Works With 2021

SEC-301: Hands-On with CPMS Security

Table of Contents

Overview

As IoT devices have become more secure, device configuration has become more complex. The Silicon Labs Custom Part Manufacturing Service (CPMS) makes it easy to configure and order parts with custom security specifications. This lab will walk through the steps for using Simplicity Studio v5 along with CPMS to order custom parts.

Setup

1. *CPMS:* To access CPMS, go to https://cpms.silabs.com/login and log in using your Silicon Labs credentials. If you do not have access initially, just `register` your account.

2. *Simplicity Commander:* This lab uses the Windows Command Prompt to access the Simplicity Commander CLI. All of the commands line instructions in this lab manual are executed from the *WorksWith-SEC301* directory. If you wish to work out of a different directory, full paths to relevant files must be given.

   The pre-work for this course includes instructions for how to add commander to your path. The command line instructions in this lab manual assume that Commander is in your path. If it is not, the full path to commander will need to be given: *C:\SiliconLabs\SimplicityStudio\<version>\developer\adapter_packs\commander\commander.exe*.

References

More information about Silicon Labs security features can be found in the following App Notes:

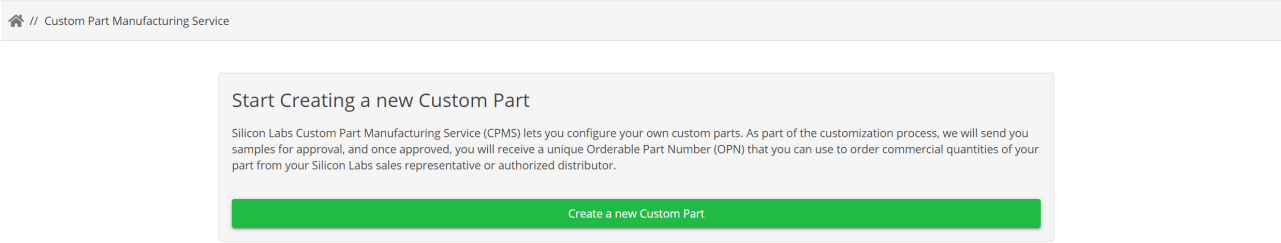
- AN1190: Secure Debug
- AN1218: Secure Boot with RTSL
- AN1222: Production Programming
- AN1247: Secure Vault Tamper
- AN1268: Secure Identity

Part 1

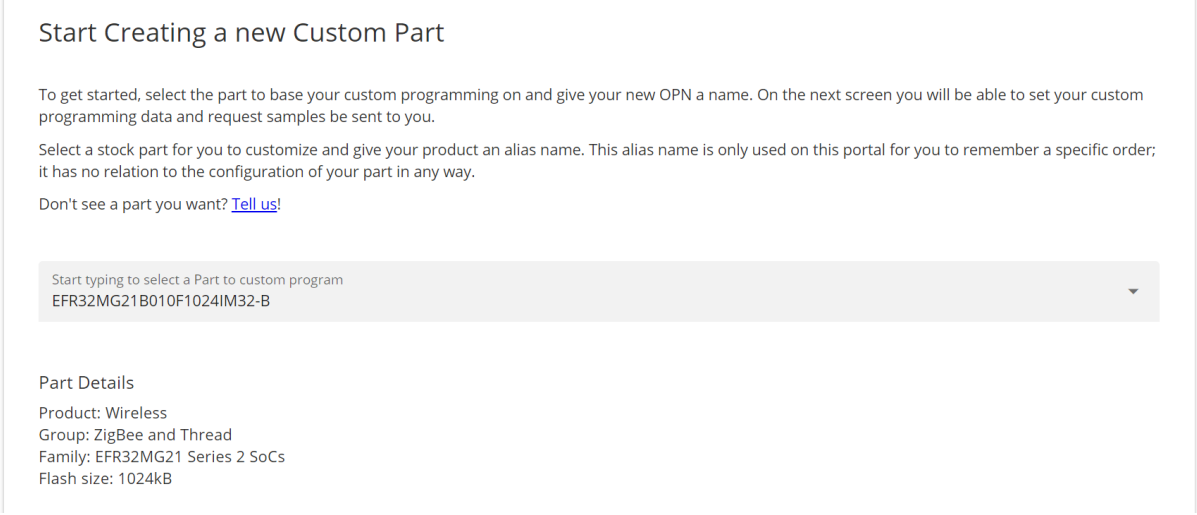
A simple part with a pre-flashed user application

CPMS

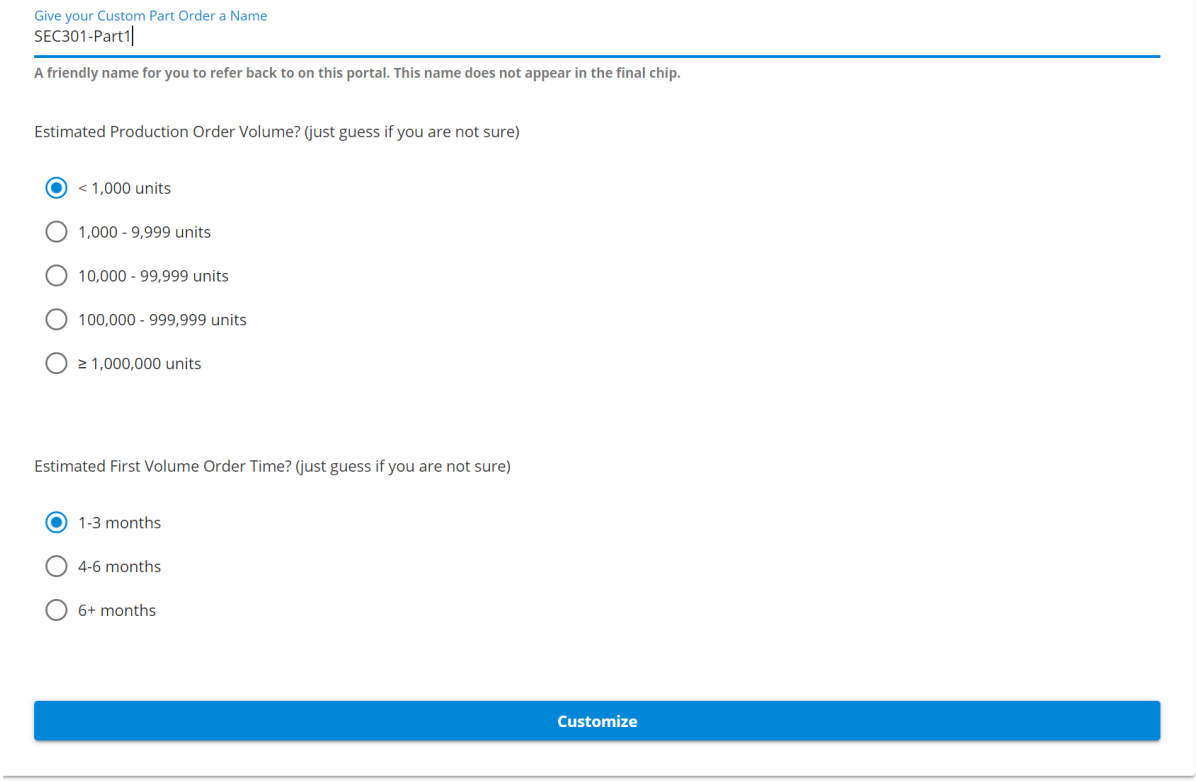
1. In a browser, open CPMS at https://cpms.silabs.com/login
2. Log in using your www.silabs.com account credentials
3. Click `Create a new Custom Part`



1. *Part:* Select "*EFR32MG21B010F1024IM32-B*"



2. *Name:* Enter "*SEC301-Part1*". This name will be used within CPMS to help differentiate between custom devices.
3. *Estimated Product Order Volume:* Select `< 1,000 units`
4. *Estimated First Volume Order Time:* Select `1-3 months`

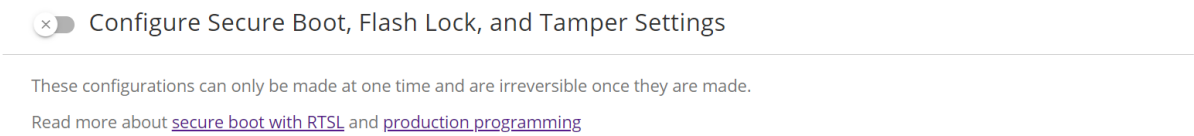


4. Click `Customize`.This takes you to the part customization page. Change the following configurations (configurations not listed can be left as the default):

    1. *Debug Lock:* This example is only going to set up the firmware, so you can set the debug lock to the minimum setting, `Unlocked`.

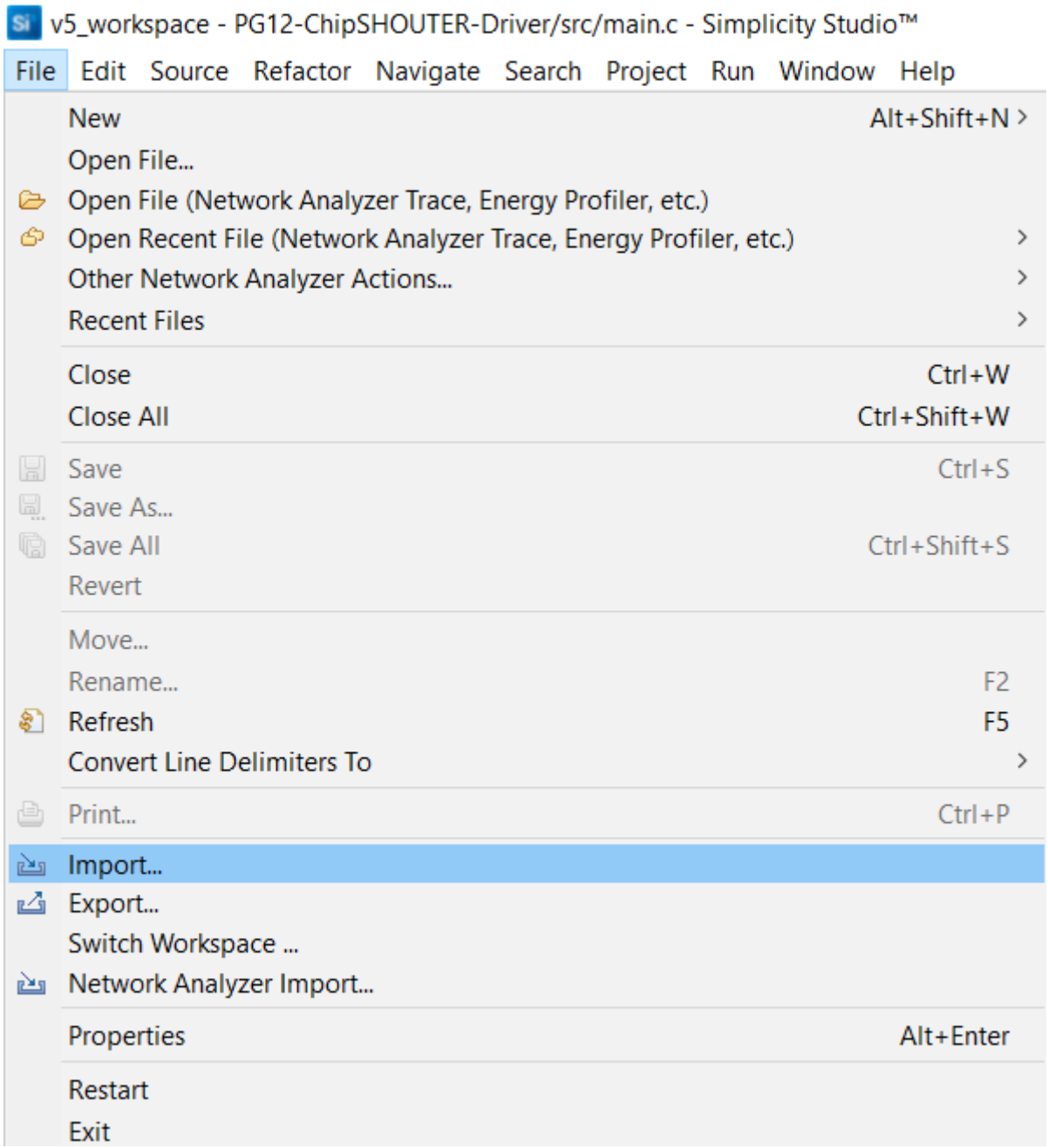
    2. *Configure Secure Boot, Flash Lock, and Tamper Settings:* Off. This will be used in later examples in this lab.

