TrustZone and ATECC608 on SAML11

Quang Hai Nguyen
Revision 2.0, 02.07.2019

Hands-on

Five Years Out

©2017 by ARROW

Revision History

Revision, Date	Editor	Subject (major changes)
Revision 0.1, 03.06.2019	Quang Hai Nguyen	Preliminary
Revision 1.0, 06.06.2019	Quang Hai Nguyen	Release
Revision 2.0, 02.07.2019	Quang Hai Nguyen	Change the hands-on flow

Table of Contents

List of Figures

List of Icon Identifiers

Table 1: Icon Identifiers List

(i)	Extra information about a topic
✎	Task needs to be done
⚠	Important information or a warning
✓	Result expected to see

Overview

Introduction

This hands-on shows you how to create a secure application using Secure Elements and Arm TrustZone on SAML11 microcontroller. It demonstrates the use case IP (Intellectual Property) protection and authentication.
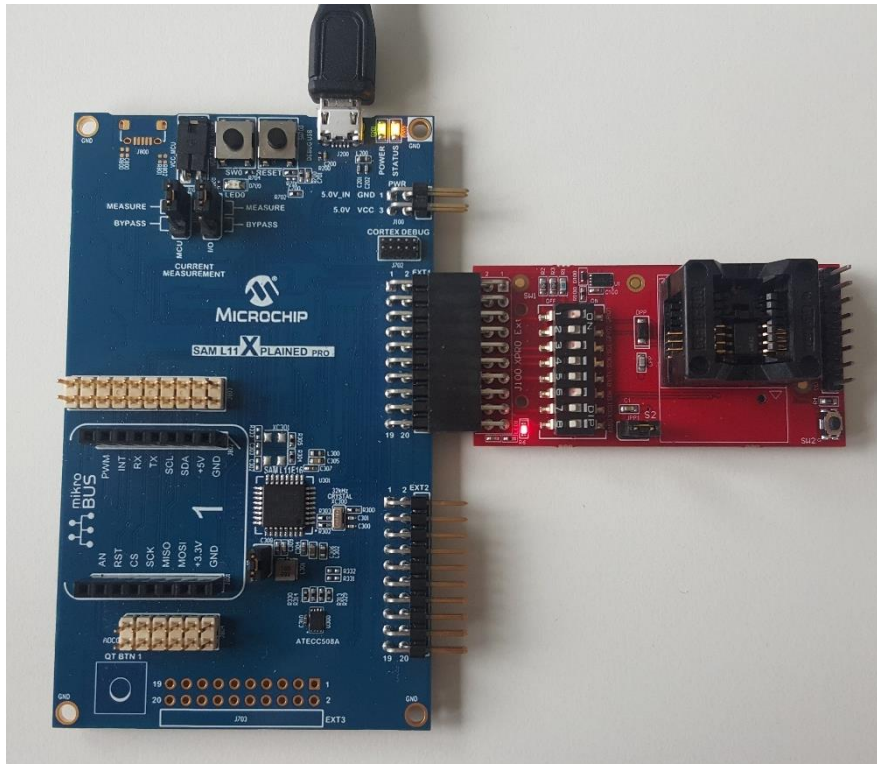
Figure 1: Use case setup

The application includes two sub-applications – the secure application which is hosted in the trusted domain of the SAML11 and the non-secure application, which locates in the non-trusted domain of the SAML11.
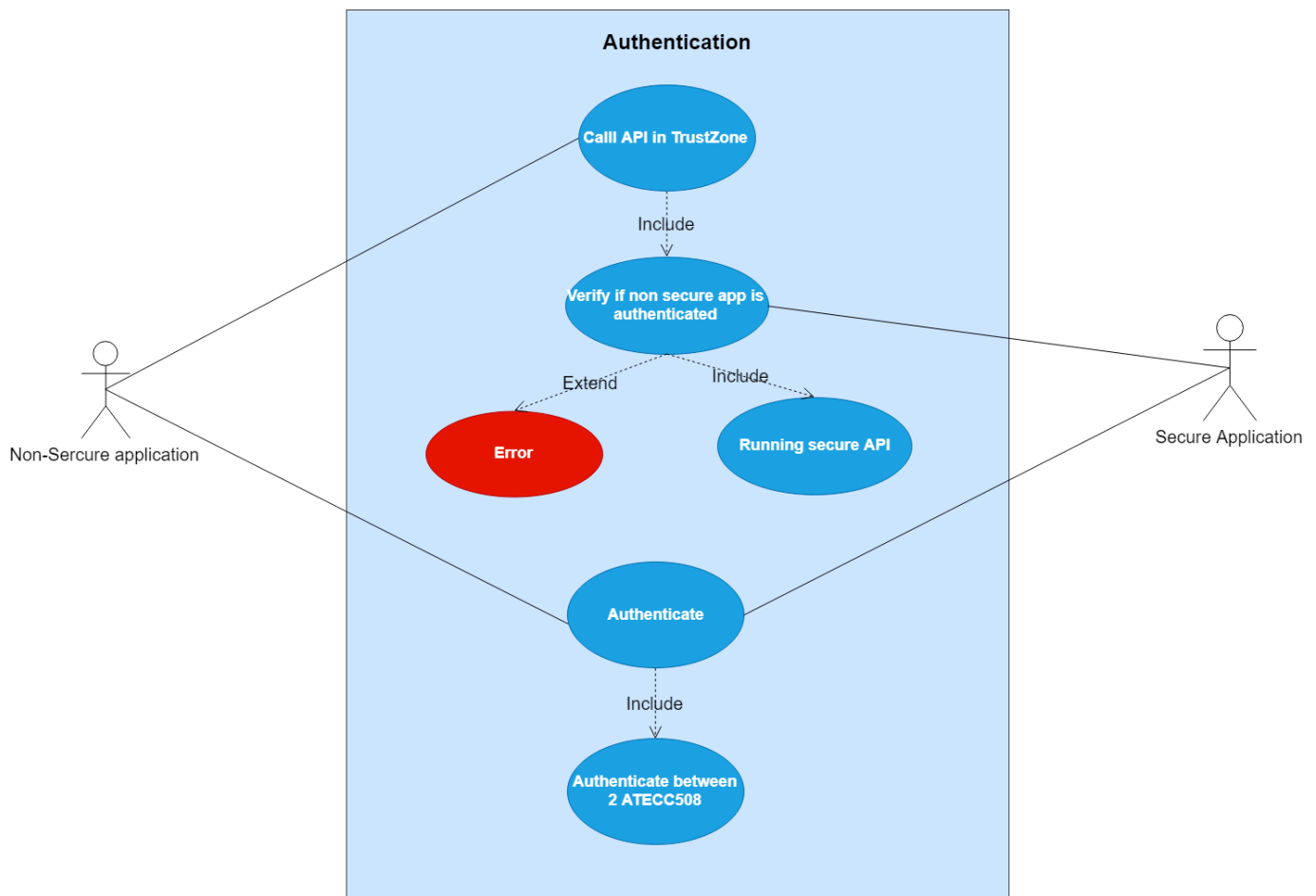
Figure 2: Use case diagram

When the non-secure application calls the API belonging to the secure application, the secure application checks if the non-secure one is already authenticated. If the authentication process is carried out and succeeded, the secure application allows the API to be executed. If the authentication process fails, an error message will be shown.

Hands-on description

During this hands-on, you will learn how to create a TrustZone application from scratch by using Atmel Start to set up the peripherals and middlewares. In addition, you will learn how to port the cryptoauthlib, which is provided by Microchip for interfacing the secure element, into your project.

Arm TrustZone

TrustZone provides the flexibility for hardware isolation of memories and peripherals, therefore reinforcing the ability of Intellectual Properties (IP) and Data protection. SAML11 provides up to six regions for the Flash, up to two regions for Data Flash, up to two regions for SRAM and the ability to assign peripherals, I/O pins, interrupts to secure or non-secure application.

ATEC608

The Microchip ATECC608A integrates ECDH (Elliptic Curve Diffie Hellman) security protocol an ultra-secure method to provide key agreement for encryption/decryption, along with ECDSA (Elliptic Curve Digital Signature Algorithm) sign-verify authentication for the Internet of Things (IoT) market including home automation, industrial networking, accessory and consumable authentication, medical, mobile and more.

Assignments

- Assignment 1: Create the application in the secure domain
- Assignment 2: Create the application in the non-secure domain
- Assignment 3: Testing the application

Requirement

Hardware

- SAML11 Xplained
- Provisioned ATECC508 on the SAML11 Xplained board
- Provisioned ATECC608 with adapter
- Type A-to-micro USB cable

Software

- Atmel Studio version 7
- TeraTerm

(i) Please refer to the Installation Guide for more information on how to get these programs.

Assignments

(i) In the following part, the terms secure world, secure application, secure region are used interchangeably and referred the same terminology, which is the protected memory region in the controller.

The same principle is applied for the term non-secure application, non-secure world, non-secure region.

Assignment 1: Create the application in the secure domain

Assignment 1.1: Setup a project from Atmel Start

Create a secure project from scratch

Start Atmel Studio and then choose File → New → Atmel Start Project
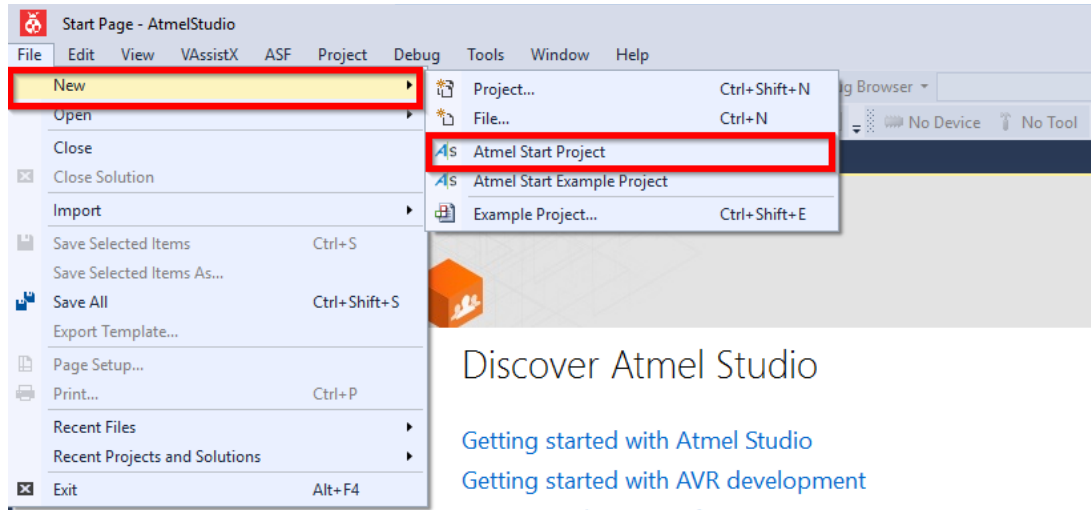
Atmel Start is the online graphical tool letting you easily set up the configuration of the microcontroller's peripherals, clock tree, and middlewares.

Because Atmel Start is an online tool, please make sure that you have the internet connection

