PSA Crypto API Lab

This lab procedure walks through the steps to generate a wrapped key in the Secure Vault High device and use the PSA Crypto API to perform the ECDSA digital signature.

KEY POINTS

- Secure key storage in the Secure Vault High device
- ECDSA sign and verify

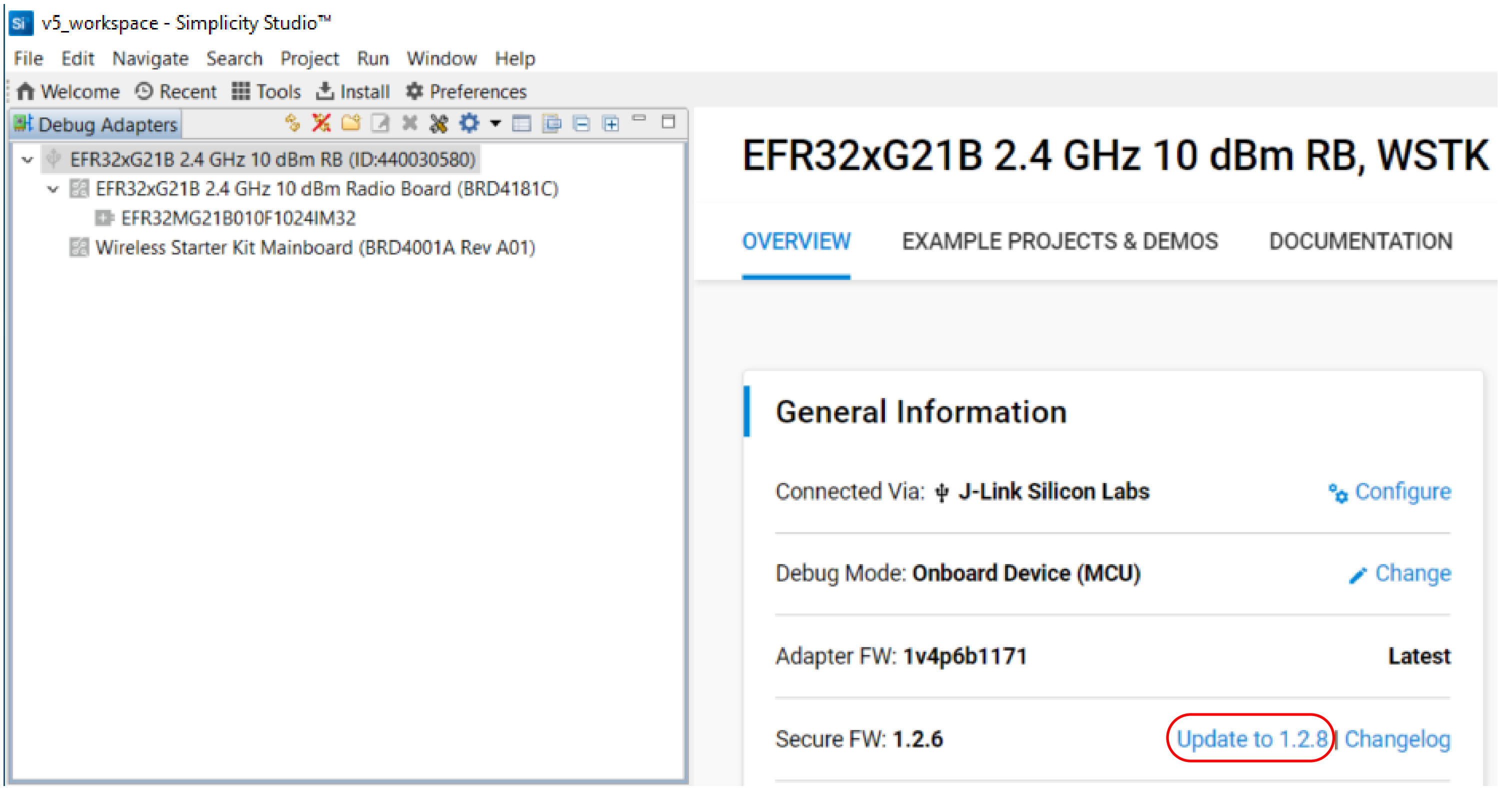
1 Prerequisites

For this lab you will need the following:

- EFR32xG21B 2.4G Hz 10 dBm Radio Board (BRD4181C)
- Simplicity Studio v5 with Gecko SDK Suite v3.2.0
- Terminal program to receive UART communication form the WSTK

1.1 Update the SE firmware

Click the **Update to <version>** in **Update to <version> | Changelog** link (if it appears) to update the **Secure FW** (version **1.2.8 or above**).



