Hello and welcome to this presentation describing the secure data storage features of the STM32H5.
It will cover the main features of the STM32H5, that enable users to protect sensitive data. It is part of a set of presentations, that describe the security mechanisms offered by the STM32H5 and related secure firmware.

Agenda

- # STM32H5 Secure Data Storage
- # Secure Data Storage – Flash interface access control
- # Secure Data Storage – Data Encryption with SAES/DHUK
- # Secure Data Storage – Combining flash interface access control and SAES/DHUK
- # Secure Data Storage - Provisioning

life.augmented

2

The following topics are going to be explained:
First, an overview of the STM32H5 secure data storage features.
Second, the flash interface access control, that filters out requests that violate access permissions.
Third, the combining of flash interface access control and Secure AES co-processor (SAES) with Derived Hardware Unique Key (DHUK) to encrypt / decrypt sensitive data.
Fourth, the provisioning of keys and firmwares.

STM32H5 Secure Data Storage
Flash interface – Option Byte Keys

- Secure Data Storage
  - High level of protection of data thanks to:
    - Flash Interface → Option Byte Key access control
      - Up to 5 domains
    - SAES
      - DHUK per domain isolation
      - RHUK: Anti-cloning
      - EPOCH for Anti-replay

life.augmented

The Flash interface implements the logic necessary to carry out the Flash memory operations (program/erase) controlled through the Flash registers plus security access control features.
The embedded flash memory includes a set of nonvolatile option bytes. They are loaded at power-on reset and can be read and modified only through configuration registers.
A new area called Option Byte Keys is used to store not only cryptographic keys, but also any kind of information (mapping, configuration, etc.).
The secure storage is tied with SAES and OBK-HDPL value set by the System Configuration, Boot And Security (SBS).
In order to enforce temporal isolation: boot levels are

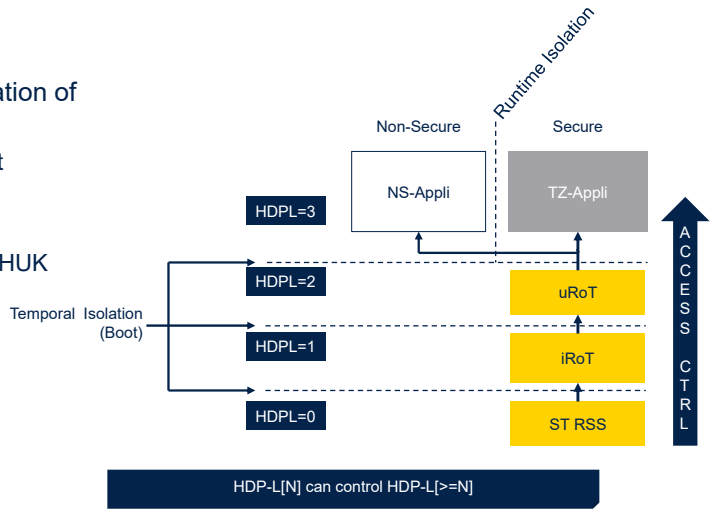
isolated thanks to the Hide Protect Level (HDPL) monotonic counter.
The secure storage features:

- Five nonvolatile areas dedicated to secure storage. Areas are protected with HDPL and TrustZone
- Battery-powered volatile secure storage, automatically erased in case of tamper
- Write-only key registers in the AES engines – Device 96-bit unique ID and JTAG 32-bit device-specific ID
- Secure storage can rely upon SAES engine to encrypt the stored data, to benefit of Derived Hardware Unique Key (DHUK) properties.

Data are isolated: all data encrypted/decrypted relying upon SAES+DHUK benefit of the variation of the DHUK for each different isolated area (HDPL0, HDPL1, HDPL2, HDPL3S, HDPL3NS).
The Root Hardware Unique Key (RHUK) is a non-volatile hardware secret unique per device, allowing anti-cloning protection.
All keys encrypted using the SAES/DHUK inherit of the RHUK property: unique per device.
The EPOCH is a counter incremented each time a regression is done, allowing antirollback protection.

STM32H5 Secure Data Storage

- Secure Data Storage
  - The Secure Data Storage allows isolation of data
    - Following HDPL rules: HDPL[N] cannot access HDPL[N-1]
    - Including access control
    - Including data encryption with SAES/DHUK



life.augmented

4

This figure explains the temporal isolation based on Hide Protection Levels (HDPL), which is implemented as a monotonic counter: it can only be incremented.
The various programs involved in boot-time and run-time are presented on the right, chronologically, from bottom to top:

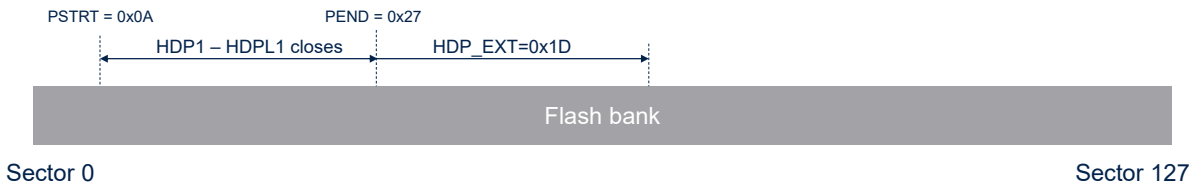
- The ST Root Secure Service (RSS)
- The ST Immutable Root Of Trust (iROT)
- The ST Updatable Root Of Trust (ST-uROT) or proprietary one (uROT)
- The Secure and non-secure applications.

Each of these programs is assigned an HDPL.
Access to the hide protection area can be denied by progressing the HDPL level in the System configuration,

boot and security (SBS).
For example, the part of the flash containing the RSS becomes hidden when the RSS transitions the HDPL from 0 to 1.
The HDPL level can be only cleared by a system reset, there is no means to deactivate the area protected by HDP.
The protected HDP area is defined by setting its size using start and end sectors in a similar way as the secure watermark.
The current HDPL also determines the access control rules and keys used for encryption / decryption.

