AN6008

Application note

Getting started with debug authentication (DA) for STM32 MCUs

Introduction

This document describes the debug authentication (DA) security service. When not specified, STM32 refers to all applicable products present in Table 1. Applicable products.

The STM32 debug authentication controls the product life cycle, such as regressions (for more details about the life cycle, refer to the reference manual), and debug reopening:

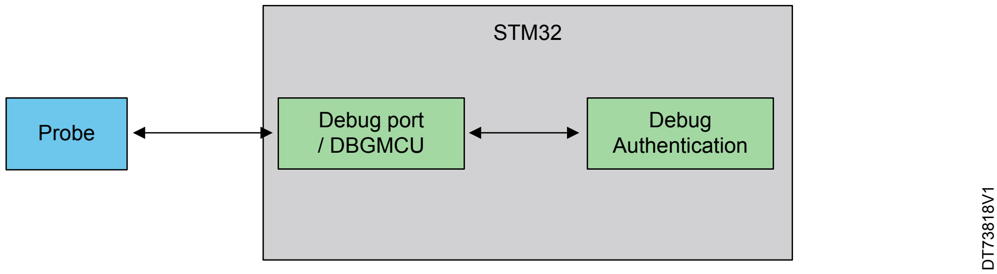
- **Regression**
  The user leverages the regression service to erase the user firmware and data within the user flash memory, SRAM, and option-byte keys (OBK) when OBK are supported by STM32. After a regression, STM32 falls back in product state open. Depending on STM32 products, there can be several kinds of regression: full regression and partial regression Refer to Section 3: STM32 debug authentication services description for more details.
- **Debug reopening**
  The user leverages the debug reopening to safely reopen the debug on STM32 when it is in a product state different than open.

When the STM32 product state is not open, the user can trigger the debug authentication services by sending a password or a certificate chain to the STM32 device.

These two options are named the debug authentication methods.

The debug authentication protocol uses the STM32 device debug access port 0 (DAP0) and the DBGMCU IP for communication.

Figure 1. Debug authentication interface



The STM32 debug authentication implements the Arm® PSA ADAC (authenticated debug access control) specification.

The Arm® PSA ADAC protocol is based on the certificate chain and the challenge/response principle.

Table 1. Applicable products

Type	Product
Microcontrollers	STM32H5 series, STM32H7R3/7S3, STM32H7R7/7S7 lines

1 General information

This document applies to

- STM32H5 series Arm® Cortex®-M33-based microcontrollers.
- STM32H7Rx/7Sx Arm®Cortex® M7 based microcontrollers.

Note: Arm is a registered trademark of Arm Limited (or its subsidiaries) in the US and/or elsewhere.

Table 2. Terms and abbreviations

Acronym	Definition
AES	Advanced encryption standard
ADAC	Authenticated debug access control
DA	Debug authentication
EPOCH	Epoch
HDP	Hidden protection
HDPL	Hidden protection level
iRoT	Immutable root of trust
JTAG	Joint test action group
OBK	Option-byte keys
OFTDEC	On-the-fly decryption
PKA	Public key accelerator
PSA	Platform security architecture
SAES	Secure advanced encryption standard
SoC	System on chip
SDM	Secure debug manager
STiRoT	ST immutable root of trust
SWD	Serial wire debug
TZ	Arm® TrustZone®
TZEN	Arm® TrustZone® enabled
uRoT	Updatable root of trust
WRP	Write protection

Reference documents

Reference	Name/address	Title
[1]	DEN 0101	Authenticated Debug Access Control 1.0(1)
[2]	RM0481	STM32H573/56x & STM32H533/523 reference manual
[3]	RM0492	STM32H503 reference manual
[4]	AN6007	Application note for STiRoT
[5]	Security features on STM32H5 MCUs	https://wiki.st.com/stm32mcu/wiki/Security:Security_features_on_STM32H5_MCUs
[6]	RM0477	STM32H7Rx/7Sx reference manual

1. *This URL belongs to a third-party. It is active at document publication. However, STMicroelectronics shall not be liable for any change, move, or inactivation of the URL or the referenced material.*

Products compatibility

Table 3. Synthesis table for SMT32H5 products

-	STM32H573xx/H533xx	STM32H563xx/H523xx	STM32H503xx
DA data storage location	HDPL1 OBK (@ 0x0FFD0100)	HDPL1 OBK (@ 0x0FFD0100)	OTP (@ 0x08FFF000)
DA data encryption	Yes	No	No
Available DA methods	Password (TZEN=0xC3) / certificate (TZEN=0xB4)	Password (TZEN=0xC3) / certificate (TZEN=0xB4)	Password
Discovery	Yes	Yes	Yes
Full regression	Yes	Yes	Yes
Partial regression	Yes (if TZEN=0xB4)	Yes (if TZEN=0xB4)	No
Open debug	Yes (if TZEN=0xB4)	Yes (if TZEN=0xB4)	No
Close debug	Yes	Yes	No

Table 4. Synthesis table for STM32H7Rx/7Sx products

	STM32H7Sxx	STM32H7Rxx
DA storage location	HDPL0 OBK	HDPL0 OBK
DA data encryption	Yes	No
Available DA methods	Password / certificate	Password / certificate
Discovery	Yes	Yes
Full regression	Yes	Yes
Open debug	Yes	Yes
Close debug	Yes	Yes
Forced download	Yes	Yes

2 Overview

2.1 Debug authentication provisioning overview

Before using the debug authentication services, the user must provision STM32 with its credentials. The debug authentication allows two types of credential: password or certificates:

- **Password method**, the user must provision a password hash (SHA256) within STM32.
- **Certificate method**, the user must provision the hash of the public key carried by the debug authentication root certificate and the debug authentication authorized permissions (for regression and / or debug reopening).

The user must provision the DA credentials in product state provisioning.