In the *CREATE NEW PROJECT* window, we tick on the option *Show only boards*, choose *SAML11 Xplained Pro* and click *CREATE NEW PROJECT*.
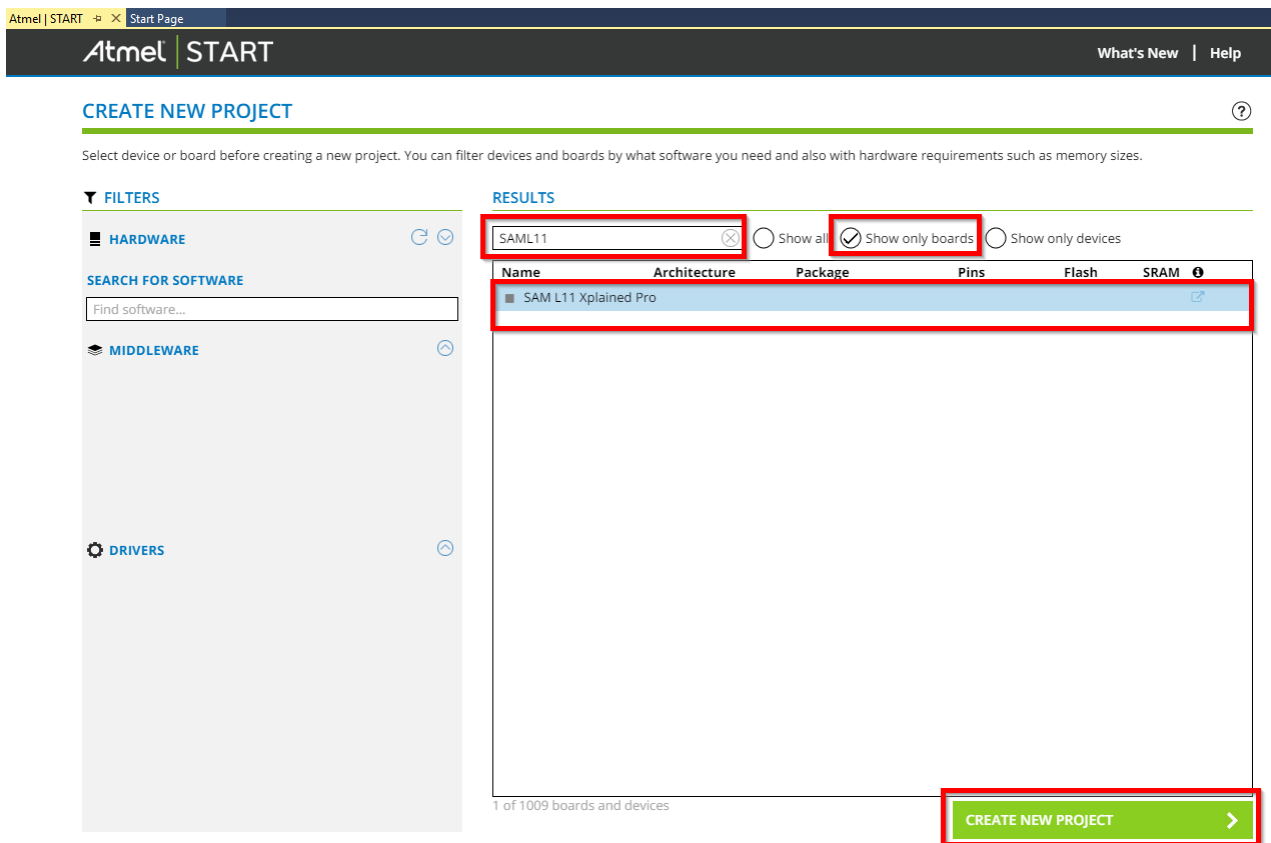
If Atmel Start started correctly, we should see the window as below:
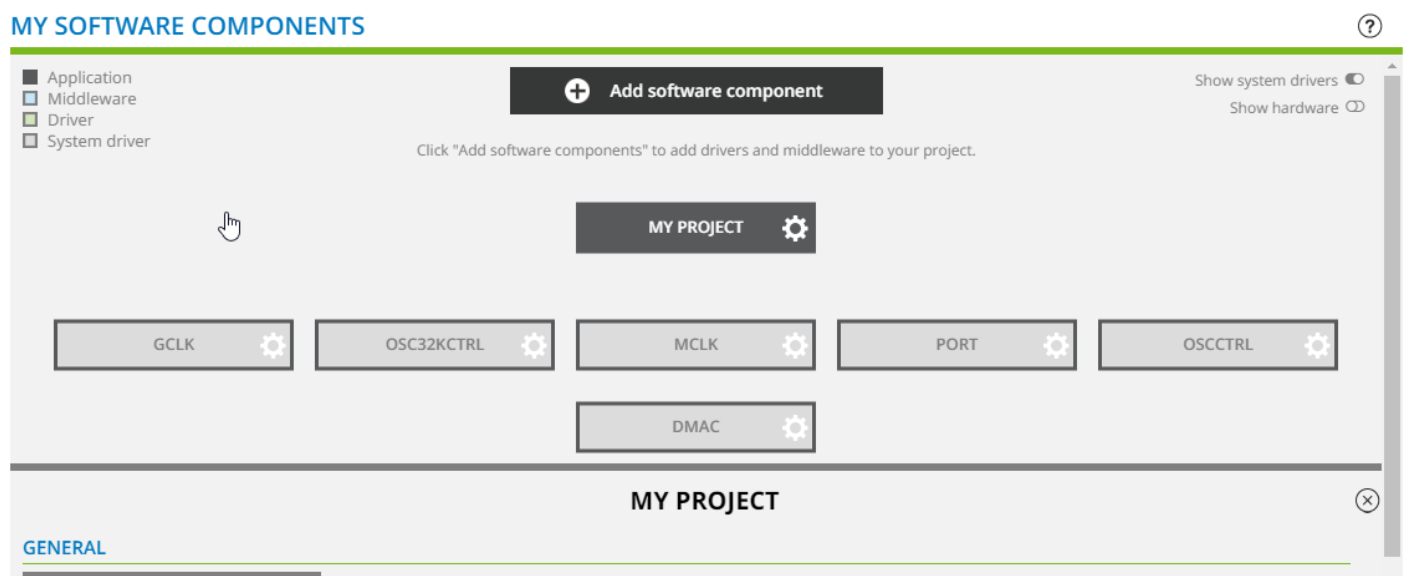
Lets' start the configuration by giving our project a more intuitive name. Click on *MY PROJECT* and choose *Rename component*.
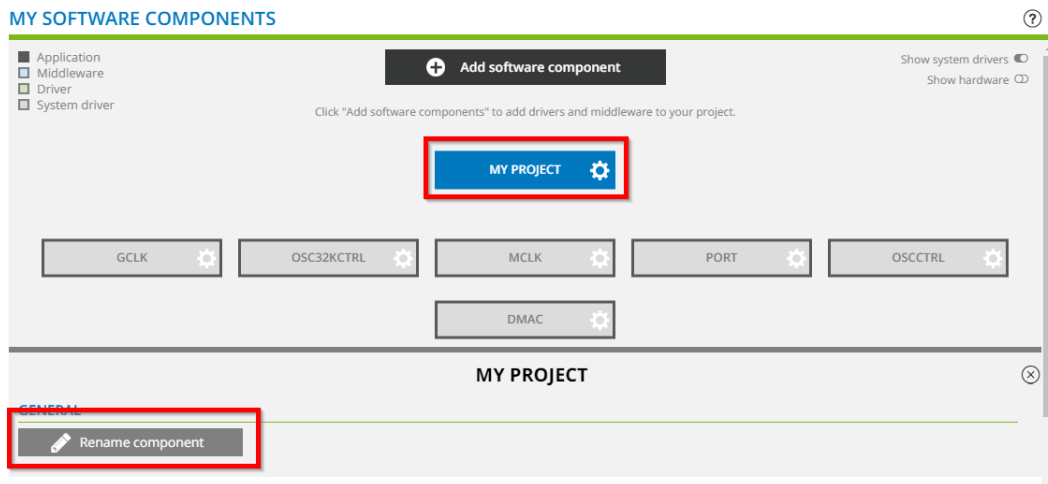
Let's name our project "Secure Project"
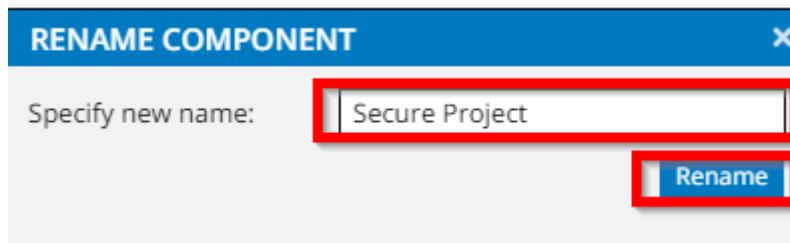
For the Secure Project, we must set up three components: *I2C bus* to interface with the secure elements*, TrustZone management* to configure the trusted and non-trusted memory area and *Stdio Redirect* to enable debug information on the terminal.

Configure the I2C bus

Click *Add software component.* In the search field, enter I2C and choose *I2C* in the *Drivers* section.
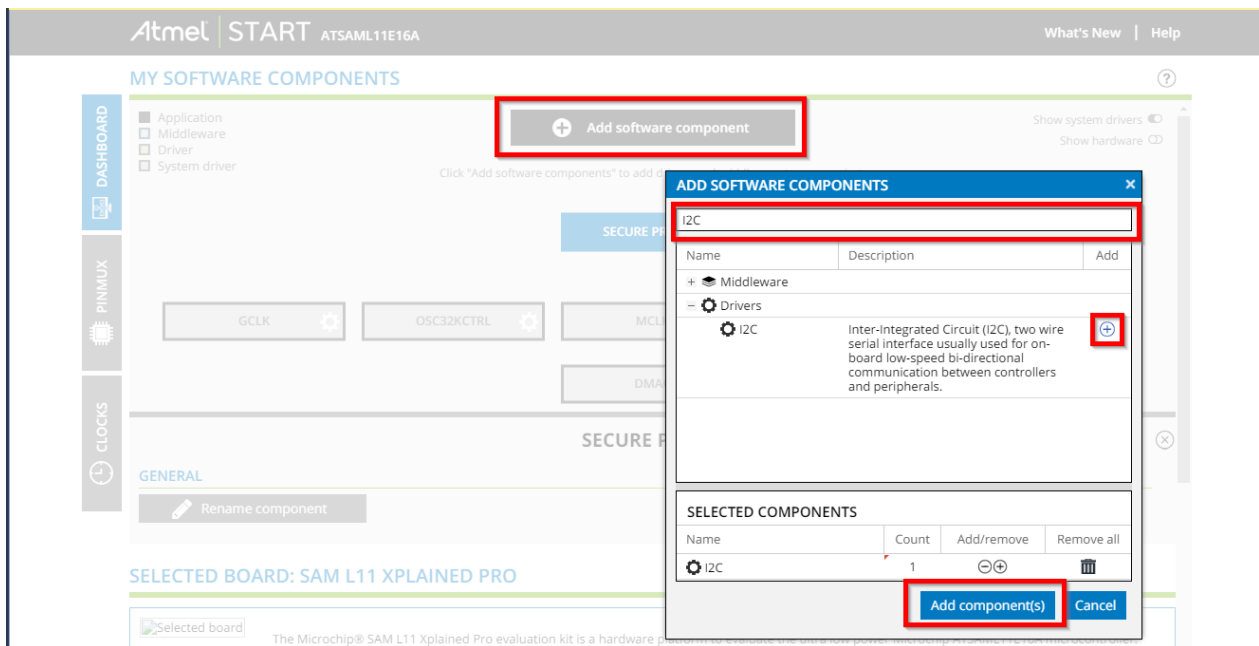
Click on the *I2C_0* to observe the configuration:
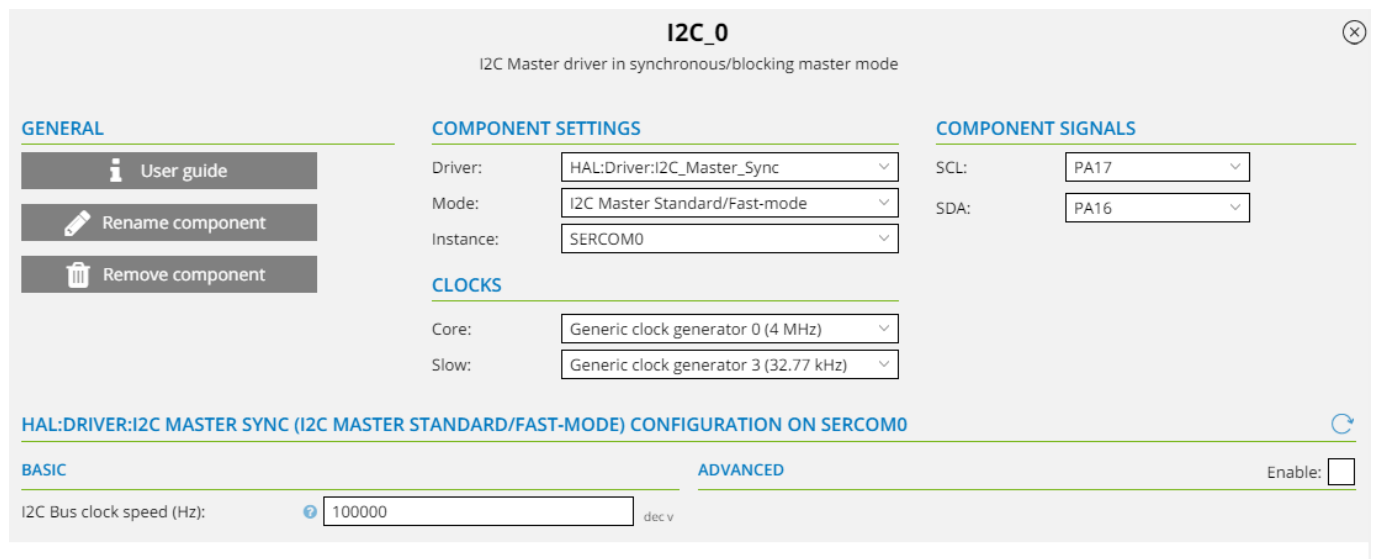
According to the getting started of the SAML11 Xplained Pro board, the pins for I2C bus on the Extension 1 header are PA17 and PA16. Therefore, we configure SCL as pin PA17 and SDA as pin PA16 and this configuration corresponds to using the SERCOM1.

 Enable debug interface on the terminal

We click again on the *Add software component.* This time, in the search field, we enter "Stdio" and choose the STDIO Redirect in the Middleware section.

