Hello and welcome to this presentation describing the STM32H5 security feature called debug authentication control.

Agenda

1. Introduction
2. Permissions
3. Certificates
4. Processing permissions
5. Provisioning
6. ARM PSA protocol

life.augmented

2

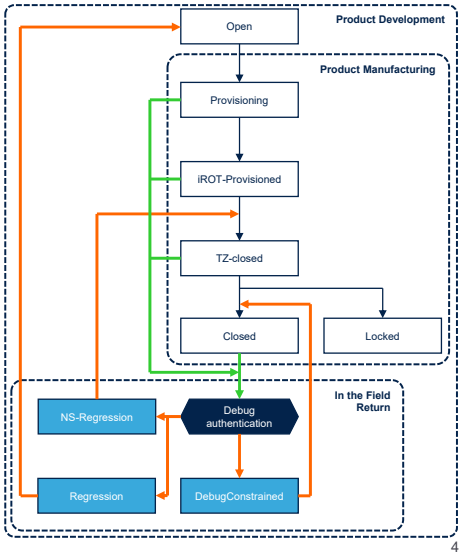
The following topics will be explained:

- Introduction to debug authentication control
- Definition of permissions
- Use of certificates
- Handling of permissions
- Provisioning the debug authentication configuration by using STM32CubeProgrammer
- Description of the Arm Platform Security Architecture (PSA) protocol, particularly the Authenticated Debug Access Control (ADAC).

Let us start with an introduction to debug authentication control.

STM32H5 Security
Debug Authentication Control

- Debug Authentication allows to control
  - Debug re-opening ➔secure controlled
  - Lifecycle management / regressions
- The system has to GUARANTY
  - Root Of Trust ASSETS protection
  - User information confidentiality
- To be used
  - During Development, manufacturing and in the field
- WARNING: DA-Config to be done to allow regression



life.augmented

4

The debug authentication is one of the most critical security features of the system considering that with a debugger the user can access a large part of the system.
To control re-opening of the debug, the device imposes a debug authentication protocol.
If the device is in CLOSED (product life cycle) state, the debug state is CLOSED.
The debug authentication procedure allows a trusted debugger to reopen access without compromising sensitive information called the Root Of Trust (ROT).
Reopening the debug is possible only if sensitive asset security is ensured and TrustZone is enabled. This is called Constrained Debug, as constraints ensure the security of the ROT information.
Alternatively, a partial or full regression mechanism can be

used when security of sensitive information cannot be guaranteed.
This is called Regression, as it ensures removal of the sensitive information before reopening the debug.

- Partial regression corresponds to releasing non-secure code and assets; the intermediate state which allows partial regression management is called NS-Regression.
- Full regression corresponds to releasing all codes and assets; the intermediate state allowing full regression management is called Regression.

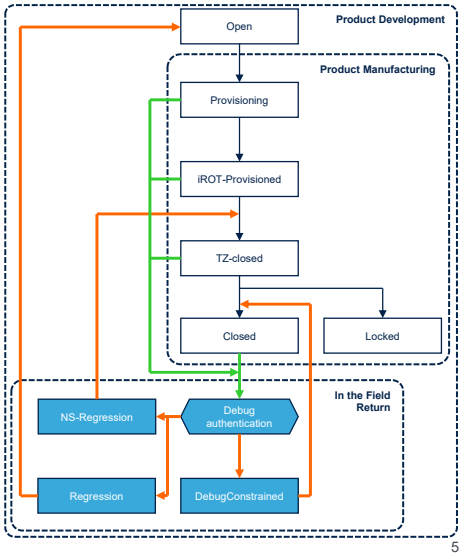
The debug authentication configuration must be done only when the product state is “Provisioning”, it cannot be performed when product state is “Open”.

STM32H5 Security
Debug Authentication Control

- To Launch Debug Authentication (DA), post “STDA” in DBGMCU, then trigger a reset
  - Can be done “Green Arrows” from all states except Open & Locked
- ARM PSA security model / definition
  - Recommend: to limit debug on Non-Secure

Debug state	Scope	Secure and attestable state	Security implications
Non-PSA-RoT debug	Non-Secure Processing Environment (NSPE)	YES	Compromises only NSPE operation and data ➢ Application RoT and PSA-RoT remain intact and trustworthy
	Application RoT	YES	Compromises only application RoT operation and data ➢ PSA-RoT remain intact and trustworthy
PSA-RoT debug	Non-recoverable	NO	Compromises PSA-RoT operation and parameters
	Recoverable	NO	May compromise the PSA-RoT operation, but cannot compromise the PSA secret parameters, that are stored in isolated locations or shielded locations

life.augmented



5

The transition to Debug Authentication state can be performed from Provisioning, iROT-Provisioned, TZ-closed and Closed product states, as indicated by the green arrows in the figure. The transition is caused by writing the STDA string of characters into the DBGMCU register and then triggering a reset.

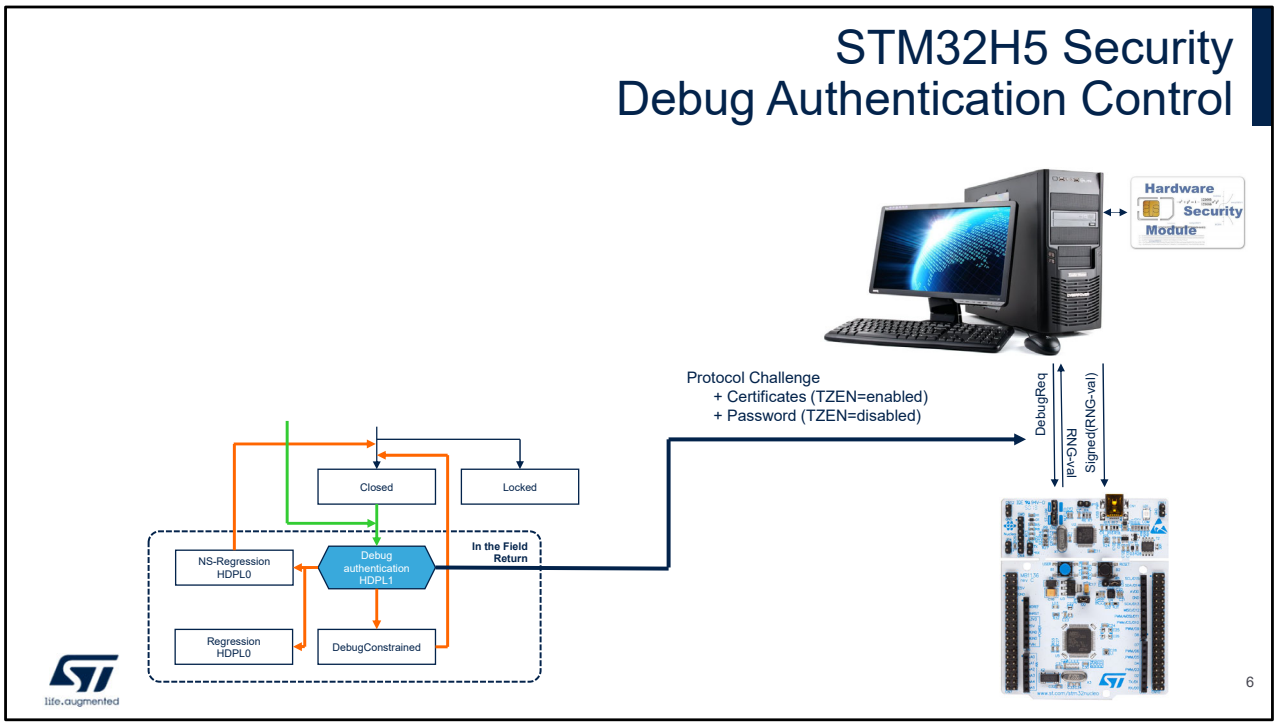
The Platform Security Architecture (PSA) Security Model is a specification for securing device.

Recoverable PSA-RoT debug applies to devices that can deny access to production PSA secret parameters when in PSA-RoT debug state and can be restored to a fully trustworthy and attestable state on exit if the debug activity has not compromised the Secured state.

- Non-recoverable PSA-RoT debug applies to devices that are not able to protect the PSA parameters and are not

able to ensure that the debug has not compromised the Secured state. This type of device should make the PSA parameters permanently inaccessible on entry to debug. The only valid next state is Regression of NS-Regression.

Non-PSA-RoT debug is restricted to non-secure processing environment, and application RoT operation and data. PSA-RoT remains intact and trustworthy.



During the Debug Authentication sequence, the PRODUCT_STATE is not changed. It will be changed depending on the success of the sequence if the request is for one of the regressions.

1. The external host requests to launch the debug authentication protocol, via the DBGMCU access port mailbox; the rest of the device is kept under reset.
2. The System, Boot and Security (SBS) selects the STMicroelectronics RSS-DA (debug authentication library) boot address, and requests the CPU to be released from reset
3. The CPU running RSS-DA library executes the debug authentication protocol in the system flash memory. If the device is closed, the access port mailbox is closed until RSS-DA acknowledges the authentication sequence start

request.

4. The authentication method depends on TrustZone activation:
    a. When TrustZone is activated, the authentication method is based on certificates.
    b. When TrustZone is disabled, the authentication method is based on password. This method only allows the full regression of the product to be controlled.
5. The above reopenings are effective only when HDPL in SBS_HDPLSR has a value equal or superior to the value programmed in DBG_AUTH_HDPL field in SBS_DBGCR register.

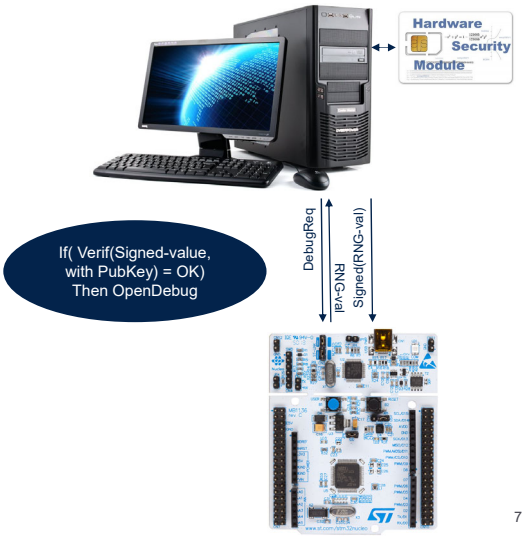
In the case of an authentication failure, the user is informed through the host interface.

The debug authentication library in in the system flash memory is available only when HDPL = 0 or 1. Only this library can perform the steps 3 and 4 described above.

STM32H5 Security
Debug Authentication Control

- When product is in the field, or under provisioning, OEMs can control distribution of maintenance rights to field technicians

TZEN	Scope
Enabled	Via Debug Authentication protocol using **Challenge-Response mechanism:** ➢ Open Debug without compromising the Root Of Trust ASSETs (Constrained Debug) ➢ Launch regression (Unconstrained Debug) when Root Of Trust ASSETS security cannot be guaranteed
Disabled	Only Full regression is allowed ➢ Controlled thanks to a **Password mechanism**



life.augmented

7

To ease the product maintenance in-the-field, debug authentication control enables the maintenance of product by activating the debug and makes it possible to manage regressions while considering the security of the sensible information.