Application

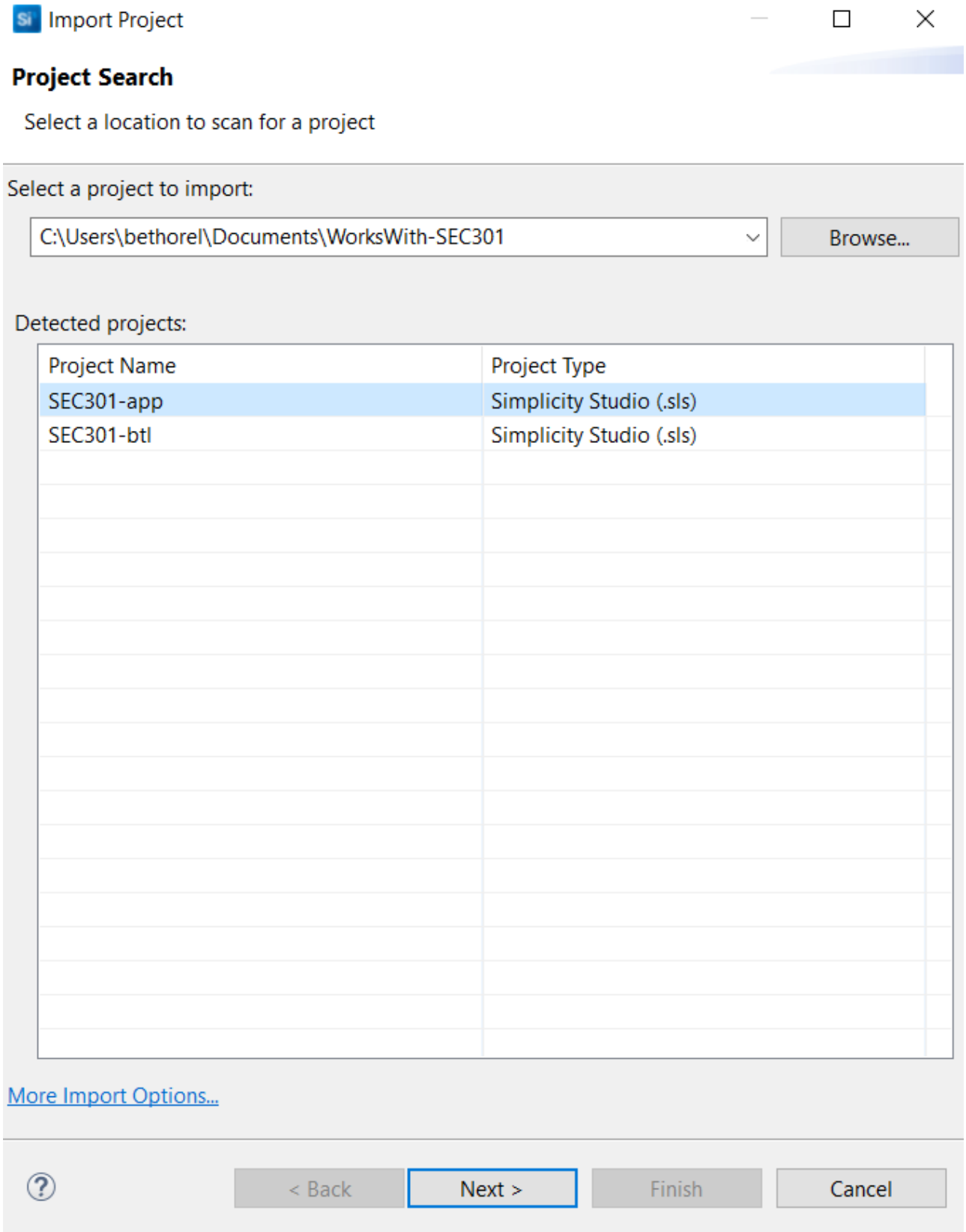
1. Open *Simplicity Studio*

2. Click `File > Import...`



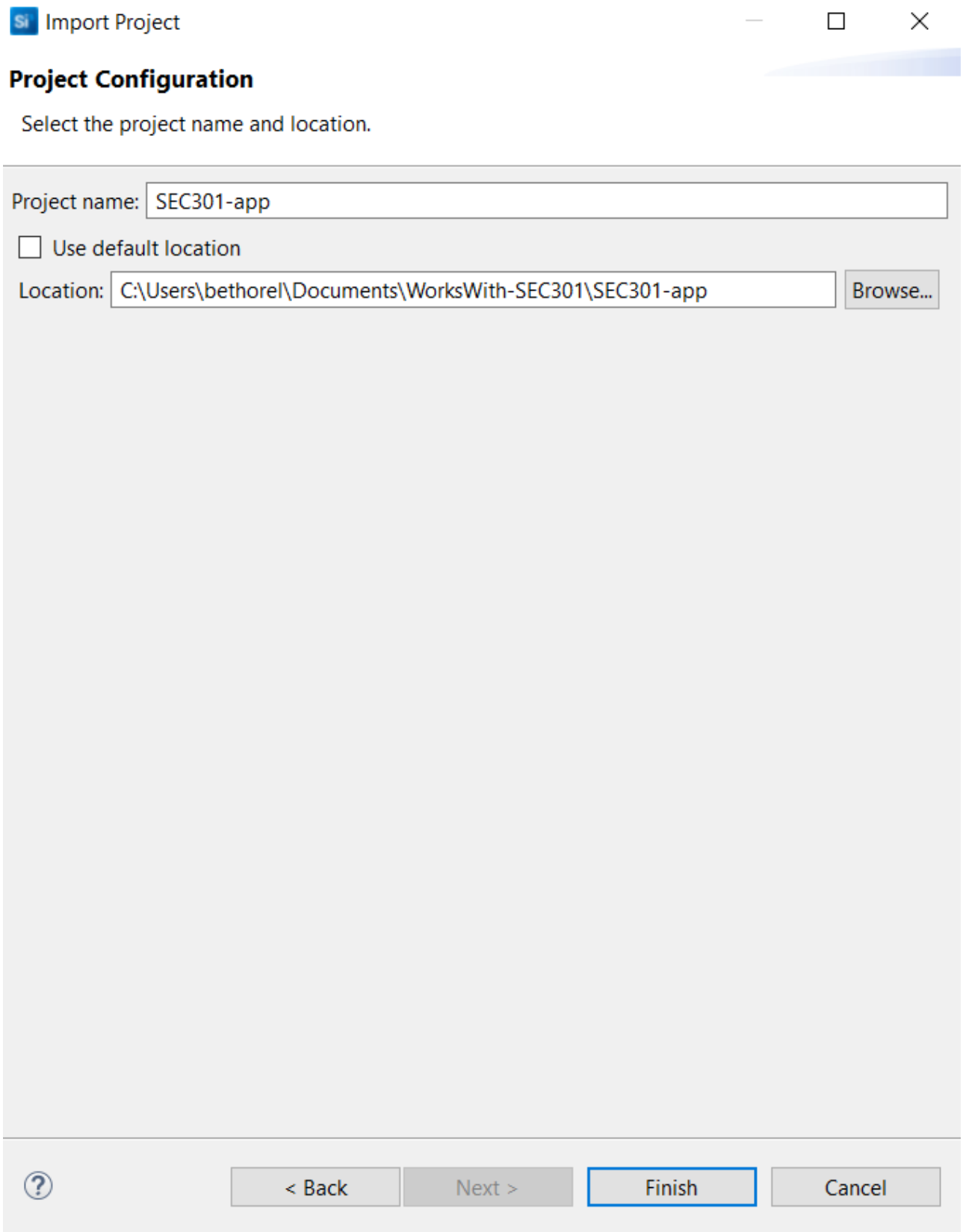
3. Click `Browse...` and navigate to the *WorksWith-SEC301* directory on your computer

4. You should see *SEC301-app* and *SEC301-btl*. For now, select *SEC301-app*.



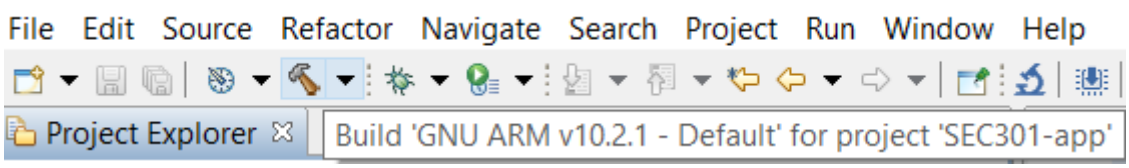
5. Click `Next >`, then `Next >`

6. Uncheck the `Use default location` option to choose a new location to import the project. Navigate to the *WorksWith-SEC301* directory and click `Select Folder`. This is done so that the project files are all in once place and close to the key files that will be used later in the lab.



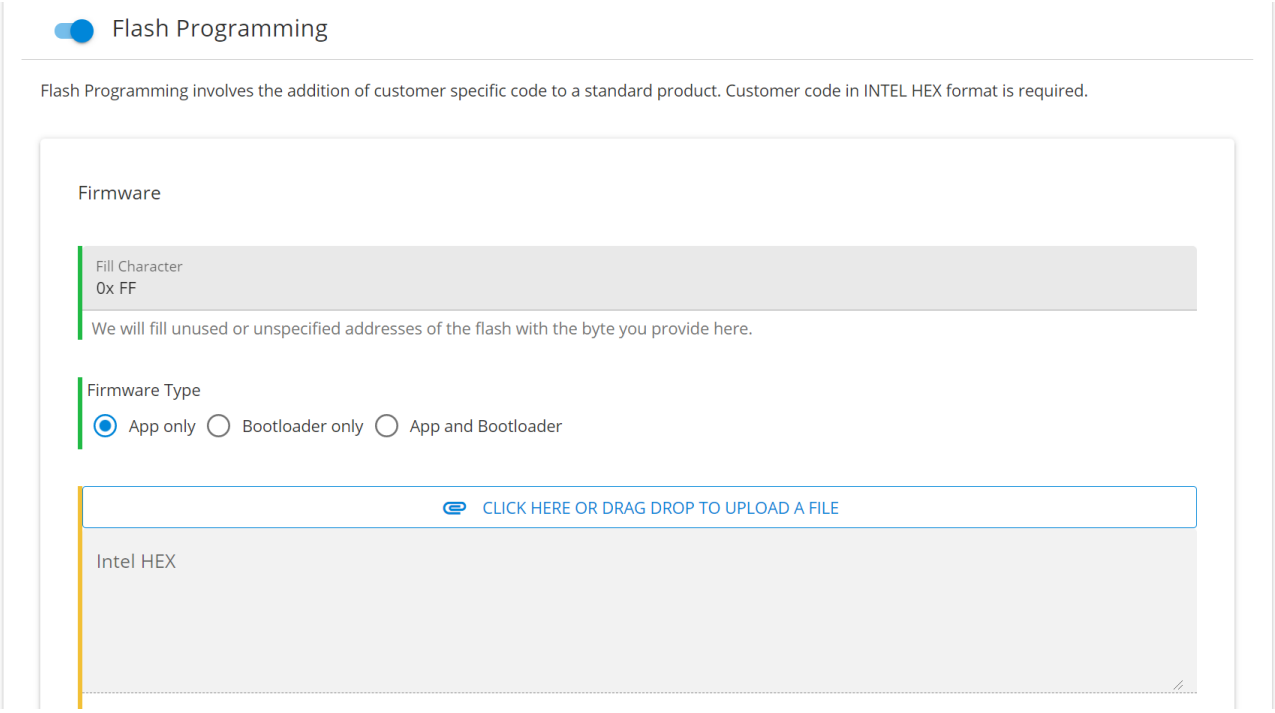
7. Click `Finish`

8. `Build` the project. This will generate binaries for the project.