STM32H5 Secure Data Storage
HDP control

- Each bank has independent controls
- HDPL1 HDP area is programmed in User OB by start and end sector
- HDP area can be extended with several sectors using FLASH_HDP_EXT
  - Register value starts as 0 on reset
  - New written value must always be same or larger than previous
  - Extension is set by HDPL=2 code and prevents access from HDPL=3



life.augmented

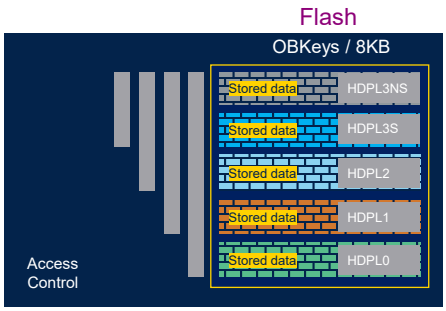
5

Secure Hide Protection (HDP) is controlled by on Hide Protection Levels (HDPL), used for Arm Platform Security Architecture (PSA).
Three levels are available to facilitate secure boot process stages isolation.
Each level-increase blocks access to the lower levels, which only become available again on next reboot.
Same mechanism works on smaller scale in dedicated key storage mechanism called the OBK.
The area associated with HDPL 1, which is blocked by incrementing to level 2, is configured using option bytes.
The threshold between HDPL 2 and HDPL 3 is a volatile setting in flash register, called the HDP extension.
If the two banks are swapped, the protection defined to

physical Bank1 remains on the physical Bank1, unaffected by swapping

Flash interface – Option Byte Keys

- The Flash interface include a new area called Option Byte Keys (OBKeys)
  - 2* 8KB sectors (9-bit ECC) supporting swapping
  - Only accessible by Secure Privileged programs
  - Access control to 5 domains: OBK_HDPL+EPOCH_SEL
    - HDPL0 (255 Bytes),
    - HDPL1 (2047 Bytes),
    - HDPL2 (767 Bytes),
    - HDPL3S (3071 Bytes),
    - HDPL3NS (2031 Bytes)
  - Regressions
    - HDPL3NS = Automatically (by ST-DA) erased with NS-Regression
    - HDPL1,2,3S, 3NS = Automatically (by ST-DA) erased with Regression
    - HDPL0 never erased

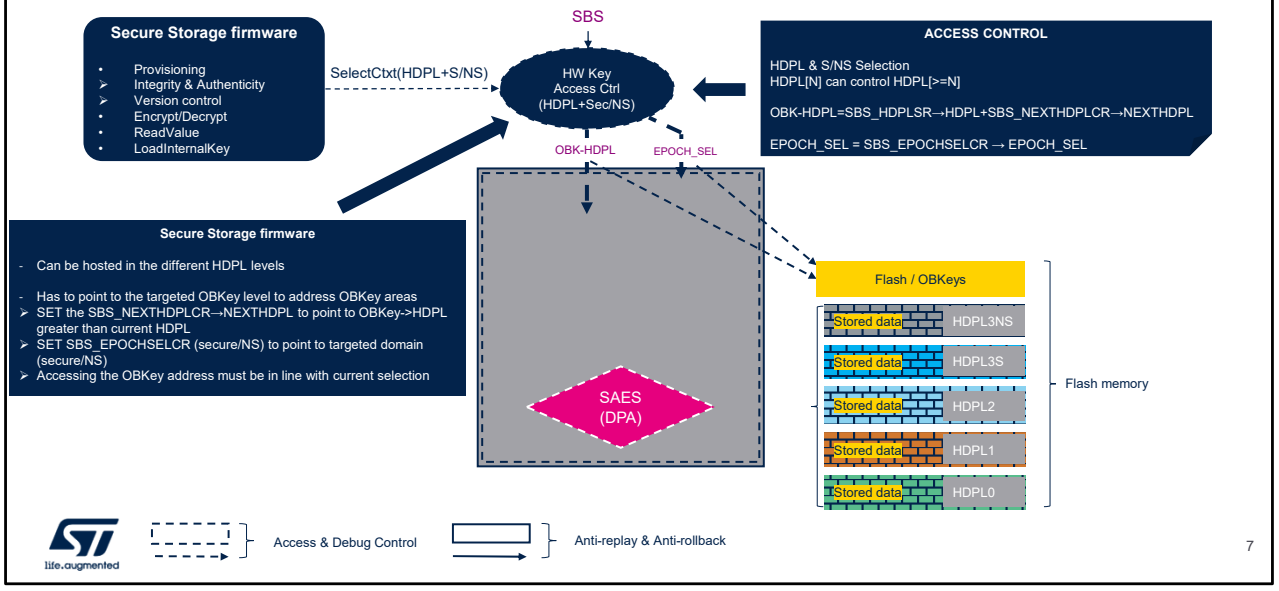


The embedded flash memory includes two memory sectors (2 x 8 Kbytes) of secure storage, OBKey only accessible by secure privileged state.
OBKEY storage can contain any sensitive information, keys of course, but also data, such as mapping information, flags, etc.
Contents are protected with an 9-bit Error Correcting Code (ECC).
Commands enable the user to swap keys from one sector to the alternate sector.
The OBKey storage is split into five domains, for which access control is determined by current HDPL and EPOCH value, sent to the SAES module.
Note that HDPL0 is never accessible to the user