This implies the following actions:

- Set the PRODUCT_STATE in Closed state when the product has been configured
- When debug authentication control is based on certificates (TZEN enabled), then provision the OEM-PublicKey and the SOC-Mask. Setting the SOC-Mask to only propose the non-secure-debug and partial and full regression guaranty to be compliant with the Arm PSA security model, rather than constrained debug
- When the debug authentication control is based on

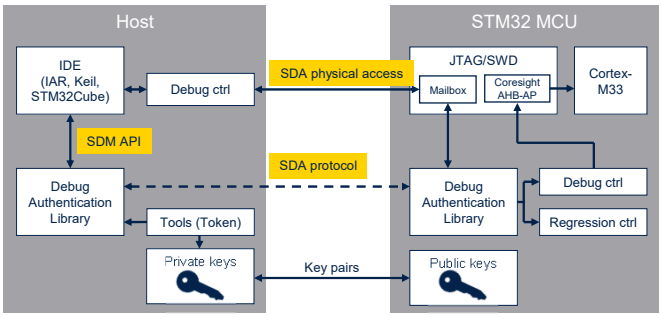
password (TZEN disabled), then only the HASH of the password has to be provisioned.

STM32H5 Security
Debug Authentication Control

- Debug Authentication ecosystem
  - Requires integration in the IDEs
    - IDE customization
    - Debug Authentication Library
    - Tool to generate responses to challenges (with PrivateKey)
    - Tools to setup the DA-config, and tools to generate certificates

Based on ARM PSA ADAC v1.0
- SDA = protocol
- SDM = Debug Authentication library (DLL) API

- Refer to https://developer.arm.com/documentation/den0101/latest

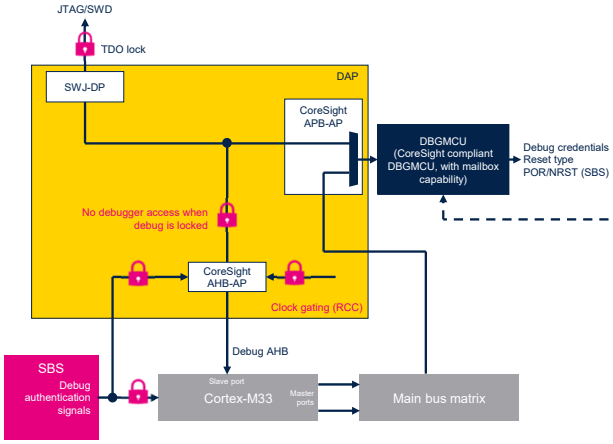


life.augmented

This figure details the various components required both in the host and the microcontroller to implement the debug authentication control.
In the host, the integrated development environment (IDE) relies on a Debug Authentication library through the Secure Debug Manager (SDM).
The SDM component asks the Secure Debug Authentication Control for debug access.
The Secure Debug Authenticator (SDA) accepts commands sent by the SDM via the debugger mailbox. It issues an authentication challenge and validates the credentials provided in response. Upon successful authentication, the SDA handles the hardware or software aspects of enabling debug access to target resources. In addition, upon request it can provide the SDM with information about the debug target.

This information can be used for discovery and identication purposes, to pass to the credential provider, to present the user with data to make an informed decision when choosing credentials, or other purposes.
The SDM resides on the debug host, and the DA resides on the debug target. The two components establish a logical communications link between themselves. This logical link is routed through the debug client, debug link, and debugger mailbox.
The SDA protocol is transported over Serial Wire Debug or JTAG link. The mailbox present in the debug access port is used to exchange messages.

STM32H5 Security
Debug Authentication Control



- DAP
  - DBGMCU : APB0
  - Mailbox: 2 * 32 bits + Status_register
    - PC_To_Device: DBGMCU_DBG_AUTH_HOST
    - Device_To_PC: DBGMCU_DBG_AUTH_DEVICE
    - Status register: DBGMCU_DBG_AUTH_ACK
- Hardware Implementation
  - DBGMCU on APB0 ➔always available
  - Cortex-M33: debug re-opening control (SBS)

life.augmented

9

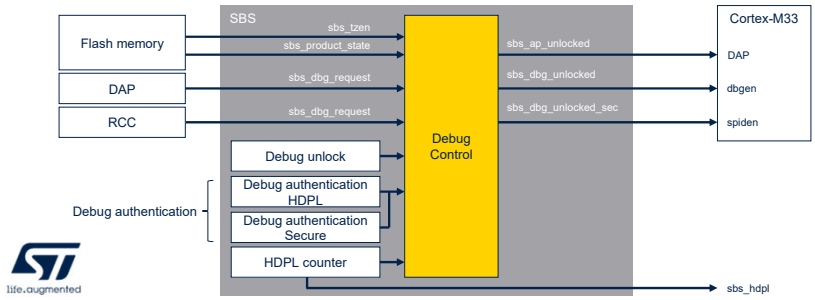
This figure details the units present in the Debug Access Port, which is connected to the debug probe by using the port represented on the top.
The DAP is based on a unique Debug Port (DP), connected to two Access Ports (APs), one supporting the APB protocol and the other one supporting the AHB protocol.
Note that the resources that the host debugger can access through the APB-AP are also accessible from the Cortex-M33 through main bus matrix and CoreSight APB-AP.
The DBGMCU is therefore accessible from both the remote host debugger and the local Cortex-M33 processor core
Even when the AHB-AP is locked, the debugger can access the DBGMCU. This is required to re-open the debug.
The DBGMCU includes two 32-bit mailboxes, one per direction and a status register.

In this figure, the locking mechanisms are highlighted:

- TDO Test Debug Out signal can be locked, preventing any communication on JTAG and also serial wire output
- Access to AHP-AP is locked when debug is locked, preventing debugger access to Cortex-M33 registers, as well as memory and peripherals
- The clock of the AHB-AP can be gated off for security and also power management purposes
- The debug authentication signals DBGEN and SPIDEN determine whether debug is permitted in non-secure and possibly secure states. They are generated by the System configuration Boot and Security module (SBS).

STM32H5 – Presentation
Debug Authentication / Debug Control

- Principle ➔ should allow debug reopening control
  - The Intrusive debug of the Cortex-M33, is opened only when the HDPL counter becomes equal to DBG_AUTH_HDPL
    - DBG_UNLOCK : To activate the unlock mechanism
    - DBG_AUTH_HDPL can be set with HDPL1, 2, 3
    - DBG_AUTH_SEC to select opening debug for Secure or Non-Secure



life.augmented

10

This figure describes the SBS debug control unit.

Debug opening depends on the states of inputs on the left of the figure.
The debug control unit is configured by the SBS sub-units located on the bottom left.
The outputs of the debug control unit provides the debug authentication signals to the Cortex-M33 and debug-related IPs, such as the DAP.
The following inputs control the debug opening:

- sbs_dbg_req: input signal that a host requests to launch the debug authentication protocol
- sbs_tzen: informs on the selected platform configuration related to TrustZone activation
- sbs_product_state: provides the current

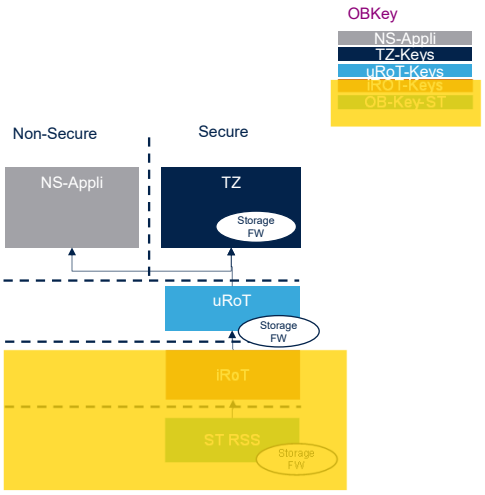
PRODUCT_STATE
- sbs_dbg_reset: reset information coming from the RCC and DAP.

Configuration of the Debug Control units is based on the following registers:
- DBG_UNLOCK in SBS_DBGCR: debug unlock when DBG_AUTH_HDPL is reached
- AP_UNLOCK in SBS_DBGCR: access port unlock
- DBG_AUTH_HDPL in SBS_DBGCR: authenticated debug temporal isolation level, defines the HDPL value from which the debug can be opened (value from 1 to 3)
- DBG_AUTH_SEC in SBS_DBGCR: specifies if the debug reopening is for non-secure only or for both (secure and non-secure)
- DBGC_LOCK in SBS_DBGLOCKR: locks the current debug configuration, released only by a reset.

The outputs of the Debug Control unit indicate wether:
- Debug is locked
- Debug is only enabled for non-secure state
- Debug is enabled for both non-secure and secure states.

STM32H5 – Presentation
Debug Authentication / Debug Control

- Debug re-opening Control typical example
  - DBG_UNLOCK = 0xB4
  - DBG_AUTH_HDPL=0x8A ➔ HDPL2
  - DBG_AUTH_SEC=0xB4 ➔ Secure+NonSecure



life.augmented

11

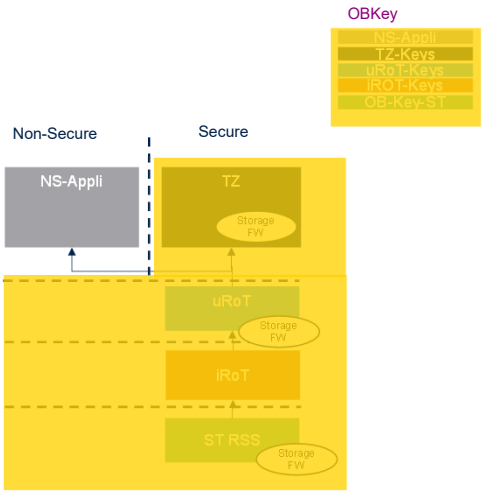
This slide and the the following one describe debug reopening typical examples.
In this one, assumptions are:

- DBG_UNLOCK=0xB4, enabling debug unlock when HDPL in SBS_HDPLSR equals DBG_AUTH_HDPL
- DBG_AUTH_HDPL=0x8A, defining HDPL2 as Hide Protect Level at which debug re-opens
- DBG_AUTH_SEC=0xB4, enabling debug unlock for secure and non-secure states.

Thus, debug for Updatable Root of Trust (uROT), non-secure and secure images are re-opened.

STM32H5 – Presentation
Debug Authentication / Debug Control

- Debug re-opening Control typical example
  - DBG_UNLOCK = 0xB4
  - DBG_AUTH_HDPL=0x6F ➔ HDPL3
  - DBG_AUTH_SEC=0x00 ➔ NonSecure



life.augmented

12

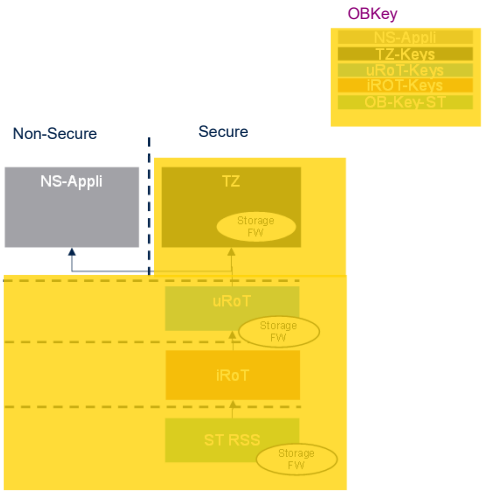
In this example, assumptions are:

- DBG_UNLOCK=0xB4, enabling debug unlock when HDPL in SBS_HDPLSR equals DBG_AUTH_HDPL
- DBG_AUTH_HDPL=0x6F, defining HDPL3 as Hide Protect Level at which debug re-opens
- DBG_AUTH_SEC=0x00 (actually different than 0xB4), enabling debug unlock for non-secure state only.

Thus, debug for non-secure image is re-opened.