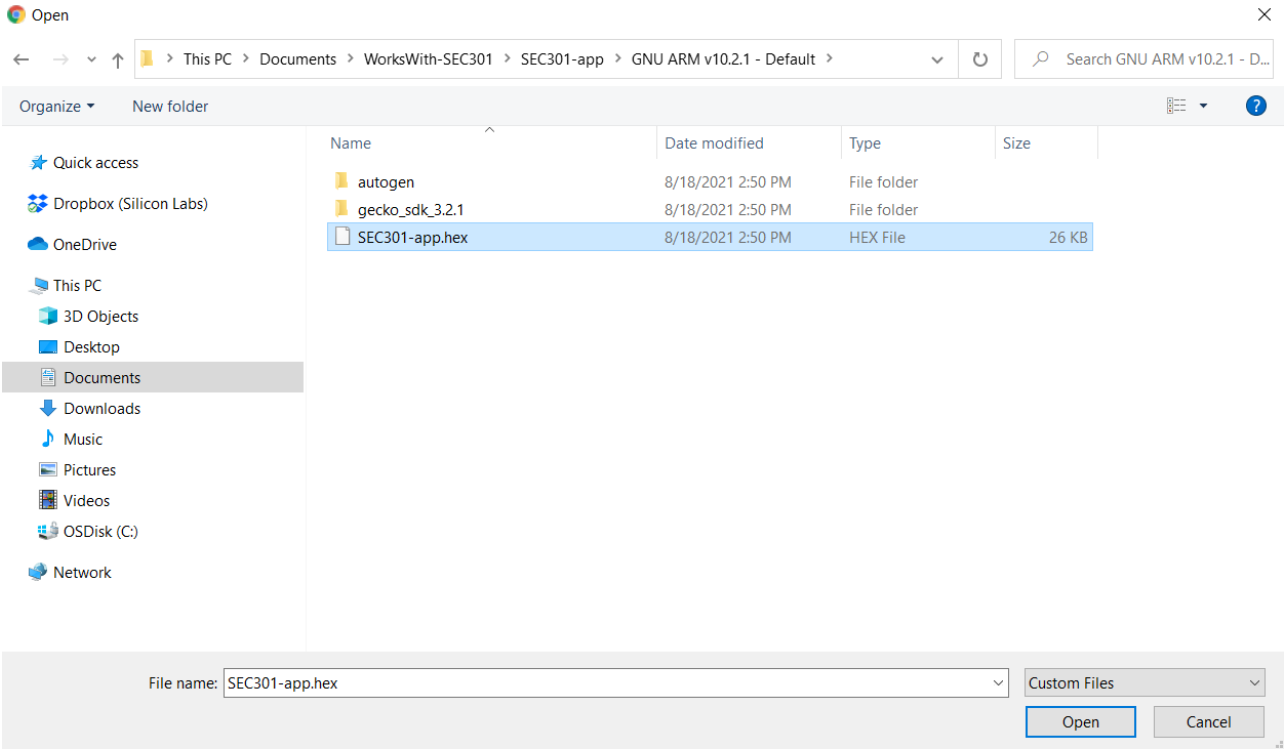


Flash Programming

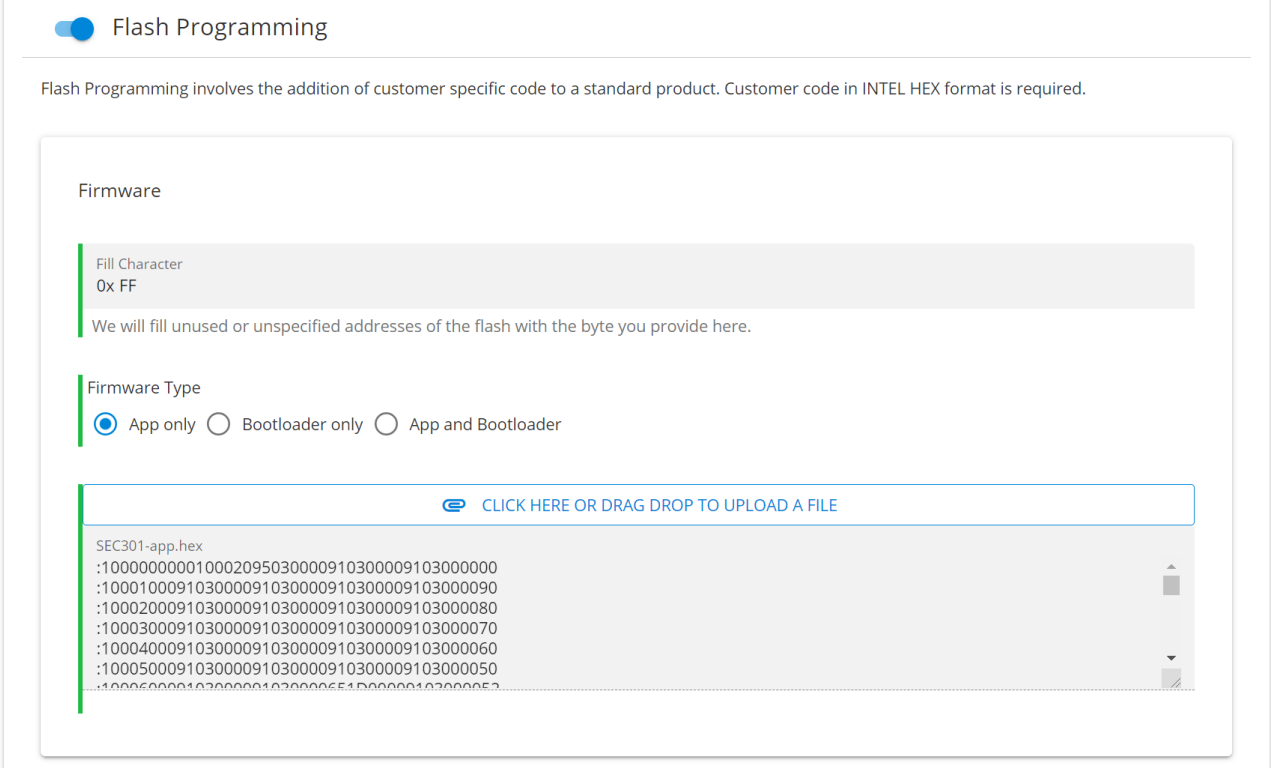
1. In CPMS, return to the *Flash Programming* section
2. *Firmware Type:* Select `App only`



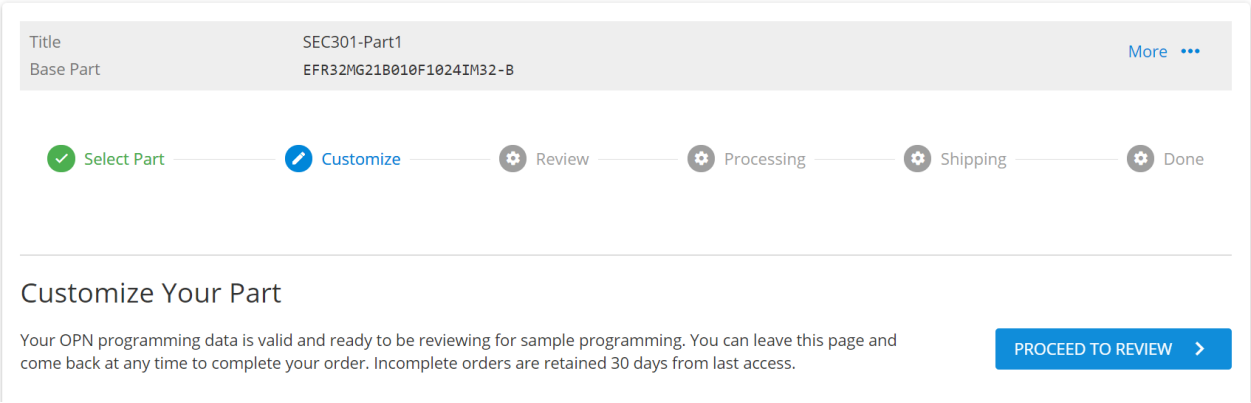
3. Click on `CLICK HERE OR DRAG DROP TO UPLOAD A FILE`
4. Navigate to the directory where you imported *SEC301-app*. If you chose to use the default directory, on Windows this will be in *C:/Users/<username>/SimplicityStudio/v5_workspace*.
5. In the *GNU ARM v10.2.1 - Default* directory, select *SEC301-app.hex* and click `Open`. CPMS only accepts Intel Hex files for firmware images.



6. You should now be able to see the binary for the application in CPMS



7. Scroll to the top of the page, and click `PROCEED TO REVIEW`



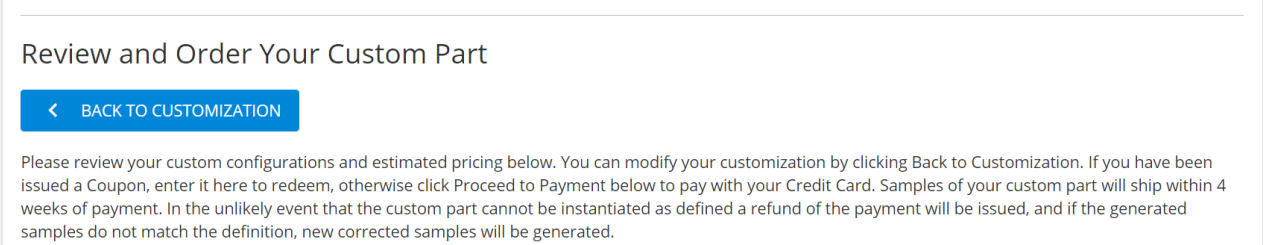
8. You can now review the pricing for your custom part and the security configurations you've entered. You will ***NOT*** actually be ordering parts for these examples, so do ***NOT*** enter any banking information for this lab.