EFR32xG21B 2.4 GHz 10 dBm RB (ID:440030580)
EFR32xG21B 2.4 GHz 10 dBm Radio Board (BRD4181C)
EFR32MG21B010F1024IM32
Wireless Starter Kit Mainboard (BRD4001A Rev A01)

EFR32xG21B 2.4 GHz 10 dBm RB, WSTK

General Information

Connected Via: J-Link Silicon Labs
Debug Mode: Onboard Device (MCU)
Adapter FW: 1v4p6b1171 Latest
Secure FW: 1.2.6

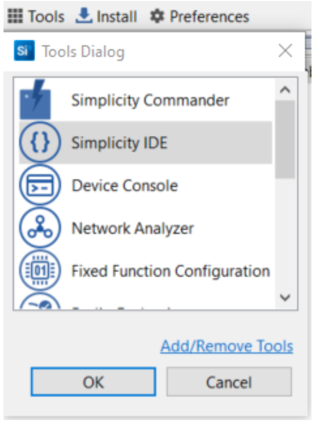
1.2 Erase the flash

Run the *security erasedevice* command in Simplicity Commander to erase the main flash. This procedure is to make sure there is no security key in NVM3.

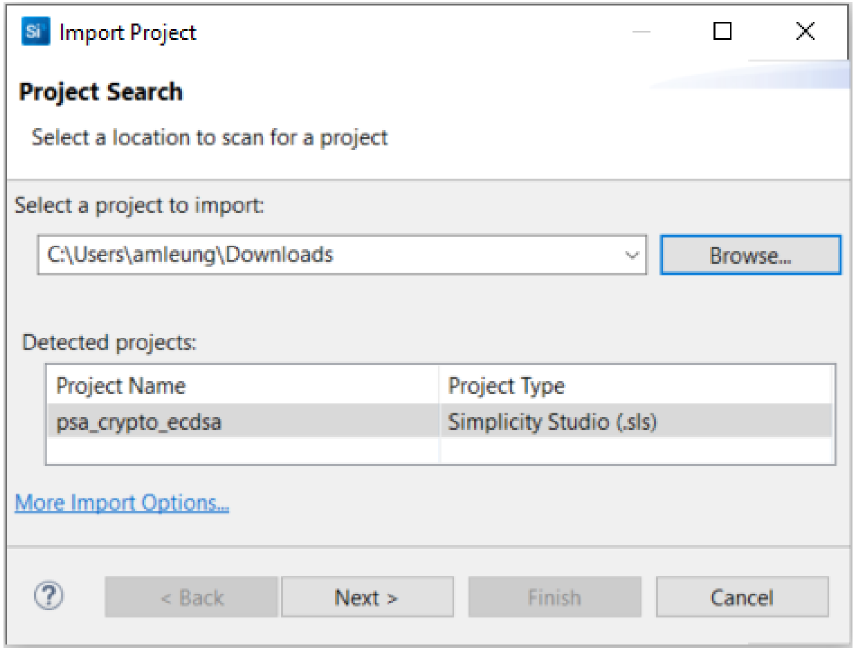
```
commander security erasedevice
```

1.3 Import the Project to Simplicity IDE

1. Go to Launcher Perspective of Simplicity Studio, click the **Tools** icon to select **Simplicity IDE**
2. Click **OK** to launch the Simplicity IDE