STM32H5 – Presentation
Debug Authentication / Debug Control

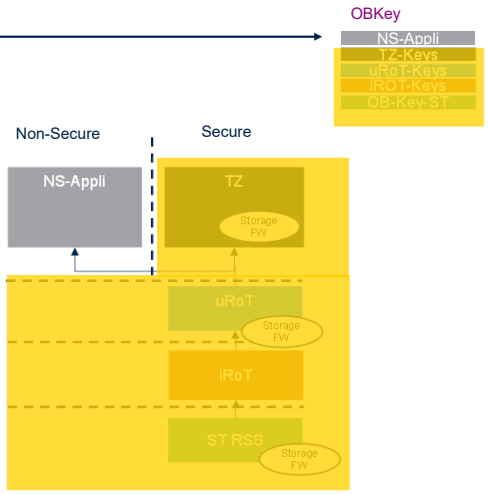
- Debug re-opening Control persistence
  - The debug opening configuration can reset with a System Reset or a POR
    - This is controlled thanks to
      - DBGMCU->DBGMCU_CR->DCRT bit field
        - 0: System Reset
        - 1: Power reset



The type of reset that re-opens the debug facilities is configurable:
- Either system reset
- Or power-on reset.

STM32H5 – Presentation
Debug Authentication / Debug Control

- NS-Regression control
  - Erase OBK-HDPL3NS
  - Erase All NS sectors in user flash
  - Increment the EPOCH-NS
  - Then Reset



life.augmented

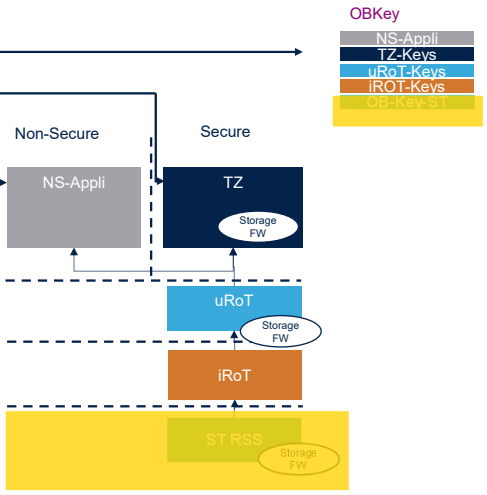
14

Re-opening debug with NS-regression automatically causes:
- The erasure of the NS sectors in user flash
- The erasure of Option Byte Keys related to NS-application (HDPL3NS).

EPOCH-NS is incremented, this a countermeasure against rollback.
Then, reset is applied.

STM32H5 – Presentation
Debug Authentication / Debug Control

- Regression Control
  - Erase OBK-HDPL3NS, 3S, 2, 1
  - Erase All S and NS sectors in user flash
  - Increment the EPOCH-NS & EPOCH
  - Then Reset



life.augmented

15

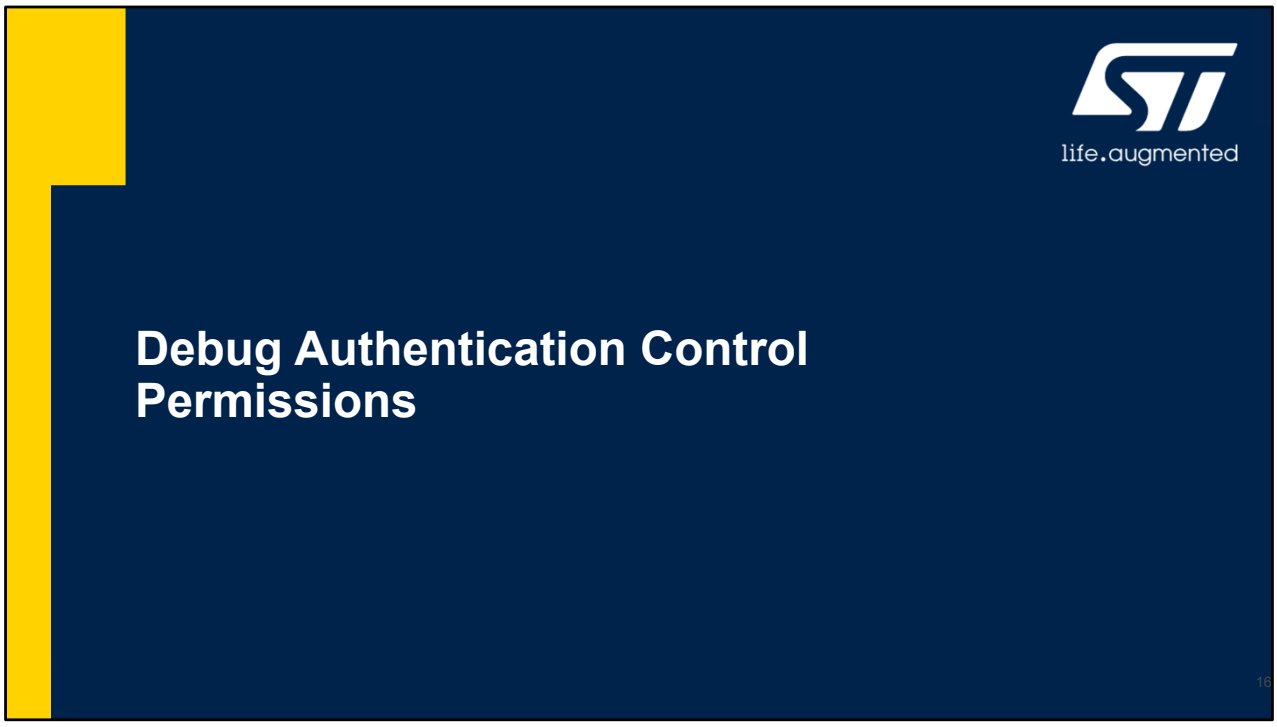
Re-opening debug with non-secure and secure regression automatically causes:

- The erasure of the NS sectors in user flash
- The erasure of the secure sectors in user flash
- The erasure of Option Byte Keys related to NS-application, TrustZone, Updatable ROT and Immutable ROT.

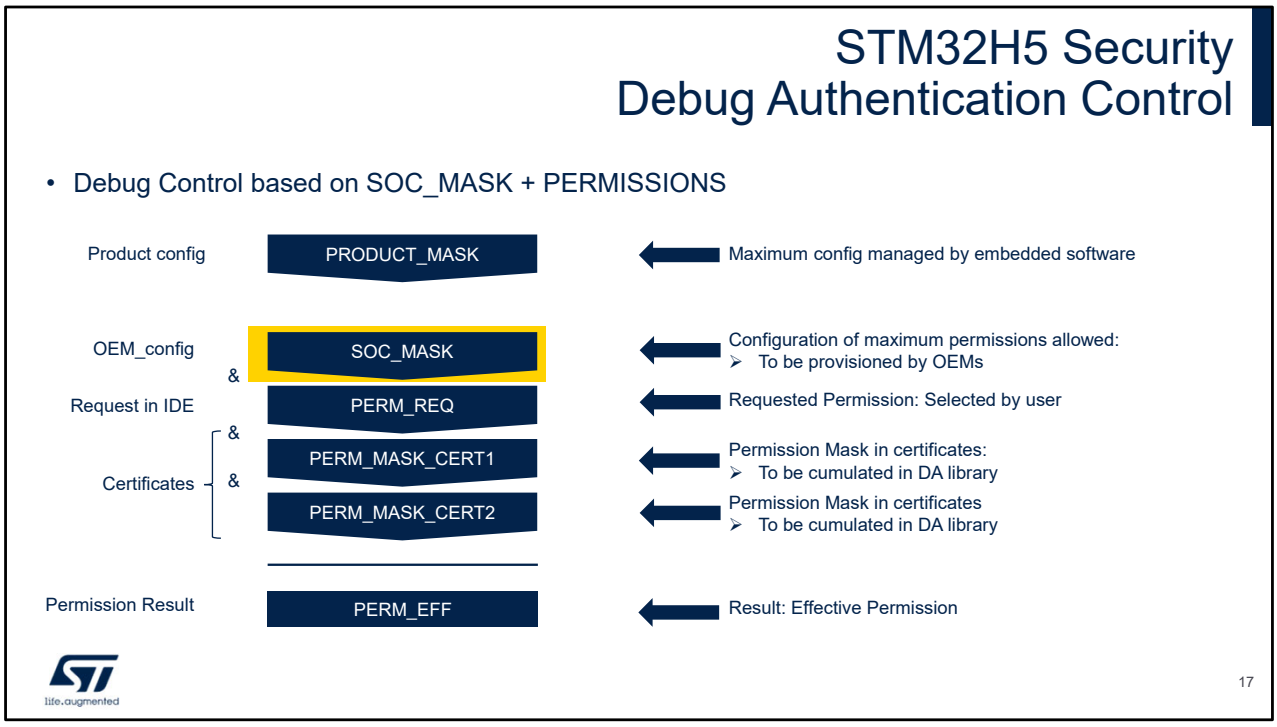
EPOCH-NS and EPOCH are incremented, this a countermeasure against rollback.

Then, reset is applied.

Therefore, all keys and flash contents are lost, except resources belonging to the RSS, that can never be changed.



The next subchapter describes the debug authentication control programmable permissions.



For fine-grained access-control, the PSA-ADAC specification defines a standard mechanism to associate certificates in the chain with a bitmap of logical permissions.
The debug host requests access to a set of permissions via the authentication token. The effective permissions are computed by masking requested permissions with permission-limiting constraints from the certificate chain.
This mechanism allows for certification authorities to restrict permissions of issued certificates.
Transition from closed to open or TZ-closed states is a matter of correct product configuration and provisioning.

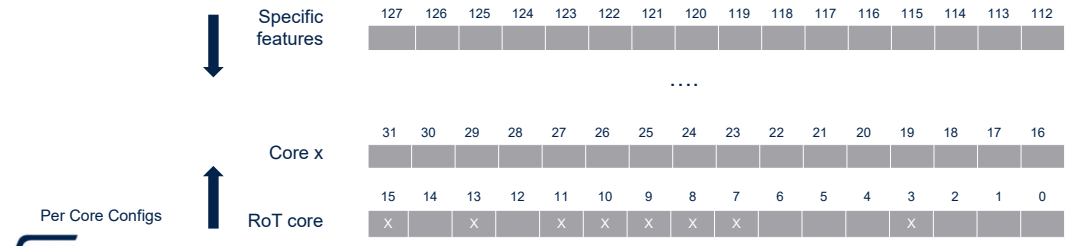
The figure describes the steps required to compute the effective permission.
The SOC_MASK is a static value set by the OEM, defining the

maximum permissions allowed.
It is anded with the permission vector requested by the debug host.
The resulting value is also anded with masks set by certificates cumulated in the Debug Authentication library.
The final permission result determines the scope of authentication and restricts permissions that its holder can unlock.

Distribution models
Permissions

- Permissions principles
  - The Debug Authentication embedded software will cumulate the limitations on permissions
  - Starting point: soc_perm = device setting the max allowed permissions
  - Each certificate: brings limitations ➔ applying masks

- ARM specification permissions defined on 128 bits



life.augmented

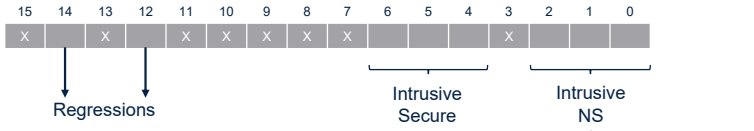
18

The permission mask is programmed in a 128-bit bitmap:
- Bits 15 down to 0 are related to the RoT core
- Bits 31 down to 16 are related to core x, they are reserved for STM32H5
- Bits 127 down to 32 are related to specific features, they are also reserved for STM32H5.