Click on the *TARGET_IO,* to observe the default configuration.
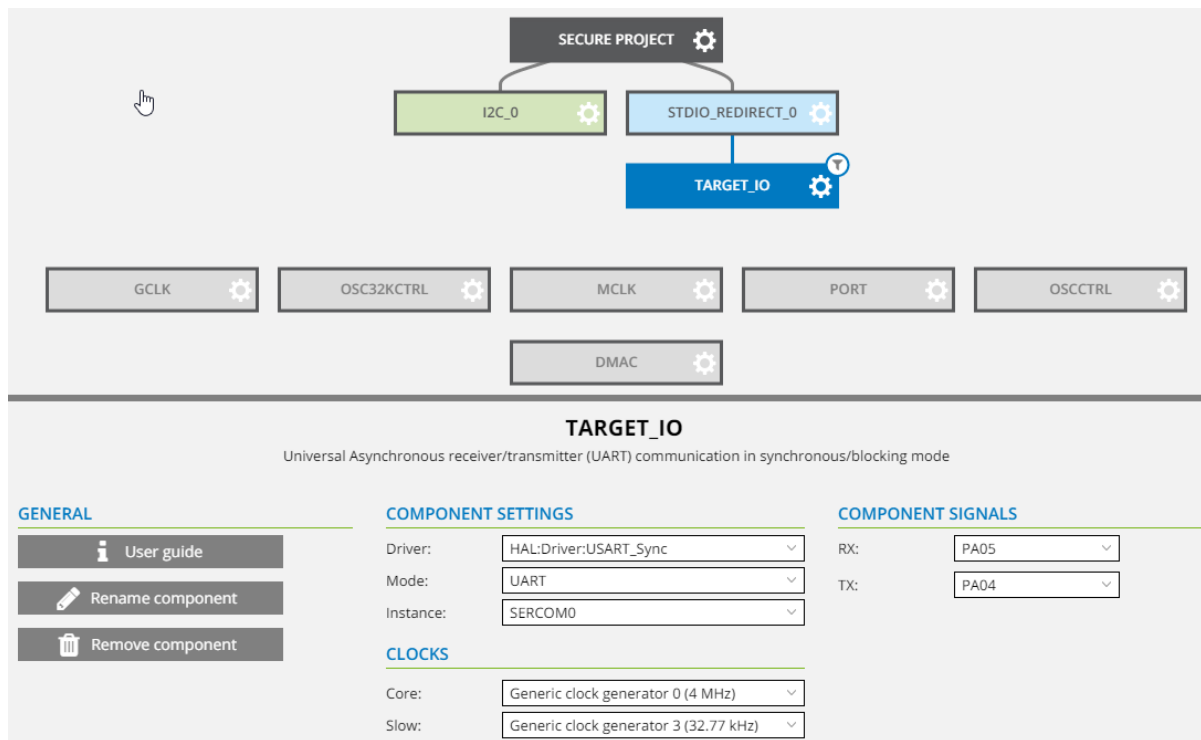
According to the Getting Started, the pin PA25 and PA24 are routed to the programming and debugging interface of the evaluation board, so we set RX to pin PA25 and TX to PA24.

We can keep the rest as default value.
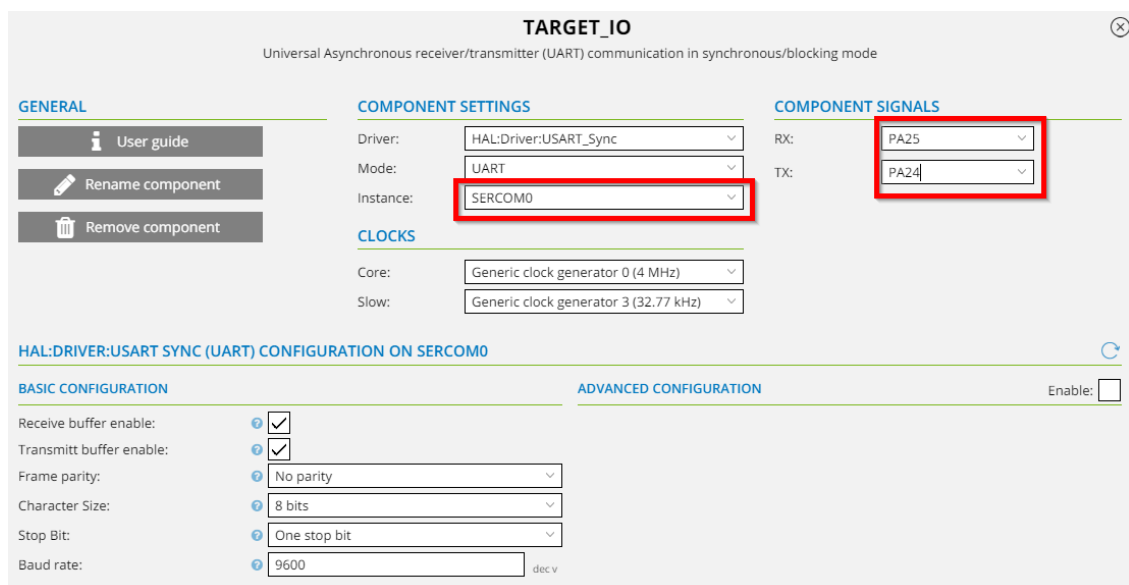
Finally, we add the *TrustZone* component to our project.
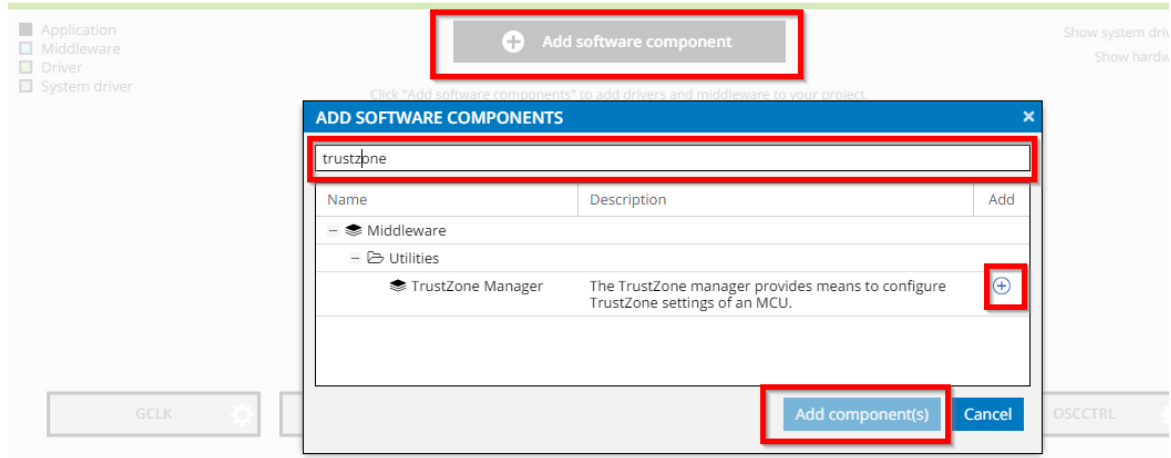
In the configuration of the *TrustZone*, we set the memory region as following:
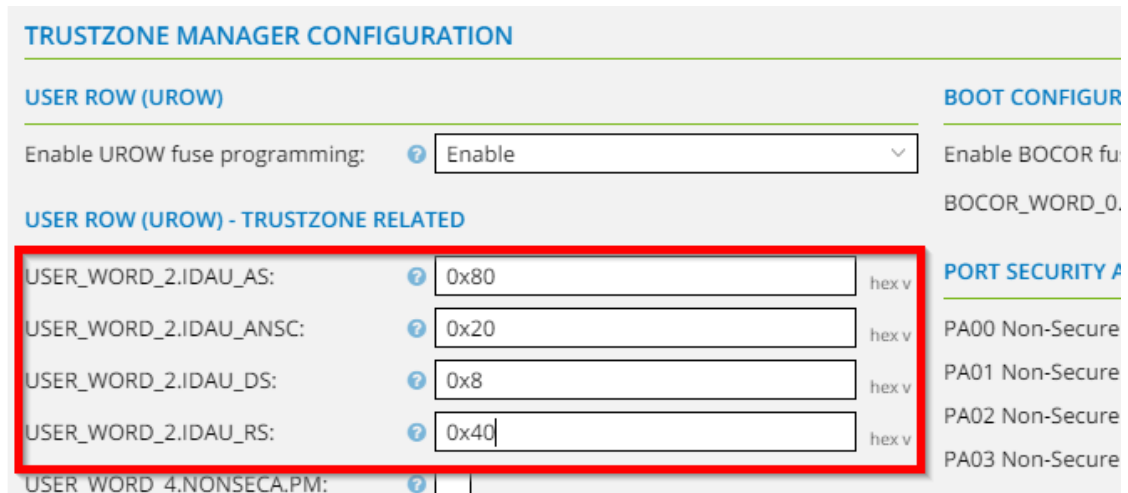
With the above configuration, the memory of the SAML11 is divided as follows:

The diagram shows memory layout:

FLASH:
- 0x0000 0000 — APP_S
- (AS x 0x100) − (ANSC x 0x20) = 0x7C00 — APP_NSC
- (AS x 0x100) = 0x8000 — APP_NS
- 0x1 0000

SRAM:
- 0x2000 0000 — RAM_S
- 0x2000 0000 + (RS*0x80) = 0x2000 2000 — RAM_NS
- 0x2000 4000

Click on *GENERATE PROJECT*
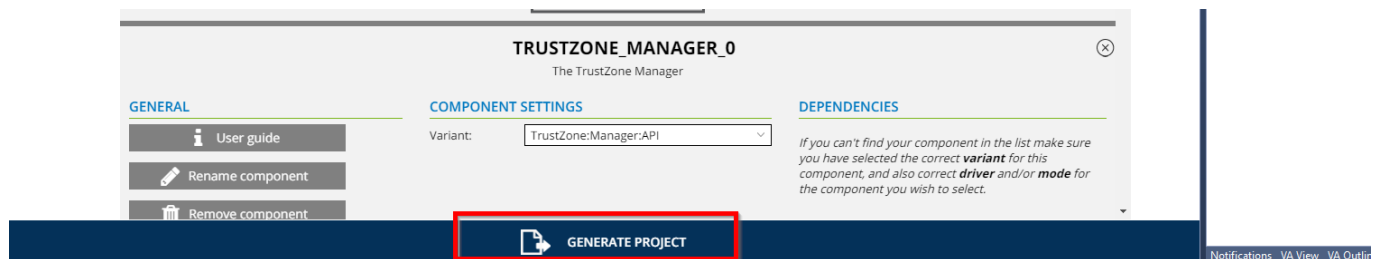
Let's name the solution *SAML11_TZ_ATECC* and click *OK.*
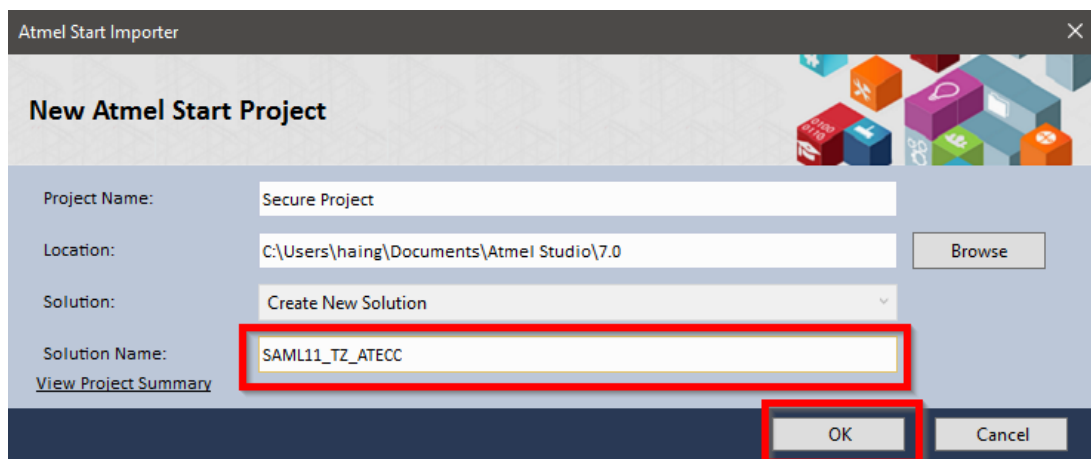
After the generation, click Build Secure Project to verify of the generation process is successful.
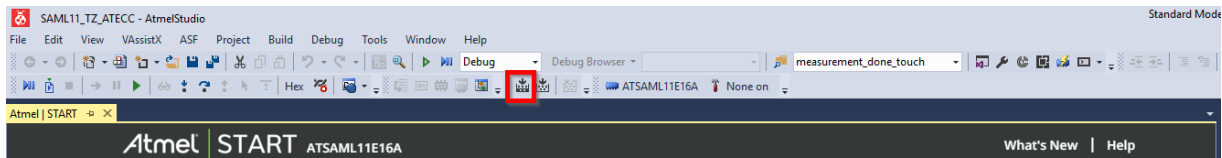
After the build, the output message should be as follows:

ⓘ If you face any error, please try to re-generate the project

In the solution explorer, focus on Secure project and choose Atmel Start icon

✏️ Writing our first hello world

Locate the main.c file and add the following line:

```
printf("hello world\r\n");
```

under atmel_start_init().

```
#include <atmel_start.h>

int main(void)
{
    /* Initializes MCU, drivers and middleware */
    atmel_start_init();
    printf("hello world\r\n");
    /* Replace with your application code */
    while (1) {
    }
}
```