On STM32:

- The password method is only supported when Arm® TrustZone® is disabled (`TZEN=0xC3`).
- The certificate method is only supported when Arm® TrustZone® is enabled (`TZEN=0XB4`).

Caution: *For STM32H5, beware when provisioning the device. Do not provision a password hash when TZ is enabled, or a root certificate public key hash when TZ is disabled. The device can be permanently locked.*

For more details about DA provisioning, refer to Section 4.1: Provisioning.

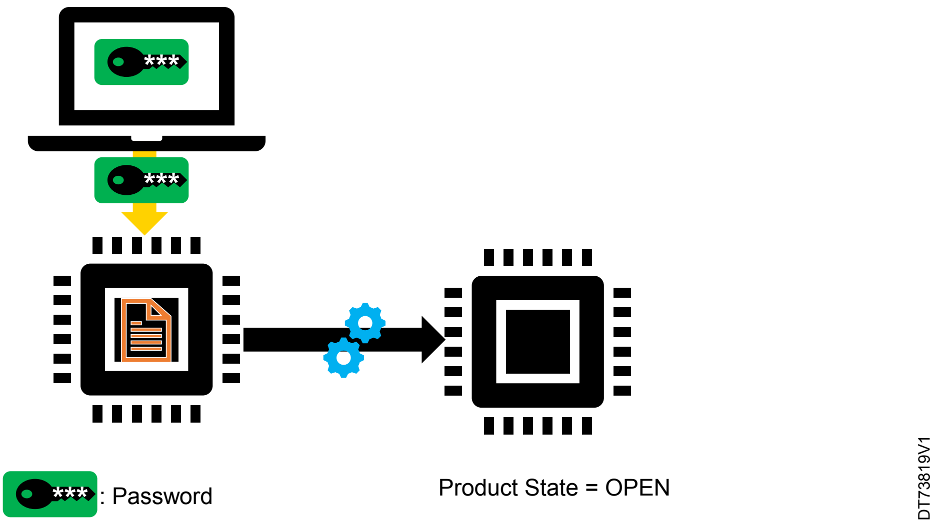
On STM32H7Rx/7Sx: the password method or certificate method is selected through the option bytes DBG_AUTH. Refer to document [6].

2.2 Debug authentication using password overview

When using the password method, only a full regression is possible.

The figure below shows how the user triggers the debug authentication service using the password method.

Figure 2. **Debug authentication using a password**



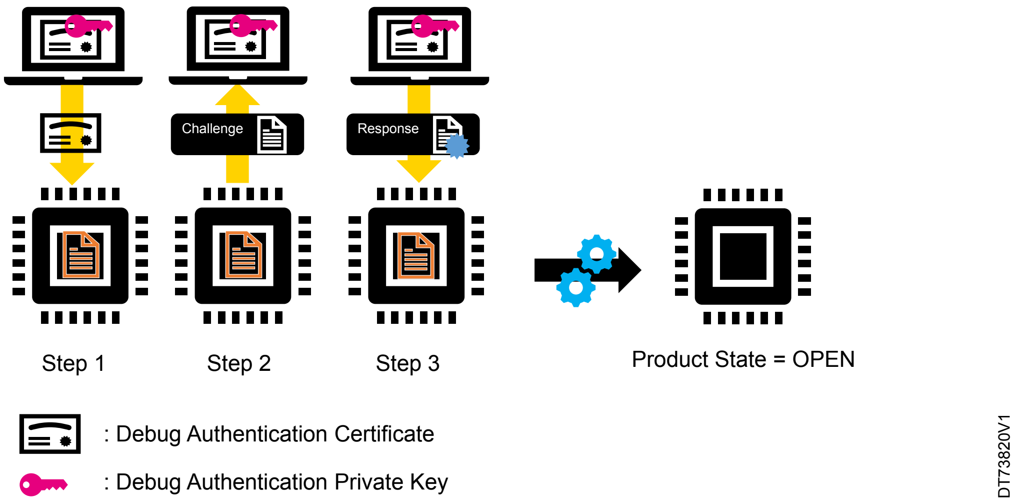
To access the debug authentication feature, the host must send the password to the STM32 device. When the STM32 device receives the password, it verifies that its hash corresponds to the one that is provisioned inside the key storage.

2.3 Debug authentication using certificates overview

To help with understanding of the debugging authentication using certificates process, this section focus on the simplest certificate chain. However, in reality, the STM32 device receives a chain of certificates instead of just one. The principle of the debug authentication certificate chain is similar to the X509 certificate chain, but the certificate format used here is proprietary.

Figure 3 shows how the user triggers the debug authentication service using the certificate method.

Figure 3. Debug authentication using certificate



When the user triggers the debug authentication feature (regression or debug reopening), they first send a certificate and an action request to the STM32 device.

1. On certificate chain reception, the STM32 device:
   – verifies that the root key embedded in the certificate corresponds to the hash of the root public key stored in the device.
   – manages the permissions embedded in the certificates (refer to Section 5.4.3: Permission masks for more details.
   – checks that the requested action fits with the authorized actions list carried by the certificate chain.
2. The STM32 device sends a challenge to the host.
3. The STM32 device verifies that the host owns the debug authentication private key before performing the requested action (regression or debug reopening). The certificate carries the authorized actions. Finally, a token which carries the requested action and the response to the challenge is sent to the device.

3 STM32 debug authentication services description

3.1 Discovery service

3.1.1 Discovery service details

The discovery service allows the user to get information about the device state, especially when debug is closed. The discovery service is available through the STM32CubeProgrammer GUI or CLI.