Part 1.1

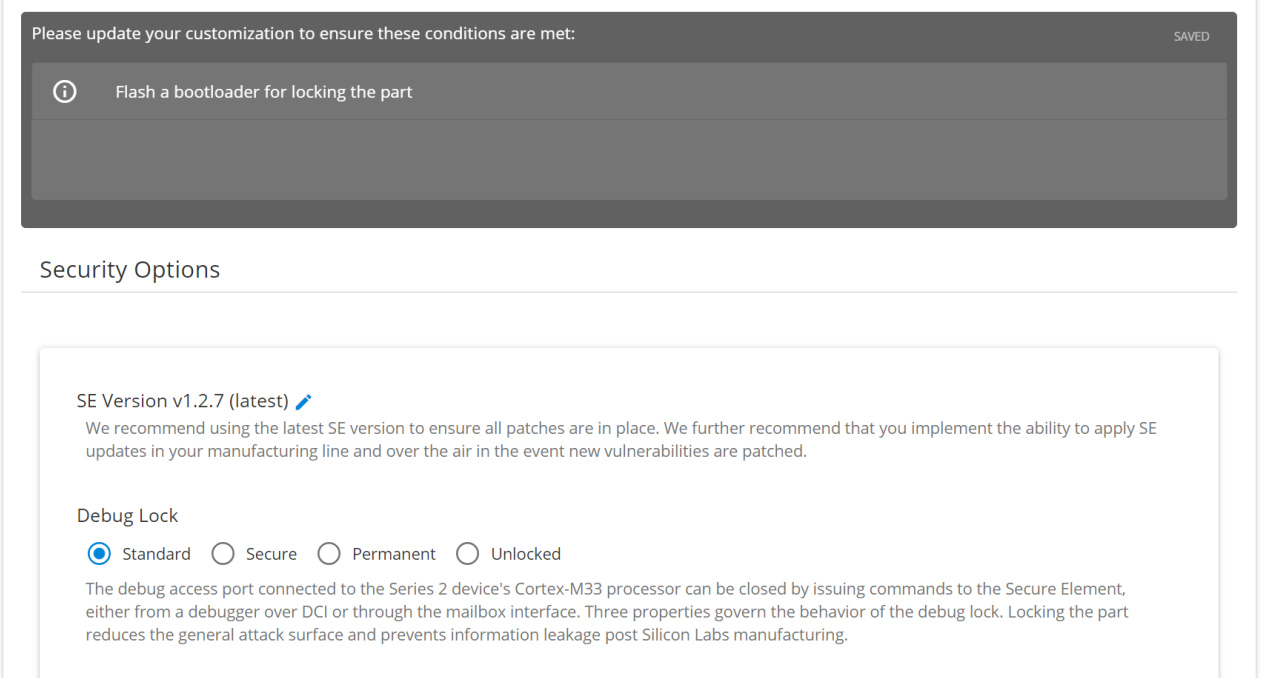
Improving security using standard lock

CPMS

1. Click on `BACK TO CUSTOMIZATION` to go back to the Customization page



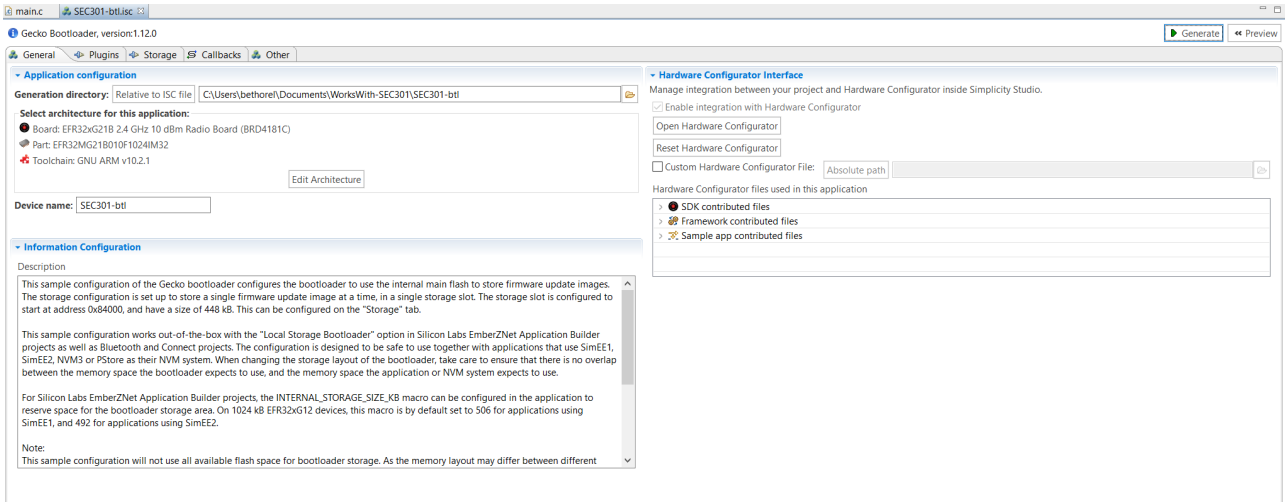
2. Select `Standard` for Debug Lock. You will see a warning appear indicating that you need to flash a bootloader in order to apply the lock.



Bootloader

1. In Simplicity Studio, import *SEC301-btl* (refer to Part 1 - Application for instructions on how to import an SLS project)

2. Open *SEC301-btl.isc*



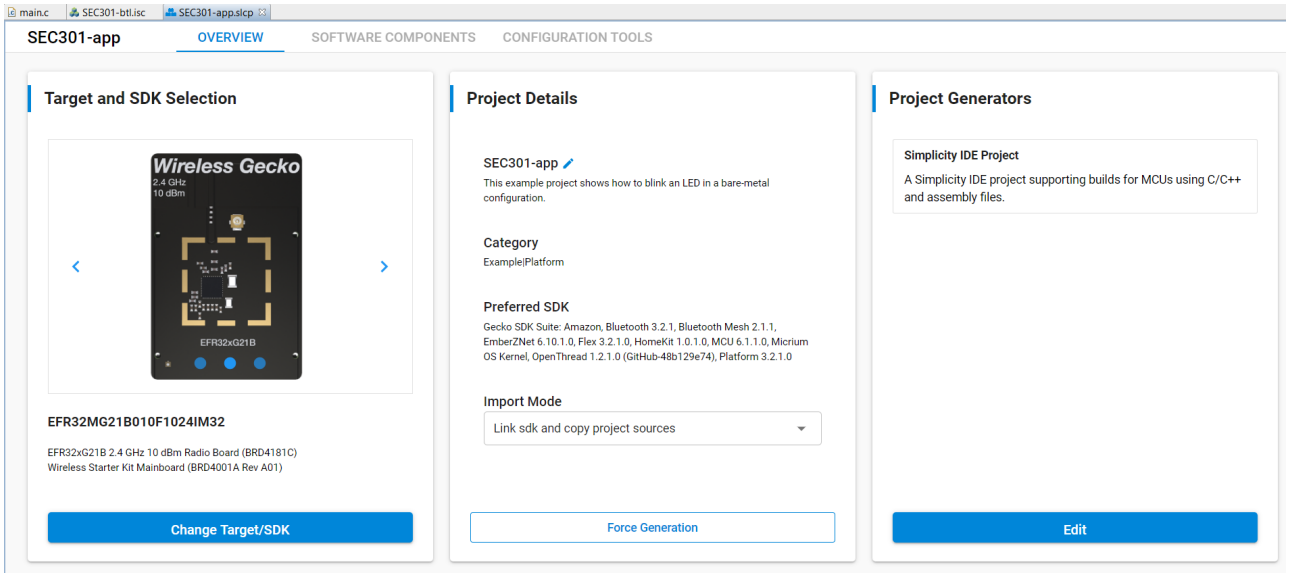
3. At the top right, click on `Generate`

4. Now that the files have been generated, `Build` the project (if the build button is greyed out, you may need to click on the project in the Project Explorer)

Application

1. You also need to tell the application to accommodate a bootloader:

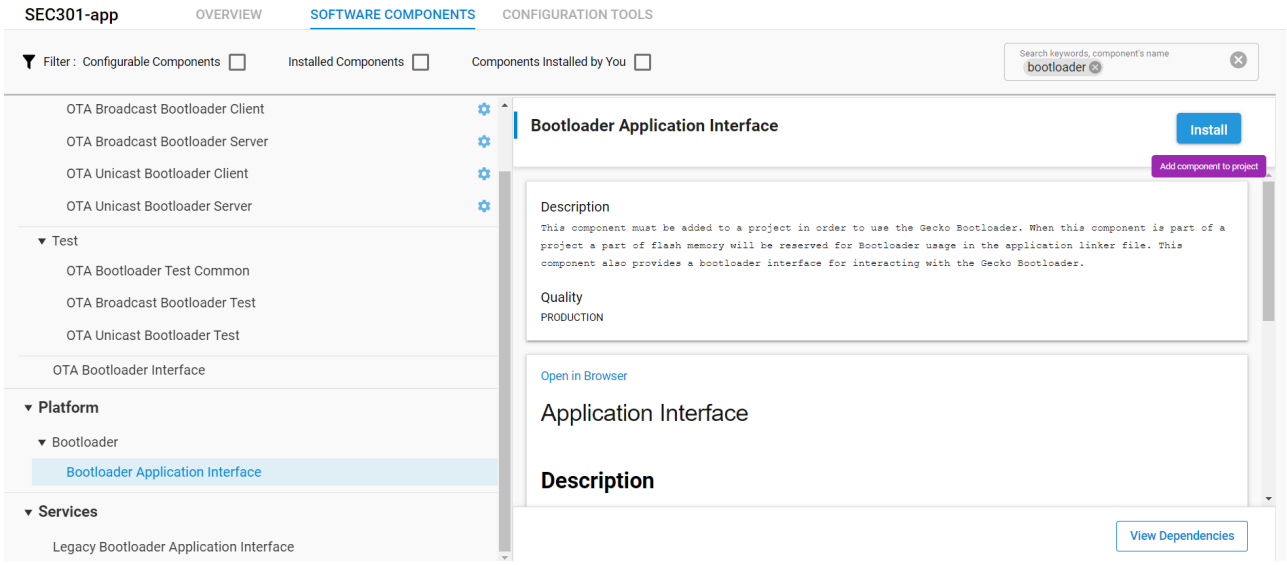
2. In the SEC301-app project, open *SEC301-app.slcp*



3. Click on the `SOFTWARE COMPONENTS` tab

4. In the Search bar, search for "*bootloader*"

5. Click on *Platform* > *Bootloader* > *Bootloader Application Interface*, and click `Install`



6. Now that the configuration is set, `Build` the project

Image Preparation

1. CPMS requires the firmware image to be in one file, so you need to merge application and bootloader hex files. This can be done with various tools. This lab uses the Simplicity Commander command line

interface.