In the lower halfword, only bits 14, 12, 6-4 and 2-0 are relevant.
Starting point is defined by the SoC permissions, that set the maximum allowed permissions.
Certificates can bring stronger limitations.

Distribution models
Permissions

- Permissions proposal (STM32H5)
  - Invasive Debug
    - NS-Application (Level3 NS)
    - Secure-Application (Level 2S+NS)
    - ROT-Debug (Level 1S+NS)
  - Regression
    - Partial to TZ
    - Full regression



Regressions	
0 0	No regression
0 1	TZ-Regression
1 x	Regression

Intrusive S-Debug	
0 0 0	No debug
0 0 1	Level 1
0 1 0	Level 2
1 0 0	Level 3

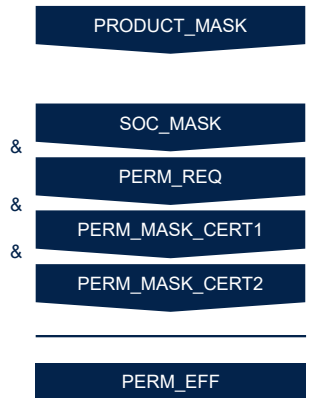
Intrusive NS-Debug	
0 0 0	No debug
0 0 1	Level 1
0 1 0	Level 2
1 0 0	Level 3

This slide describes the lower halfword of the mask.
Bits 2-0 set the HDPL at which debug can reopen for non-secure debug.
Bits 6-4 set the HDPL at which debug can reopen for secure debug.
Bits 14 and 12 control the regression.

Distribution models Permissions

- Permissions:
  - From higher permissions to lower
  - Perm_eff = (Perm_req & Perm_mask & Soc_mask)| (Soc_value & ~Soc_mask), where:
    - PRODUCT_MASK: Value hard coded to apply product specific limitations
    - SOC_MASK is the default provisioned value allowed
      - Allows to set limitations ➔for certification
    - PERM_REQ: Permissions vector requested by the debug host
    - PERM_MASK: Mask of permissions allowed by the certificate chain to be enabled
      - Perm_mask is the result of certificate masks: From OEM_PERMISSION to TECHNICIAN
- The Debug Authentication library ensures the processing of PERM_EFF

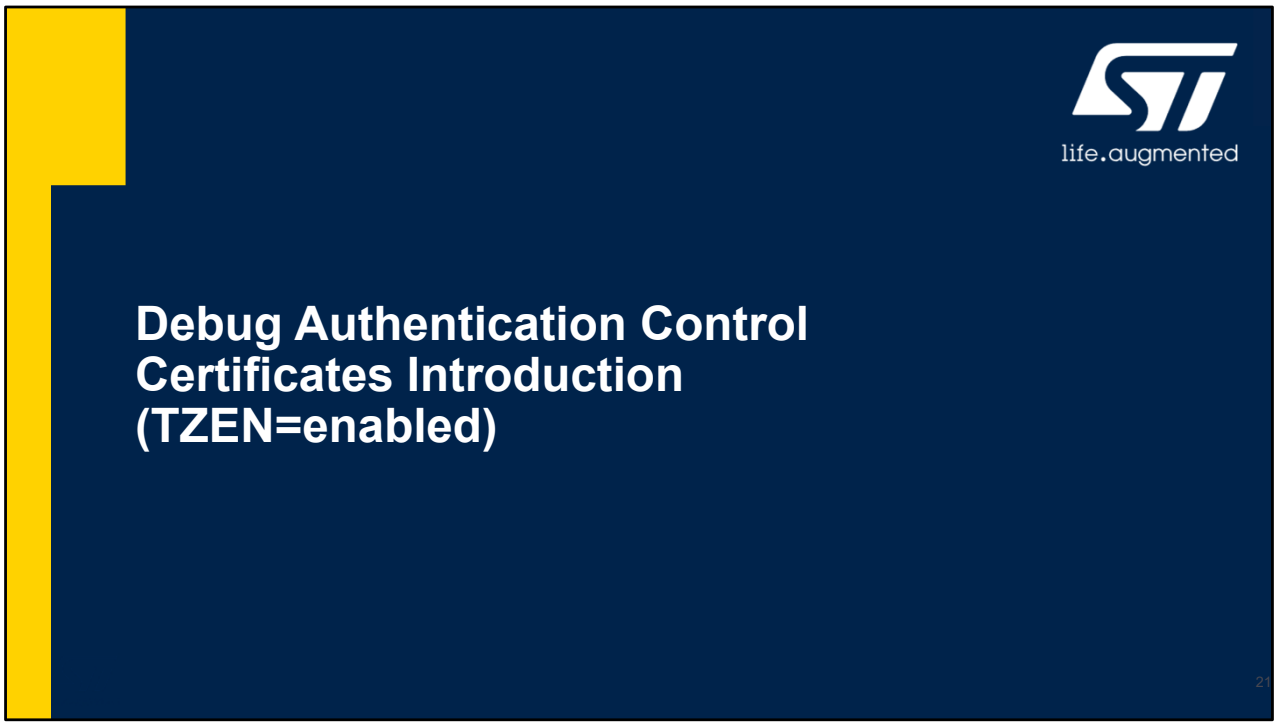
Allows a high level of flexibility



life.augmented

PERM_EFF is the effective permission resulting from the compute sequence described in this slide.
SoC_Mask is a value to be provisioned by the OEM, in the DA-config state.
Soc_Mask is a static value corresponding to all features supported by the implementation.
The intended use case is to allow the provisioning process of the target to set permanent restrictions on permissions requested by the host, and further, to set the values for those permissions that are restricted.
Steps to compute the effective permissions:
· The host requests a set of debug signals (Perm_req).
· The target combines the permission masks present in the certicates of the trust chain (Perm_mask).
Finally, the target computes the effective permissions

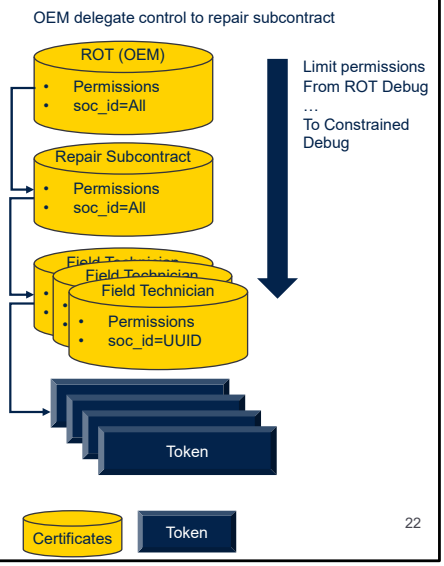
(Perm_eff), merging with the SoC-programmed permission constraints (Soc_mask and Soc_value).
The debug authentication library performs these computations, transparently for the end-user.
This masking mechanism allows a high level of flexibility.



The next sub-chapter introduces debug authentication certificates.

Debug Authentication Control Certificates introduction

- Certificates
  - Form: ARM PSA ADAC proprietary format
  - Certificates types
    - Root: Top certificate with maximum permissions
    - Intermediate: allow to consider different levels
    - Leaf: Terminal certificate
  - A Root can be used as Leaf certificate

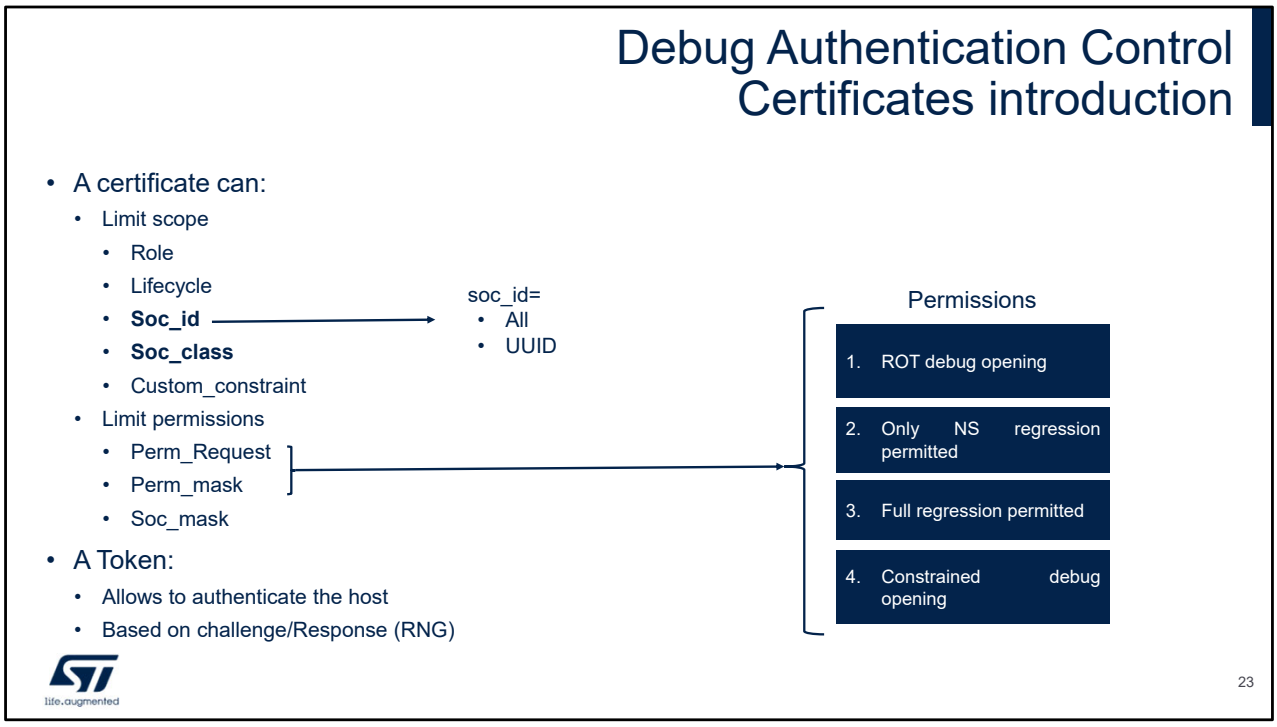


life.augmented

22

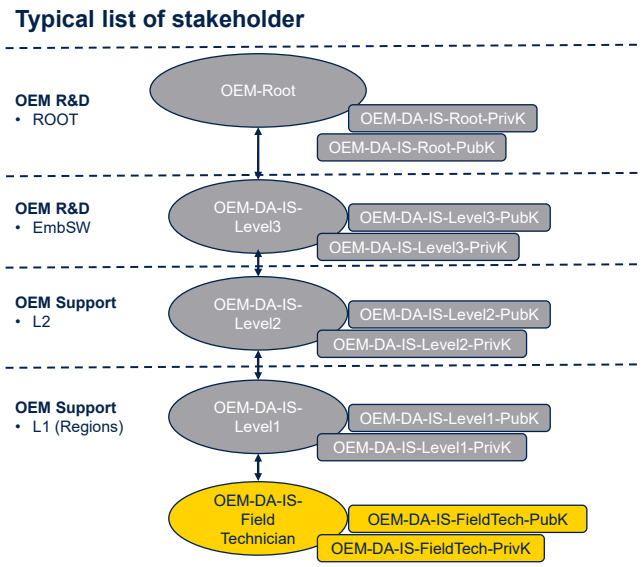
When TrustZone is activated, the authentication method is based on certificates.
As soon as a debug certificate chain is fully verified by the device, if the certificate concerns a debug permission, the RSS-DA programs the debug opening of the Cortex-M33.
Alternatively, the certificate can authorize partial or full regression, allowing debug on a regressed part.
The default mechanism for authentication relies on a challenge-response protocol:
The response to the challenge is a signed authentication token, also called the debug token.
A chain of certificates links the key used to sign to the authentication token to a set of one or more trusted anchors (roots of trust).
Based on the vendor's needs, the chain can be of arbitrary

length, ending in a key directly linked to a programmed root authority.
Certificates allow to have a very limited provisioning but offers a lot of flexibility: different models to distribute the right to access to different parts of the system can be implemented.
However, intermediate certificates also limit the exposure of the most sensitive keys, allowing that those keys can be used less often and protected with added security.
The Arm PSA-ADAC specification defines a certificate format to build trust chains and offer the flexibility to deal with complex scenarios.



