y	y
PCROP	Outer/Inner	Avoid direct read of the code protected in flash (via debugging link or from core). This could allow to protect key values from direct read in the flash but execution will drive to show values in RAM	All the flash region protected (possibly static keys)	x	x	y	n	y	n
MPU CM4/CM0+	Inner	Allows isolation between privilege and unprivilege mode on the M4 or CM0+ core from any core access. Any DMA could bypass this protection-> need to control DMA configuration possibility.	On CM4, isolation between secure boot and application On CM0+, isolation between RF middleware and KMS storage	x	x	y	n	y	y
Cortex M0+ flash/RAM isolation	Inner	Allows isolation between CM4 and CM0+ flash/RAM. Any code injected on CM4 side can't access CM0+ resources	All the flash/RAM assigned to CM0+ (RF Middleware/Secure Engine/SBSFU M0+/static Keys and KMS)	x	x			y	y
Cortex M0+ TZ privilege/unprivilege	Inner	Allows isolation in flash/RAM inside the CM0+ (and on some specific peripherals) . Any code injected on CM0+ unprivileged side can't access privileged resources	Flash (image header/Static keys/secure engine), SRAM, DMA,AES,PKA and TrueRNG IP	x	x			y	y
Cortex M0+ HDP	Inner	As the portion of flash disappears on software request until the next boot, avoid an injected code to access the protected values.	statics keys/image header	x	x			y	y

y : yes
n : no
a : activable

life.augmented

STM32WLEx\STM32WL5x Security comparison

With both architectures, you ensure integrity and authenticity of the firmware thanks to secure boot but ….

- STM32WLEx security level allows you to mitigate only attacks through debugging link.
  Any successful inner attack can lead to the loss of confidentiality of assets.
  This could be mitigated using MPU mechanism which would increase implementation complexity.

- STM32WL5x security level addresses outer attacks like for the single core, but thanks to the additional isolation mechanisms, it also addresses inner attacks.

life.augmented

Documentation

- UM2767 Getting started with the SBSFU of STM32CubeWL

https://www.st.com/resource/en/user_manual/dm00731353-getting-started-with-the-sbsfu-of-stm32cubewl-stmicroelectronics.pdf

- AN5544 Integration guide of SBSFU on STM32CubeWL (including KMS)

https://www.st.com/resource/en/application_note/dm00725183-integration-guide-of-sbsfu-on-stm32cubewl-including-kms-stmicroelectronics.pdf