The information provided by the discovery service is the following:

- **Target ID**: ID of the product. For example, `0x484` for STM32H573 or STM32H563.
- **SoC ID**: ID of the SoC, which is different from one device to another. This ID can be used to restrict a certificate to only one device.
- **SDA version**: version of DA named sda_id in Arm® PSA ADAC specification document.
- **Vendor ID**: it is a two-byte value defined by JEP106 ID spec. IT is SDM which translates the two bytes to "STMicroelectronics".
- **PSA life cycle**: it is a two-byte value representing the PSA life cycle for the upper byte, then the STM32H5 PRODUCT_STATE for the lower byte.
- **PSA auth version**: fixed to “1.0”.
- **ST HDPL1 status**: value coming from HDPL1 OBK.
  - These values are defined by the user when creating the firmware running in HDPL1 (if needed).
  - Or if the user uses STiRoT, the values are defined in document [4].
- **ST HDPL2 status**: value coming from HDPL2 OBK
  - These values are defined by the user when creating the firmware running in HDPL2 (if needed).
- **ST HDPL3 status**: value coming from HDPL3 OBK.
  - These values are defined by the user when creating the firmware running in HDPL3 (if needed).
- **Token formats**: fixed to “`0x200`”.
- **Certificate formats**: fixed to “`0x201`”.
- **Cryptosystem**: “Ecdsa-P256 SHA256” (certificate) or “ST password”.
- **ST provisioning integrity**: indicates if integrity of provisioned DA data is correct (`0xeaeaeaea`) or wrong (`0xf5f5f5f5`).

Table 5. **Discovery service availability**

Part number	Availability
STM32H5xx	Every product state except Locked
STM32H7Rs/H7Sx	Provisioning and Closed product states only

3.1.2 CLI command for discovery command

In order to launch a discovery command, use the following command:

```
.\STM32_Programmer_CLI.exe -c port=SWD debugauth=2
```

3.2 Full regression service

3.2.1 Full regression details

The full regression service changes the product state to open.

Debug authentication filters the regression request according to the user credential that comes with it. Therefore, if a full regression is requested and it is not authorized by the device configuration, the request is rejected.

When launching the full regression service, the following actions are performed:

- Erase fully the memories (user flash memory, OBK, SRAM, and back-up RAM).

- Increments secure and nonsecure EPOCH.
- Remove HDP, secure watermarks, and WRP protections.
- Reset TZEN and boot lock to default value.
- Erases key registers within cryptographic peripherals (AES, SAES, PKA, OTFDEC).
- Change product state to open.

The full regression service is not available in the product state open or locked.

3.2.2 STM32 series full regression support

Table 6. STM32 series full regression support

Part number	Full regression supported
STM32H573xx	Yes
STM32H563xx/STM32H562xx	Yes
STM32H533xx	Yes
STM32H523xx	Yes
STM32H503xx	Yes
STM32H7Rx/7Sx	Yes

3.2.3 CLI commands for full regression

In order to launch a full regression by using certificate method, use the following command:

```
.\STM32_Programmer_CLI.exe -c port=SWD speed=fast per=a key=.\key.pem cert=.\certificates_cha
in debugauth=1
```

with

- key.pem = .pem file containing the user private key
- certificates_chain = .b64 file containing the chain of certificates

In order to launch a full regression by using password method, use the following command:

```
.\STM32_Programmer_CLI.exe -c port=SWD per=a pwd=.\password.bin debugauth=1
```

with

- password.bin = binary file containing the password

3.3 Partial regression service

3.3.1 Partial regression service details

The partial regression service changes the product state to TZ-closed.

When launching the partial regression service, the following actions are performed:

- Erase the STM32 nonsecure memories:
  - Nonsecure user flash memory. User flash sectors are not covered by flash secure watermark registers.
  - HDPL3 nonsecure OBK
  - Back up RAM
- Increments the nonsecure EPOCH
- Change the product state to TZ-closed

The partial regression service is only available in the product state closed.

3.3.2 Partial regression service and EEPROM emulation

If EEPROM emulation is activated, the partial regression deactivates it before performing the erasure, and reactivates it at the end of the process. Hence, the data in the EEPROM zone is fully erased after a partial regression.

3.3.3 Partial regression service and HDP areas

If an HDP area is defined and covers secure and nonsecure sectors, the partial regression service erases the nonsecure sectors and resizes the HDP area to cover only the secure sectors after the partial regression processing.

3.3.4 Partial regression service and WRP

When launching the partial regression service, the WRP protection is removed on nonsecure sectors.

As the WRP protection is defined on a eight Kbytes sector, it is mandatory to match the secure/nonsecure area limits with the eight Kbytes sector mapping.

In other words, the start and the end of the secure area must match a multiple of four sectors if the WRP is activated on the secure area. Mixing secure and nonsecure sectors in the WRP zone is not authorized.

If this rule is not applied, an error blocks the partial regression service from running.

3.3.5 STM32 series partial regression support

Table 7. STM32 series partial regression support

Part number	Partial regression supported
STM32H573xx	Yes
STM32H563xx/STM32H562xx	Yes
STM32H533xx	Yes
STM32H523xx	Yes
STM32H503xx	No
STM32H7Rx/7Sx	No

3.3.6 CLI command for partial regression

In order to launch a partial regression by using certificate method, use the following command:

```
.\STM32_Programmer_CLI.exe -c port=SWD speed=fast per=b key=.\key.pem cert=.\certificates_cha
in debugauth=1
```

3.4 Debug reopening service

3.4.1 Debug reopening service details

The debug reopening service modifies the debug state from the default one (see table below). This is done in a way to never expose assets to people not allowed to access them.

Debug authentication filters the debug reopening request according to the user credential that comes with it.

Indeed, STM32 is in a different debug state according to the current STM32 product state.

The table below depicts the STM32 default debug state for each product state.