✏️ Setup TeraTerm

Connect the board to your PC via USB cable and start TeraTerm. In TeraTerm, choose File → New Connection → Serial → COMXY: EDBG Virtual COM Port (COMXY)
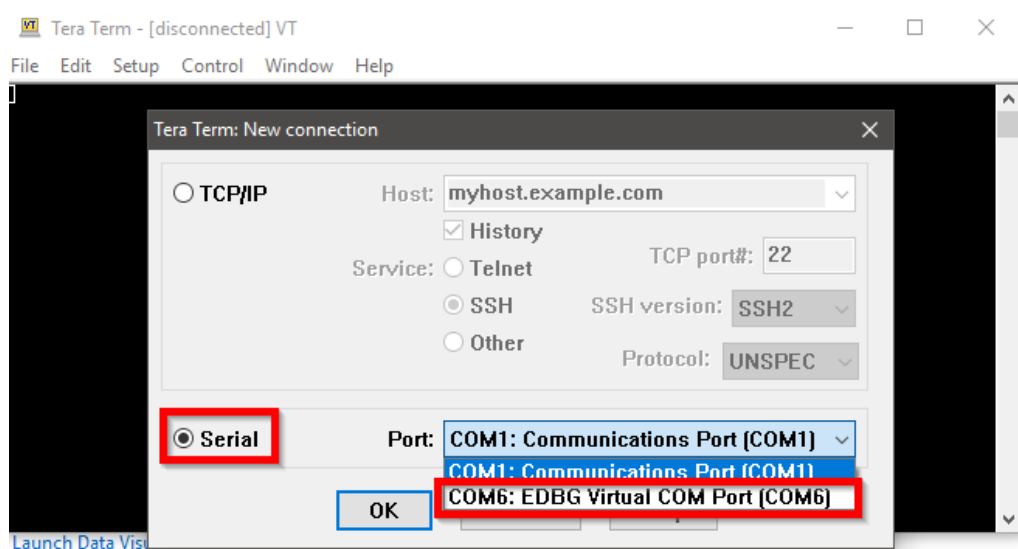
ⓘ The COM port value may be different on your machine. For example, mine is COM6

In TeraTerm, choose Setup → Terminal
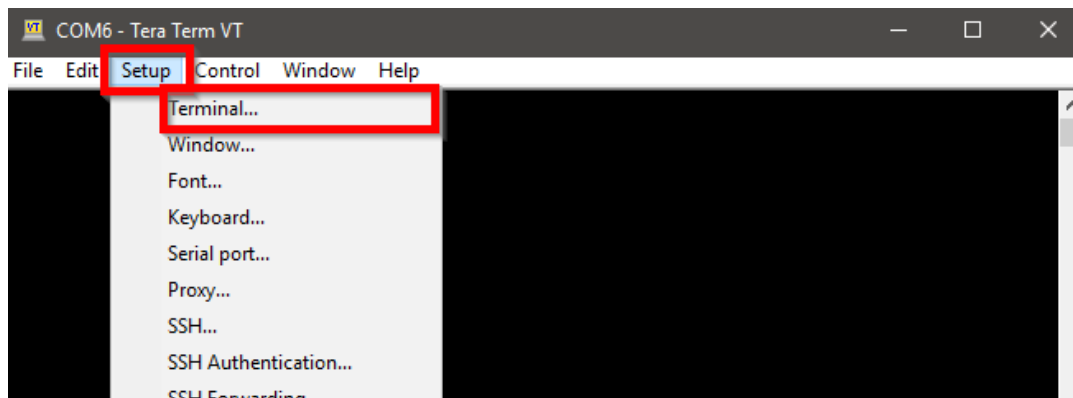
then configure as follows:
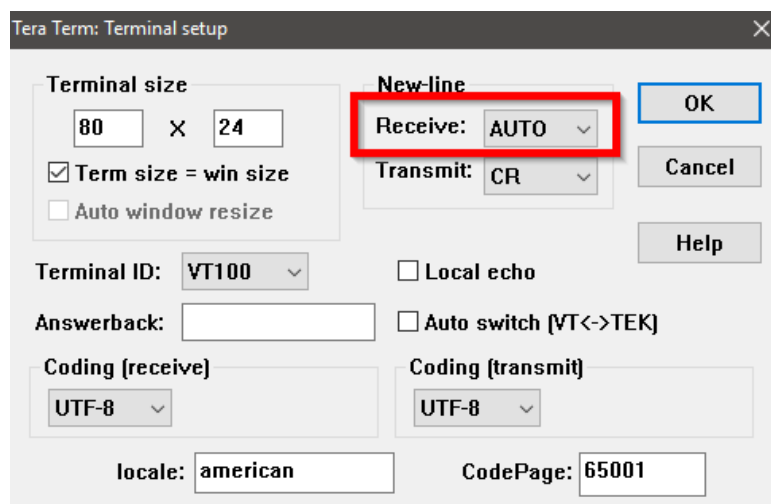
In Atmel Studio, build and run the project
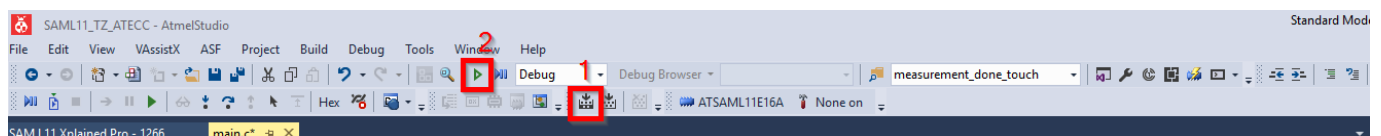
In TeraTerm, you can see the "Hello World" printed.
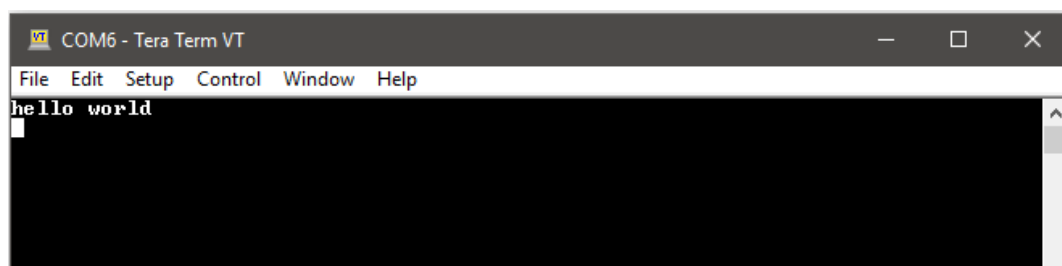
We have finished setting up the based project. Now we will integrate the cryptoauthlib into our project so we can easily interface with the secure element.

Assignment 1.2: Integrate the cryptoauthlib into the secure project

Download the cryptoauthlib from Microchip GitHub link.

Copy the folder *lib* of the *cryptoauthlib* into the *Secure Project* folder
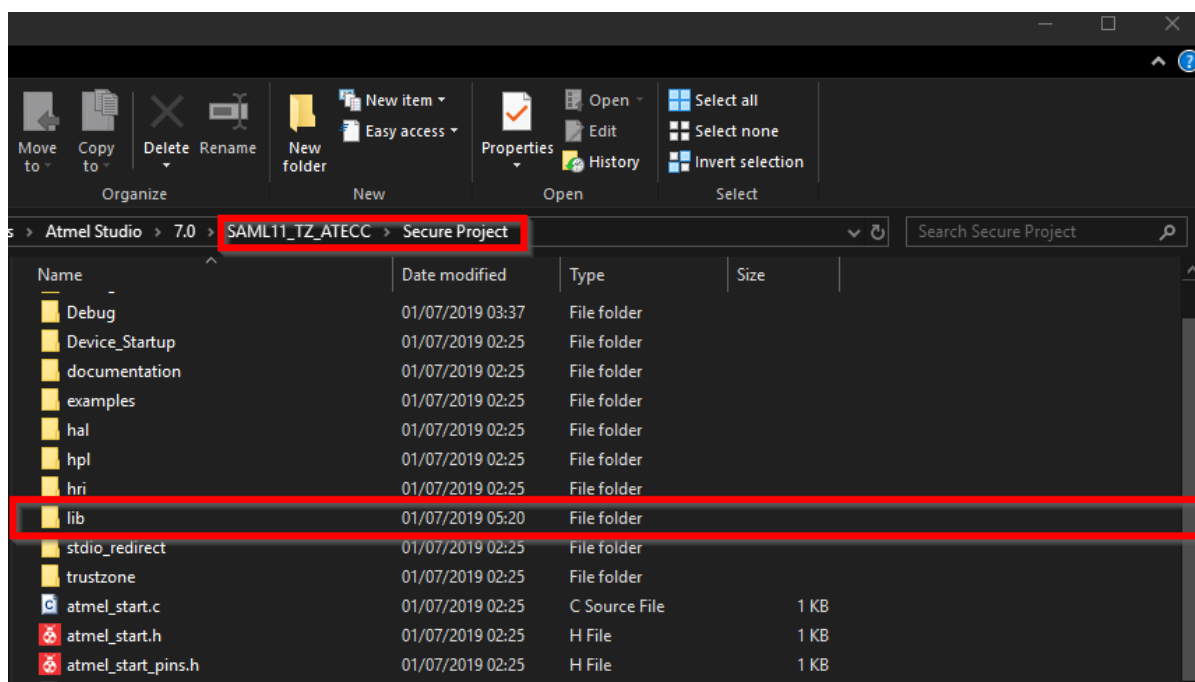
In Atmel Studio, Solution Explorer section, choose *Show All Files*
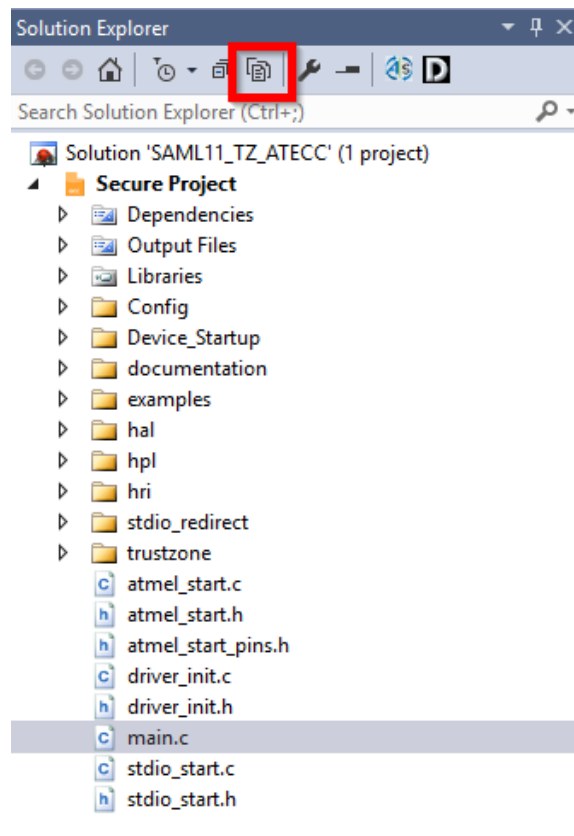
Locate the *lib* folder, right-click and choose *Include In Project.*
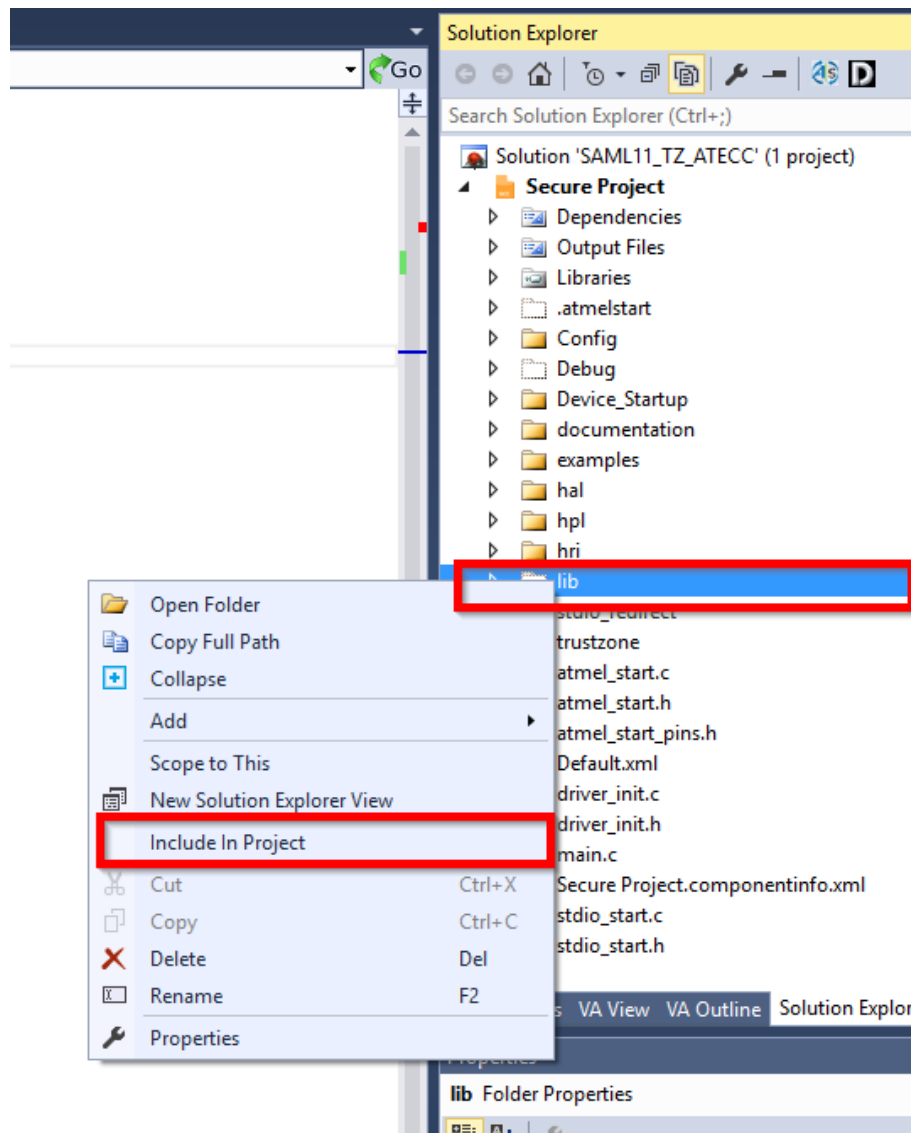
Now the library is added into the project. Next step, we must port the library to make it works with the SAML11 controller.