2. Open a terminal in the *WorksWith-SEC301* directory.

3. Run:

```
commander convert "SEC301-app\GNU ARM v10.2.1 - Default\SEC301-app.hex"
"SEC301-btl\GNU ARM v10.2.1 - Default\SEC301-btl.hex" -o SEC301-merged.hex
```

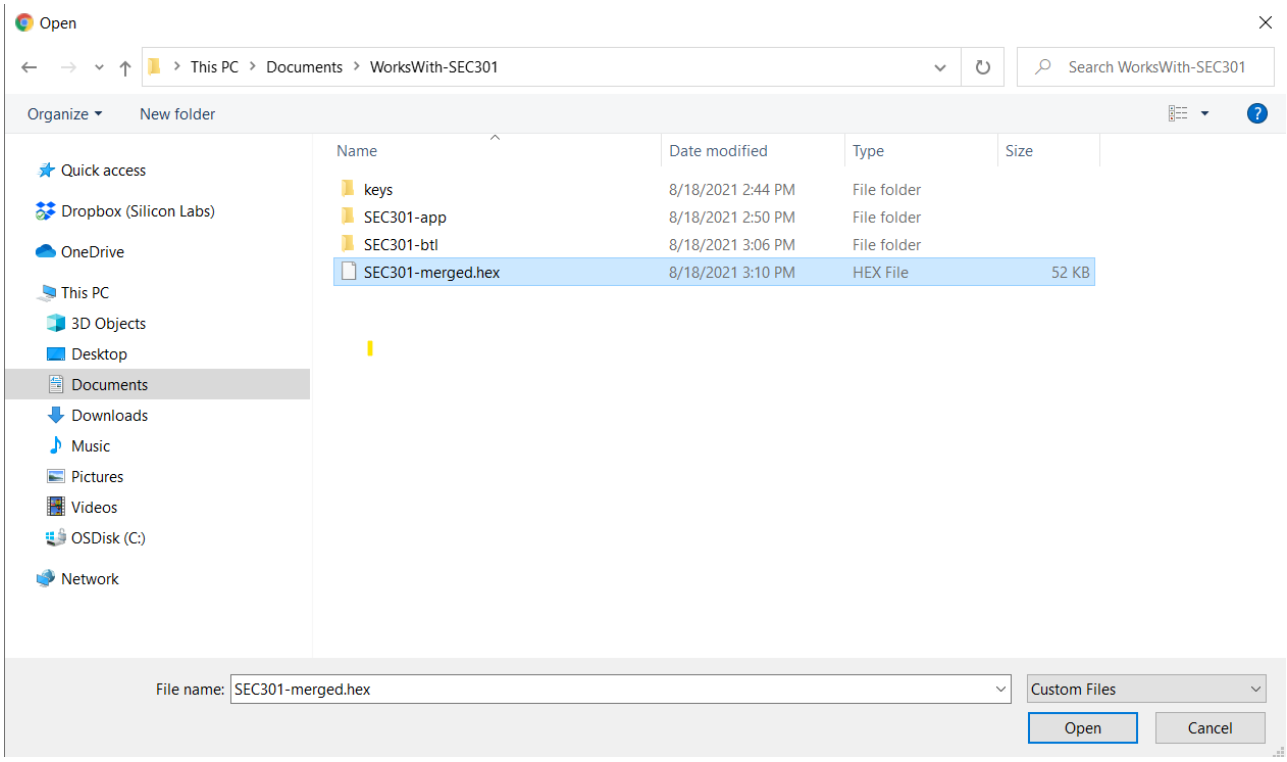
```
Parsing file SEC301-app\GNU ARM v10.2.1 - Default\SEC301-app.hex...
Parsing file SEC301-btl\GNU ARM v10.2.1 - Default\SEC301-btl.hex...
Writing to SEC301-merged.hex...
DONE
```

This will create *SEC301-merged.hex* in the *WorksWith-SEC301* directory

Flash Programming

1. In CPMS, scroll down to *Flash Programming* and for *Firmware Type* select `App and Bootloader`

2. Upload the merged hex file, *SEC301-merged.hex*



Flash Programming

Flash Programming involves the addition of customer specific code to a standard product. Customer code in INTEL HEX format is required.

Firmware

Fill Character
0x FF

We will fill unused or unspecified addresses of the flash with the byte you provide here.

Firmware Type

- ○ App only
- ○ Bootloader only
- ◉ App and Bootloader

CLICK HERE OR DRAG DROP TO UPLOAD A FILE

SEC301-merged.hex

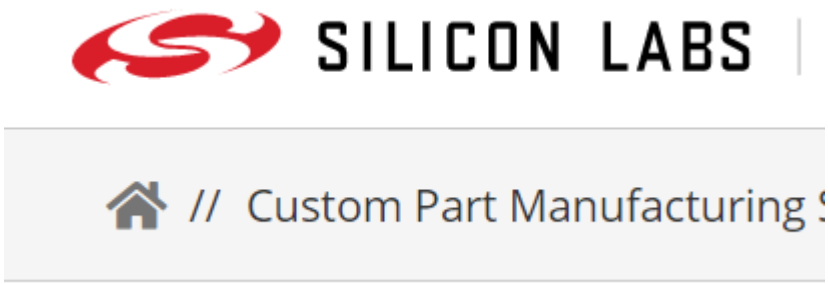
```
:1000000000800020A10000009D0000000B01000006
:100010009D0000009D0000009D0000009D0000006C
:100020009D0000009D000000682400009D0000006D
:100030009D000000A82400009D0000009D0000001D
:1000400010B5054C237833B9044B13B1044800E0D4
:1000500000BF0123237010BD080100200000000034
```

Part 2

A part secured against untrusted CMs

CPMS

1. In CPMS, click on the `Home icon` at the top of the page to go to the CPMS home page



2. Create a new custom part. Besides a name of "*SEC301-Part2*", all the other device ordering information are the same as in Part 1 - CPMS (the part is "*EFR32MG21B010F1024IM32-B*")

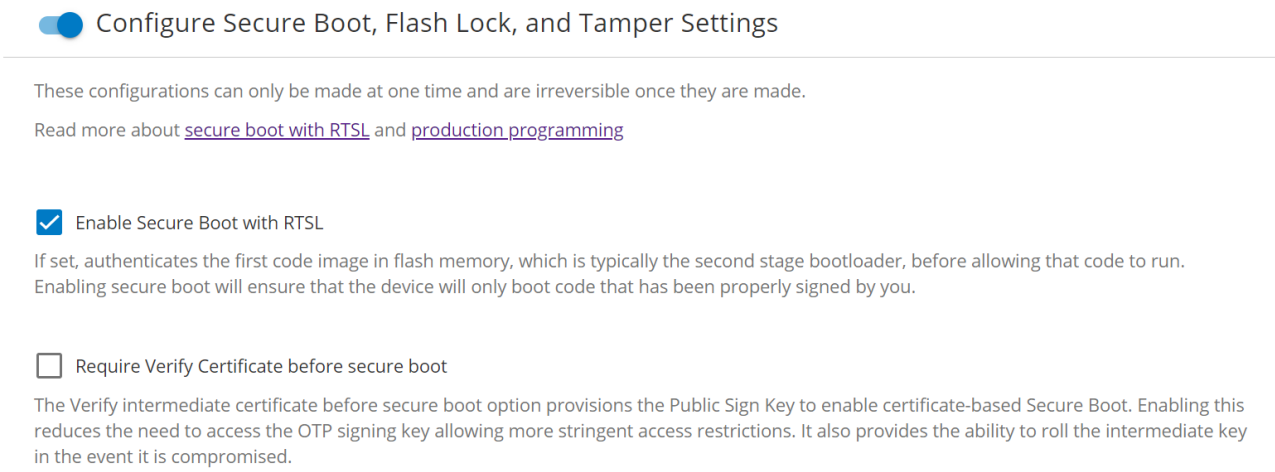
3. Customize (as before, configurations not listed can be left as the default):

    1. *Debug lock:* `Standard`

    2. *Configure Secure Boot, Flash Lock, and Tamper Settings:* Yes

        - *Enable Secure Boot:* yes

        - *Require Verify Certificate:* no

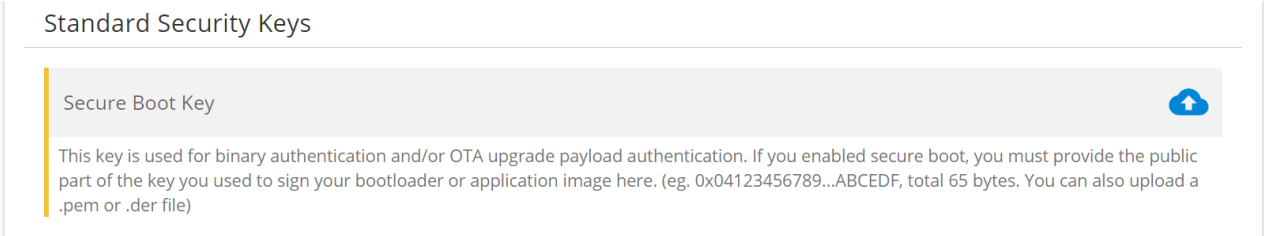


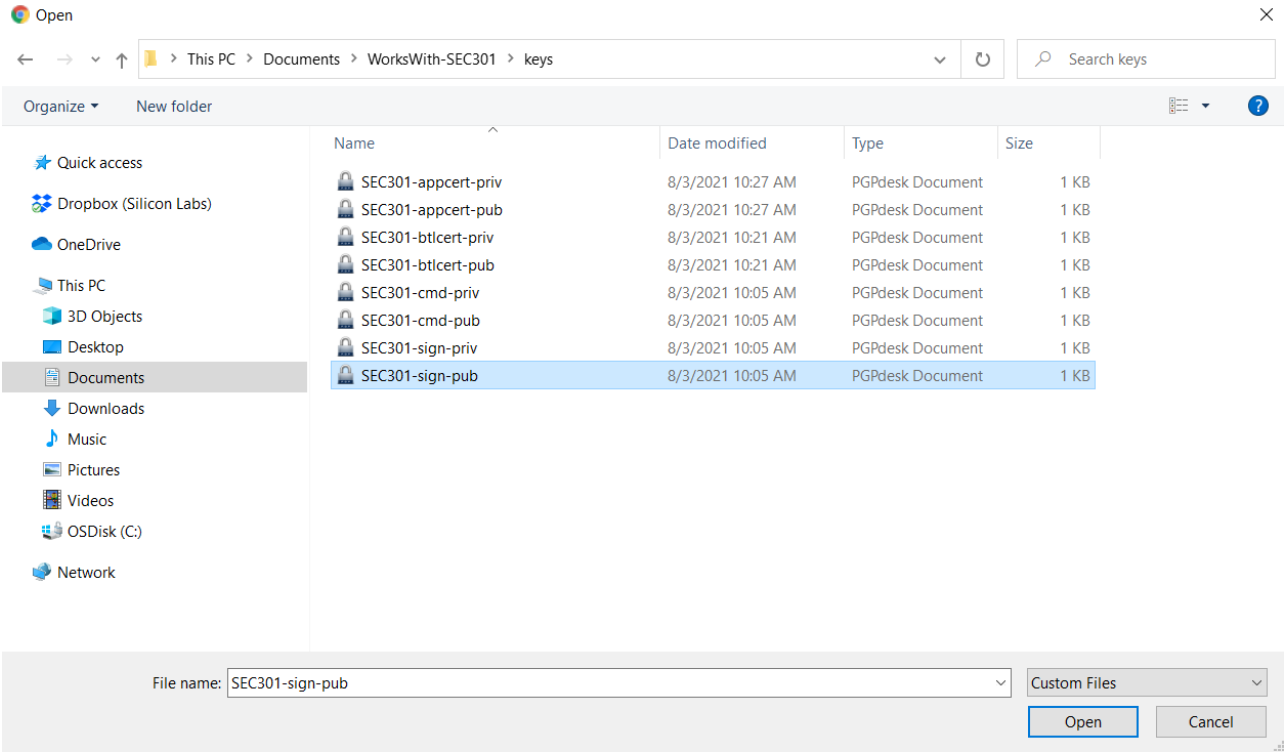
4. *Standard Security Keys:* Because Secure Boot is enabled, you need to supply a Secure Boot Key. Ideally, this key would be generated and managed by an HSM. This lab will use commander.

    All the keys for this lab have already been generated and are been provided in the *WorksWith-SEC301/keys* directory. For reference, the keys were generated with the command:

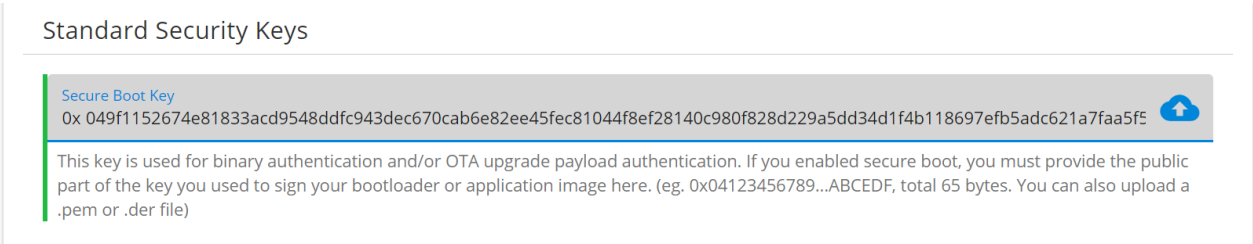
```
  commander security genkey --type ecc-p256 --privkey keys\SEC301-<type>-
priv.pem --pubkey keys\SEC301-<name>-pub.pem
```

To provision the Secure Boot Key in CPMS, click on the blue upload button in the *Secure Boot Key* field, then select the *WorksWith-SEC301/keys/SEC301-sign-pub.pem* file.