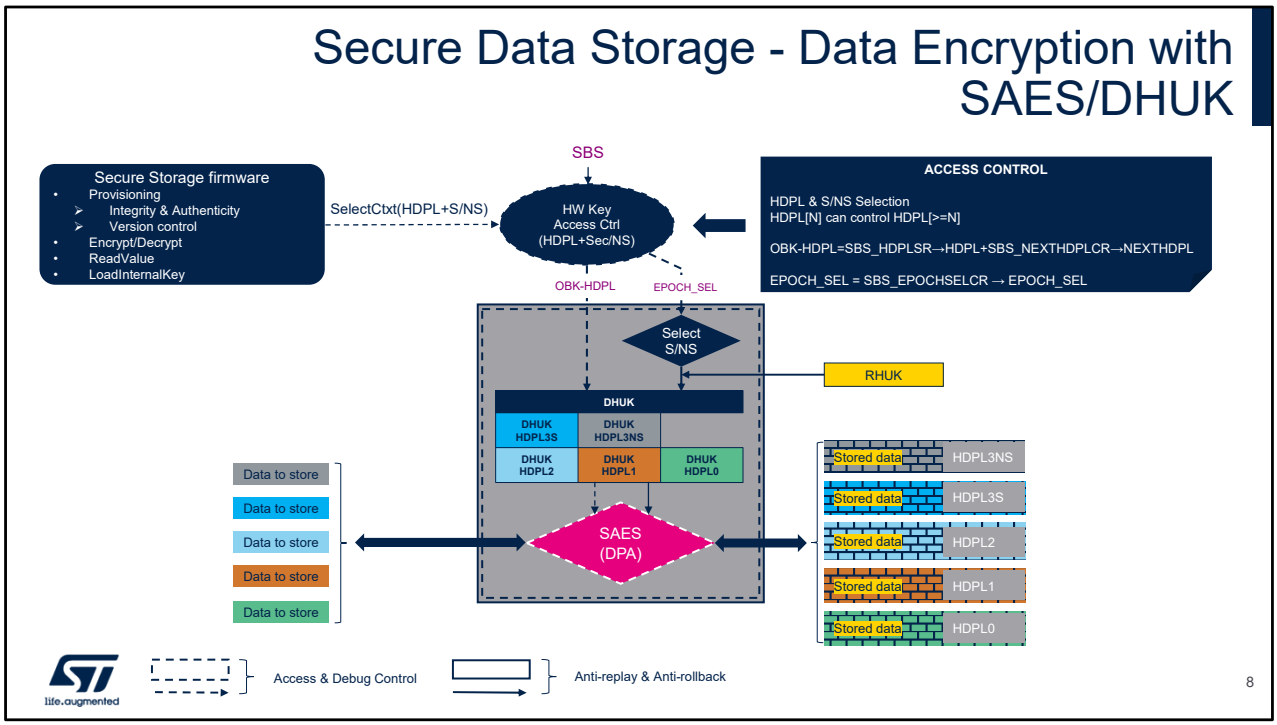
application; it is reserved for ST application.
Regression from one version to a lower one requires the erasure of the keys related to the lower level.
A key feature is that data are never disclosed when opening access with debugger. A partial or full regression is necessary to erase and reprogram the data.
This is done automatically by the ST Debug Authentication (DA) library, that manages the debug authentication protocol by allowing to securely reopen the debug or to launch regressions on secured products in the field.
HDPL0 OBKeys are never erased.
The figure explains the temporal isolation feature:

- A code running in HDPL0, can access all HDPLs
- A code running in HDPL1, can access HDPL1 to 3
- A code running in HDPL2 can access HDPL2, 3
- A code running in HDPL3 secure can access HDPL3 Secure + Non-Secure.

Secure Data Storage - Data Encryption with SAES/DHUK



A Secure Storage firmware can be developped to be used in the different HDPLs: HDPL0,1,2,3
It is key to understand that when such code is executed in a particular HDPL level, if it needs to access another level, then it has to explicitly commit this transition, by programming NEXTHDPL (compare to current), and also the EPOCH to select Secure/Non-Secure.
This selection is used for Flash Interface OBKey access control, but also used by the SAES to select different DHUK.



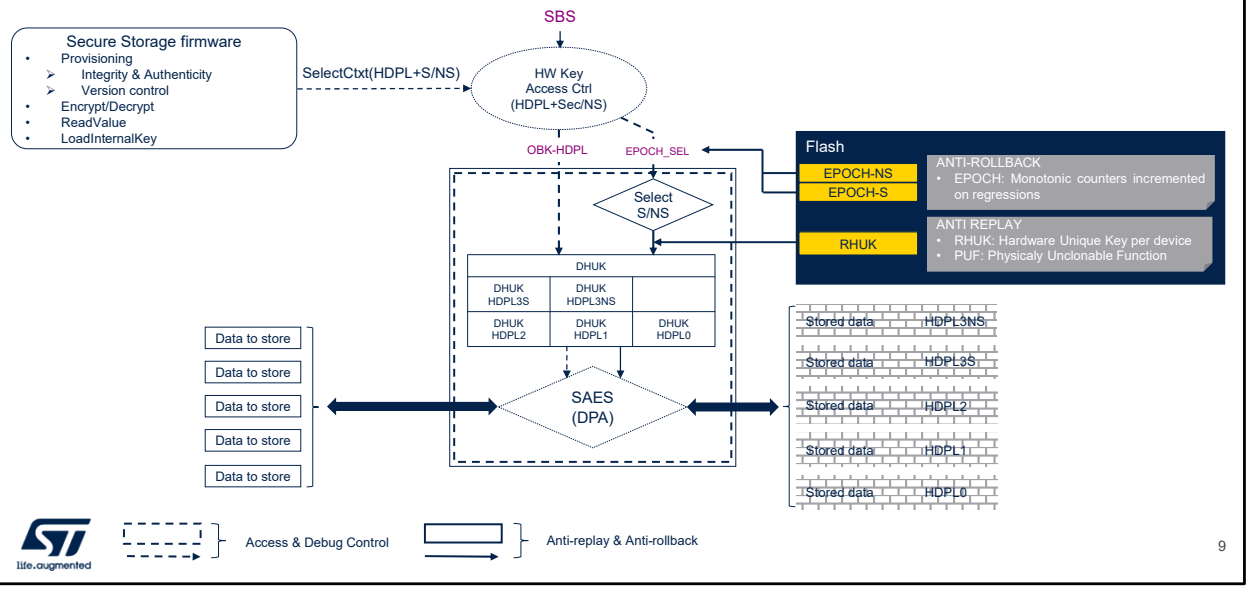
This figure represents the data protection performed by the flash security mechanisms, based on SAES and DHUK.
To distinguish two types of protection mechanisms, the following convention is used:

- Dotted lines identify mechanisms related to access and debug control
- Solid lines identify mechanisms related to anti-replay and anti-rollback.

The secure storage firmware selects the context composed of HDPL and Secure / Non-secure state, which is passed to the hardware key access control unit.
This unit selects the OBK-HDPL and EPOCH according to this current context and SBS register settings.
OBK-HDPL selects the area of secure storage of the Flash

memory and selects the corresponding DHUK in SAES. The EPOCH selector has to be set by the application wishing to use the secure storage.
sbs_epoch_s and sbs_epoch_ns are 24-bit values coming from the Flash memory and representing regression counters respectively for secure and non-secure states. Then, data to be stored to flash memory will first be checked against accessible domains and if no violation is detected, are encrypted according to the selected DHUK.