life.augmented

Thank you

life.augmented

Key management system architecture

life.augmented

Crypto key management system

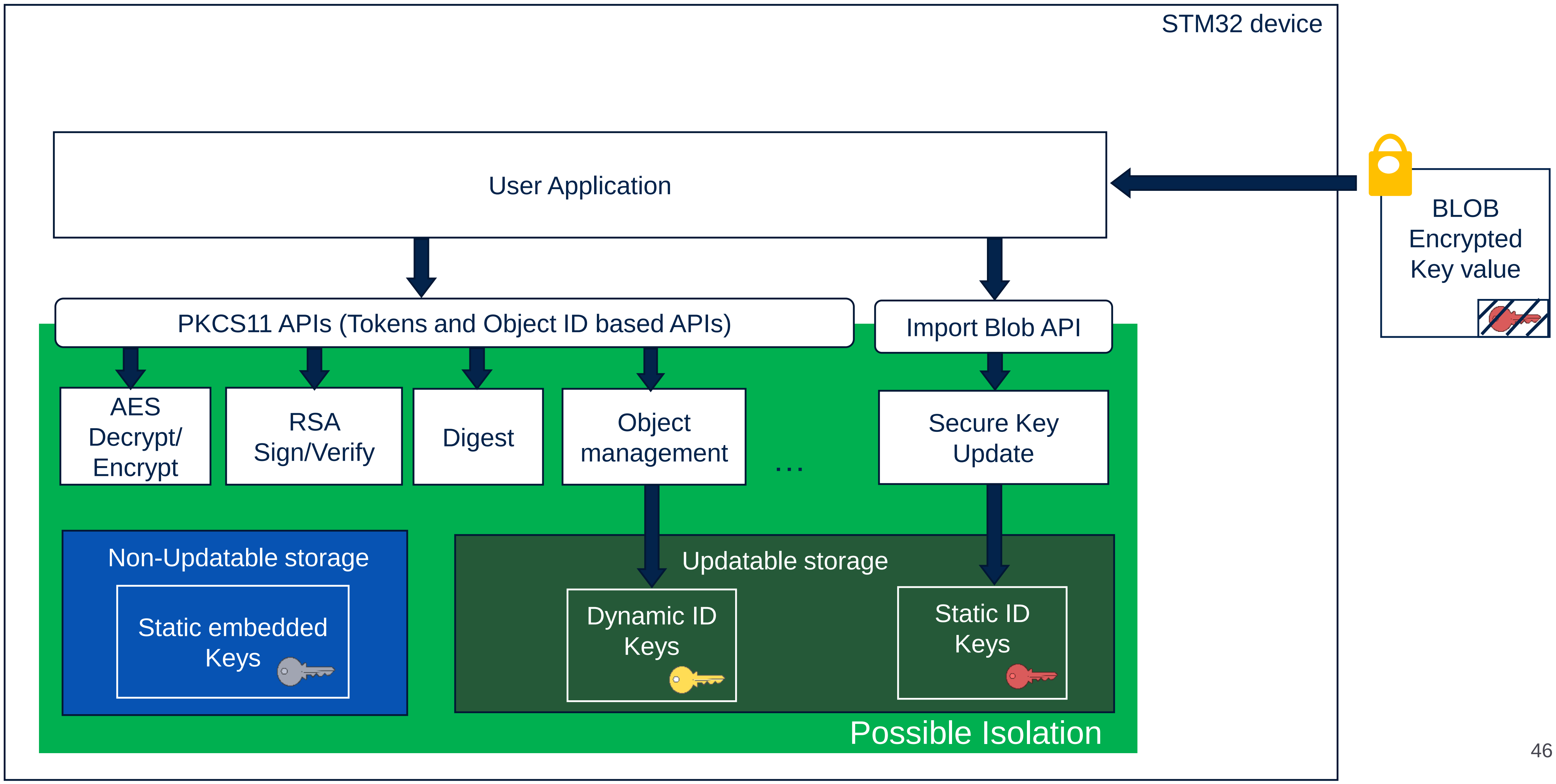
- Key management purpose:
  - Unified the way to manage and use keys (minimize exposition of the value)
  - Standard API PKCS#11 APIs
  - Ease a potential isolation
- KMS : Key management services
  - Object management (create, update, delete)
  - AES encryption/decryption
  - Digest functions
  - RSA sign/verify
  - Key management functions: Key generation/derivation