Please go to folder lib/hal. This folder contains the HAL driver for I2C bus for multiple driver frameworks and multiple microcontroller platforms. Since we have initialized the I2C bus with Atmel Start so we delete all the files **except**:

- atca_hal.c
- atca_hal.h
- atca_start_config.h
- atca_start_iface.h
- hal_i2c_start.c
- hal_i2c_start.h
- hal_timer_start.c
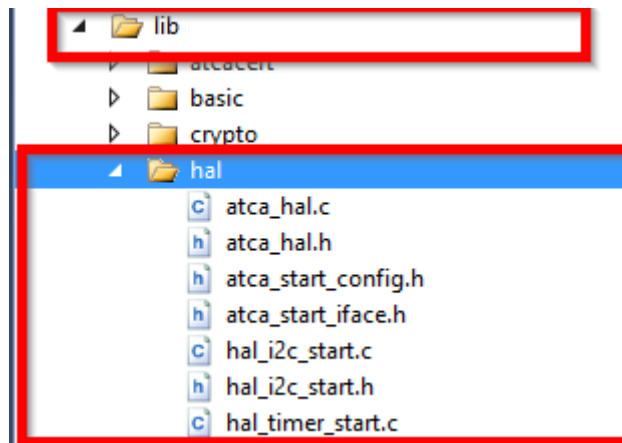
Then we delete completely the folder *mbedtls*

Next step, go to Project → Secure Project Properties
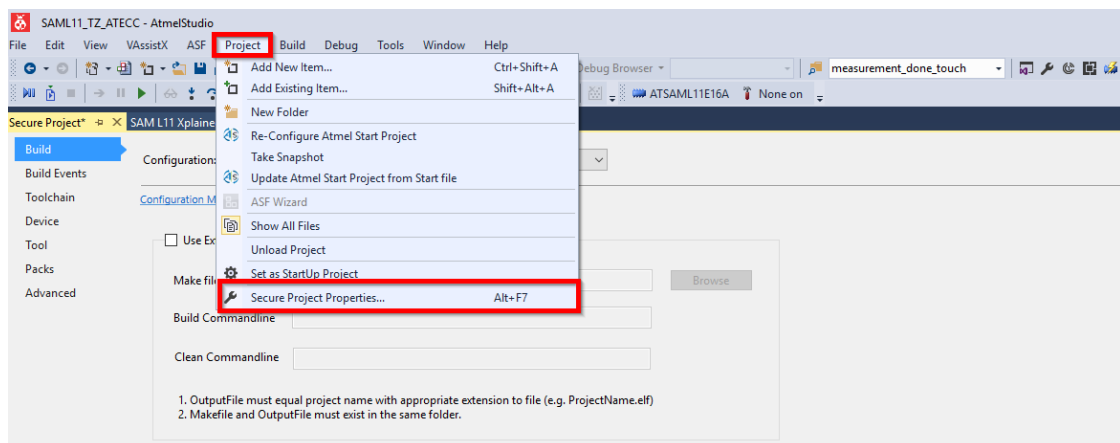
Go to Toolchain → ARM/GNU C Compiler → Directories → Add Items → Browse
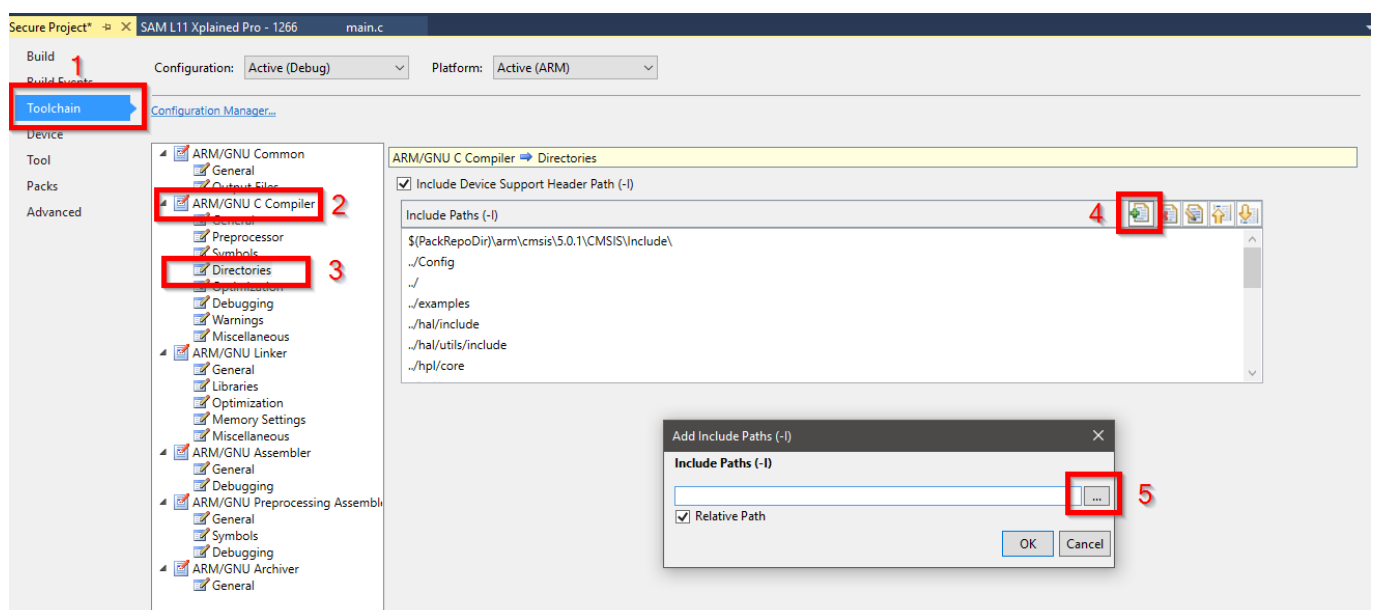
Add the following paths into the project

In the same location, go to Symbols → Add Item → enter *ATCA_HAL_I2C* to tell the cryptoauthlib that we want to use I2C to interface with the secure element.
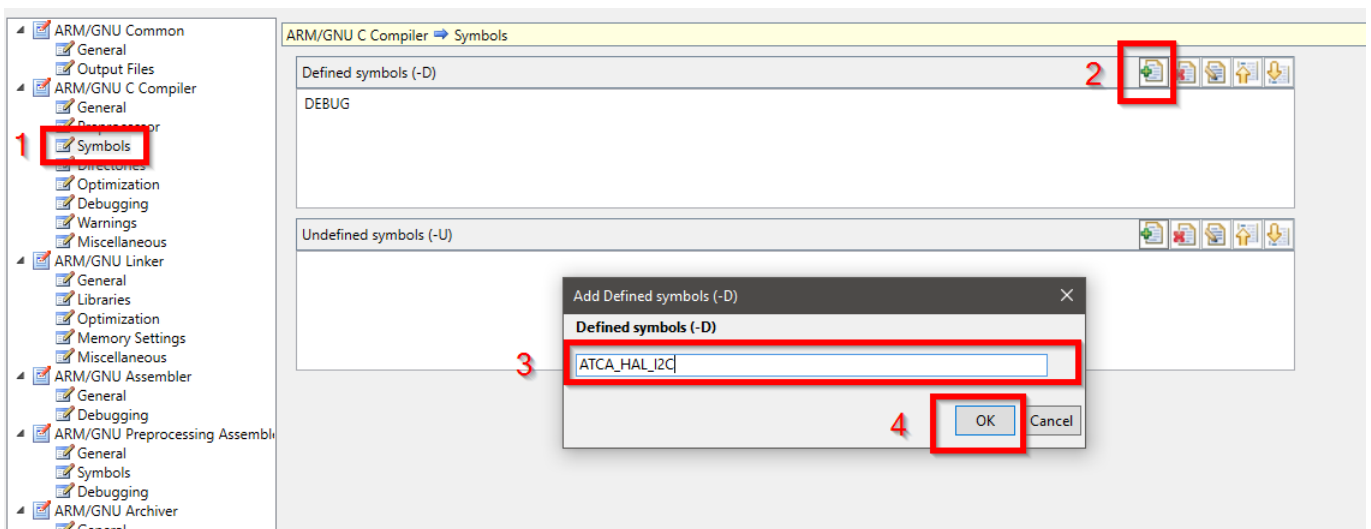
Go to file hal_i2c_start.c in lib/hal, in the function hal_i2c_init(param), modify *case 2* to *case 1* (since we are using SERCOM 1)

```
239 ⊟ATCA_STATUS hal_i2c_init(void *hal, ATCAIfaceCfg *cfg)
240   {
241       if (cfg->atcai2c.bus >= MAX_I2C_BUSES)
242       {
243           return ATCA_COMM_FAIL;
244       }
245       ATCAI2CMaster_t* data = &i2c_hal_data[cfg->atcai2c.bus];
246
247       if (data->ref_ct <= 0)
248       {
249           // Bus isn't being used, enable it
250           switch (cfg->atcai2c.bus)
251           {
252           case 1://case 2:
253               memcpy(&(data->i2c_master_instance), &i2c_0, sizeof(struct i2c_m_sync_desc));
254               data->sercom_core_freq = I2C_SERCOM_CORE_FREQ;
255               break;
256           default:
257               return ATCA_COMM_FAIL;
258           }
259
```

Build the project again and check for the error message.

	Description
❌	recipe for target 'lib/hal/hal_i2c_start.o' failed
❌	'I2C_SERCOM_CORE_FREQ' undeclared (first use in this function)
ⓘ	each undeclared identifier is reported only once for each function it appears in

Double click on the error. Replace the macro I2C_SERCOM_CORE_FREQ with the number 4000000. It is the value of the I2C core clock.

```
247       if (data->ref_ct <= 0)
248       {
249           // Bus isn't being used, enable it
250           switch (cfg->atcai2c.bus)
251           {
252           case 1://case 2:
25                                                                           _desc));
25               data->sercom_core_freq = 400000; //I2C_SERCOM_CORE_FREQ;
25               break;
256           default:
257               return ATCA_COMM_FAIL;
258           }
```

Save and build the project again. This time there should be no error.

The cryptoauthlib is fully integrated into the project, now we will write a small application to test it.