Secure Data Storage - Data Encryption with SAES/DHUK



This slide focuses on data protection (SAES + DHUK) to support anti-roll-back and anti-replay attacks.

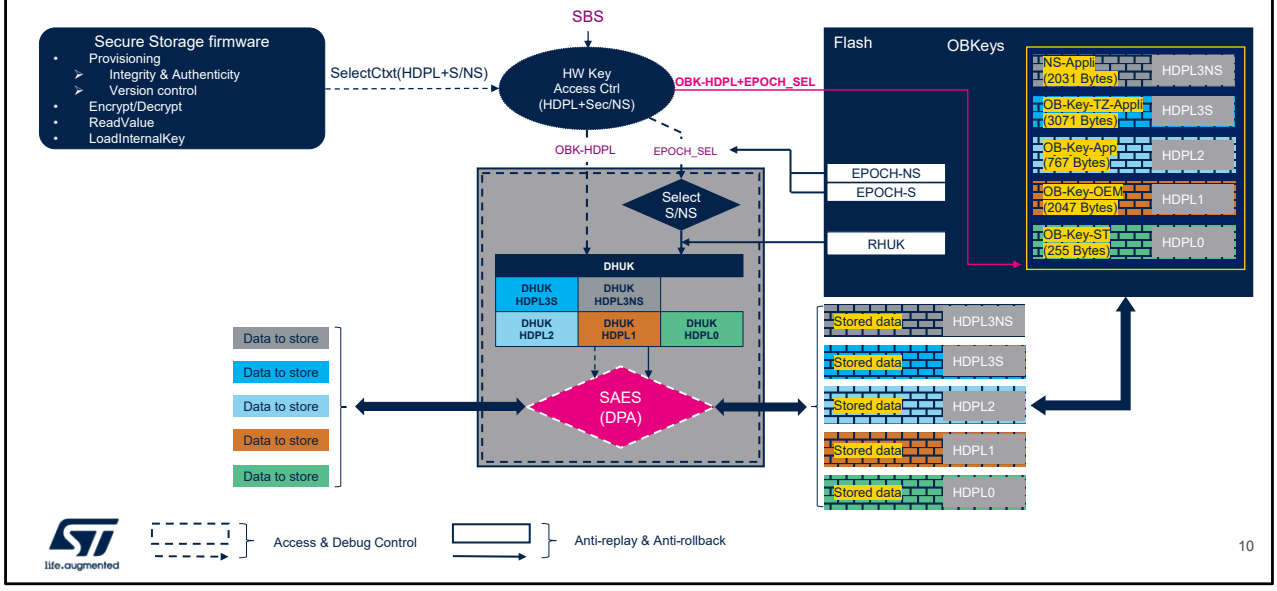
- The Anti-replay protection is guaranteed thanks to the use of the RHUK (Root Hardware Unique Key) to process the SAES-DHUK.
- The Anti-rollback protection is guaranteed thanks to the use of one of the 2 EPOCH counters to process the SAES-DHUK.

- The RHUK is
  - Uniquely programmed by ST in the system flash memory and cannot be erased or reprogrammed.
  - It is a 256 bits value used as an entry by the SAES, to process the DHUK.
  - It allows anti-replay feature, as all data encrypted with

DHUK will have different values compared to any other devices.

- Two EPOCH counters are present:
  - One for Non-Secure (EPOCH-NS) and one for Trust-Zone (EPOCH-S)
  - They are only accessible by the Debug Authentication,
    - EPOCH-NS: automatically incremented on partial-regression and full-regression
    - EPOCH-S: automatically incremented on full-regression
    - Debug Authentication guarantees the incrementation of this counters
  - EPOCH value is used in the process of the DHUK (depends on epoch-select)
  - These counters allow the support of anti-rollback, as even if data are not erased, they will be no more retrievable

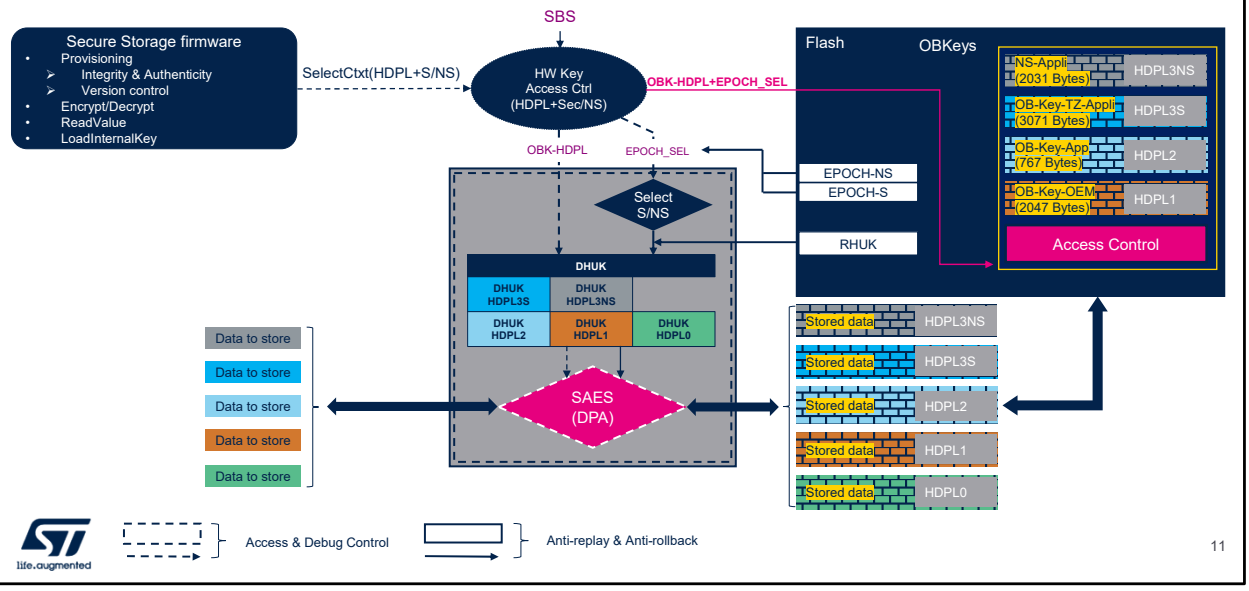
Combining Flash interface access control and SAES/DHUK