Table 8. Debug connection vs product state

Product state	Debug connection		
-	Secure	Nonsecure	Debug open
Open	Yes	Yes	Yes for HDPL 1, 2, and 3
Provisioning	No(1)	Yes(2)	Yes for HDPL 1

Product state	Debug connection		
iRoT provisioned	No[1]	Yes[2]	Yes for HDPL 3
TZ-Closed	No[1]	Yes[2]	Yes for HDPL 3
Closed	No[1]	No	No
Locked	No[3]	No[3]	No[3]

1. *STM32 secure resources cannot be accessed even if the user establishes a debug connection to STM32.*
2. *The user can only establish a debug connection to STM32 when Cortex®-M33 is running in nonsecure domain. With a debug connection, the user can access every nonsecure resource of the system.*
3. *The debug authentication service is not available in product state LOCKED.*

The debug reopening service can reenable:

- Secure debug connection whereas it is not supported by the STM32 product state.
- Nonsecure debug connection whereas it is not supported by the STM32 product state.
- Debug connection in an HDP level whereas it is not supported by the STM32 product state.

For example, the STM32 is in the product state closed.

In this product state, the user cannot establish a debug connection neither in secure nor in nonsecure whatever the HDP level.

By using the debug reopening service, the user requests STM32 to reopen the debug in nonsecure with HDP level 3. Nonsecure with HDP level 3 is the debug reopening context request.

With such a request, the user can connect via debug to STM32 when it is only running in a nonsecure domain with HDP level equal to 3. Before being able to establish such a debug connection, STM32 must complete the following boot path to reach the debug reopening context request:

- STM32 starts in the secure domain with HDP level set to 1 (secure boot, for example, iRoT, usual context). STM32 does not reach the debug reopening context request, users cannot connect to STM32 via debug.
- iRoT jumps to uRoT and increments the HDP level to 2. STM32 does not reach the debug reopening context request, the user cannot connect to STM32 via debug.
- uRoT jumps to the secure application and increments the HDP level to 3. STM32 does not reach the debug reopening context request, the user cannot connect to STM32 via debug.
- Secure application jumps to nonsecure application. STM32 reaches the debug reopening context request, the user can connect to STM32 via debug.

Note: *When the debug reopening context request targets an HDP level n, the user can establish a debug connection in an HDP level x with x higher or equal to n and x lower or equal to 3.*

For example, with a debug reopening context request set to secure HDP level 1, thanks to debug authentication reopening service, the user can establish a debug connection in secure HDP level 1, 2, and 3.

The user cancels the debug reopening effect by applying a power cycle on the STM32 (power off and power on). After such an action, the STM32 debug state comes back to the one driven by the STM32 product state. That also means that the debug opening is persistent to reset.

The debug reopening service can be launched from every product state except locked.

3.4.2 Debug reopening and STiRoT

When the user sets the STM32 boot entry on STiRoT within the STM32 system flash memory, the debug reopening service does not grant reopening debug with the debug reopening context request set to the secure HDP level 1.

3.4.3 STM32 debug reopening support

Table 9. **STM32 debug reopening support**

Part number	Debug reopening supported
STM32H573xx	Yes
STM32H563xx/STM32H562xx	Yes

Part number	Debug reopening supported
STM32H533xx	Yes
STM32H523xx	Yes
STM32H503xx	No
STM32H7Rx/7Sx	Yes

3.4.4 CLI commands to reopen debug on STM32H5

In order to reopen debug from HDPL1 secure, use the following command:

```
.\STM32_Programmer_CLI.exe -c port=SWD speed=fast per=e key=.\key.pem cert=.\certificate_chai
n debugauth=1
```

In order to reopen debug from HDPL2 secure, use the following command:

```
.\STM32_Programmer_CLI.exe -c port=SWD speed=fast per=f key=.\key.pem cert=.\certificate_chai
n debugauth=1
```

In order to reopen debug from HDPL3 secure, use the following command:

```
.\STM32_Programmer_CLI.exe -c port=SWD speed=fast per=g key=.\key.pem cert=.\certificate_chai
n debugauth=1
```

In order to reopen debug from HDPL1 nonsecure, use the following command:

```
.\STM32_Programmer_CLI.exe -c port=SWD speed=fast per=h key=.\key.pem cert=.\certificate_chai
n debugauth=1
```

In order to reopen debug from HDPL2 nonsecure, use the following command:

```
.\STM32_Programmer_CLI.exe -c port=SWD speed=fast per=i key=.\key.pem cert=.\certificate_chai
n debugauth=1
```

In order to reopen debug from HDPL3 nonsecure, use the following command:

```
.\STM32_Programmer_CLI.exe -c port=SWD speed=fast per=j key=.\key.pem cert=.\certificate_chai
n debugauth=1
```

3.4.5 CLI commands to reopen debug on STM32H7Rx/7Sx

In order to reopen debug from HDPL1, use the following command:

```
.\STM32_Programmer_CLI.exe -c port=SWD speed=fast per=d key=.\key.pem cert=.\certificate_chai
n debugauth=1
```

In order to reopen debug from HDPL2, use the following command:

```
.\STM32_Programmer_CLI.exe -c port=SWD speed=fast per=c key=.\key.pem cert=.\certificate_chai
n debugauth=1
```

In order to reopen debug from HDPL3, use the following command:

```
.\STM32_Programmer_CLI.exe -c port=SWD speed=fast per=b key=.\key.pem cert=.\certificate_chai
n debugauth=1
```

3.5 Close debug service

3.5.1 Close debug details

The close debug service enables closing the debug once it has been opened through the debug reopening service.

3.5.2 STM32 close debug support

Table 10. STM32 close debug support

Part number	Close debug supported
STM32H573xx	Yes
STM32H563xx/STM32H563xx	Yes
STM32H533xx	Yes
STM32H523xx	Yes
STM32H503xx	No
STM32H7Rx/7Sx	Yes

3.5.3 CLI command for close debug command

In order to close the debug, use the following command:

```
.\STM32_Programmer_CLI.exe -c port=SWD debugauth=3
```

3.6 Forced download service

3.6.1 Forced download details

The forced download service allows the connection to ST bootloader, for example, to be able to download an image when using an iRoT.