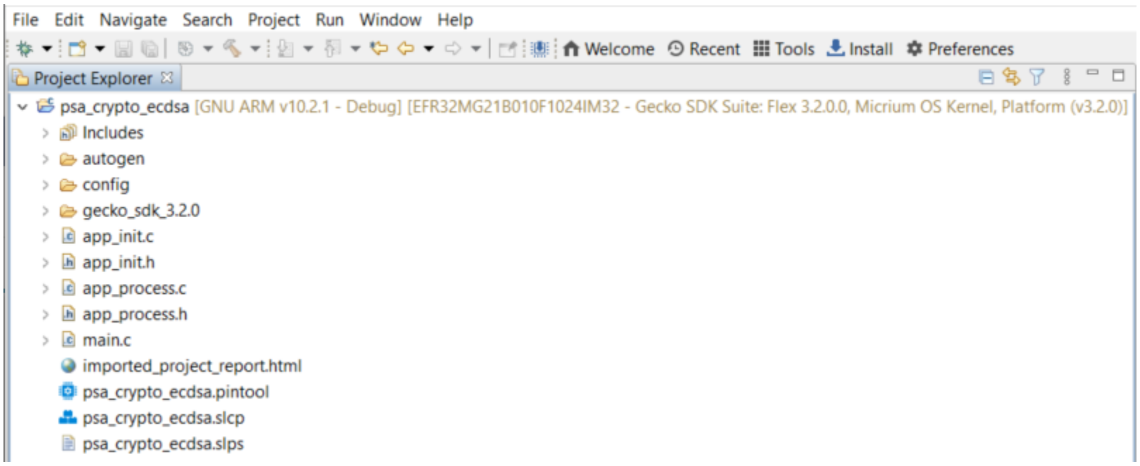
3. Import the *psa_crypto_ecdsa* project from the folder containing the *psa_crypto_ecdsa.sls* (in psa_crypto_api_lab.zip) file



C:\Users\amleung\Downloads

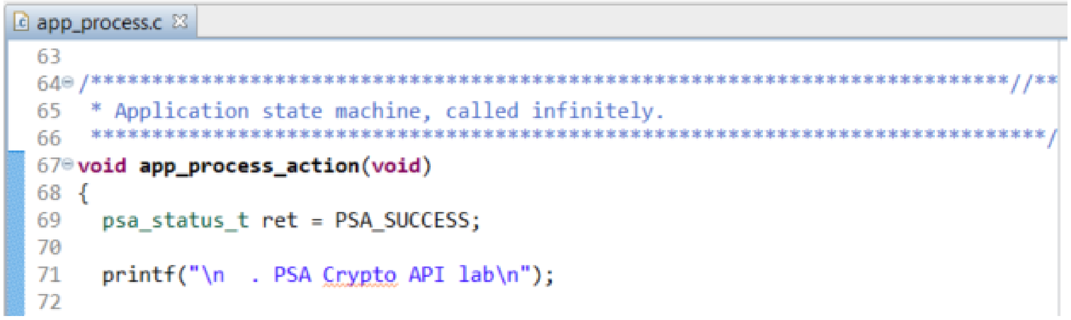
Project Name	Project Type
psa_crypto_ecdsa	Simplicity Studio (.sls)

4. Click **Next**, **Next**, and **Finish** to generate the project



psa_crypto_ecdsa [GNU ARM v10.2.1 - Debug] [EFR32MG21B010F1024IM32 - Gecko SDK Suite: Flex 3.2.0.0, Micrium OS Kernel, Platform (v3.2.0)]
- Includes
- autogen
- config
- gecko_sdk_3.2.0
- app_init.c
- app_init.h
- app_process.c
- app_process.h
- main.c
- imported_project_report.html
- psa_crypto_ecdsa.pintool
- psa_crypto_ecdsa.slcp
- psa_crypto_ecdsa.slps

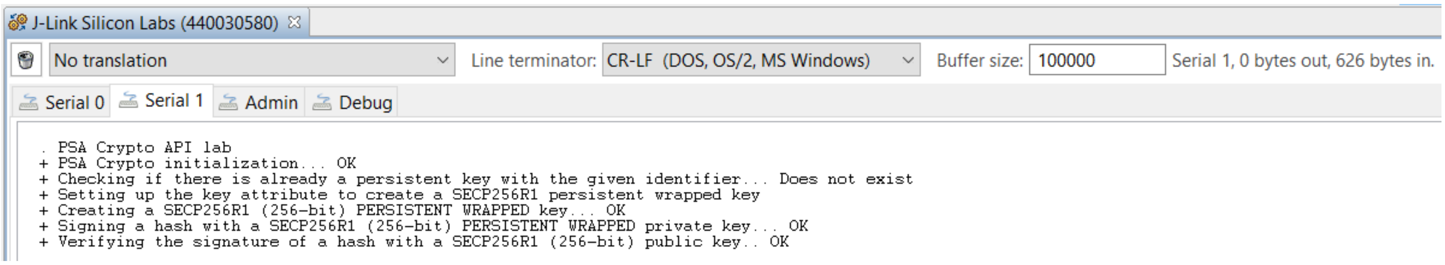
5. Open the *app_process.c* file to start the lab



```
/***************************************************************************//**
 * Application state machine, called infinitely.
 ******************************************************************************/
void app_process_action(void)
{
  psa_status_t ret = PSA_SUCCESS;

  printf("\n  . PSA Crypto API lab\n");

```

2 Coding

The figure below is the expected output on the terminal program.