In this slide, we show the combination of the 2 isolation mechanisms: ACCESS CONTROL (thanks to the Flash) and DATA PROTECTION (SAES+DHUK).
The current HDPL is used by the flash memory to select the right area, and by the SAES when processing the derived key to generate a key derivation related to HDPL context.
This figure highlights the access control mechanism present in the flash controller.
Option byte keys have to be provisioned in locations of the flash depending on the current HDPL value.
Increasing the HDPL hides the locations associated with lower HDPLs.
Data to be programmed are encrypted by SAES, that

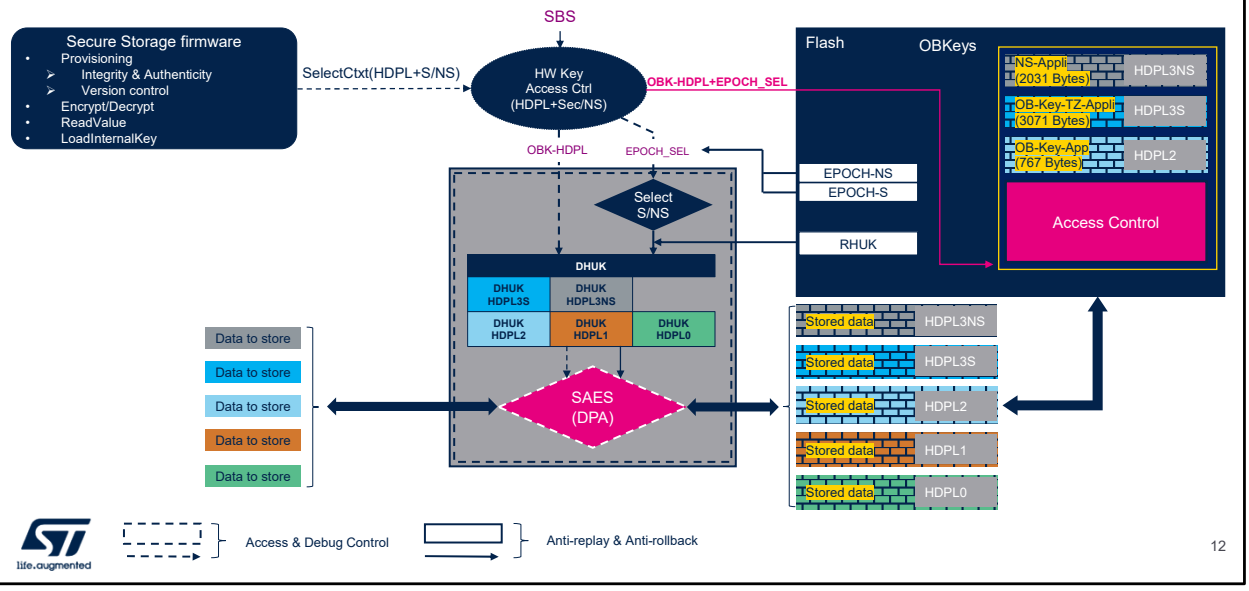
receives the DHUK associated with the current HDPL.
This slide and the following three explain the OBKey temporal isolation mechanism.
Assumption for this slide: HDPL0 access control: all the OBKeys are accessible.

Combining Flash interface access control and SAES/DHUK



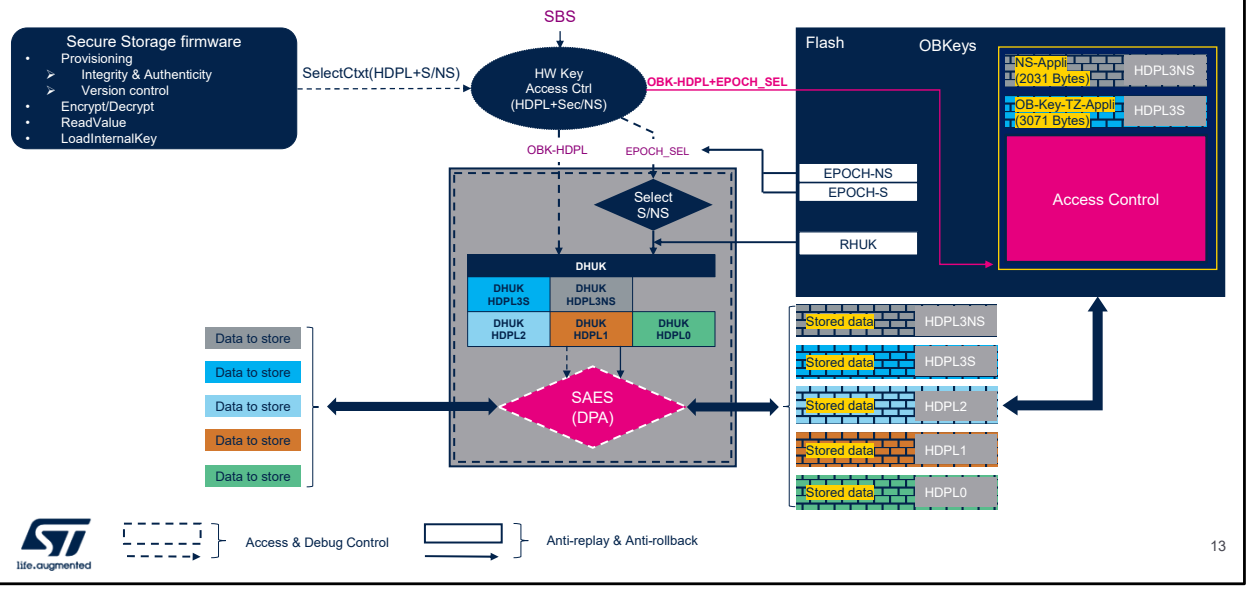
Assumption for this slide: HDPL1 access control: only OBKeys HDPL1 to 3 are accessible.