Crypto key management

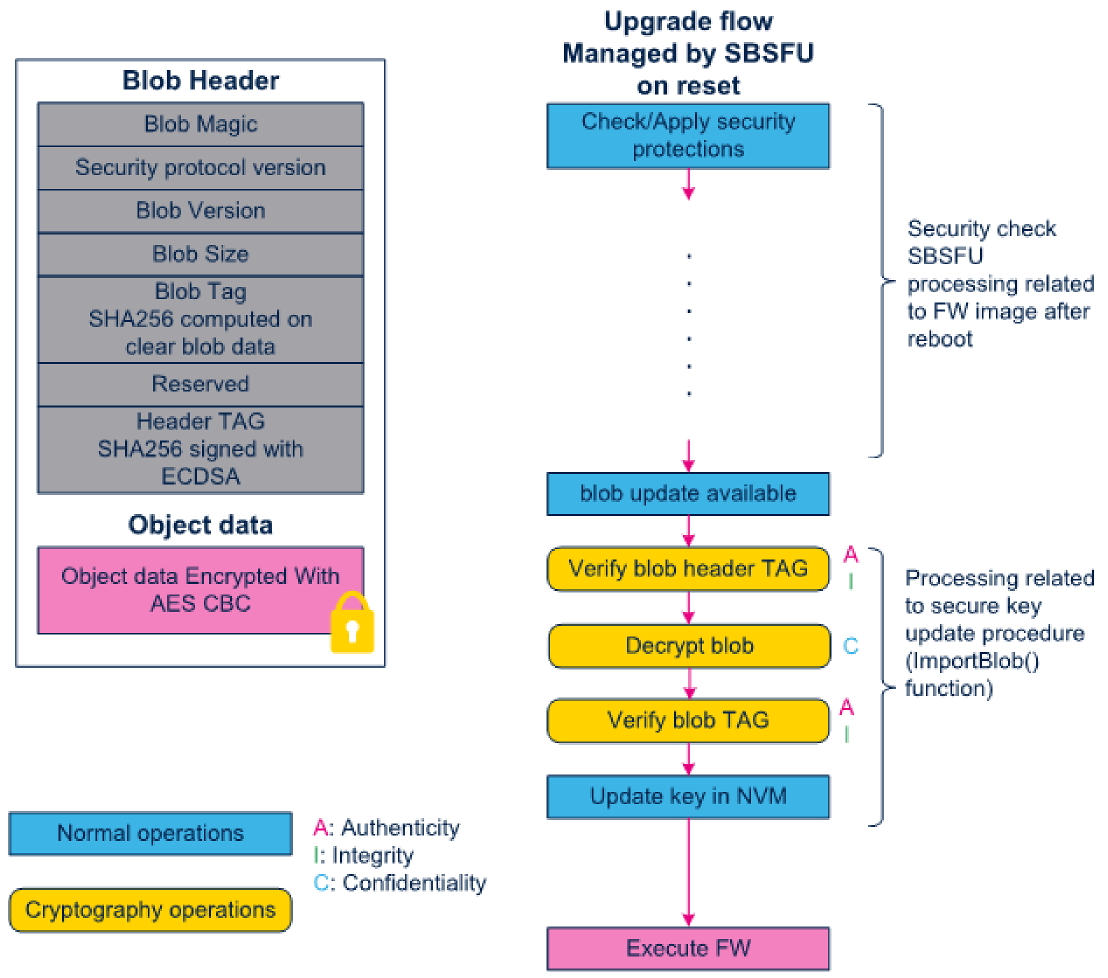
- KMS manages 3 types of keys
  - Static embedded keys
    - **Predefined** keys embedded within the code
    - Such keys **can not be modified**
  - Updatable keys with static ID
    - Key IDs are **predefined** in the system
    - key value **can be updated** in a NVM storage via a secure procedure using static embedded root keys
    - Such keys **can not be deleted**
  - Updatable keys with dynamic ID
    - Key IDs are defined when creating the keys
    - Key value is **created using internal functions**. Typically, the DeriveKey() function creates dynamic objects
    - Such keys **can be deleted**

life.augmented

KMS - Overview



SBSFU-KMS integration for provisioning



Blob Header
Blob Magic
Security protocol version
Blob Version
Blob Size
Blob Tag SHA256 computed on clear blob data
Reserved
Header TAG SHA256 signed with ECDSA
Object data
Object data Encrypted With AES CBC
Upgrade flow Managed by SBSFU on reset
Check/Apply security protections
Security check SBSFU processing related to FW image after reboot
blob update available
Verify blob header TAG
A
I
Decrypt blob
C
Verify blob TAG
A
I
Update key in NVM
Processing related to secure key update procedure (ImportBlob() function)
Execute FW
Normal operations
Cryptography operations
A: Authenticity
I: Integrity
C: Confidentiality

- Loader will flash the Blob binary in the download memory slot
- SBSFU will detect the new blob and will call the KMS API to install this new keys in a secure way
- 3 mains steps :
  - Blob header authenticity and integrity check
  - Blob data decryption
  - Blob data authenticity and integrity check

ST KMS delivery

- KMS code :

  STM32Cube_FW_WL\Middlewares\ST\STM32_Key_Management_Services

- KMS usage code example :

  STM32Cube_FW_WL\Projects\NUCLEO-WL55JC\Applications\KMS

  - KMS_Blob_Binary
  - KMS_Blob_Example
  - KMS_Derive_Key
  - KMS_Embedded_AES_Keys
  - KMS_Embedded_RSA_Key