J-Link Silicon Labs (440030580)
No translation
Line terminator: CR-LF (DOS, OS/2, MS Windows)
Buffer size: 100000
Serial 1, 0 bytes out, 626 bytes in.
Serial 0
Serial 1
Admin
Debug

```
. PSA Crypto API lab
+ PSA Crypto initialization... OK
+ Checking if there is already a persistent key with the given identifier... Does not exist
+ Setting up the key attribute to create a SECP256R1 persistent wrapped key
+ Creating a SECP256R1 (256-bit) PERSISTENT WRAPPED key... OK
+ Signing a hash with a SECP256R1 (256-bit) PERSISTENT WRAPPED private key... OK
+ Verifying the signature of a hash with a SECP256R1 (256-bit) public key.. OK
```

Hints:

- The slides in the *PSA Crypto.pptx*
- The PSA Crypto API document - https://docs.silabs.com/mbed-tls/latest/
- The PSA Crypto macros are defined in the crypto_values.h (*C:\SiliconLabs\SimplicityStudio\v5\developer\sdks\gecko_sdk_suite\v3.2\util\third_party\crypto\mbedtls\include\psa*)
- The source code of the PSA Crypto ECDSA platform example (*C:\SiliconLabs\SimplicityStudio\v5\developer\sdks\gecko_sdk_suite\v3.2\app\common\example\psa_crypto_ecdsa*)

2.1 PSA Crypto Initialization

Write the code for PSA Crypto initialization in the *app_process.c*.

```
// Initialize the PSA Crypto
printf("  + PSA Crypto initialization... ");

// Write your code here
```

2.2 Set Up the Key Attributes

Write the code to set up the key attributes in the *app_process.c*.

```
// Set up the key attribute to create a SECP256R1 persistent wrapped key, key ID is
PERSISTENT_KEY_ID
printf("  + Setting up the key attribute to create a SECP256R1 persistent wrapped key\n");

// Write your code here
```

2.3 Generate a SECP256R1 Persistent Wrapped Key

Write the code to generate a SECP256R1 persistent wrapped key in the *app_process.c*.

```
// Generate a SECP256R1 persistent wrapped key
printf("  + Creating a SECP256R1 (256-bit) PERSISTENT WRAPPED key... ");

// Write your code here
```

2.4 Sign a Hash with the SECP256R1 Private Key

Write the code to sign a hash in the *app_process.c*.

```
// Sign a hash with the SECP256R1 private key
printf("  + Signing a hash with a SECP256R1 (256-bit) PERSISTENT WRAPPED private key... ");

// Write your code here
```

2.5 Verify the Signature with the SECP256R1 Public Key

Write the code to verify the signature in the *app_process.c*.

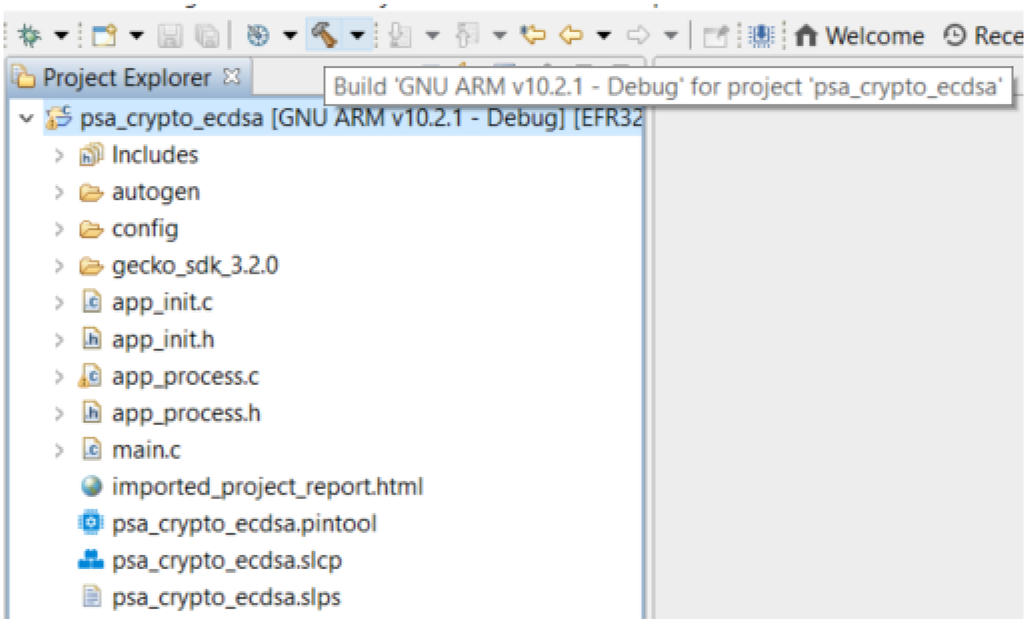
```
// Verify the signature with the SECP256R1 public key
printf("  + Verifying the signature of a hash with a SECP256R1 (256-bit) public key.. ");

// Write your code here
```