5. You should now be able to see the public boot key in CPMS



6. *Flash Programming:* Because Secure Boot is enabled, the images need to be signed.

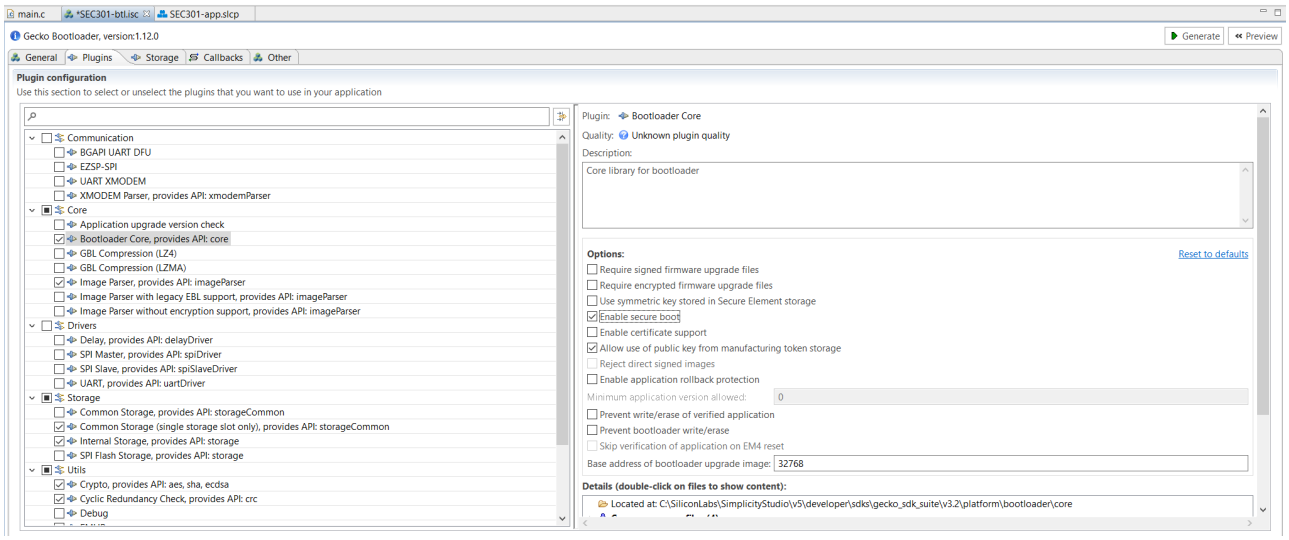
Bootloader

1. Prepare the bootloader to use a signature:

2. In Simplicity Studio, open *SEC301-btl.isc*

3. Click on the `Plugins` tab, then select `Bootloader Core, provides API: core`

4. Click `Enable Secure Boot`



5. Click `Generate`

6. `Build` the project

Application

1. Prepare the application to use a signature:

2. Copy the *application_properties.c* file (found in the *WorksWith-SEC301* directory) to the *SEC301-app* directory. This prepares space in the application for a signature.

```
copy application_properties.c SEC301-app
```

3. `Build` the project

Image Preparation

1. Prepare the hex files in commander:

2. Sign the bootloader

```
commander convert "SEC301-btl\GNU ARM v10.2.1 - Default\SEC301-btl.hex" --secureboot --keyfile keys\SEC301-sign-priv.pem --outfile SEC301-btl-signed.hex
```

This will create the *SEC301-btl-signed.hex* signed image file in your *WorksWith-SEC301* directory.

```
Parsing file SEC301-btl\GNU ARM v10.2.1 - Default\SEC301-btl.hex...
Found Application Properties at 0x000024a8
Writing Application Properties signature pointer to point to 0x000025e0
Setting signature type in Application Properties: 0x00000001
Image SHA256: ca36debc860cdb720aabe9fdd37dc730172fe34571aedc452b52f9ef5a824264
R = 746AF8EB33BF0432286B2D60E23C827B5CFBF1ED5BB078C3C19F30E36988EA04
S = CE64F9A71A4C69B1C759FEFF6D2F6CA6F1A0D9CC151F7B447D31B8EE0E94770D
Writing to SEC301-btl-signed.hex...
DONE
```

3. Sign the application in commander:

```
commander convert "SEC301-app\GNU ARM v10.2.1 - Default\SEC301-app.hex" --secureboot --keyfile keys\SEC301-sign-priv.pem --outfile SEC301-app-signed.hex
```

This will create the *SEC301-app-signed.hex* signed image file in your *WorksWith-SEC301* directory.

```
Parsing file SEC301-app\GNU ARM v10.2.1 - Default\SEC301-app.hex...
Found Application Properties at 0x00006198
Writing Application Properties signature pointer to point to 0x0000643c
Setting signature type in Application Properties: 0x00000001
Image SHA256: 731ec2f815e56e7d12e4fb1c28275d50134906157aaecdd3dcc9b37794c25aae
R = 43E31A583BC63439C464B60314EE311320A3862B3FCA94368C04F53522C2B7AA
S = 6027746C1CFB7FD56F6DC489E8EC455738BB4BB45CF19C7075A679AEDE34A8D7
Writing to SEC301-app-signed.hex...
DONE
```

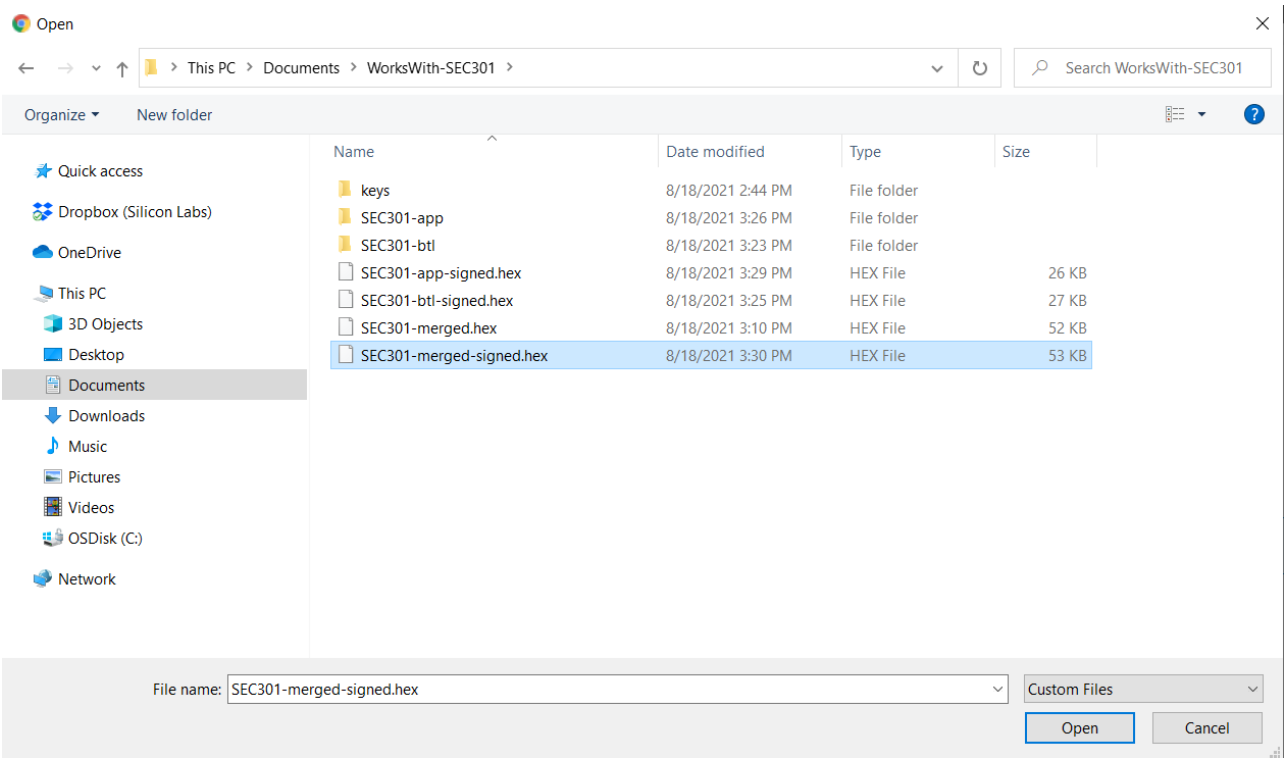
4. Merge the signed hex files:

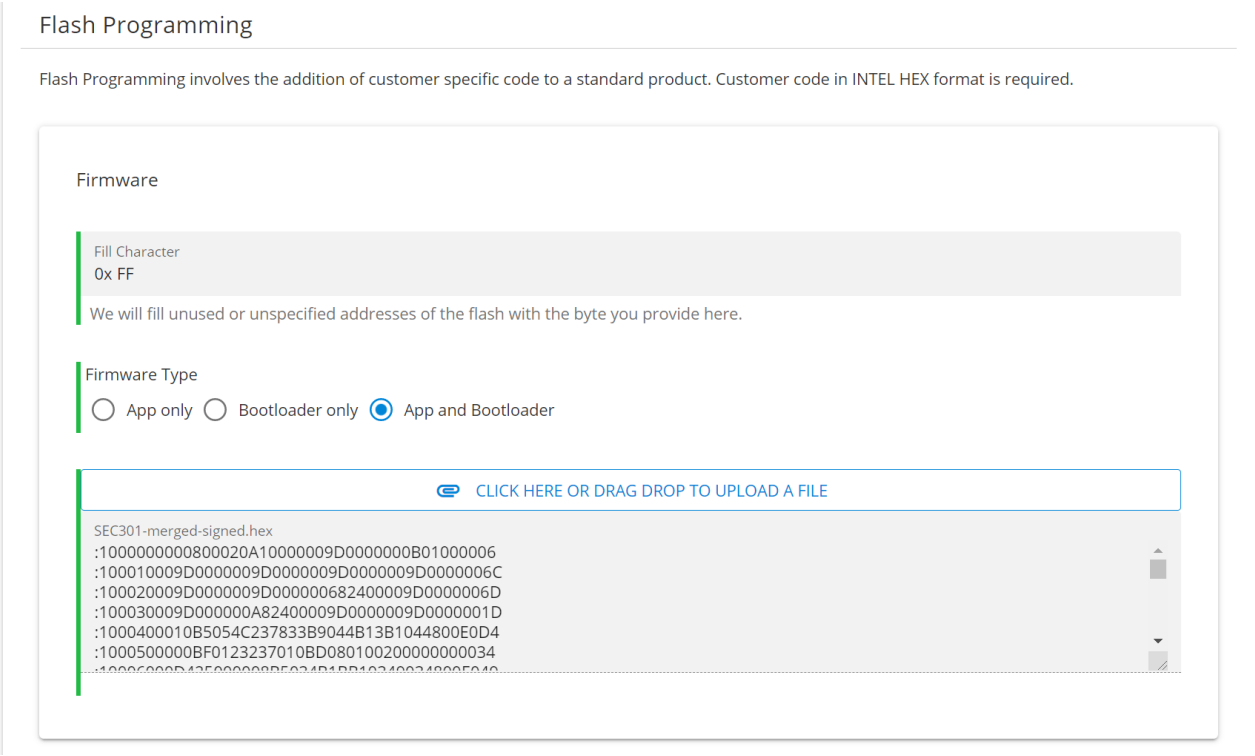
```
commander convert SEC301-app-signed.hex SEC301-btl-signed.hex -o SEC301-merged-signed.hex
```

```
Parsing file SEC301-app-signed.hex...
Parsing file SEC301-btl-signed.hex...
Writing to SEC301-merged-signed.hex...
DONE
```