Let us describe the contents of a certificate:
Scope is limited by fields defining the role (root, intermediate or leaf), soc id, soc_class.
Debug permissions are limited by fields defining the permission request, permission mask and soc mask.
A token is issued by the host debugger.
It allows the microcontroller to authenticate the host, through the challenge / response mechanism.

Debug Authentication Control
Certificate introduction - Generation

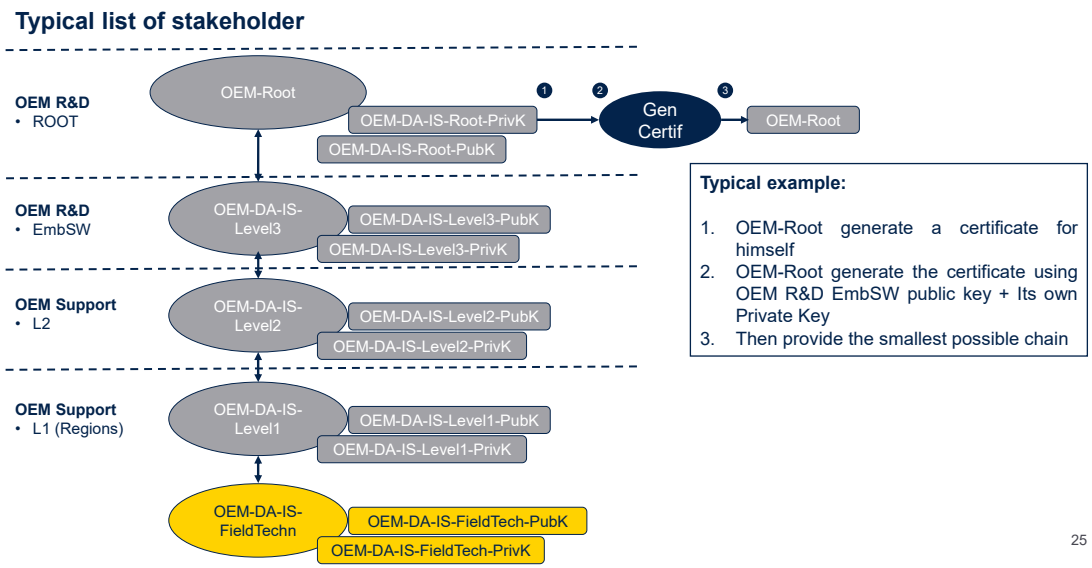


life.augmented

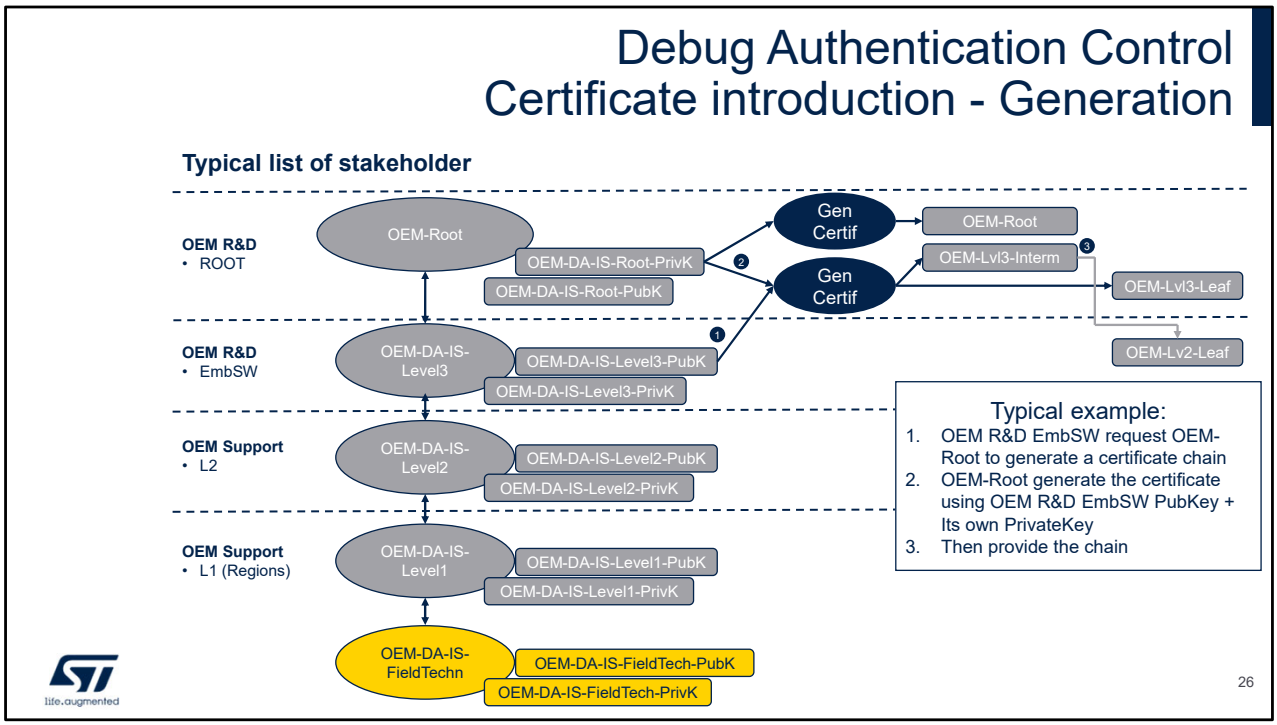
24

This diagram exemplifies a chain of trust.
The certificate chain is anchored to the device's root of trust.
Four additional levels are implemented, restricting debug permissions from OEM R&D down to Level 1 support field technicians.
At each level, an independent pair of keys is required: public and private (to authenticate each stakeholder).
ADAC assumes that public keys of trusted authorities have been pre-provisioned on devices and are available to the ADAC target-side code.

Debug Authentication Control
Certificate introduction - Generation

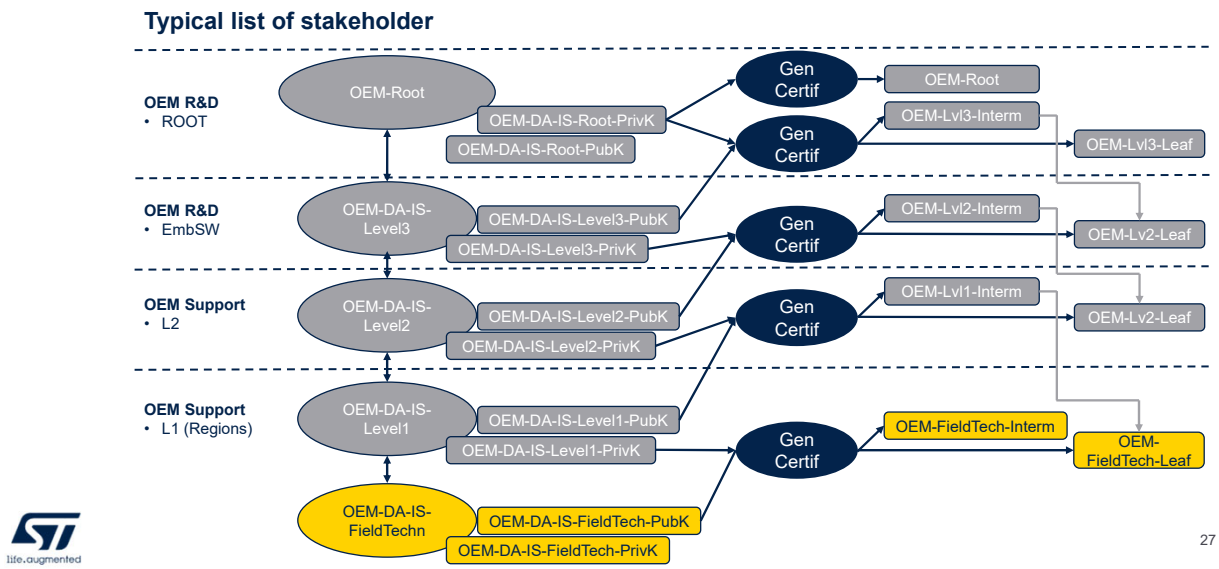


The certificate is generated by the OEM company, this is basically a structure.
This structure, that includes the public key, is hashed and signed, by using the private key.
Once generated, the certificate is stored in the OBKey related to the minimum HDPL accessible from the OEM.

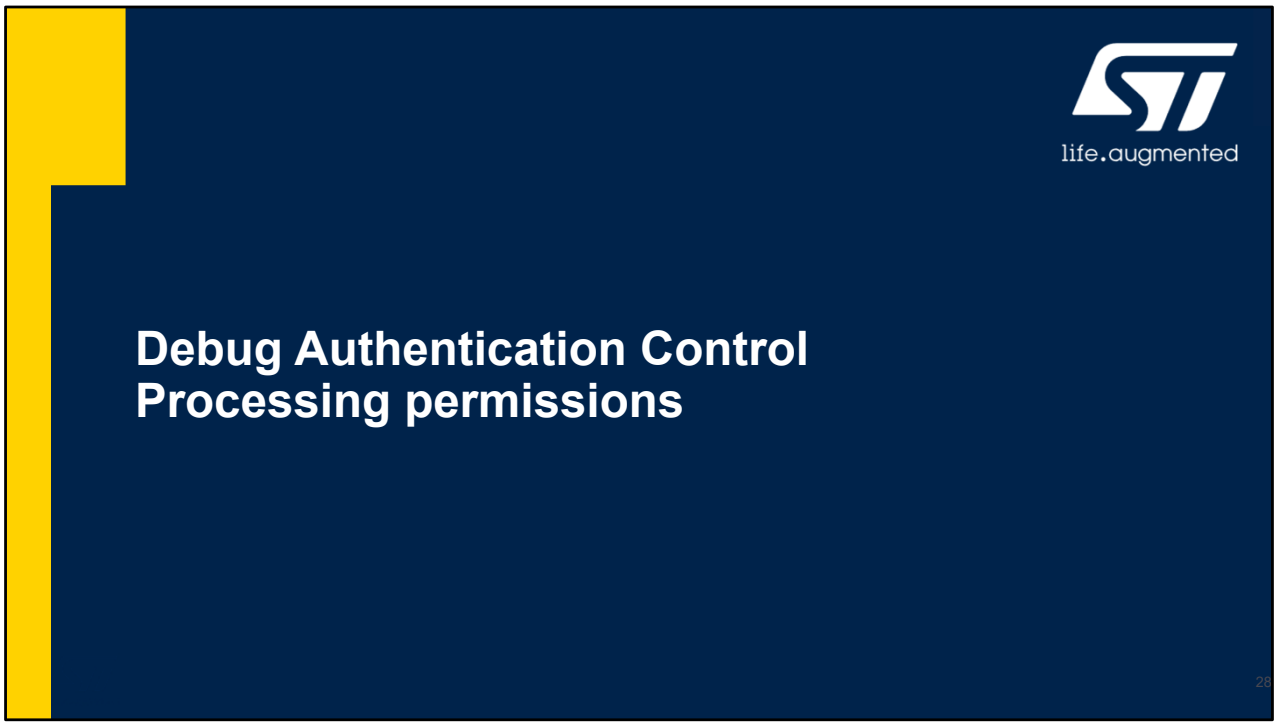


In the OEM company, two levels are distinguished: development of secure applications and development of general-purpose embedded software.
Isolation is needed between the two teams.
This is why a separate certificate is generated for the second team.
It is obtained by requesting the OEM root to generate a certificate chain.
The OEM root generates the certificates that includes the OEM R&D embedded software public key. It is signed by using the OEM root private key. This certificate will be used by the challenge / response protocol, that enables the debug of OEM embedded software.