2.6 Reference solution

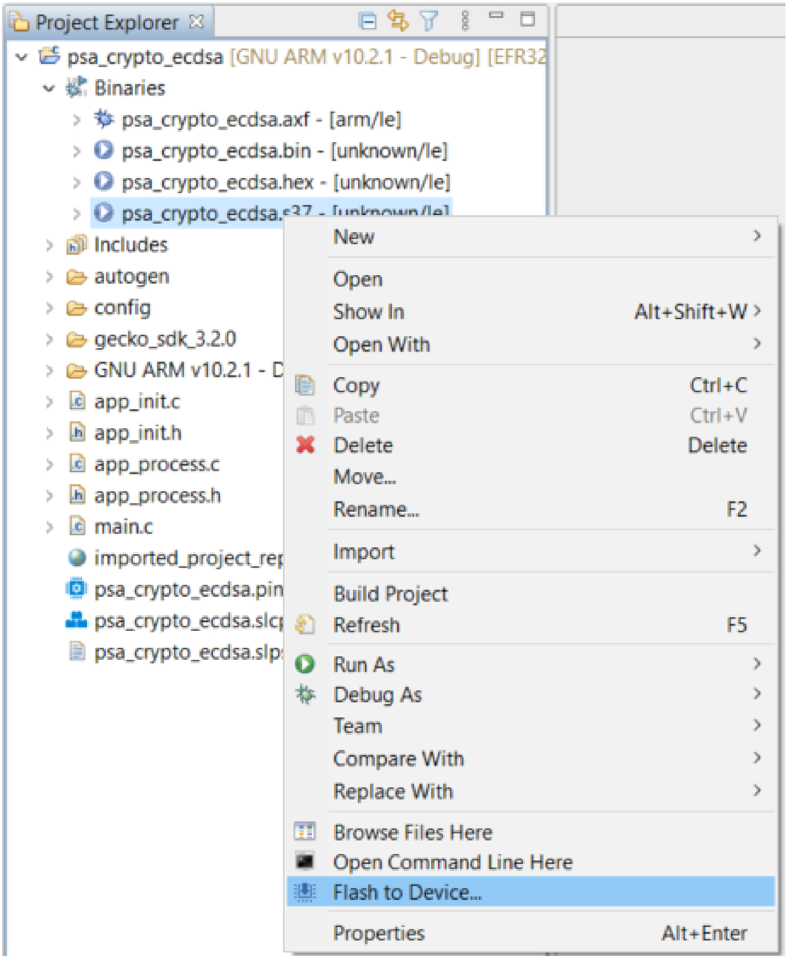
The reference solution can be found in the *app_process_solution.c* (in psa_crypto_api_lab.zip) file.