✎   Test criptoauthlib

Replace the **complete** main.c file with the code below:

```c
#include <atmel_start.h>
#include "cryptoauthlib.h"
#include "atca_host.h"

/* Define section ------------------------------------------------*/
#define CHECK_STATUS(s)                                                             \
if(s != ATCA_SUCCESS) {                                                             \
        printf("status code: 0x%x\r\n", s);                                        \
        printf("Error: Line %d in %s\r\n", __LINE__, __FILE__);                    \
        while(1);                                                                   \
}


/* Local variable section ---------------------------------------*/
ATCAIfaceCfg cfg_ateccx08a_i2c_host = {
        .iface_type                   = ATCA_I2C_IFACE,
        .devtype                      = ATECC508A,
        .atcai2c.slave_address        = 0xC2,
        .atcai2c.bus                  = 1,
        .atcai2c.baud                 = 400000,
        .wake_delay                   = 800,
        .rx_retries                   = 20,
        .cfg_data =                   &I2C_0
};

ATCAIfaceCfg cfg_ateccx08a_i2c_remote = {
        .iface_type                   = ATCA_I2C_IFACE,
        .devtype                      = ATECC508A,
        .atcai2c.slave_address        = 0xC0,
        .atcai2c.bus                  = 1,
        .atcai2c.baud                 = 400000,
        .wake_delay                   = 800,
        .rx_retries                   = 20,
        .cfg_data =                   &I2C_0
};


/* Local function prototype section -------------------------------*/
void TestPorting(void);
void print_bytes(uint8_t * ptr, uint8_t length);
int main(void)
{
        /* Initializes MCU, drivers and middleware */
        atmel_start_init();

        TestPorting();

        while (1) {
        }
}
```

```c
void print_bytes(uint8_t * ptr, uint8_t length)
{

        uint8_t i = 0;
        uint8_t line_count = 0;
        for(;i < length; i++) {
                printf("0x%02x, ",ptr[i]);
                line_count++;
                if(line_count == 8) {
                        printf("\r\n");
                        line_count = 0;
                }
        }

        printf("\r\n");
}

void TestPorting(void)
{
        printf("\x1b[2J");
        printf("Test Porting\r\n");

        volatile ATCA_STATUS status;

        status = atcab_init( &cfg_ateccx08a_i2c_host );
        CHECK_STATUS(status);
        printf("Device init complete\n\r");

        /*Getting serial from host*/
        uint8_t serial_number[ATCA_SERIAL_NUM_SIZE];

        status = atcab_read_serial_number((uint8_t*)&serial_number);
        CHECK_STATUS(status);
        printf("Serial Number of host\r\n");
        print_bytes((uint8_t*)&serial_number, 9);
        printf("\r\n");

        delay_ms(2000);

        status = atcab_init( &cfg_ateccx08a_i2c_remote );
        CHECK_STATUS(status);
        status = atcab_read_serial_number((uint8_t*)&serial_number);
        CHECK_STATUS(status);
        printf("Serial Number of remote\r\n");
        print_bytes((uint8_t*)&serial_number, 9); printf("\r\n");
}
```
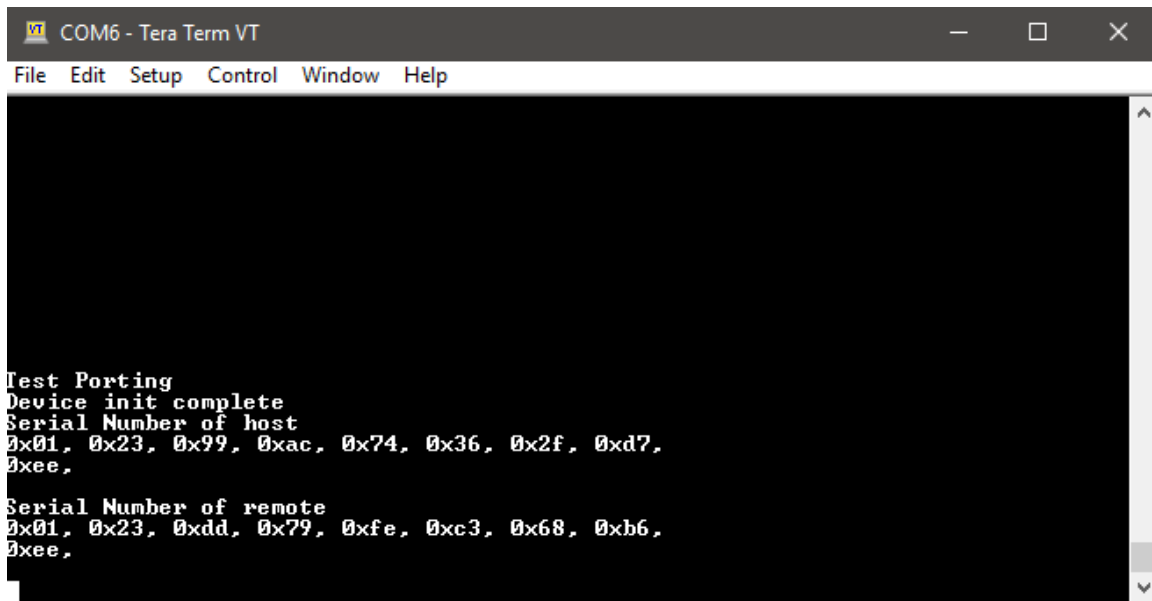
From the snippet above, what we have done is, we have created two i2c instances for two secure elements, the one on SAML11 Xplained board and the other on the adapter board. Then we initialized the communication to the two secure elements and read back the serial number from them.

Build and Run the application again. If there are no issues, we will be prompted with the serial number on the terminal as follows:

Assignment 1.3: Develop a secure application

So far we have initialized the peripherals and port the cryptoauthlib into the project. Now we start developing our application. Create a new folder called Secure Functions, this folder contains the functions belonging to the secure application. Some of these functions are shared with the non-secure application through the veneer table, which will be mentioned later.

In the *Solution Explorer*, right-click at Secure Project → Add → New Folder and name it Secure Functions

Right-click at the *Secure Functions* folder → Add → New Item and create two files SecureFunctions.c and SecureFunctions.h
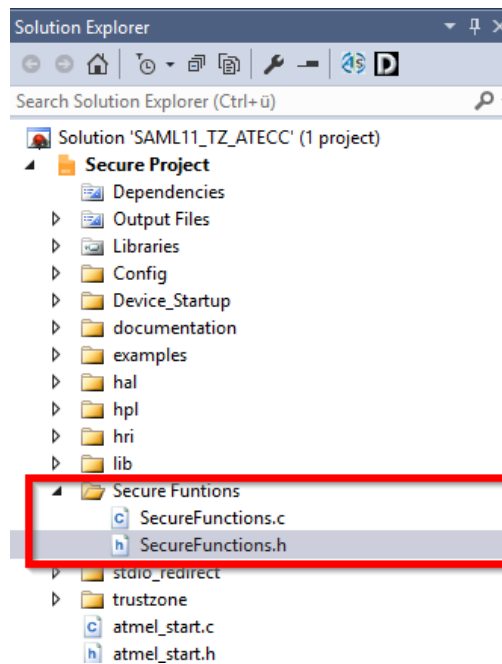
Modify SecureFuntions.h

In the file SecureFunction.h. add the following lines between the #ifndef and #endif

```
#include <atmel_start.h>

void secure_console_puts (uint8_t * string);
void non_secure_console_puts (uint8_t * string);
void SymmetricAuthentication(void);
bool IsAuthenticated(void);
uint8_t APIInTrustZone(int * a, int * b, int * sum);
```

The first two functions simply print the text on the terminal screen in different colors. The green text means it is printed from the secure application and red text means it is printed from the non-secure application.

The function SymmetricAuthentication executes the authentication between the host and remote.

The function IsAuthenticated returns the status if the host and the remote is authenticated. It is used to notify the non-secure application to perform the authentication by calling the SymmetricAuthentication before calling any other APIs.

The function APIInTrustZone simulate a special IP software or library that we want to protect. It is only allowed to be executed after successful authentication. For our case, the protected IP is just a sum function.

Next Task is writing the body of those functions.

Modify SecureFuntions.c

In the SecureFuntions.c file, add the following lines:

```c
#include "Secure Functions/SecureFuntions.h"
#include "cryptoauthlib.h"
#include "atca_host.h"


#define CHECK_STATUS(s)                                                     \
if(s != ATCA_SUCCESS) {                                                     \
        printf("status code: 0x%x\r\n", s);                                \
        printf("Error: Line %d in %s\r\n", __LINE__, __FILE__);            \
        return;                                                            \
}                                                                          \


/* Local variable section ------------------------------------------*/
ATCAIfaceCfg cfg_ateccx08a_i2c_host = {
        .iface_type                    = ATCA_I2C_IFACE,
        .devtype                       = ATECC508A,
        .atcai2c.slave_address         = 0xC2,
        .atcai2c.bus                   = 1,
        .atcai2c.baud                  = 400000,
        .wake_delay                    = 800,
        .rx_retries                    = 20,
        .cfg_data                      = &I2C_0
};

ATCAIfaceCfg cfg_ateccx08a_i2c_remote = {
        .iface_type                    = ATCA_I2C_IFACE,
        .devtype                       = ATECC508A,
        .atcai2c.slave_address         = 0xC0,
        .atcai2c.bus                   = 1,
        .atcai2c.baud                  = 400000,
        .wake_delay                    = 800,
        .rx_retries                    = 20,
        .cfg_data                      = &I2C_0
};
```

```c
void SymmetricAuthentication(void)
{

        /* Set display foreground color to green */
        printf("\033[0;32m");
        printf("Symmetric Authentication\r\n");

        printf("Authentication in progress\r\n");
        volatile ATCA_STATUS status;
        status = atcab_init( &cfg_ateccx08a_i2c_host ); CHECK_STATUS(status);
        printf("Host init complete\r\n");


        uint8_t serial_number[ATCA_SERIAL_NUM_SIZE];
        status = atcab_read_serial_number((uint8_t*)&serial_number);
        CHECK_STATUS(status);
        printf("Serial Number of host\r\n");
        print_bytes((uint8_t*)&serial_number, 9); printf("\r\n");


        uint8_t nonce[32];
        status = atcab_random((uint8_t*)&nonce);
        CHECK_STATUS(status);
        printf("Random from host\r\n");
        print_bytes((uint8_t*)&nonce, 32);


        status = atcab_init( &cfg_ateccx08a_i2c_remote );
        CHECK_STATUS(status);

        status = atcab_read_serial_number((uint8_t*)&serial_number);
        CHECK_STATUS(status);

        printf("Serial Number of remote\r\n");
        print_bytes((uint8_t*)&serial_number, 9); printf("\r\n");


        uint8_t mac[32];
        uint8_t slot = 0; uint8_t mode = (1<<6); // include serial number
        status = atcab_mac(mode, slot, (const uint8_t*)&nonce, (uint8_t*)&mac);
        CHECK_STATUS(status);
        printf("MAC from remote\r\n");
        print_bytes((uint8_t*)&mac, 32);
```

```
status = atcab_init( &cfg_ateccx08a_i2c_host );
        uint8_t otherdata[CHECKMAC_OTHER_DATA_SIZE];
        memset(otherdata, 0x00, CHECKMAC_OTHER_DATA_SIZE);
        otherdata[0] = 0x08;
        otherdata[1] = 0x40;
        otherdata[7] = serial_number[4];
        otherdata[8] = serial_number[5];
        otherdata[9] = serial_number[6];
        otherdata[10] = serial_number[7];
        otherdata[11] = serial_number[2];
        otherdata[12] = serial_number[3];
        mode = 0;

        status = atcab_checkmac(mode, slot, (const uint8_t*)&nonce, (const uint8_t*)&mac, (const
uint8_t*)&otherdata);

        if(status == ATCA_SUCCESS)
        {
                printf("Authenticated by host\r\n\r\n");
                authen_status = true;
        }
        else
        {
                printf("Failed to authenticate\r\n\r\n");
                authen_status = false;
        }
}

uint8_t APIInTrustZone(int * a, int * b, int * sum)
{

        if(authen_status)
        {
                *sum = *a + *b;
                return 0;
        }
        else
        {
                return 1;
        }

}
```

The most important function is SymmetricAuthentication. In that function, the I2C communication of the two secure elements are initialized. Then the host requests the serial number of the remote for later usage. After that, the host sends a challenge (random number) to the remote. The remote uses its serial number to sign the challenge coming from the host and sends the signature back to the host. The host uses the remote seriyl number to verify the signature from the remote. If there is a matched, the host and the remote are authenticated.

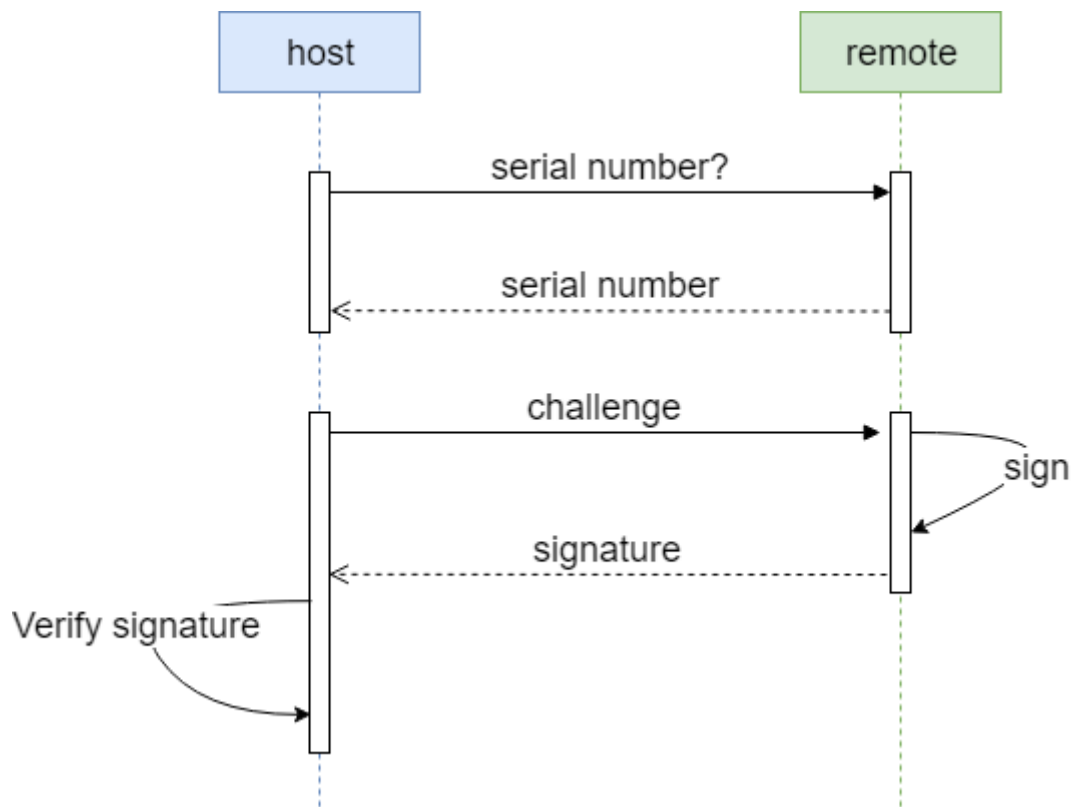The process above is explained again by the sequence diagram

Figure 3: Authentication sequence diagram

Next step, we modify the veneer functions. The veneer functions allow non-secure application calls the designated APIs from the secure application.

Modify trustzone_veneer.h

Open the file trustzone_veneer.h locating in trustzone folder.