  STM32Cube_FW_WL\Projects\NUCLEO-WL55JC\Applications

  - BFU_1_Image
  - SBSFU_2_Images_DualCore
  - LoRaWAN_FUOTA

life.augmented

Thank you

life.augmented

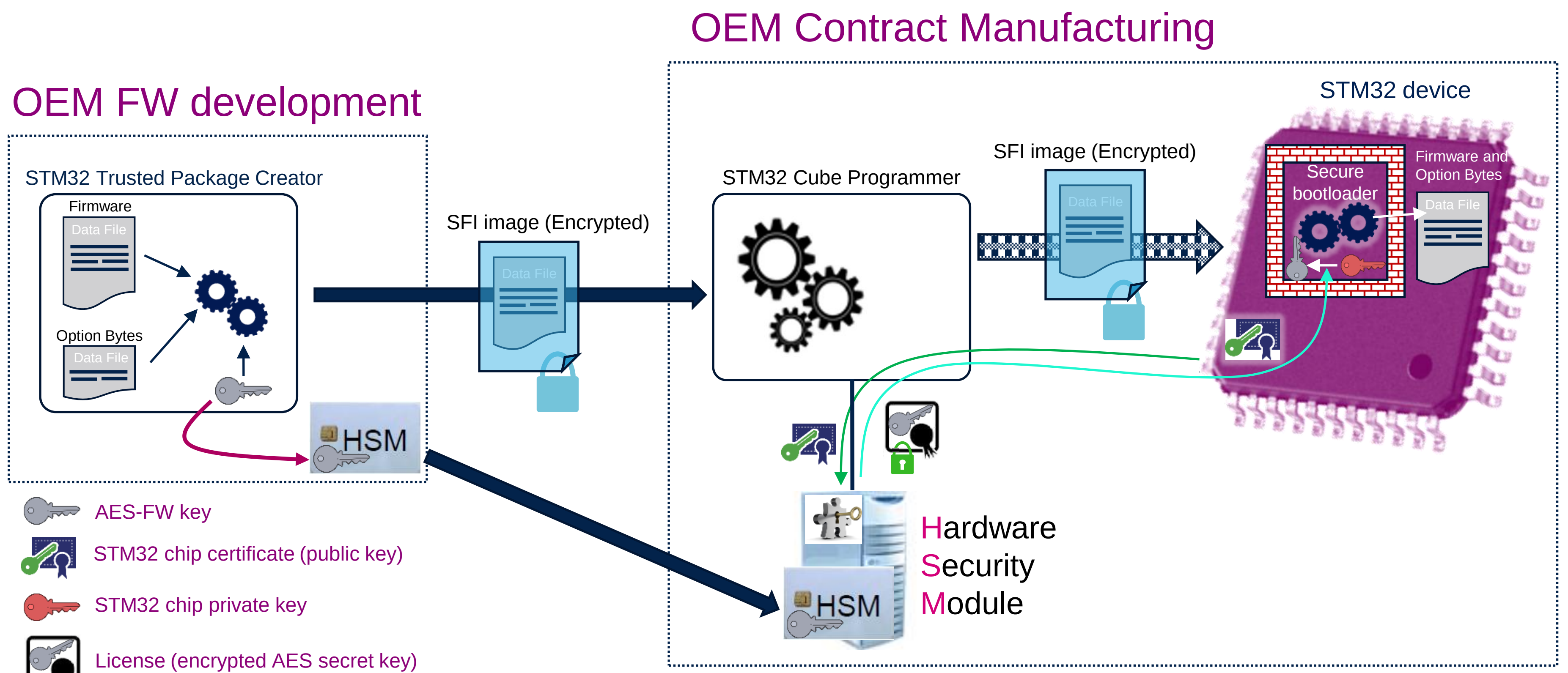
SFI

life.augmented

SFI overview

- Secure Firmware install purpose :
  - Allows to program a first encrypted FW in an unsecure environment
  - Allows to avoid over production (grey market)
- It relies on preprogrammed keys inside the chip at ST production level
- SFI only available on dual core
- SFI does not manage diversity