3 Testing

3.1 Build and Download the Code to the BRD4181C Radio Board

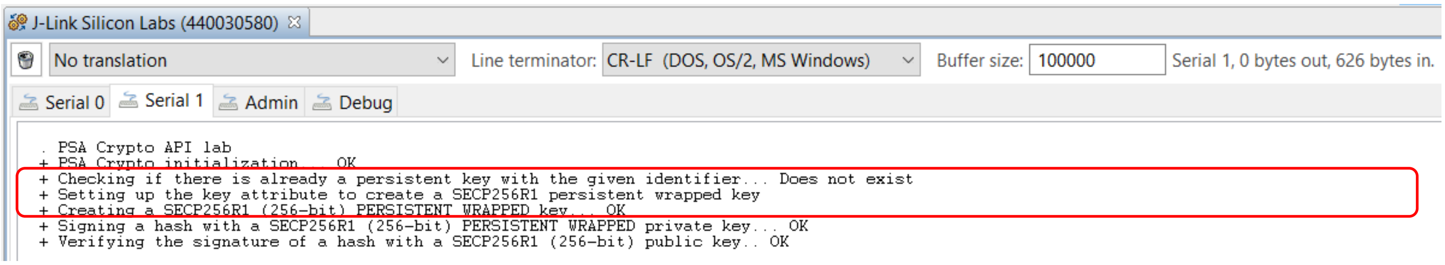


Build 'GNU ARM v10.2.1 - Debug' for project 'psa_crypto_ecdsa'



3.2 First Run

There is no key in the device, the persistent wrapped key is generated and stored in the flash through the NVM3 driver.

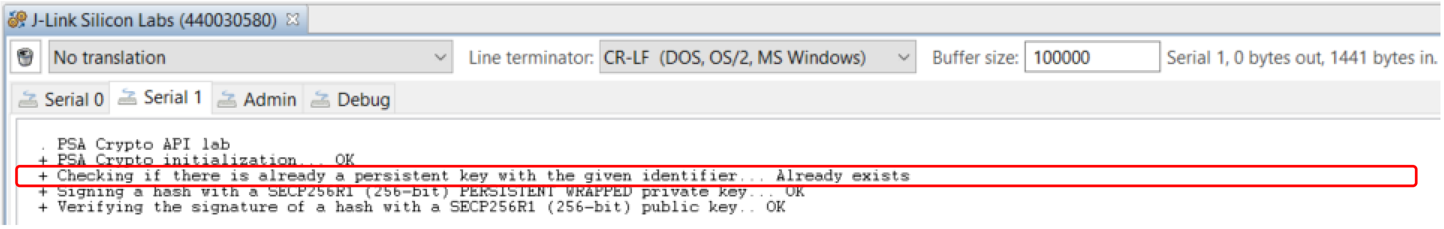

J-Link Silicon Labs (440030580)
No translation
Line terminator: CR-LF (DOS, OS/2, MS Windows)
Buffer size: 100000
Serial 1, 0 bytes out, 626 bytes in.
Serial 0
Serial 1
Admin
Debug

```
 . PSA Crypto API lab
 + PSA Crypto initialization... OK
 + Checking if there is already a persistent key with the given identifier... Does not exist
 + Setting up the key attribute to create a SECP256R1 persistent wrapped key
 + Creating a SECP256R1 (256-bit) PERSISTENT WRAPPED key... OK
 + Signing a hash with a SECP256R1 (256-bit) PERSISTENT WRAPPED private key... OK
 + Verifying the signature of a hash with a SECP256R1 (256-bit) public key.. OK
```

3.3 Consecutive Run

Move the **Power Source Select** switch to the **USB** position to power off the radio board, then back to the **AEM** position to power on the radio board to re-run the program.

The persistent wrapped key is retrieved from the flash for ECDSA operations.


J-Link Silicon Labs (440030580)
No translation
Line terminator: CR-LF (DOS, OS/2, MS Windows)
Buffer size: 100000
Serial 1, 0 bytes out, 1441 bytes in.
Serial 0
Serial 1
Admin
Debug

```
 . PSA Crypto API lab
 + PSA Crypto initialization... OK
 + Checking if there is already a persistent key with the given identifier... Already exists
 + Signing a hash with a SECP256R1 (256-bit) PERSISTENT WRAPPED private key... OK
 + Verifying the signature of a hash with a SECP256R1 (256-bit) public key.. OK
```

3.4 Destroy the Key

There are two ways to destroy the persistent wrapped key in the flash.

1. API
   - Add code below to *app_process.c* to destroy the key.

```
printf("  + Destroying the SECP256R1 (256-bit) PERSISTENT WRAPPED key.. ");
ret = psa_destroy_key(key_id);
if (ret != PSA_SUCCESS) {
  printf("Failed: %ld\n", ret);
  goto exit;
}
printf("OK\n");
```

2. Simplicity Commander
   - Erase the NVM3 area (default size is 40 kB) to destroy the key.

```
        commander device pageerase --range 0xf4000:0xfe000
        Erasing range 0x000f4000 - 0x000fe000
        DONE
```