Flash Programming

1. In CPMS, for *Firmware Type* select `App and Bootloader` and upload the merged signed hex.



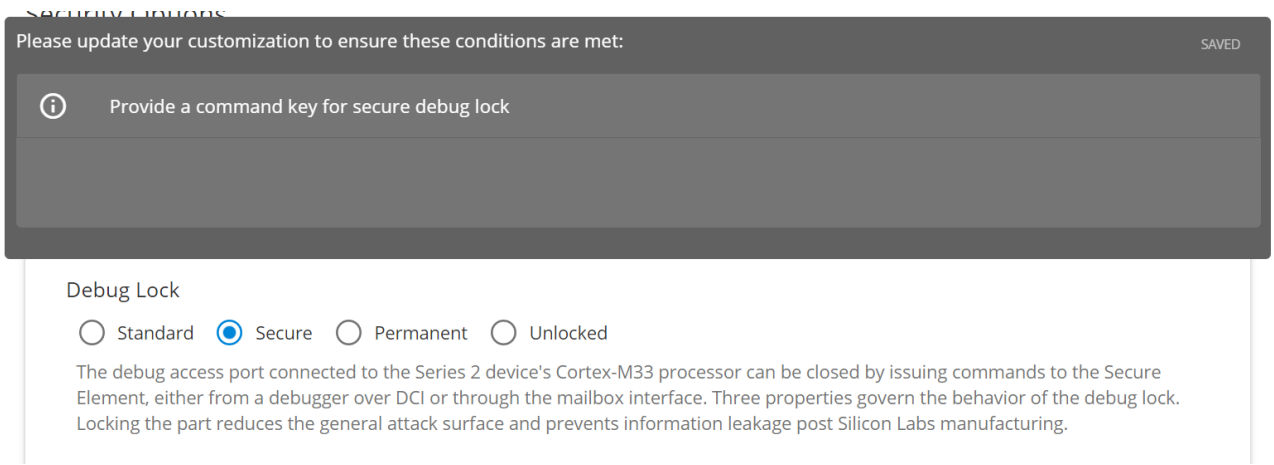


Part 2.1

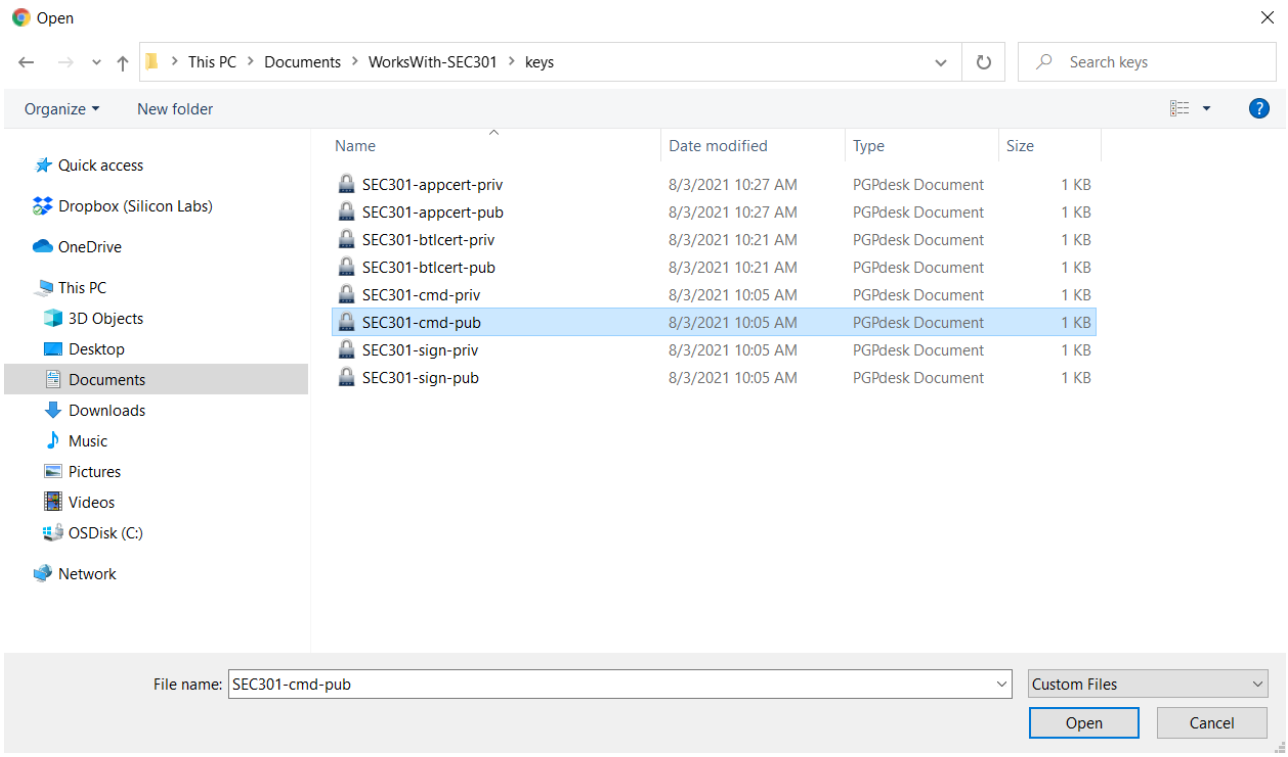
Improving security using Secure lock

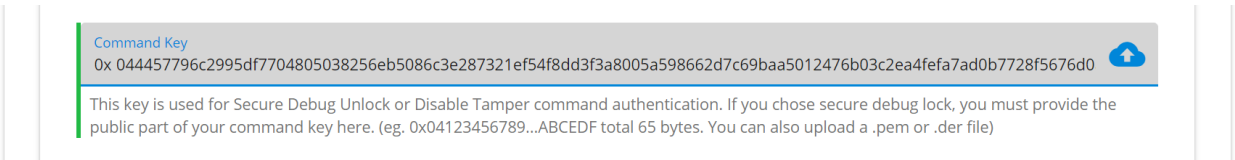
CPMS

1. In CPMS, click on `BACK TO CUSTOMIZATION` to go back to the Customization page
2. Scroll up to *Debug Lock* and select `Secure`. You will see a warning appear indicating that you need to provide a command key in order to apply the lock.



3. Providing a command key works just like the secure boot key. Scroll to the *Standard MCU Keys* and upload the *keys/SEC301-cmd-pub.pem* file into the *Command Key* slot.





Now you have a part that only boots signed images, doesn't allow debug access except to those with the private key, and comes shipped with your own firmware.

Part 3

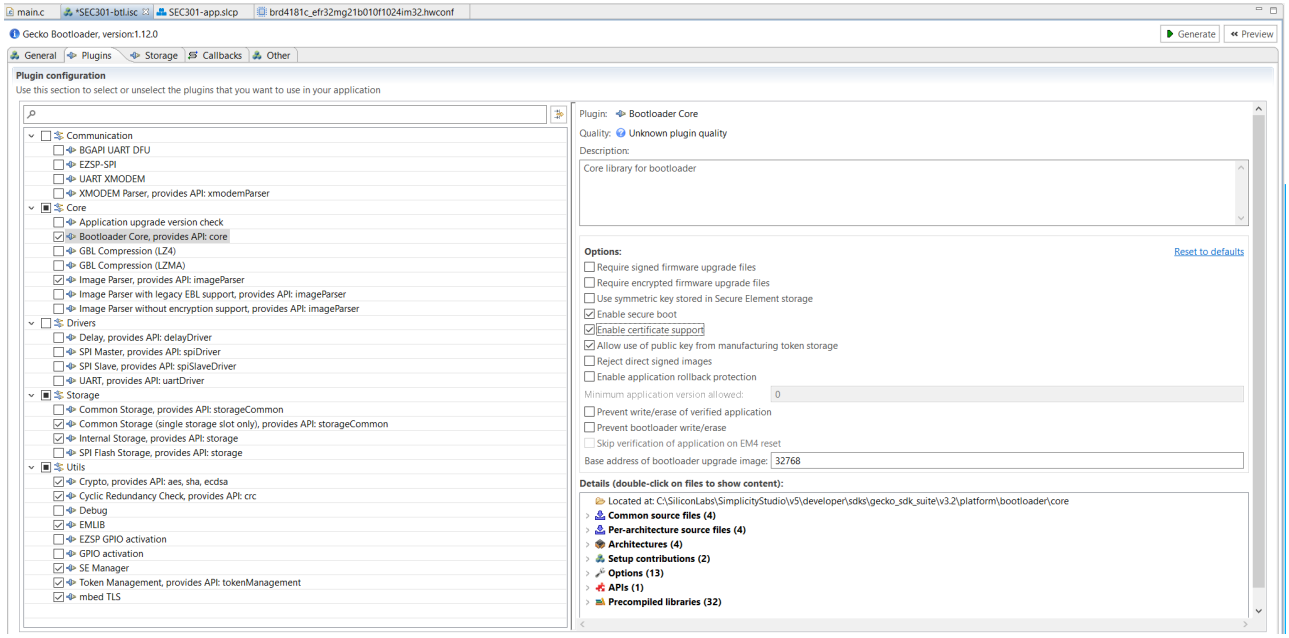
A secure part using certificate chains

CPMS

1. In CPMS, click on the `Home icon` at the top of the screen to go to the CPMS home page
2. Create a new custom part. Besides a name of "*SEC301-Part3*", all the other device ordering information are the same as in Part 1 - CPMS (the part is "*EFR32MG21B010F1024IM32-B*")
3. Customize:
    1. *Debug lock:* `Secure`
    2. *Configure Secure Boot, Flash Lock, and Tamper Settings:* Yes
        - *Secure Boot:* Yes
        - *Require Verify Certificate:* Yes. Enabling this feature will prevent the device from booting any images that are not signed with a valid certificate. To use this feature, you need to update your signature configurations to use certificates.
        - *Enable Anti Rollback:* Yes. Enabling this feature will prevent the device from "upgrading" to firmware with a firmware version number that is lower or equal to the current firmware version. This prevents attackers from exploiting patched vulnerabilities and allows for the revocation of compromised certificates.
        - *Flash Page Locking:* `Full`. This setting will lock all pages including the second stage bootloader and application. If the image and signature does not end on a page boundary and your system is tight on memory, you can select `Narrow` to leave the last (partially unused) page unlocked for future use.
    3. *Standard Security Keys:*
        1. Import the secure boot key (*keys/SEC301-sign-pub*) as in Part 2 - CPMS
        2. Import the command key (*keys/SEC301-cmd-pub*) as in Part 2.1 - CPMS
        3. OTA Decryption Key: If you plan to encrypt future firmware upgrades, you should provision an OTA key and set appropriate settings in the btl configuration (highly recommended, but not covered by this lab)

Bootloader

1. Prepare the Bootloader to use certificates:
2. In Simplicity Studio, open *SEC301-btl.isc*
3. Click on the `Plugins` tab, then select `Bootloader Core, provides API: core`
4. Click `Enable certificate support`