Debug Authentication Control
Certificate introduction - Generation



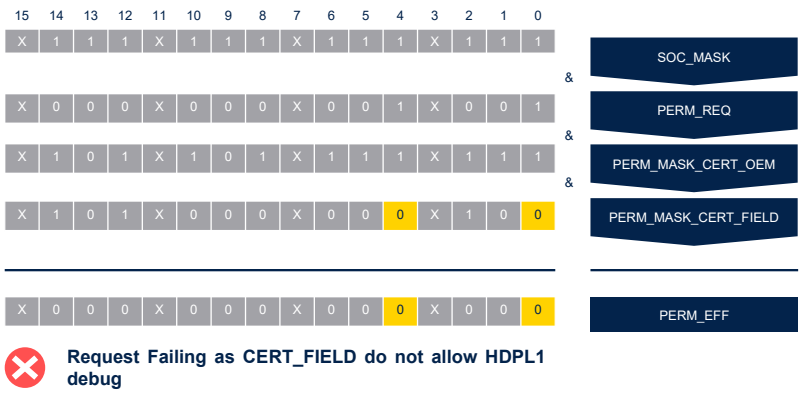
At each level, a certificate is generated, that includes the public key of that level and a signature obtained by the private key of the upper level.
A chain of certificates links the key used to sign to the authentication token to a set of one or more trusted anchors (roots of trust)



Now, after having described permissions and certificates, we can explain how permissions are dynamically processed.

Distribution models Permissions

- Permission example 1: SOC All possible + PERM_REQ_HDPL1 (ROT)
  - Soc_perm =
    - IntrusiveDebug (Full)
    - NonIntrusive (Full)
    - Regression (Full)
  - OEM Certif
    - Intrusive-Debug (Full)
    - Non-Intrusive (AuditLog+NSLogs)
    - Regression (Full + PartialTZ)
  - Technician Certif – type1
    - Intrusive-Debug (NS-Appli)
    - Non-Intrusive (AuditLog+NSLogs)
    - Regression (Full + PartialTZ)



Request Failing as CERT_FIELD do not allow HDPL1 debug

life.augmented

29

Three examples of permission handling are going to be explained.
When a bit equals one in the mask, the feature is enabled, else it is disabled.
In this example, the debug is requested for Temporal Isolation Level 1, which is the Root of Trust software (executed in HDPL1).
SOC_MASK does not provide restrictions.
However, PERM_REQ and the permission masks contain bits equal to zero.
The resulting value PERM_EFF sets the debug permissions:

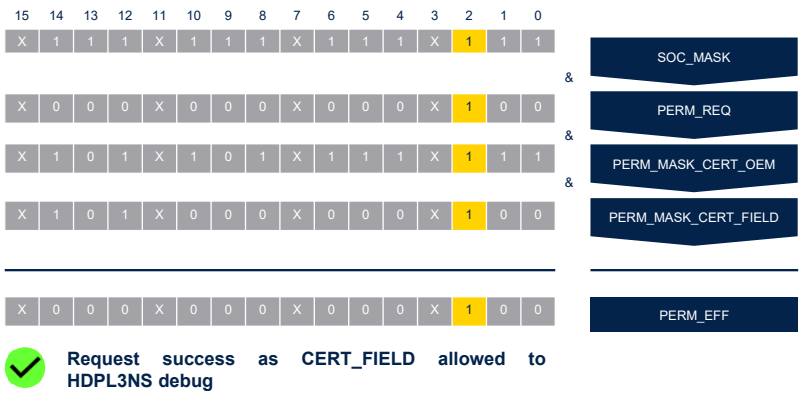
- Bits 2-0 = 0, no HDPL enables opening of non-secure intrusive debug
- Bits 6-4 = 0, no HDPL enables opening of secure intrusive debug

- Bits 14 and 12 = 0, no regression is permitted.

In this configuration, a debug request from HDPL1 results in a failure.

Distribution models Permissions

- Permission example 2: SOC All possible + PERM_REQ_HDPL3NS
  - Soc_perm =
    - IntrusiveDebug(Full)
    - NonIntrusive(Full)
    - Regression(Full)
  - OEM Certif
    - IntrusiveDebug(Full)
    - NonIntrusive(AuditLog+NSLogs)
    - Regression(Full + PartialTZ)
  - Technician Certif – type1
    - IntrusiveDebug(NS-Appli)
    - NonIntrusive(AuditLog+NSLogs)
    - Regression(Full + PartialTZ)



life.augmented

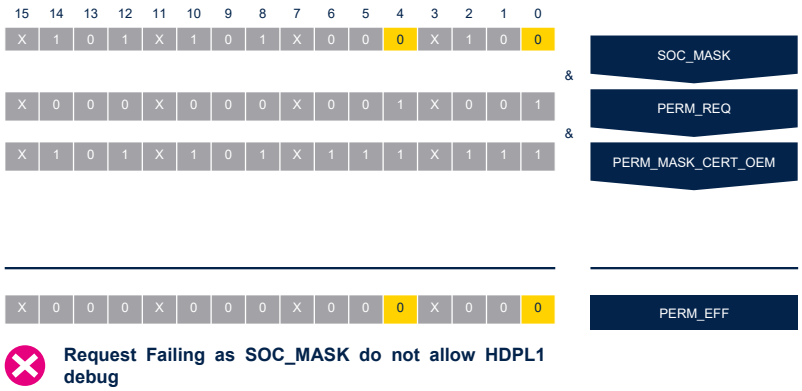
In this scenario, the debug is requested for Temporal Isolation 3 Non-Secure (HDPL3NS).
SOC_MASK does not provide restrictions.
The resulting value of PERM_EFF is:

- Bits 2-0 = 4, HDPL3 enables opening of non-secure intrusive debug
- Bits 6-4 = 0, no HDPL enables opening of secure intrusive debug
- Bits 14 and 12 = 0, no regression is permitted.

In this configuration, a debug request for non-secure application debugging succeeds, without regression.

Distribution models Permissions

- Permission example 3: SOC HDPL3NS only + PERM_REQ_HDPL1 (ROT)Typically for a certified config
  - Soc_perm =
    - IntrusiveDebug(HDPL3NS)
    - NonIntrusive(AuditLog+NSLogs)
    - Regression(Full+TZ)
  - OEM Certif
    - IntrusiveDebug(Full)
    - NonIntrusive(AuditLog+NSLogs)
    - Regression(Full + PartialTZ)



Request Failing as SOC_MASK do not allow HDPL1 debug

life.augmented

31

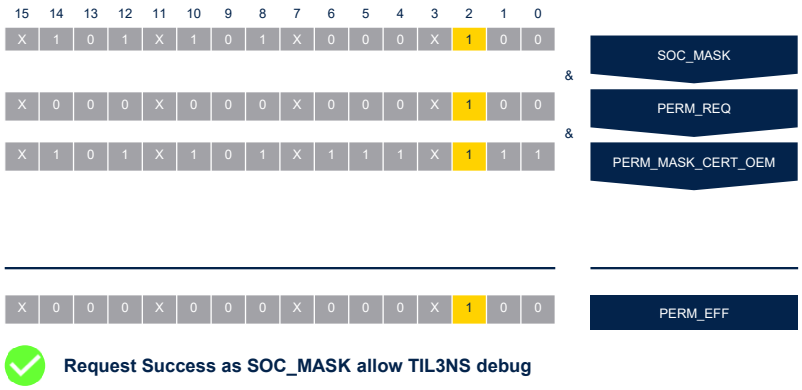
In this scenario, the debug is requested for Temporal Isolation 1 (HDPL1).
SOC_MASK disables HDPL1-NS and HDPL1-S debug opening.
The resulting value of PERM_EFF is:

- Bits 2-0 = 0, no HDPL enables opening of non-secure intrusive debug
- Bits 6-4 = 0, no HDPL enables opening of secure intrusive debug
- Bits 14 and 12 = 0, no regression is permitted.

In this configuration, a debug request for HDPL1 software debugging fails.

Distribution models Permissions

- Permission example 4: SOC HDPL3NS only + PERM_REQ_HDPL3 NS
  - Soc_perm =
    - IntrusiveDebug(HDPL3NS)
    - NonIntrusive(AuditLog+NSLogs)
    - Regression(Full+TZ)
  - OEM Certif
    - IntrusiveDebug(Full)
    - NonIntrusive(AuditLog+NSLogs)
    - Regression(Full + PartialTZ)



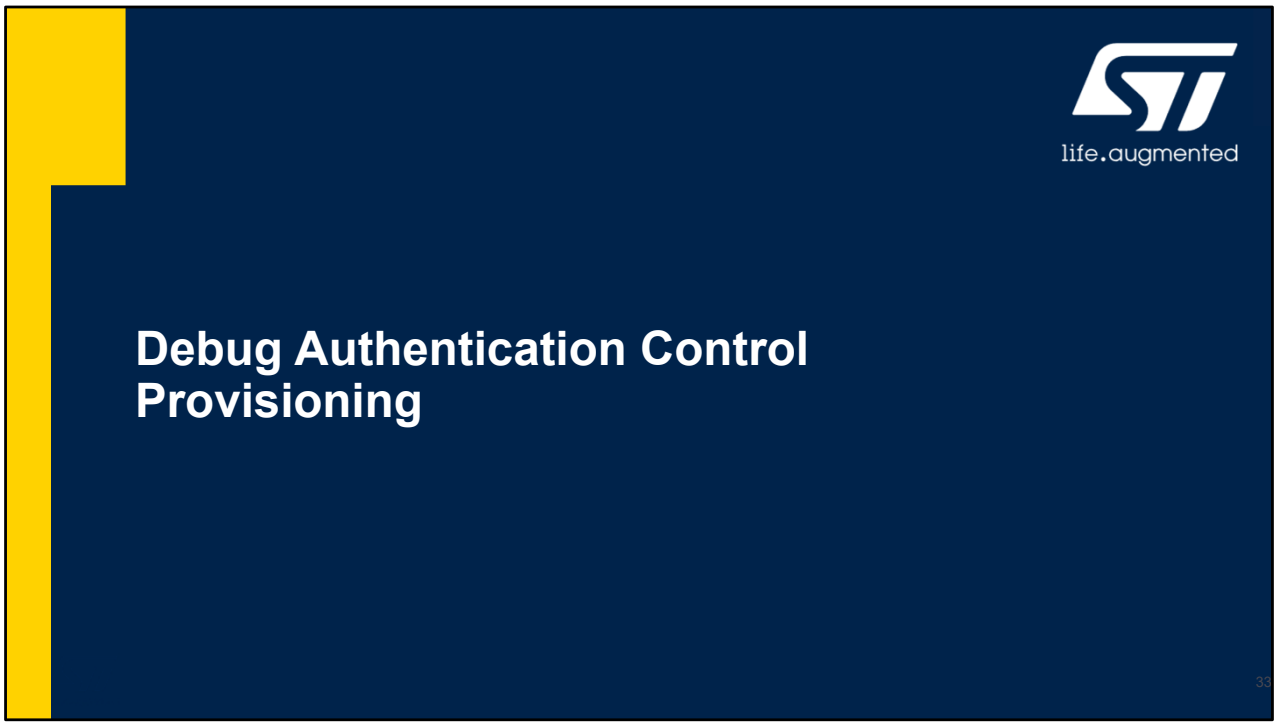
life.augmented

32

In this scenario, the debug is requested for Temporal Isolation 3 non-secure (HDPL3NS).
SOC_MASK enables HDPL3-NS debug opening.
The resulting value of PERM_EFF is:

- Bits 2-0 = 4, HDPL3 enables opening of non-secure intrusive debug
- Bits 6-4 = 0, no HDPL enables opening of secure intrusive debug
- Bits 14 and 12 = 0, no regression is permitted.