Combining Flash interface access control and SAES/DHUK

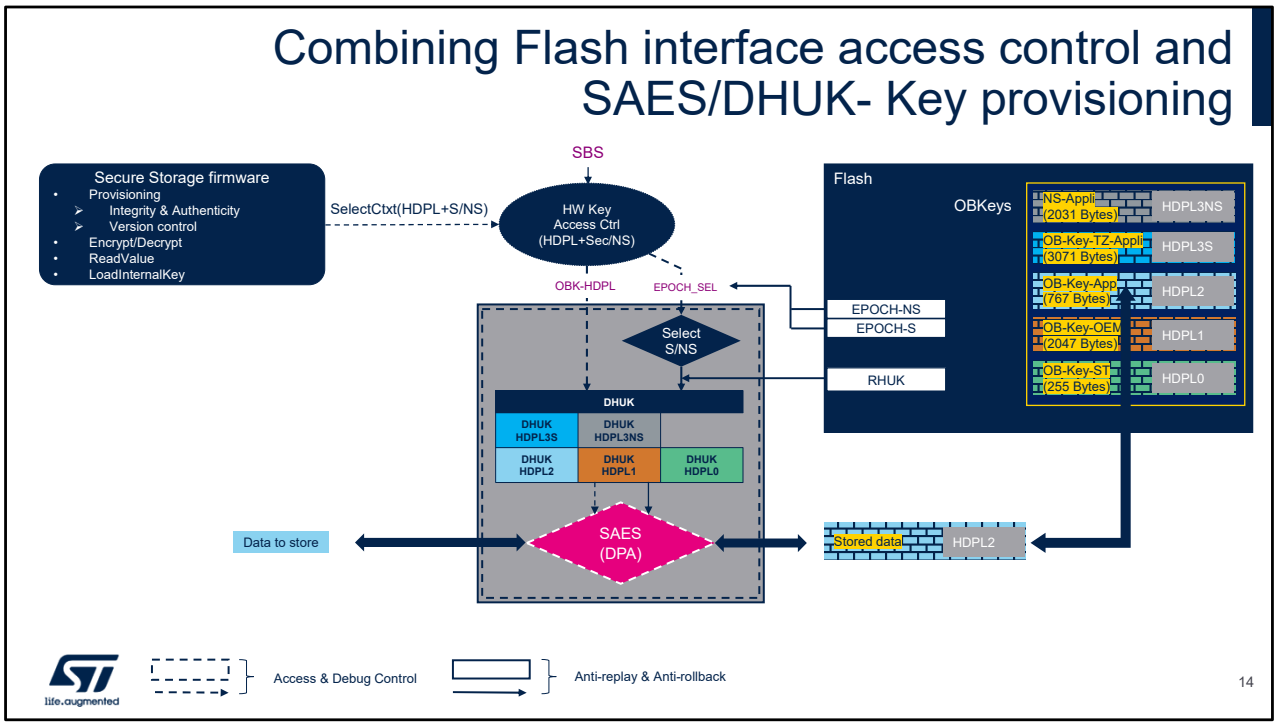


Assumption for this slide: HDPL2 access control: only OBKeys HDPL2 to 3 are accessible.

Combining Flash interface access control and SAES/DHUK



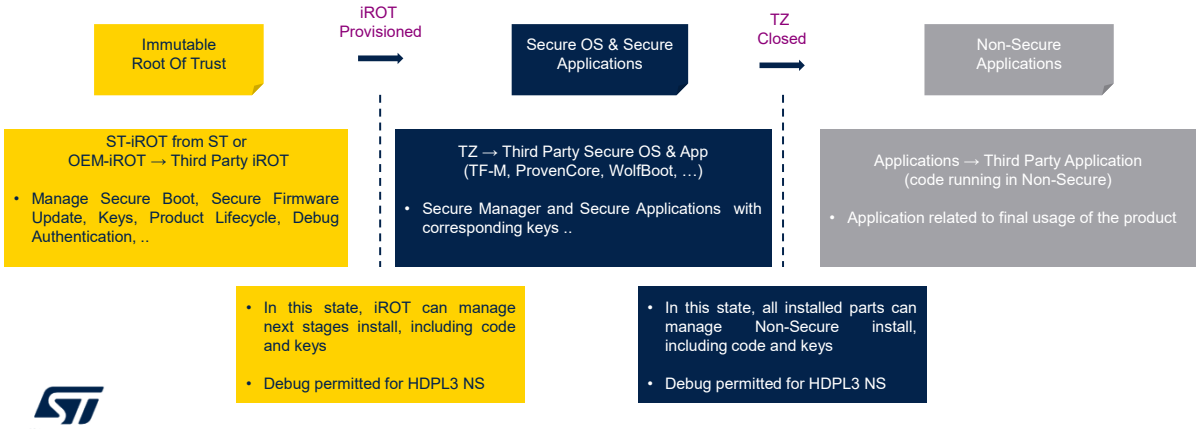
Assumption for this slide: HDPL3 access control: only OBKeys HDPL3 are accessible.



An option-byte key can be accessed only if the OBK-HDPL (set in SBS) matches the HDPL associated to the storage offset. If it is not the case, an OBKERR error is raised
This figure describes the OBkey programming sequence.
We assume that the current HDPL level is two.
Thus, only the HDPL2 and HDPL3 areas of the OBKey memory can be accessed.
As previously explained, an OBKEY can be any sensitive information, not only a key. It is encrypted according to the current DHUK prior to being programmed in the related region of the OBKey memory.

Provisioning

- ST Lifecycle consider 3 main bricks to be installed in the product that could rely on 3 different parties



The STM32H5 life-cycle is designed to support provisioning of the product with up to 3 different parties.

- The First party is the Immutable Root-of-Trust (iROT).
- The Second party is the secure operating system and secure applications.
- The Third party is the non secure application.

We are differentiating the Initial provisioning, from the full product provisioning.

Typically, for an install with 3 parties, we can consider:

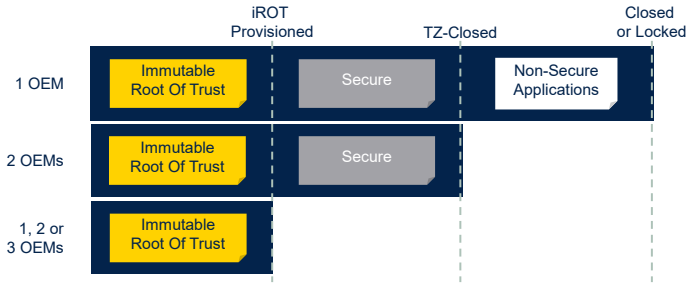
- iROT (firmware and data) to be provisioned in Provisioning PRODUCT_STATE, then to move to iROT-Provisioned.
- Then Secure OS & Secure application being installed (update mechanism of iROT), then PRODUCT_STATE

passed in TZ-Closed

- Then the Non-Secure application being installed (update mechanism of uROT (part of SecureOS)), then PRODUCT_STATE passed in Closed.