3.6.2 STM32H7Rx/7Sx series forced download support

Table 11. STM32H7Rx/7Sx series forced download support

Part number	Forced download support
STM32H7Sxx	Yes
STM32H7Rxx	Yes

3.6.3 CLI command for forced download

In order to launch the forced download service, use the following command:

```
.\STM32_Programmer_CLI.exe -c port=SWD speed=fast per=e key=.\key.pem cert=.\certificate_chai
n debugauth=1
```

4 Debug authentication activation

4.1 Provisioning

4.1.1 Introduction

The debug authentication use two kinds of methods:

- A password (maximum length of password is 128 bits/16 bytes).
- A certificate chain.

Before using the debug authentication service, the user must provision STM32 with data for debug authentication configuration.

The debug authentication configuration includes three main information:

- A SHA256 hash on below data, for integrity check
- A DA credential related to the chosen authentication method, named debug authentication root key in TPC
- A permission mask

On STM32H503xx, debug authentication data is provisioned and not encrypted in OTP (at address `0x08FFF000`).

OTP programming is done by using STM32CubeProgrammer.

On STM32H573xx/STM32H533xx (crypto device), DA data is stored encrypted (AES-CBC) in HDPL1 OBK (at address `0x0FFD0100`).

On STM32H563xx/STM32H523xx (noncrypto device), DA data is stored not encrypted in HDPL1 OBK (at address `0x0FFD0100`).

OBK generation is done by using the STM32 Trusted Package Creator.

OBK provisioning is done by using the STM32CubeProgrammer.

OBK generation is done by the STM32 Trusted Package Creator.

OBK provisioning is done by using the STM32CubeProgrammer.

For more details, also refer to document [5].

On STM32H7Rx/7Sx, DA data is stored encrypted (AES-CBC) in HDPL0 OBK (at index 8).

4.1.2 Debug authentication using password

When using the password method, the debug authentication root key is the SHA256 hash of the password to use for debug authentication.

4.1.3 Debug authentication using certificates

When using the certificate method, the debug authentication root key is the SHA256 hash of the root certificate public key.

4.1.4 Permission mask

A permission mask is only relevant when using the certificate method.

In addition to debug authentication credentials, the user must provision the permission mask, which defines the service the user wants to authorize on the device.

The permission mask is a field of 128 bits with only the first 16 LSB bits used to authorize services (regressions, debug reopening).

Bit 127	Bit 126	Bit 125	Bit 124	Bit 123	Bit 122	Bit 121	Bit 120
Forced download	*Reserved*	*Reserved*	*Reserved*	*Reserved*	*Reserved*	*Reserved*	*Reserved*

Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8
Reserved	Full regression	*Reserved*	Partial regression	*Reserved*	*Reserved*	*Reserved*	*Reserved*

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Reserved	Open debug from HDPL3 S/NS	Open debug from HDPL2 S/NS	Open debug from HDPL1 S/NS	*Reserved*	Open debug from HDPL3 NS	Open debug from HDPL2 NS	Open debug from HDPL1 NS

4.2 Debug authentication trigger

Debug host must run in sequence the two actions depicted below:

1. Debug host uses SWD/JTAG with access point 0 to write ‘STDA’ character list within `DBGMCU_DBG_AUTH_HOST` register.
2. Debug host resets the debug target.

After this sequence the STM32 device starts the debug authentication protocol.

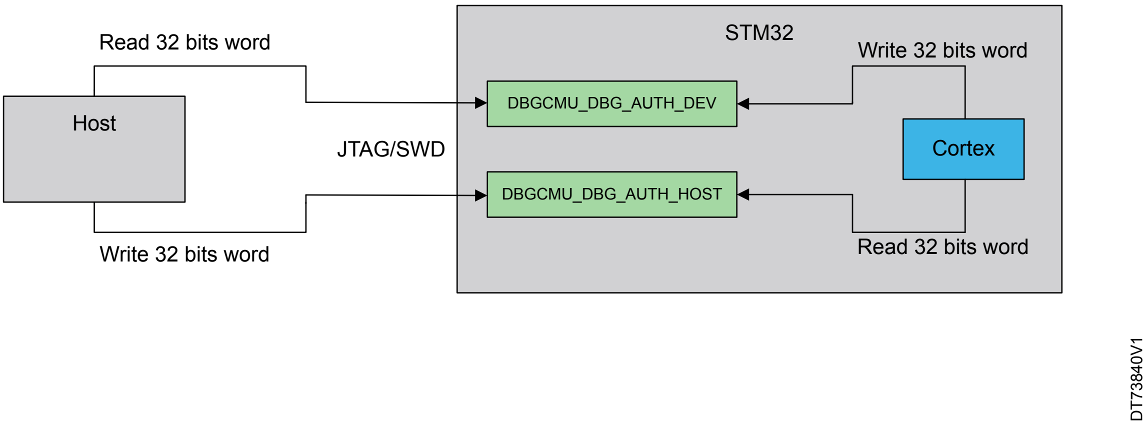
5 STM32 debug authentication protocol description

5.1 Physical link

Host and the STM32 device use JTAG or SWD physical connection over access point 0.

Using access point 0, debug transactions only access a very limited part of the STM32 device, for example, the DBGMCU IP.

Figure 4. DBGMCU usage



DBGMCU acts as a mailbox between the host and the STM32 device.

The debug host uses JTAG/SWD to write a word within the `DBGMCU_DBG_AUTH_HOST` register in order to send messages to the debug authentication service of the STM32 device.

The debug authentication service of the STM32 device reads the same register to get the messages.