In this configuration, a debug request for HDPL3 non-secure software debugging succeeds, without regression.



Debug authentication is natively supported by STM32H5 platform.
It means that the data used by ST debug authentication (ST-DA) must be provisioned at a defined location in secure key storage area (OBKeys defined in flash memory).
This will be described in the next slides.

Debug Authentication Control Provisioning

- Warning 1:
  - The provisioning has to be done when in Provisioning State on STM32H57x
    - The main reason is that it will be encrypted using the DHUK, which is not the same in Open than the rest of the states
    - Provisioning done in Open will not be valid for other states
- Warning 2:
  - Different Provisioning if you are in TZEN=enabled or TZEN=disabled
- Warning 3:
  - Moving to later states without provisioning the DA-config, the device cannot be regressed anymore

life.augmented

34

On the STM32H75x, the debug authentication configuration must be done only when the PRODUCT_STATE is “Provisioning”, it cannot be performed when PRODUCT_STATE is “Open”.
The data to provision depend upon the TZEN option byte setting.
When moving to later states without provisioning the DA-config, the device cannot be regressed anymore.
Note that RSS_Lib encryption option must be set for STM32H57x devices, while RSS_Lib encryption option must be reset for STM32H56x devices.

Debug Authentication Control Provisioning

- Provisioning different depending on TZEN enabled/disabled
  - In all cases, the provisioning is done beginning of the OBK-HDPL1
  - TZEN enabled ➔ For certificate management
    - HASH (SHA256) of the ROT Certificate Pub Key
    - SOC_PERMISSION: 128 bits defining the permissions authorized by default
  - TZEN disabled ➔Password management
    - HASH (SHA256) of the Password
  - TPC is used to generate the OBK, adding a SHA256 to ensure the integrity

life.augmented

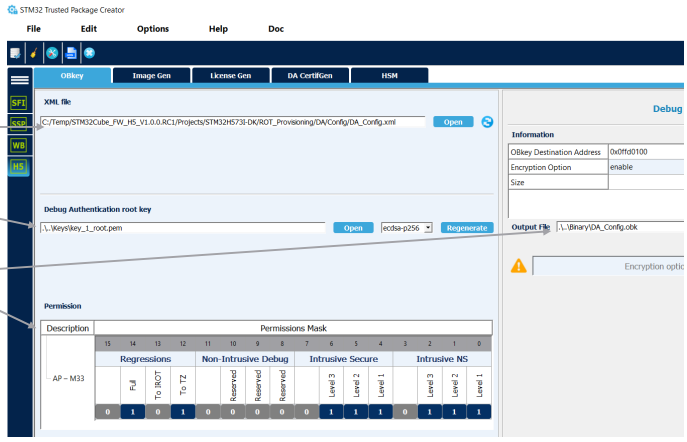
35

When trustzone is enabled, debug authentication control is based on certificates.
When trustzone is disabled, debug authentication control is based on passwords; only the HASH of the password has to be provisioned.
In all cases, provisioning is done at the beginning of the OBK-HDPL1.
A tool designed by STmicroelectronics, called Trusted Package Creator (TPC), is used to generate the OBK, adding a SHA256 to ensure the integrity of the stored information.
TPC is part of the STM32CubeProgrammer tool set.

TZEN=enabled ➔ Certificates

Debug Authentication Control Provisioning

- Provisioning – Prepare (TPC)
  - Preparation through TPC / H5 / OBkey
    - Selecting DA_Config.xml
    - TZEN=Enabled ➔ Certificates
      - SHA256 of the ROT certificate
      - SOC_MASK ➔ Select permissions (the maximum allowed to ROT)
    - Allows to Generate OBKey



life.augmented

This slide and the following one contain screenshots of the STM32CubeProgrammer tool, in order to highlight the features related to debug authentication, when trustzone is enabled.
The first step is preparing provisioning.
The following information are required:

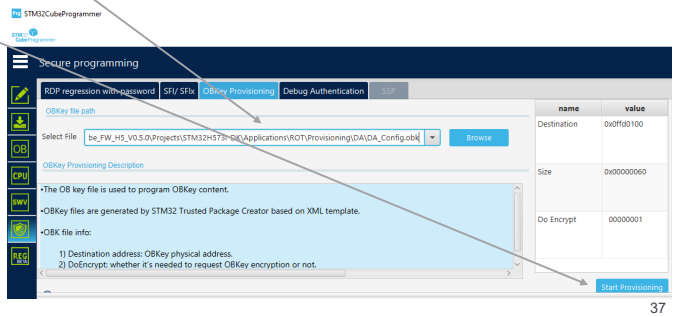
- DA_Config xml file, describing the OBK region structure
- The SHA256 of the ROT certificate is provided by the file key_1_root.pem in this example
- The SOC_MASK is defined through a bitmap that can be programmed in the GUI.

The OBKey is then generated and output in the DA_Config.obk file.

TZEN=enabled ➔ Certificates

Debug Authentication Control Provisioning

- Provisioning – Provision (CubeProgrammer)
  - Connect the device
  - Select the previously generated obk file
  - Start Provisioning



life.augmented

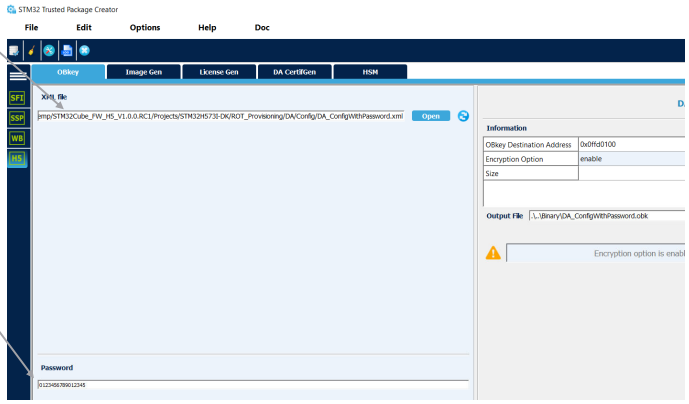
37

The next step is the provisioning.
The microcontroller has to be connected.
The previously generated .obk file is selected.
Then the user can click on the Start Provisioning dialog to start the programming.

TZEN=disabled ➔ Password

Debug Authentication Control Provisioning

- Provisioning – Prepare (TPC)
  - Preparation through TPC / H5 / OBkey
    - Selecting DA_ConfigWithPassword.xml
    - Select the Password
    - Allows to Generate OBKey



life.augmented

38

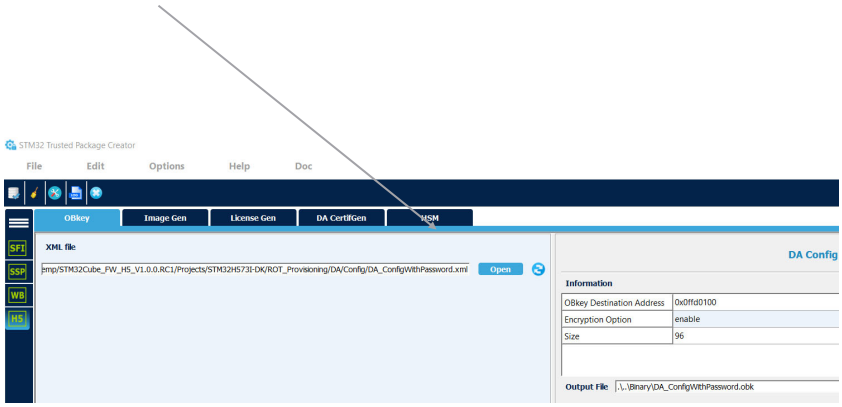
This slide and the following one contain screenshots of the STM32CubeProgrammer tool, in order to highlight the features related to debug authentication, when trustzone is disabled.
The first step is preparing provisioning.
The following information is required:
- DA_ConfigWithPassword xml file, describing the OBK region structure
- The password is directly entered in the GUI.

The OBKey is then generated and output in the DA_ConfigWithPassword.obk file.

TZEN=disabled ➔ Password

Debug Authentication Control Provisioning

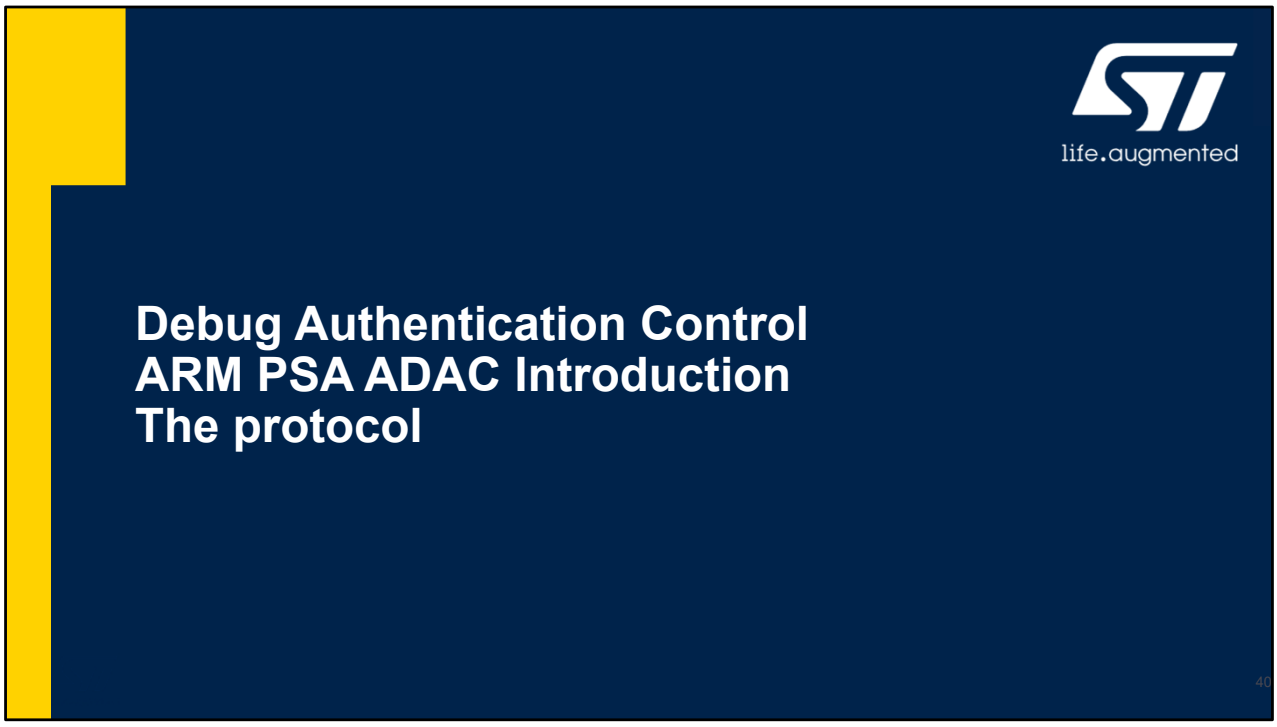
- Provisioning – Provision (CubeProgrammer)
  - Connect the device
  - Select the previously generated obk file
  - Start Provisioning



life.augmented

39

The next step is the provisioning.
The microcontroller has to be connected.
The previously generated .obk file is selected.
Then the user can click on the Start Provisioning dialog to start the programming.