Provisioning

- Initial provisioning (from Open or Provisioning)



- We recommend to use SFI for untrusted manufacturing
- We recommend to protect all key provisioning, even if the manufacturing is trusted

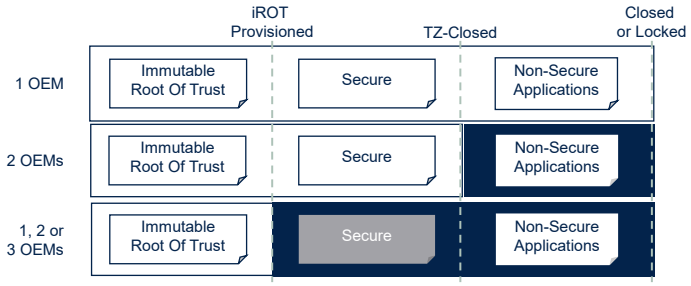
life.augmented

Initial provisioning is done when the life cycle is either open or provisioning.
STM32H5 products are delivered in Open state. This state is useful for development but cannot be used for products.
The provisioning state allows to manage the provisioning of the product. It allows to launch secure firmware Install, or bootloader to provision the product.
The first case considers that one unique OEM is responsible for the full firmware. All can be provisioned in only one step.
The second case considers that two different parties are responsible for two parts: the first party is responsible for iROT + Secure, then second party is responsible for the non-secure application.

The third case is an initial provisioning step including only the iROT (STiROT or OEMiROT) covered by a first third party, but the rest of the system is loaded by zero, one or 2 other parties.
The Secure Firmware Install (SFI) is an immutable secure service embedded by STMicroelectronics in the devices. The SFI allows secure and counted installation of OEM firmware in an untrusted production environment (such as OEM contract manufacturer).
The confidentiality of the installed images written either in the internal flash memory or encrypted in an external flash memory, is also protected, using the AES.
ST recommends to protect all key provisioning, even if the manufacturing is trusted.

Provisioning

- Completing the initial provisioning



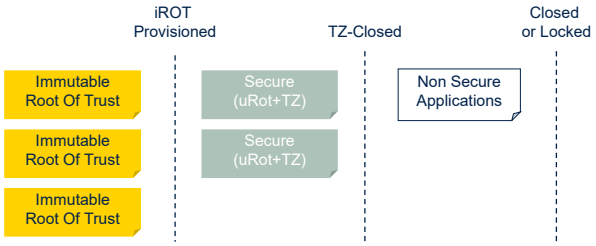
- Provisioning done based on firmware(s) already installed at initial provisioning
  - Secure Firmware Update / Secure Key Provisioning
    - In one or 2 steps

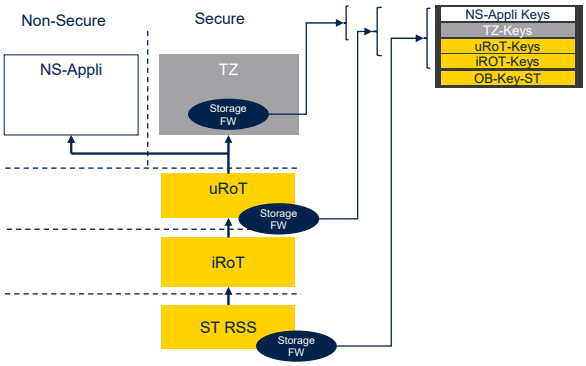
life.augmented

The figure highlights the contribution of second and possibly third OEM regarding the provisioning of firmwares and keys.
In the second case, a second OEM provides the non-secure image.
In the third case, a second OEM provides the secure image and keys, while a third OEM provides the non-secure image and keys.
This provisioning relies on services offered by the firmware installed during the initial provisioning.
These services enable the second and third OEM to update the firmware and to provision keys, which can be achieved in one or two steps.

Provisioning

- Initial provisioning
  - Done by SFI
    - Untrusted manufacturing
  - Debug interfaces or bootloader
    - Trusted manufacturing



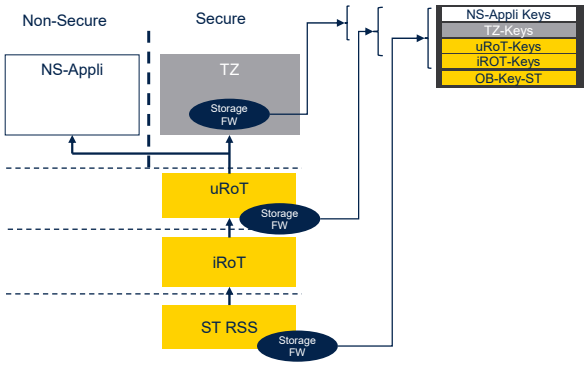


In case of untrusted manufacturing, initial provisioning should be performed through SFI.
In case of trusted manufacturing, a debugger or a bootloader can take care of this initial provisioning.
The difference between the two approaches is the encryption of all firmwares, OBKeys and Option Bytes, when manufacturing is untrusted.
Image install requires encryption and therefore keys.
Thanks to the HDPL and KeyStorage mechanisms:

- The HDPL0 can provision all HDPLs,
- HDPL1 can provision from HDPL1 to HDPL3,
- HDPL2 can provision from HDPL2 to HDPL3
- HDPL3 can only provision HDPL3.