The debug authentication service writes a word within the `DBGMCU_DBG_AUTH_DEV` register in order to send a word to the debug host.

The debug host uses JTAG/SWD to read a single word from the `DBGMCU_DBG_AUTH_DEV` register.

The debug host and the STM32 device use the `DBGMCU_DBG_AUTH_ACK` register for acknowledgment of exchanges from the STM32 device to the debug host and from the debug host to the STM32 device.

For more details about the DBGMCU register, refer to document [2].

Host and STM32 use the Arm® PSA ADAC protocol over DBGMCU mailbox.

Once the debug authentication sequence is completed, STM32 debug authentication opens access point 1 and the debug host can establish the debug connection with it.

With access point 1, the debug host accesses all STM32 resources granted by the debug reopening context request.

5.2 STM32 debug authentication protocol overview

STM32 debug authentication services use the Arm® PSA ADAC protocol depicted by document [1].
The STM32CubeProgrammer embeds this protocol and can be used without extra effort.

This protocol specifies five commands:

Command constant	Command name	Description
0x0001	ADAC_DISCOVERY_CMD	The host requests information about the debug target (STM32) using with this command.

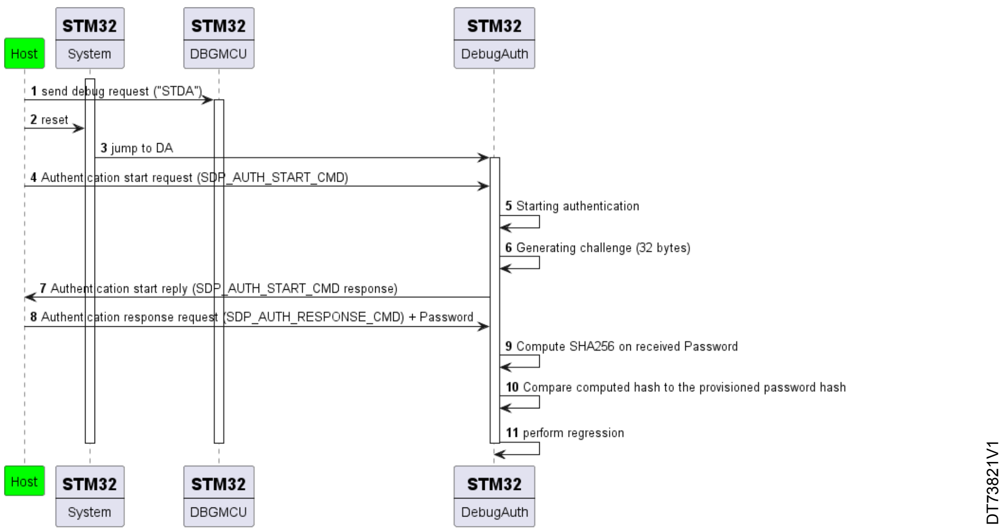
Command constant	Command name	Description
0x0002	ADAC_AUTH_START_CMD	The host sends this command to start the authentication sequence. Its primary purpose is for the target to provide a random 256-bit challenge vector used to prevent replay attacks.
0x0003	ADAC_AUTH_RESPONSE_CMD	This command is used to provide the debug token and additional credentials as part of a complete authentication response to the target.
0x0005	ADAC_CLOSE_SESSION_CMD	Not used in STM32 debug authentication service
0x0006	ADAC_LOCK_DEBUG_CMD	The lock debug command restores the device's debug access controls to the locked state, given its current life cycle state.

5.3 Debug authentication using password

The debug authentication with password method reuses protocol defined by document [1] with the debug authentication response message customized with a new field that carries the password.

The following figure describes the debug authentication messages sequence when using the password method to trigger a full regression.

Figure 5. Debug authentication using password



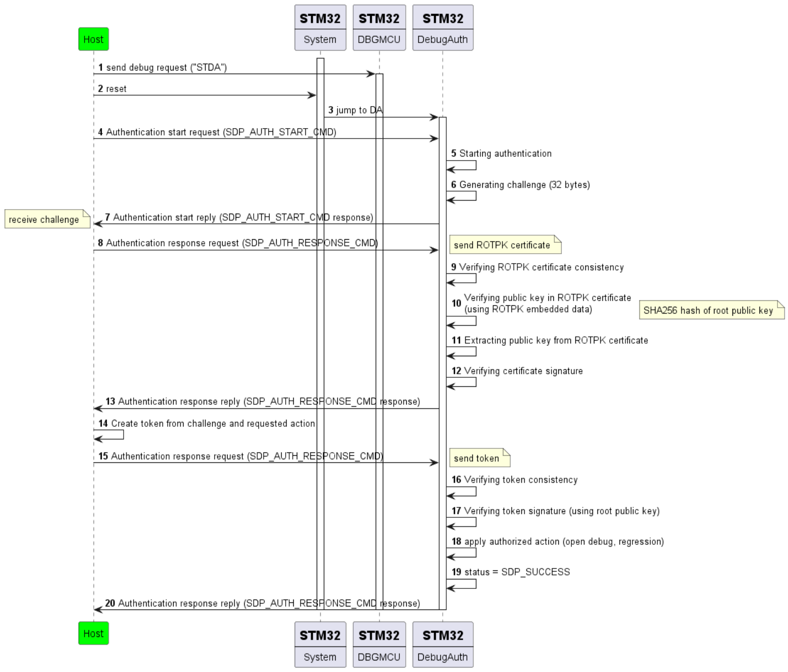
5.4 Debug authentication using a certificate chain

5.4.1 Sequence diagram

The STM32 debug authentication service uses a certificate format and certificate chain defined by document [1].

Figure 6 describes the debug authentication messages sequence when using the certificate method (in this case, only one root certificate is used).