The next subchapter is an overview of the Arm PSA ADAC protocol.

Debug Authentication Control
ARM PSA ADAC Introduction

- The ARM PSA ADAC protocol is defined here:
  - https://developer.arm.com/documentation/den0101/0001
  - It is designed to be portable
  - It allows customization ➔We use this feature to implement the Password (TZEN=disabled)
  - It proposes a list of commands:
    - Discovery, Authentication start, Response, Close sessions, Lock session
    - Define the Token
    - Define the certificates formats
  - It allows a large support of cryptographic algorithms

life.augmented

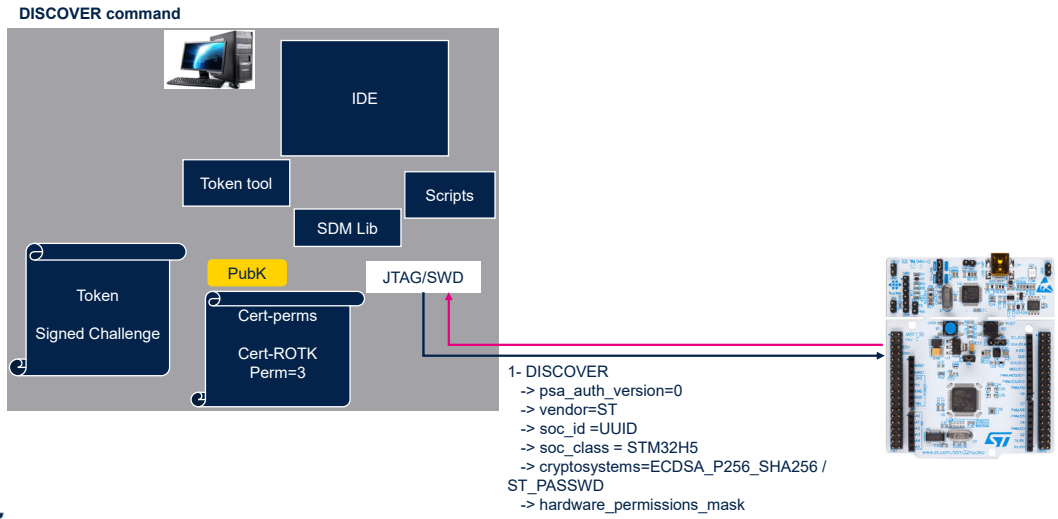
41

The ADAC specification defines the protocol used by the Secure Debug Manager to request debug access from the Secure Debug Authenticator.
It is based on 4 layers: debug link, link layer, command protocol and authentication commands
The ADAC specification covers the following topics:

- Command protocol
- Debug authentication commands
- Certificate format
- Token format.

The protocol allows customization, which has been used by STmicroelectronics to design the password mechanism, active when trustzone is disabled.
The following cryptographic algorithms are supported: ECDSA, RSA, EdDSA, ShangMi SM2, CMAC with AES, HMAC with SHA-256.

Debug Authentication Control
ARM PSA ADAC Introduction

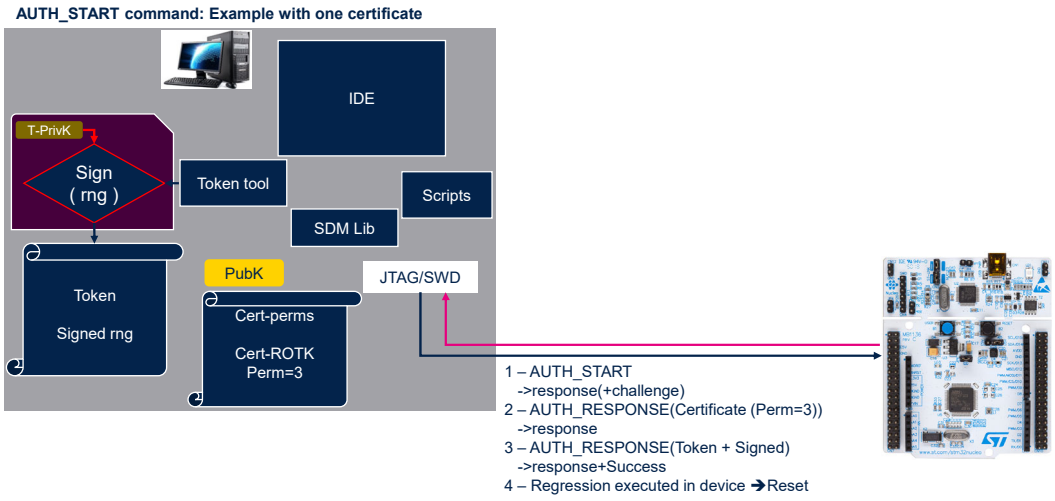


life.augmented

42

This slide explains the Discovery command.
The debug host requests information about the debug target used with this command.
The requested ID list is discretionary; the target can reply with any set of values that is equal to or a super set of the requested value, excluding any values unavailable on the target or not recognized by the target.
If a requested ID list is not included, the target must reply with all available values.
Use of this command is optional and is not part of the required authentication command sequence.

Debug Authentication Control
ARM PSA ADAC Introduction



life.augmented

43

The required sequence of commands for authentication is as follows.

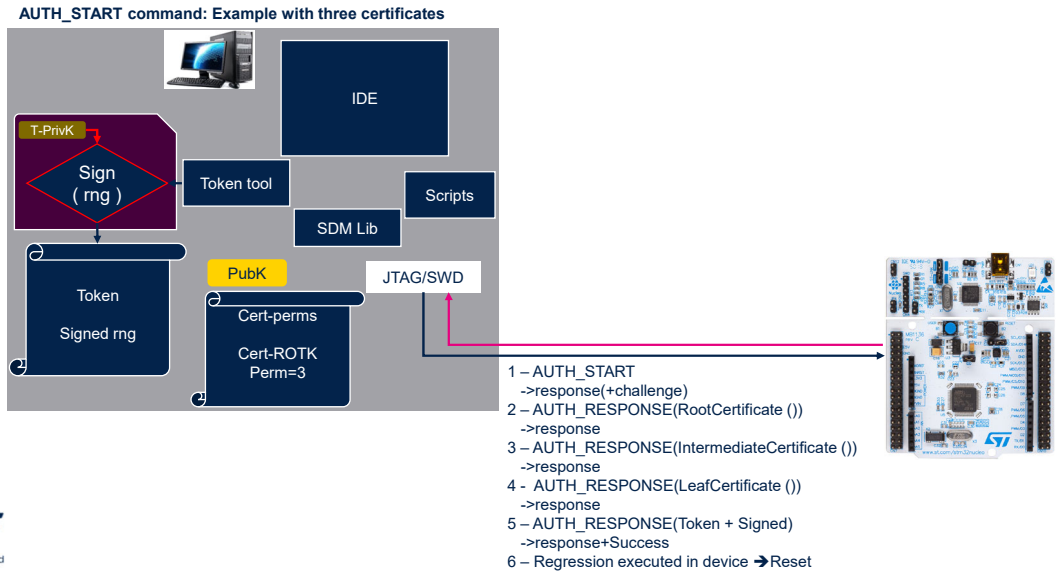
1. Authentication Start: Host requests challenge from target. Its primary purpose is for the target to provide a random 256-bit challenge vector used to prevent replay attacks.
2. Authentication Response (one or more): This command is used to provide the debug token and additional credentials as part of a complete authentication response to the target.

The target validates the provided debug token. If validation fails, the ADAC_FAILURE status is returned.

In this scenario, a unique certificate is provided by the host, which is the root certificate.

When the authentication succeeds, a regression may be necessary and a reset must be applied.

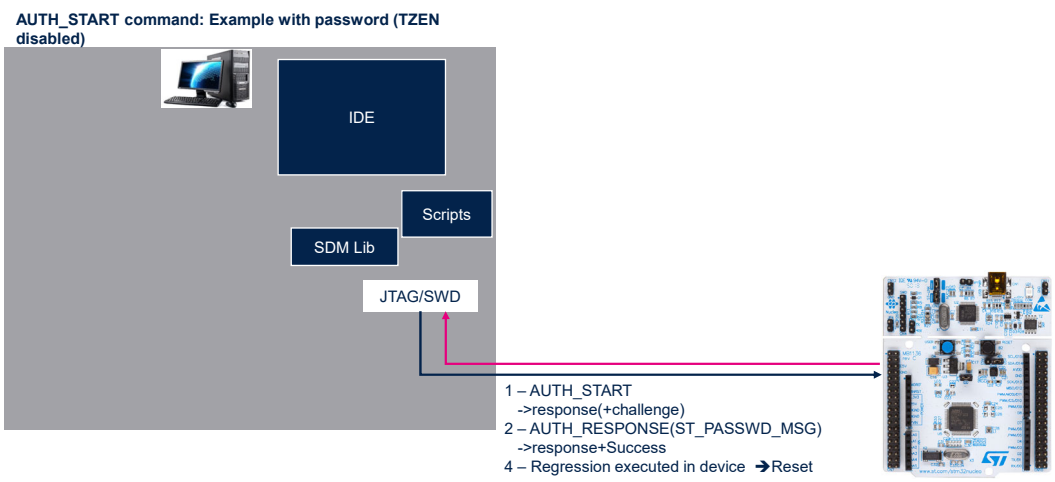
Debug Authentication Control
ARM PSA ADAC Introduction



life.augmented

The number of certificates used is a decision made by the customer to trade off security versus processing time and management complexity.
This example uses three certificates.
The root certificate signs the intermediate certificate.
The intermediate certificate signs the leaf certificate.
The leaf certificate then signs the debug token and challenge vector.

Debug Authentication Control
ARM PSA ADAC Introduction



life.augmented

In this scenario, trustzone is disabled, authentication is based on password.
In order to access the debug authentication feature, the host sends the debug authentication password to the STM32.
When the STM32 receives the password, it verifies that it corresponds to the one that is provisioned.
When the authentication succeeds, a regression may be necessary and a reset must be applied.

Thank you

© STMicroelectronics - All rights reserved.
ST logo is a trademark or a registered trademark of STMicroelectronics International NV or its affiliates in the EU and/or other countries.
For additional information about ST trademarks, please refer to www.st.com/trademarks.
All other product or service names are the property of their respective owners.

life.augmented

Thanks for attending this presentation.
You can also refer to UM2238 entitled “STM32 Trusted Package Creator tool software description”
In addition, you can refer to the following presentations:

- Security overview
- Secure data storage
- Product Lifecycle.