Provisioning

- Hardware secure storage / Use cases
  - For application (runtime isolation)
    - Secure storage firmware in secure state
      - Isolated in Secure Privileged
      - Provide services to Secure Non-Privileged
      - Provide services to Non-Secure
  - Key provisioning (see next slides)
    - Key provisioning using application services
    - During OEM product manufacturing (OEM-Provisioning state)



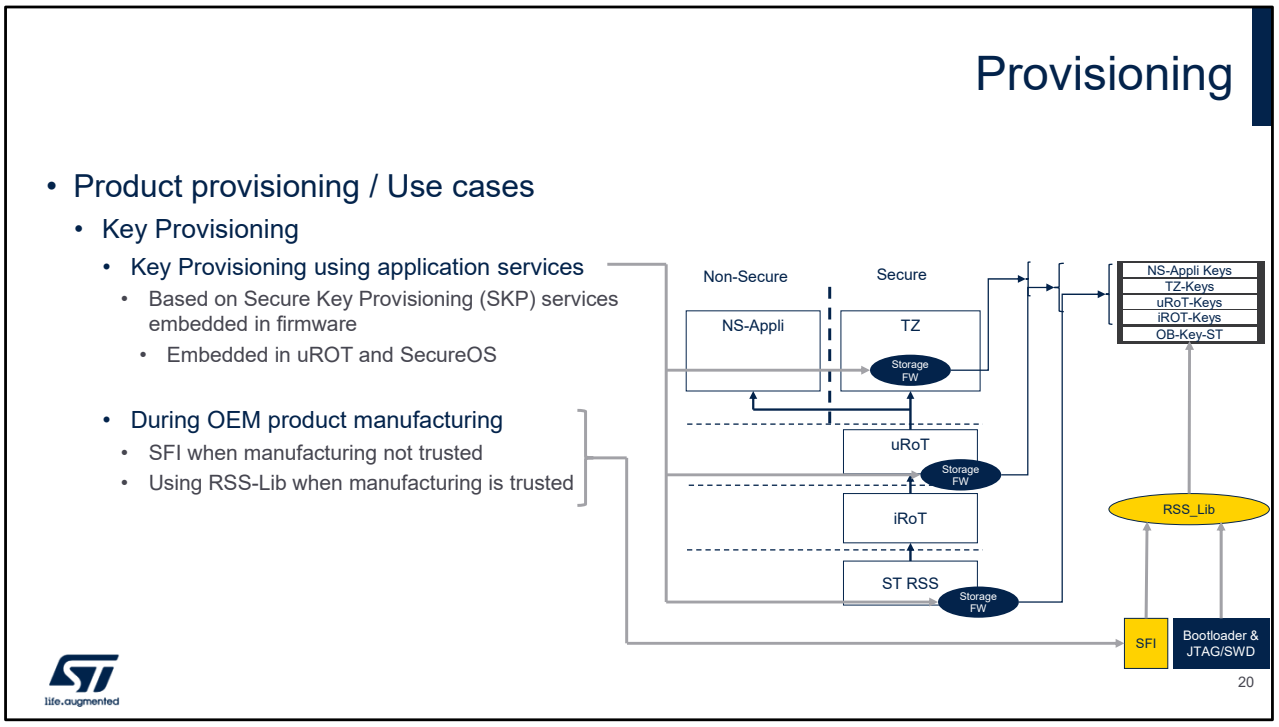
life.augmented

19

Hardware secure storage control improves the security, by isolating programs running at different privilege and security levels.
For example, the secure storage firmware that runs in secure privileged state provides services to:

- Secure non-privileged applications
- Non-secure kernel and applications.

Hardware secure storage also protects OBKeys against accesses and utilization from lower level HDPLs.
It also enables secure provisioning of firmware, by different OEMs.



This figure illustrates the various ways of provisioning keys.
First of all, let's generalize: when we are talking about keys OBKeys we are also considering Data (Mapping, configuration, status, ….).
Keys & Data Provisioning can cover Initial Provisioning but also updates.

- Initial Provisioning considers 2 kind of environments
  - Trusted environment: Provisioning done "in clear", through Bootloader or JTAG/SWD interfaces. RSS-Lib is used to manage the storage in OBKeys.
  - Untrusted environment: Provisioning done based on encrypted inputs. This is supported thanks to

the Secure Firmware Install (SFI).

- Updates are handled by applications
  - It is up to the application to define the way to provision the data. It is recommended to encrypt data to be provisioned.

Provisioning

- Data Provisioning RSSLIB API: `RSSLIB_DataProvisioning`
  - STM32H5-2M RSSLIB supports a service allowing OBKey programming
    - Host calls service via DEBUG (JTAG / SWD) or boot loader in trusted production use case
    - This service programs a buffer in OBKeys
    - Storage can be done with or without encryption (encryption uses DHUK)
    - RSSLIB_DataProvisioning verifies that input buffer is not tampered during communication between host and STM32 (i.e CRC verification over the received buffer)
    - RSSLIB_DataProvisioning verifies OBKey programming, i.e verifies that data in OBKey are the expected ones
  - C prototype:
    - `CMSE_NS_ENTRY uint32_t RSSLIB_DataProvisioning(RSSLIB_DataProvisioningConf_t * pConfig)`

life.augmented

21

A dedicated RSS service, called RSSLIB_DataProvisioning, can be called to provision data.
RSSLIB_DataProvisioning receives in input a data buffer and programs it within OBKeys.
The service programs this buffer, either in clear, or with encryption.
Encryption is only possible when the device support the crypto engine.
A CRC prevents any data and parameters tampering issue.
The service also checks that the programming is successful.
It can be called through the Bootloader.

Provisioning

- SFI supports dedicated areas to manage OBKey provisioning
  - Area O, allows to manage OBKey provisioning using SFI
  - A regular area encapsulates a sub-header, then area payload

- Area ‘O’ format description:

```
typedef struct
{
  RSSe_SFI_AreaODataHeader_t AreaOHeader;
  uint8_t AreaOPayload[];
} RSSe_SFI_AreaOData_t;

typedef struct
{
  uint32_t DoEncryption;
  uint32_t TzEn;
} RSSe_SFI_AreaODataHeader_t ;
```

life.augmented

22

The Secure firmware installation (SFI) supports dedicated areas to manage OBKey provisioning:

- Area O, that allows to manage OBKey provisioning using SFI, based on two structures detailed on the right of the slide.
- A regular area encapsulates a sub-header, present before the area payload .

The STM32 Trusted Package Creator is part of the STM32CubeProgrammer tool set and allows the generation of secure firmware and modules to be used for STM32 secure programming solutions, including SFI.

Thank you

life.augmented

23

Thanks for attending this presentation.
In addition to this presentation, you can refer to the following presentations:

- Security overview
- Life cycle
- Debug authentication.