Figure 6. **Debug authentication using a root certificate**



DT73822V1

5.4.2 Certificates and certificate chains

There are three types of certificates:

- Root certificate
- Intermediate certificate
- Leaf certificate

A certificate chain can be composed of:

- A root certificate only
- A root certificate + a leaf certificate
- A root certificate + Nx intermediate certificates + a leaf certificate

Certificates and certificate chain are created by using STM32 Trusted Package Creator.

Example of usage

A manufacturer (root level) subcontracts some services to other entities (intermediate level). These subcontractors also subcontract some of their services to other entities (leaf level).

5.4.3 Permission masks

Each certificate brings additional limitations to the authorized actions through a permission mask.

When using the certificate method, the requested action is applied only if the accumulation of the different masks of the chain authorizes this action.

Several masks are involved in the permission accumulation:

- The product mask, which is hardcoded in the device (all supported actions are authorized).
- The permission mask, which is defined in HDPL1 OBK during the provisioning sequence (refer to Section 4.1.4 for permission mask details).
- The certificate mask defined in each certificate of the certificate chain.
- The token mask defined in the token to trigger an action.

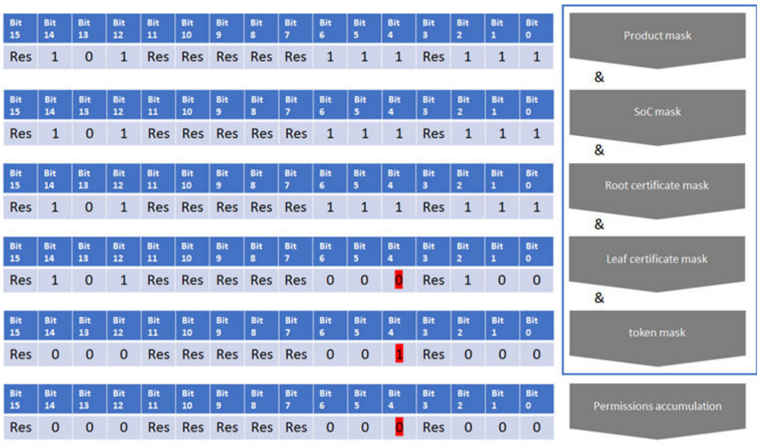
Example of a forbidden action

In this example, the token mask is used to request a debug opening from HDPL1 S/NS but the permission mask defined in the leaf certificate forbids the debug opening from HDPL1.

So finally, the requested action is rejected.

In the example described in Figure 7, the certificate chain contains a root certificate and a leaf certificate.

Figure 7. Example of a forbidden action

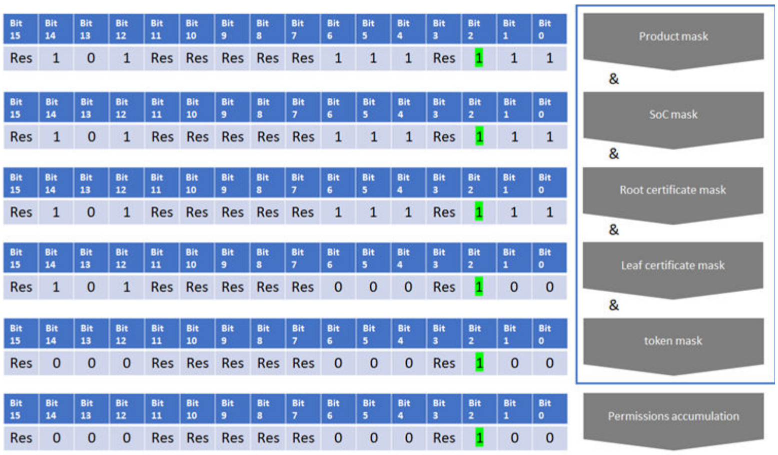


Example of an authorized action

In this example, the token mask is used to request a debug opening from HDPL3 NS. The permission accumulation allows this action so it is applied.

In the example described Figure 8, the certificate chain contains a root certificate and a leaf certificate.

Figure 8. Example of an authorized action



5.4.4 Certificates and product series/device filtering

It is possible to restrict the use of the certificates to:

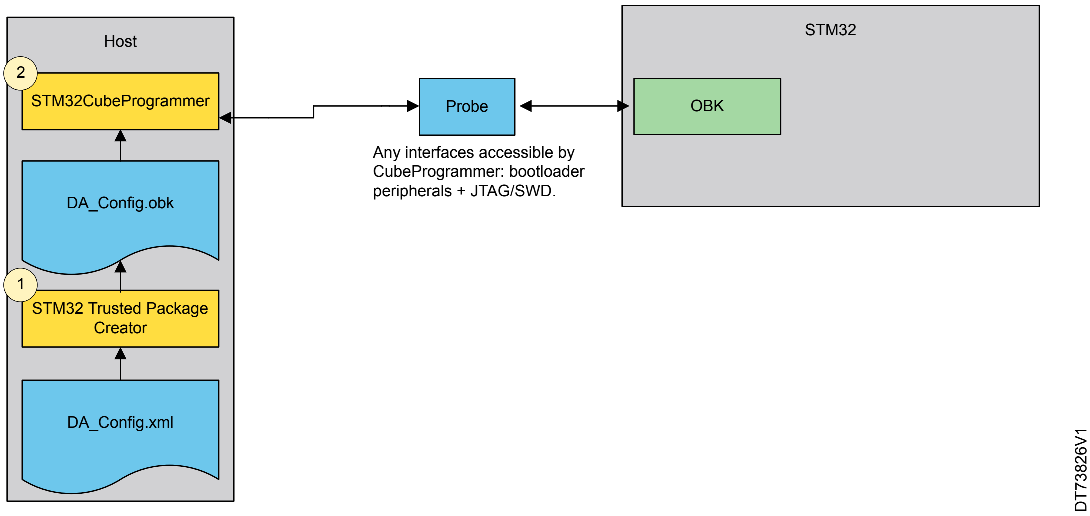
- A product series by using the target ID (also named SoC class) when generating the certificate.
  - For example, target ID for STM32H573xx is 0x484.
- A specific device by using the SoC ID when generating the certificate.