SFI process « big picture »



AES-FW key

STM32 chip certificate (public key)

STM32 chip private key

License (encrypted AES secret key)

HSM STM32HSM smartcard

life.augmented

Thank you

life.augmented

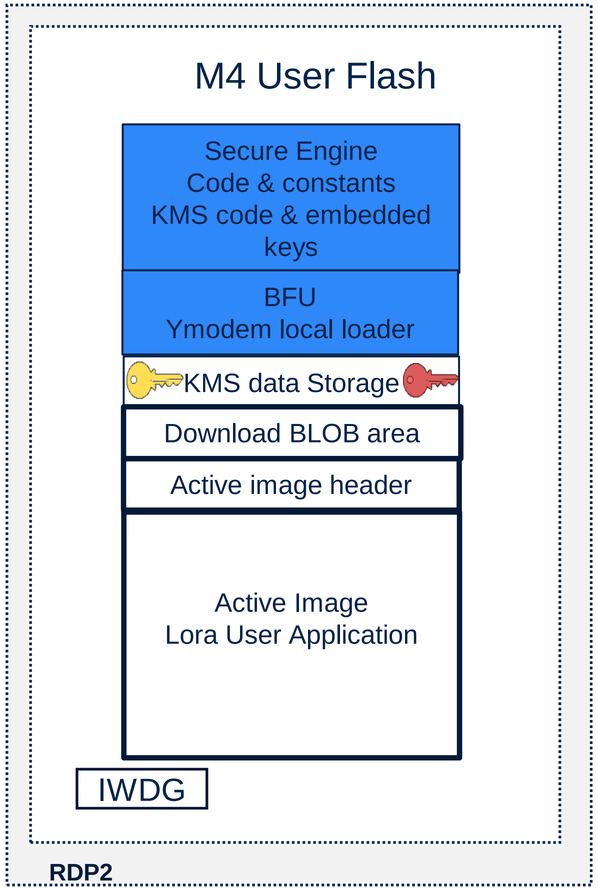
Demo : BFU-KMS provisioning

life.augmented

Demo: Secure key provisioning

- ST delivers a code example of a SBSFU including KMS
- for single core architecture
  STM32Cube_FW_WL_V1.0.0\Projects\NUCLEO-WL55JC\Applications\BFU_1_Image
- Features :
  - Secure boot
  - Secure firmware update with local loader using UART/Ymodem
  - KMS
    - Key storage containing static keys
      - Firmware keys (Public key for firmware signature check and AES key for firmware encryption)
      - Blob keys (Public key for blob signature check and AES key for blob encryption)
    - Application key provisioning through ImportBlob feature

BFU / KMS Mapping



🟦 WRP

Thank you

life.augmented

For further support in creating a PowerPoint presentation, including graphic assets, formatting tools and additional information on the ST brand

you can visit the ST Brand Portal
https://brandportal.st.com

life.augmented

Security related examples in STM32Cube_FW_WL_V0.7.0

life.augmented

Applications\KMS

- KMS_Blob_Binary - Generates the blob = encrypted key with some meta data
- KMS_Blob_Example – Runs on CM4. No isolation.
- KMS_Blob_Example_DualCore
  CM4 – Empty project. CM4 only allows booting of CM0+
  CM0+ – Runs KMS and loader. Allows to receive the blob over YMODEM
- KMS_Derive_Key
- KMS_Embedded_AES_Keys
- KMS_Embedded_RSA_Key

life.augmented

Applications\SBSFU_2_Images_DualCore

- 2_Images_KMS_Blob
- 2_Images_SBSFU
  - CM4   Loader + Jump to new app
  - CM0+ SBSFU
- 2_Images_SECoreBin – runs all KMS services
- 2_Images_UserApp_M0Plus - Loader + access SBSFU+KMS services. Runs in unprivileged
- 2_Images_UserApp_M4 - LED blinking

life.augmented

Applications\LoRaWAN_FUOTA\2_Images_KMS_Blob