At the end of the file, before the line #endif, add these following lines:

```
extern void nsc_non_secure_console_puts (uint8_t * string);

extern void nsc_SymmetricAuthentication (void);

extern bool nsc_IsAuthenticated (void);

extern uint8_t nsc_APIInTrustZone (int * a, int * b, int * sum);
```

Those functions above are the ones that can be called from the non-secure application. Next step we add the functions body to the trustzone_venner.c file

Modify trustzone_veneer.c

In file trustzone_veneer.c, add this line in the include section:

```
#include "Secure_functions/SecureFunctions.h"
```

At the very bottom of the file, at the following lines:

```
void __attribute__((cmse_nonsecure_entry)) nsc_non_secure_console_puts (uint8_t * string)
{
        non_secure_console_puts(string);
}

void __attribute__((cmse_nonsecure_entry)) nsc_SymmetricAuthentication (void)
{
        SymmetricAuthentication();
}

bool __attribute__((cmse_nonsecure_entry)) nsc_IsAuthenticated (void)
{
        return IsAuthenticated();
}

uint8_t __attribute__((cmse_nonsecure_entry)) nsc_APIInTrustZone (int * a, int * b, int * sum)
{
        return APIInTrustZone(a,b,sum);
}
```

These functions do nothing more than call the functions in the SecureFunction.c file we have created before. By this way, we allow the non-secure application calling the APIs from secure application.

Modify main.c

We will modify the main.c file to test the functions. Replace the complete main.c with the following lines:

```c
#include <atmel_start.h>
#include "Secure_functions/SecureFunctions.h"


/* Define section ------------------------------------------------*/

/* TZ_START_NS: Start address of non-secure application */
#define TZ_START_NS 0x00008000

/* typedef for non-secure callback functions */
typedef void (*ns_funcptr_void) (void) __attribute__((cmse_nonsecure_call));

int main(void)
{
        /* Pointer to Non secure reset handler definition*/
        ns_funcptr_void NonSecure_ResetHandler;

        /* Initializes MCU, drivers and middleware */
        atmel_start_init();

        /* Print Secure Hello world on the terminal window */
        secure_console_puts("\x1b[2J");
        secure_console_puts ("Secure Hello world !\r\n");

        /* - Set non-secure main stack (MSP_NS) */
        __TZ_set_MSP_NS(*((uint32_t *)(TZ_START_NS)));

        /* - Get non-secure reset handler */
        NonSecure_ResetHandler = (ns_funcptr_void)(*((uint32_t *)((TZ_START_NS) + 4U)));

        /* - Start Non-secure Application */
        NonSecure_ResetHandler();

        /* Replace with your application code */
        while (1) {
        }
}
```

In the main.c, we have done two things. The first thing is printing the text "hello world" in green color to the terminal to indicate that, the secure application is running. The second thing we have done is setting up the start address of the non-secure application and jumping to non-secure application.

Build and run the project.

We should see the result on the terminal as follows:
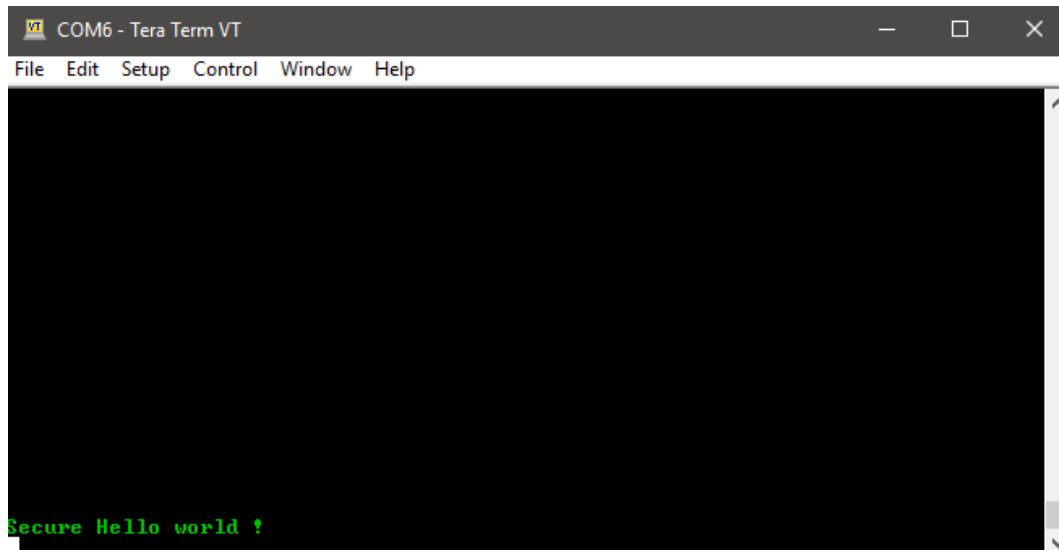
We have done the secure application part, which includes the authentication function and the protected API. The next part, we will create the non-secure application, which executes the authentication and calls protected API in the secure application.
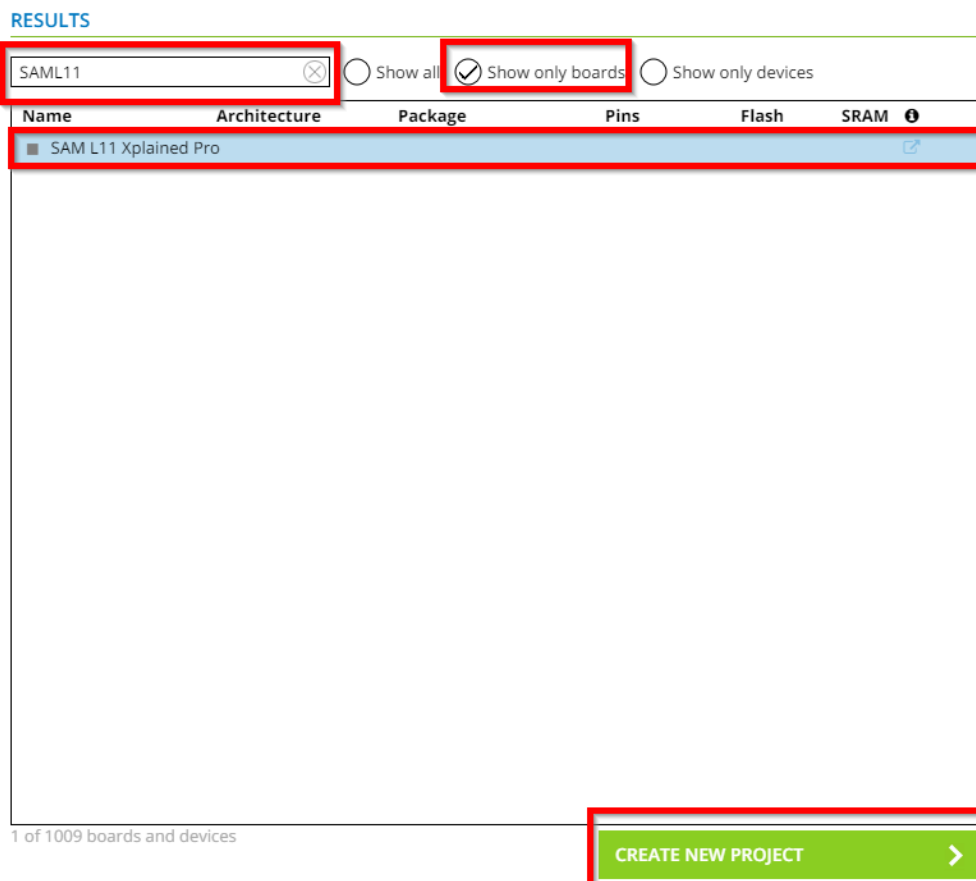
Assignment 2: Create the application in the non-secure domain

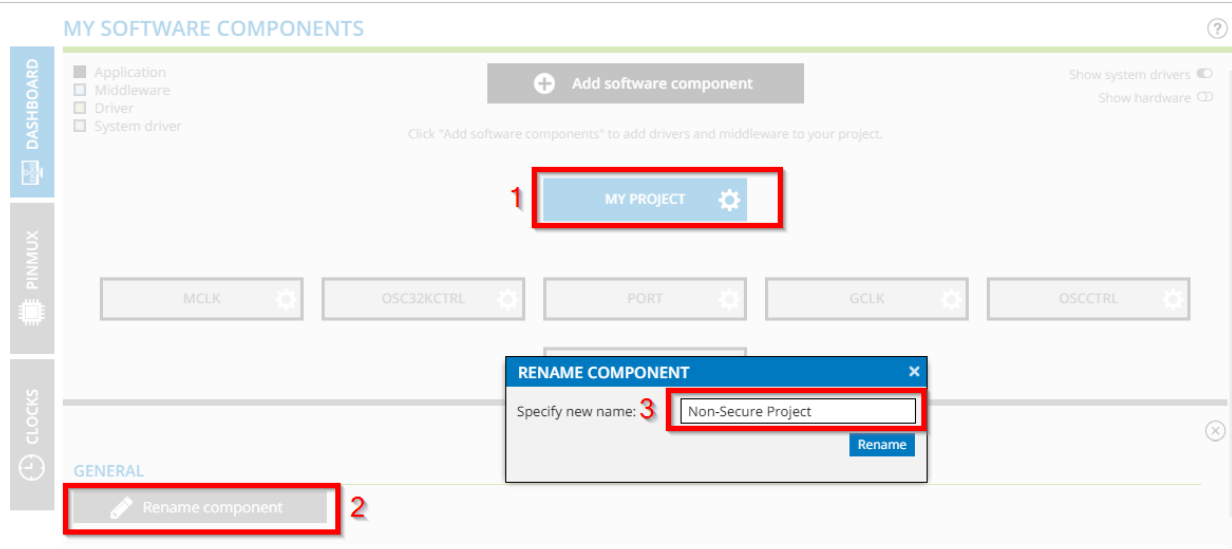### Assignment 2.1: Initialize a non-application with Atmel Start

In Atmel Studio, Choose File → New → Atmel Start Project
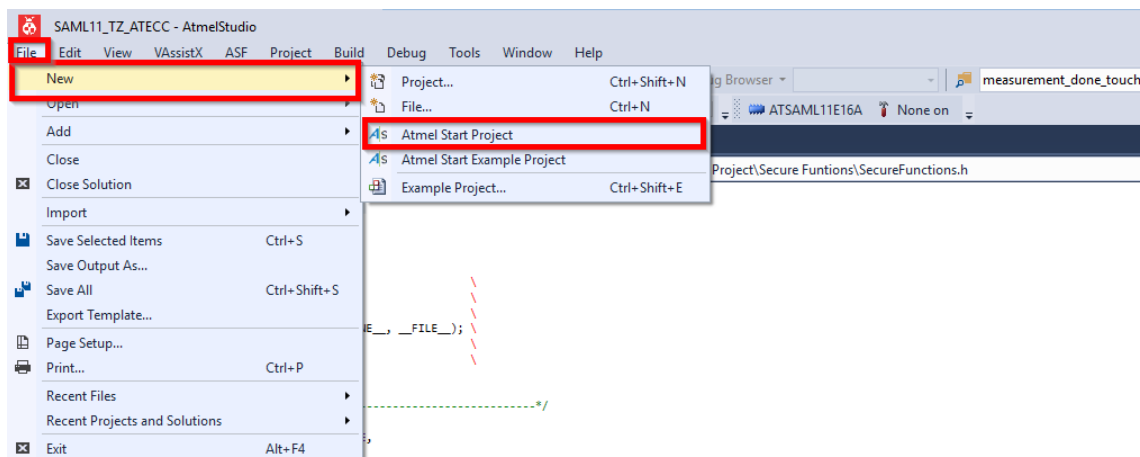
Carry out the same procedure as in the Create Secure application project section, we look for the evaluation board *SAML11 Xplained Pro* and click *CREATESA NEW PROJECT.*


Rename your project to *Non-Secure Project*

Since we don't need anything (the peripherals are initialized by the Secure Project), we click *GENERATE PROJECT.*

The program asks us to add the path of the veneer header file, export library and non-secure project linker script. These files locate in:

- Veneer header file:
  - o Your_project_location/Secure Project/trustzone/trustzone_veneer.h
- Export library:
  - o Your_project_location/Secure Project/Debug/libsecure_gateway_veneer.lib
- Non-Secure Project Linker Script:
  - o Your_project_location/Secure Project/Device_Startup/saml11_nonsecure.ld
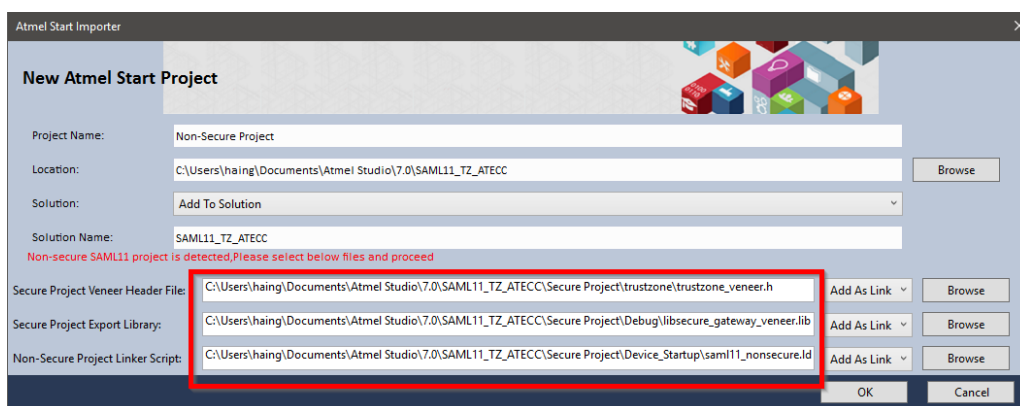
Assignment 2.2: Develop a non-secure application:

Modify main.c

Open the main.c file in the Non-Secure Project and replace it completely with the following code:

```c
#include <atmel_start.h>
#include "trustzone_veneer.h"
#include <stdio.h>

static int a = 10;
static int b = 20;
static int sum = 0;
static char s[100];


int main(void)
{


        /* Initializes MCU, drivers and middleware */
        atmel_start_init();

        /* Verify non-secure application is authenticated */
        nsc_non_secure_console_puts((uint8_t *)"Non-Secure Hello World !\r\n");

        /*Try to execute secure function call before authentication*/
        nsc_non_secure_console_puts((uint8_t *)"Verify if non-secure application is authenticated\r\n");
        if(nsc_IsAuthenticated())
        {
                nsc_non_secure_console_puts((uint8_t *)"Authenticated\r\n");
        }
        else
        {
                nsc_non_secure_console_puts((uint8_t *)"Not authenticated\r\n");
        }

        /*Execute secure function without authentication*/
        nsc_non_secure_console_puts((uint8_t *)"Try to call the API in trustzone without authentication\r\n");
        if(0 == nsc_APIInTrustZone(&a, &b, &sum))
        {
                sprintf(s, "sum of %d and %d is %d\r\n", a, b, sum);
                nsc_non_secure_console_puts((uint8_t *)s);
        }
        else
        {
                nsc_non_secure_console_puts((uint8_t *)"Function is not executed\r\n");
        }
```

```
        /*Execute secure function without authentication*/
        nsc_non_secure_console_puts((uint8_t *)"Try to call the API in trustzone without authentication\r\n");
        if(0 == nsc_APIInTrustZone(&a, &b, &sum))
        {
                sprintf(s, "sum of %d and %d is %d\r\n", a, b, sum);
                nsc_non_secure_console_puts((uint8_t *)s);
        }
        else
        {
                nsc_non_secure_console_puts((uint8_t *)"Function is not executed\r\n");
        }

        /*Authentication and call the secure function again*/
        nsc_non_secure_console_puts((uint8_t *)"Authenticating...\r\n");
        delay_ms(3000);

        nsc_SymmetricAuthentication();

        nsc_non_secure_console_puts((uint8_t *)"Verify non-secure application is authenticated\r\n");
        if(nsc_IsAuthenticated())
        {
                nsc_non_secure_console_puts((uint8_t *)"Authenticated\r\n");
        }
        else
        {
                nsc_non_secure_console_puts((uint8_t *)"Not authenticated\r\n");
        }

        nsc_non_secure_console_puts((uint8_t *)"Try to call the API in trustzone again\r\n");
        if(0 == nsc_APIInTrustZone(&a, &b, &sum))
        {
                sprintf(s, "sum of %d and %d is %d\r\n", a, b, sum);
                nsc_non_secure_console_puts((uint8_t *)s);
        }
        else
        {
                nsc_non_secure_console_puts((uint8_t *)"Function is not executed\r\n");
        }

        /* Replace with your application code */
        while (1) {
        }
}
```

What happens in the main loop is, we try to execute the protected API without the authentication process. We should get an error since the authentication is not carried out. Then we authenticate the remote with the host and call the protected API again.
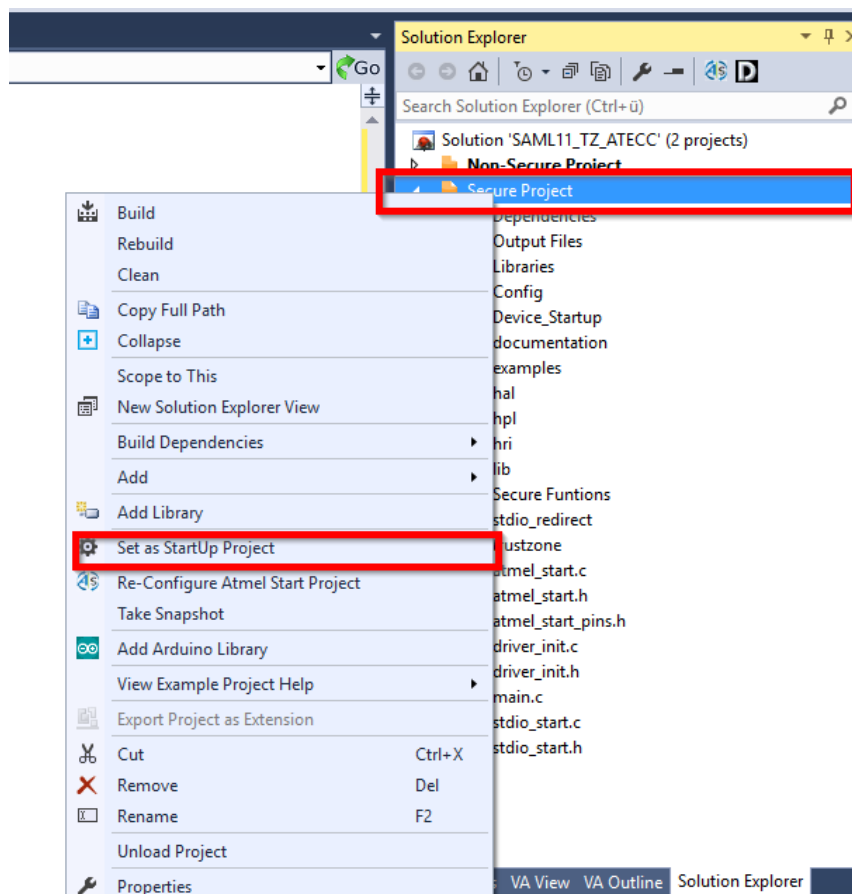
Build the Non-Secure Project.

Before build and run the complete solution, right click on the Secure Project → Properties
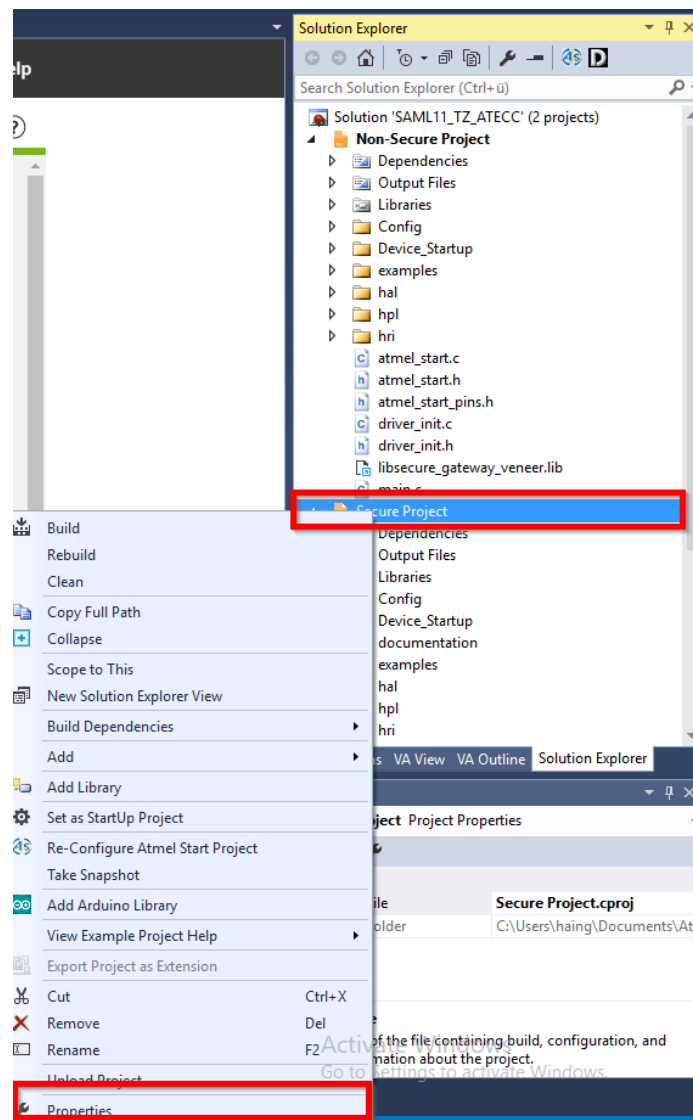
Choose Advance → Add → Navigate to Non-Secure Project/Debug/Non-Secure Project.elf → Open
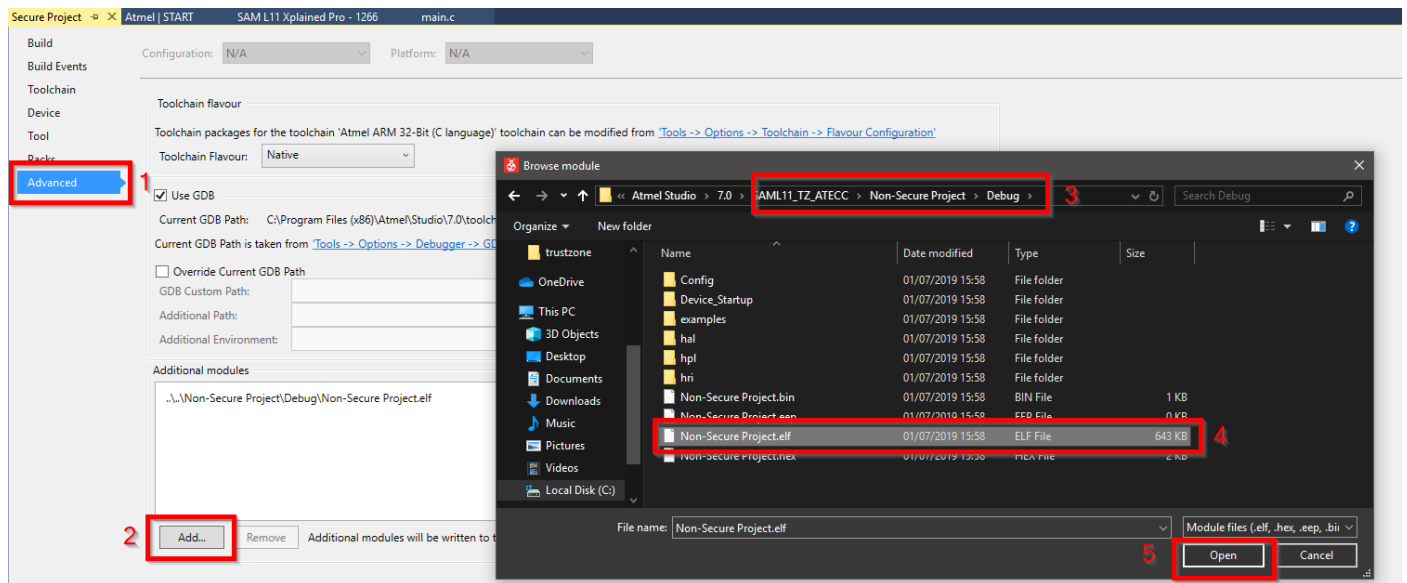
Right-click on the Secure Project again and choose *Set as StartUp Project*

Rebuild and run the complete solution.

Assignment 3: Testing the application

Press reset button on the SAML11 Xplained

✓ If it is setup correctly, you should see the following output on the terminal

In the beginning, the non-secure application tries to call a secure API (the sum function) without authentication. Because of unauthorized access, the non-application is not allowed to call the secure API, and the secure API returns an error message.

After the successful authentication, the non-secure application calls the secure API again and receives the result of the summation between 10 and 20.

✎ Repeat the step again without plugging the ATECC508 to simulate invalid hardware situation

Remove this one
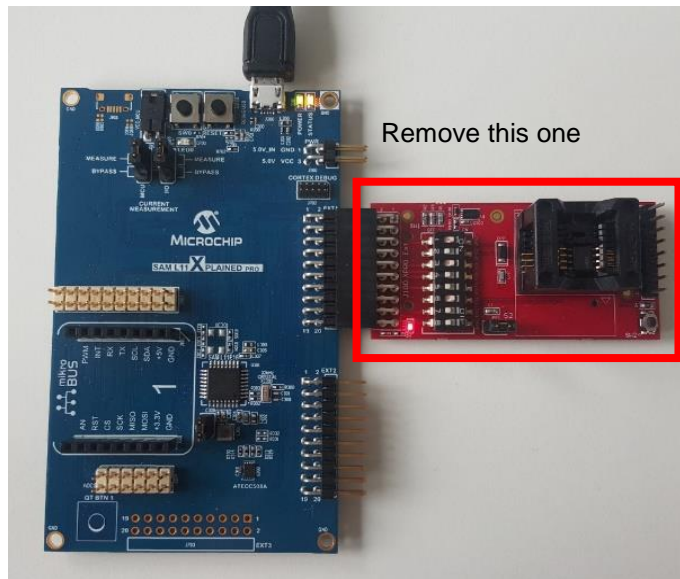
Because of without the presence of the secure element, the authentication process fails, which is demonstrated by an error code on the terminal. Without authentication, the non-secure world has no way to call the secure API.

Figure 4: Calling secure API without a secure element

Congratulation! You have finished the hands-on!

Contact information

Janus Piwek
Dipl.-Ing. (FH)
Market Development Engineer - Microchip
Arrow Central Europe GmbH

e-mail: jpiwek@arroweurope.com

THE END