5. Click `Generate`

6. `Build` the project

Application

1. Prepare the App to use certificates:

2. In Simplicity Studio, open *application_properties.c*

3. Update *application_properties.c* by adding the following lines after

```
#define APP_PROPERTIES_VERSION (0UL)
```

and before

```
// Application properties for secure boot

const ApplicationProperties_t sl_app_properties = {
```

```
// Application properties for secure boot
const ApplicationCertificate_t sl_app_certificate = {
.structVersion = APPLICATION_CERTIFICATE_VERSION,
.flags = { 0U },
.key = { 0U },
.version = 0,
.signature = { 0U },
};
```

4. Update the ApplicationProperties_t struct (also in *application_properties.c*) to point to the new certificate:

```
.cert = (ApplicationCertificate_t *)&sl_app_certificate,
```

main.c | SEC301-btl.isc | SEC301-app.slcp | brd4181c_efr32mg21b010f1024im32.hwconf | application_properties.c

```
1
2  #include <stddef.h>
3  #include "application_properties.h"
4
5  // Application version number (uint32_t) for anti-rollback
6  #define APP_PROPERTIES_VERSION (0UL)
7  // Application properties for secure boot
8  const ApplicationCertificate_t sl_app_certificate = {
9  .structVersion = APPLICATION_CERTIFICATE_VERSION,
10 .flags = { 0U },
11 .key = { 0U },
12 .version = 0,
13 .signature = { 0U },
14 };
15 // Application properties for secure boot
16 const ApplicationProperties_t sl_app_properties = {
17 .magic = APPLICATION_PROPERTIES_MAGIC,
18 .structVersion = APPLICATION_PROPERTIES_VERSION,
19 .signatureType = APPLICATION_SIGNATURE_NONE,
20 .signatureLocation = 0,
21 .app = {
22 .type = APPLICATION_TYPE_MCU,
23 .version = APP_PROPERTIES_VERSION,
24 .capabilities = 0UL,
25 .productId = { 0U },
26 },
27 .cert = (ApplicationCertificate_t *)&sl_app_certificate,
28 .longTokenSectionAddress = NULL,
29 };
30
```

5. Build the project

Key Note

1. To use certificates, more keys are needed. This lab will use 3 key pairs:

2. *SEC301-sign*: The public key is provisioned on the device and is used to verify bootloader certificates. The private key is held safe in an HSM (although this lab uses commander) and is used to sign bootloader certificates.

3. *SEC301-btlcert*: This public key is inserted into the bootloader certificate and is used to verify application certificates. The private key is used to sign bootloader images and application certificates.

4. *SEC301-appcert*: This public key is inserted into application certificates. The private key is used to sign application images.

   These 2 new key pairs (*SEC301-btlcert* and *SEC301-appcert*) are provided in the *WorksWith_SEC301* directory, and were generated with the commander instruction indicated in Part 2 - CPMS.

Image Preparation

1. Prepare the hex files in commander:

2. To generate the bootloader signature, run:

```
  commander util gencert --cert-type secureboot --cert-version 0 --cert-
pubkey keys\SEC301-btlcert-pub.pem --sign keys/SEC301-sign-priv.pem --
outfile SEC301-btlcert.bin
```

3. To sign the bootloader, run:

```
  commander convert "SEC301-btl\GNU ARM v10.2.1 - Default\SEC301-btl.hex" --
secureboot --certificate SEC301-btlcert.bin --keyfile keys\SEC301-btlcert-
priv.pem --outfile SEC301-btl-certsigned.hex
```

4. You can verify that this signed image is valid using:

```
  commander util verifysign SEC301-btl-certsigned.hex --verify keys\SEC301-
sign-pub.pem
```

   The output should look like:

```
Parsing file SEC301-btl-certsigned.hex...
Found application properties at 0x00002704
Found certificate at 0x0000267c
Successfully verified certificate signature with verification key.
Using certificate key to verify application signature.
Successfully verified application signature.
DONE
```

5. To generate the signature for the application, run:

```
commander util gencert --cert-type secureboot --cert-version 0 --cert-pubkey keys\SEC301-appcert-pub.pem --sign keys\SEC301-btlcert-priv.pem --outfile SEC301-appcert.bin
```

6. To sign the application, run:

```
commander convert "SEC301-app\GNU ARM v10.2.1 - Default\SEC301-app.hex" --secureboot --certificate SEC301-appcert.bin --keyfile keys\SEC301-appcert-priv.pem --outfile SEC301-app-certsigned.hex
```

7. You can verify that this signed image is valid using:

```
commander util verifysign SEC301-app-certsigned.hex --verify keys\SEC301-btlcert-pub.pem
```

The output should look like:

```
Parsing file SEC301-app-certsigned.hex...
Found application properties at 0x00006220
Found certificate at 0x00006198
Successfully verified certificate signature with verification key.
Using certificate key to verify application signature.
Successfully verified application signature.
DONE
```

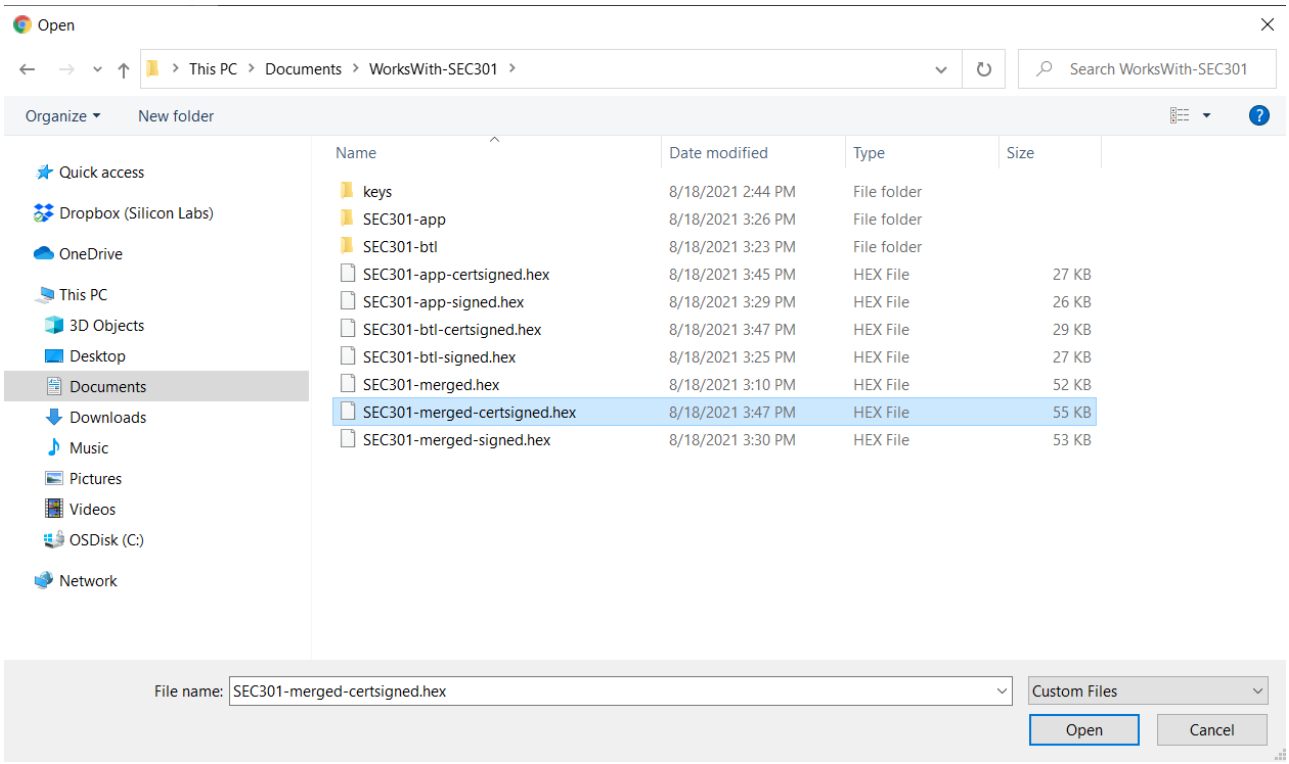
8. Lastly, combine the signed and certified images:

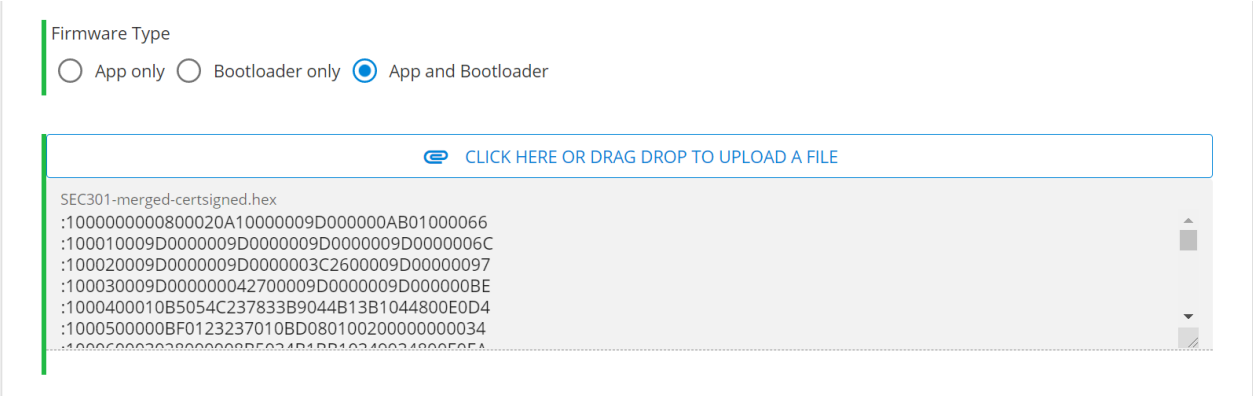
```
commander convert SEC301-app-certsigned.hex SEC301-btl-certsigned.hex -o SEC301-merged-certsigned.hex
```

Flash Programming

1. In CPMS, scroll to *Flash Programming* and select `App and Bootloader` for *Firmware Type*

2. Upload *SEC301-merged-certsigned.hex*


Open
This PC > Documents > WorksWith-SEC301 >
Search WorksWith-SEC301
Organize
New folder
Quick access
Dropbox (Silicon Labs)
OneDrive
This PC
3D Objects
Desktop
Documents
Downloads
Music
Pictures
Videos
OSDisk (C:)
Network
Name
Date modified
Type
Size
keys 8/18/2021 2:44 PM File folder
SEC301-app 8/18/2021 3:26 PM File folder
SEC301-btl 8/18/2021 3:23 PM File folder
SEC301-app-certsigned.hex 8/18/2021 3:45 PM HEX File 27 KB
SEC301-app-signed.hex 8/18/2021 3:29 PM HEX File 26 KB
SEC301-btl-certsigned.hex 8/18/2021 3:47 PM HEX File 29 KB
SEC301-btl-signed.hex 8/18/2021 3:25 PM HEX File 27 KB
SEC301-merged.hex 8/18/2021 3:10 PM HEX File 52 KB
SEC301-merged-certsigned.hex 8/18/2021 3:47 PM HEX File 55 KB
SEC301-merged-signed.hex 8/18/2021 3:30 PM HEX File 53 KB
File name: SEC301-merged-certsigned.hex
Custom Files
Open
Cancel


Firmware Type
App only
Bootloader only
App and Bootloader
CLICK HERE OR DRAG DROP TO UPLOAD A FILE
SEC301-merged-certsigned.hex
:1000000000800020A10000009D000000AB01000066
:100010009D0000009D0000009D0000009D0000006C
:100020009D0000009D0000003C2600009D00000097
:100030009D000000042700009D0000009D000000BE
:1000400010B5054C237833B9044B13B1044800E0D4
:1000500000BF0123237010BD080100200000000034