Target ID and SoC ID are information provided by the discovery service (refer to Section 3.1).

6 Debug authentication ecosystem overview

6.1 Debug authentication provisioning

Figure 9. Debug authentication ecosystem during provisioning phase



Step 1

The STM32 Trusted Package Creator is used to create the debug authentication configuration .obk file from the debug authentication configuration .xml file.

Table 12. DA configuration xml file details

Parameter	Description	Additional comment
Root public key	.pem file for root public key	-
Permission mask	Authorized permissions (128 bits)	Refer to Section 4.1.4.

Table 13. Debug authentication configuration obk file details

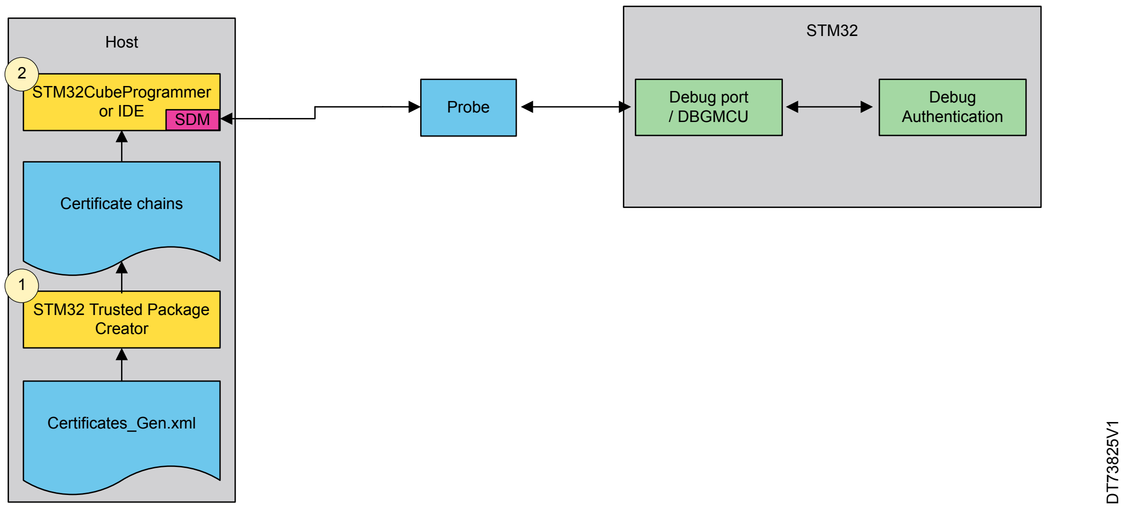
Offset	Size	Type	Description
0	4	uint32_t	HDPL1 OBK address
4	4	uint32_t	Size of data to program in OBK
8	4	uint32_t	Field to specify if encryption of OBK must be done: 1 = encryption done, 0 = encryption not done
12	32	uint8_t	Integrity SHA256 hash on data ([hash of root public key or password], [permission mask], and [reserved])
44	32	uint8_t	SHA256 hash of root public key or password
76	16	uint8_t	Permission mask
92	16	uint8_t	Reserved

Step 2

The DA configuration obk file is programmed in STM32 OBK by using the STM32CubeProgrammer.

6.2 Launch debug authentication service (certificate method)

Figure 10. **Debug authentication ecosystem for service launch**



Step 1

The STM32 Trusted Package Creator is used to create the certificate chain from the certificate generation xml file.

Table 14. **Debug authentication certificates generation xml file details**

Parameter	Description	Additional comment
Certificate role	Role of the generated certificate (root/ intermediate/leaf)	-
Root or issuer private key	Private key of root or of the issuer	-
Root or Intermediate of Leaf public key	Public key of root or intermediate or leaf	-
Usage	-	Not yet supported
PSA security life cycle	-	Not yet supported
Implementation defined state	-	Not yet supported
OEM constraint	-	Not yet supported
SoC ID	SoC ID used to generate a specific certificate to one device	By default, value is zero.
SoC class	SoC class used to generate a specific certificate to one product series	By default, value is zero.
Permission mask	Actions authorized by the generated certificate	Refer to Section 4.1.4.
Input certificate for chaining	Certificate chain to which the generated certificate must be added	-

Step 2

A debug authentication service is launched by using the STM32CubeProgrammer or an IDE (both integrating the SDM library) to send the certificates chain to the device.

7 STM32 debug authentication restrictions

7.1 Debug authentication and WWDG

The debug authentication does not manage Window WatchDog (WWDG).

It is recommended not to activate Window WatchDog when using debug authentication.

7.2 Debug Authentication and tamper management

Internal tampers 9 and 15 are systematically activated during debug authentication services call and stay activated after.

Note: They remain active if backup domain is maintained powered-on during system reset.

If a tamper event occurs during debug authentication execution, a reset is performed, and the content of the TAMP status register is written in the last 32-bits of the BKPSRAM and a 32-bits magic value (0x003981bb) is placed just before. Based on this information, the application can take the appropriate action and then clear the magic value.

Revision history

Table 15. **Document revision history**

Date	Version	Changes
22-Dec-2023	1	Initial release.
13-May-2024	2	Added mention of STM32H7Rx/7Sx throughout the document. Topic added: Section 3.6: Forced download service
13-May-2025	3	Updated: Section 6.1: Debug authentication provisioning
13-Jun-2025	4	Updated: Section 6.1: Debug authentication provisioning Added: Section 7.2: Debug Authentication and tamper management

Contents

List of tables

List of figures