TA100 Hands-on using dsPIC33CH

Microchip RTC **SEC-TA01**

Roy Yen, Taiwan ESE

2024. March

Agenda

- **TA100 Introduction** **10:00~11:00**
  - Lab1 – Create TA100 project on dsPIC33CH
  - Lab2 – Installing Trust Anchor MCC SW Module
  - Lab3 – Generate dsPIC33 code base using MCC Melody
  - Lab4 – Try running TA100 ➡ Make sure HW/SW are all good
  - Lab5 – Try your 1st TA100 function ➡ Everyone should get different result per TA100
- **TA100 Handles introduction** **11:00~11:30**
  - Lab6 – Provision TA100 ➡ Provision could only be processed 1 time
- **Asymmetric Authentication** **1:30~2:30**
  - Lab7 – ECDSA Sign & Verify using TA100 (**extra Practice**)
  - Lab8 – Read out Device Certificate (**extra Practice**)
- **Hash Function** **2:30~4:00**
  - Lab9-1 – Calculate digest using Online SHA384
  - Lab9-2 – Calculate Device Certificate TBS digest (**extra Practice**)
  - Lab9-3 – Calculate SHA384 using dsPIC33CK
  - **Lab10 – Verify Device Certificate (extra Practice)**

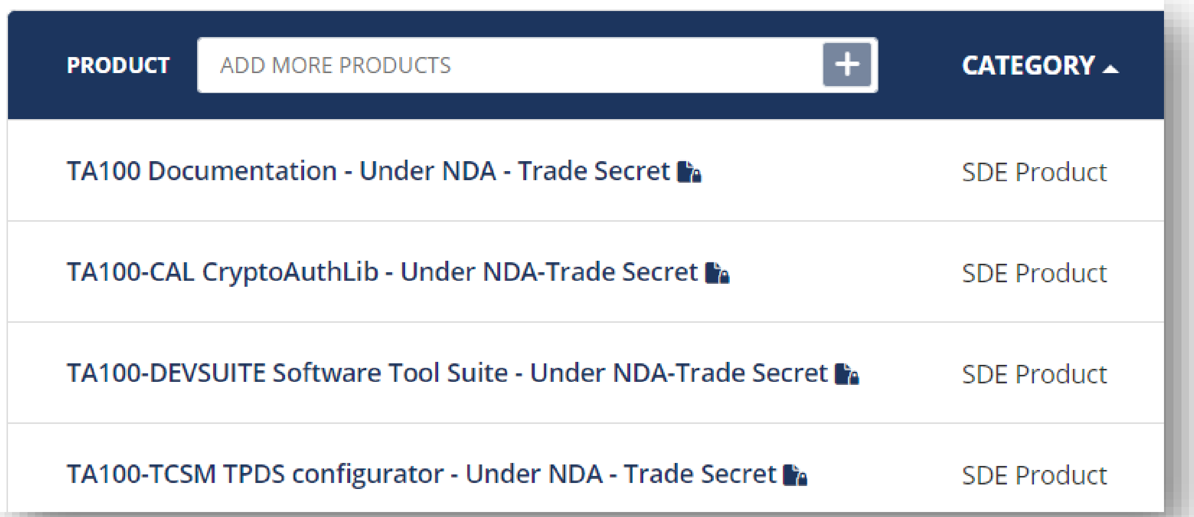
Hands-on materials preparation

Pre-Work

TA100 Hands-on Lab

Documentation

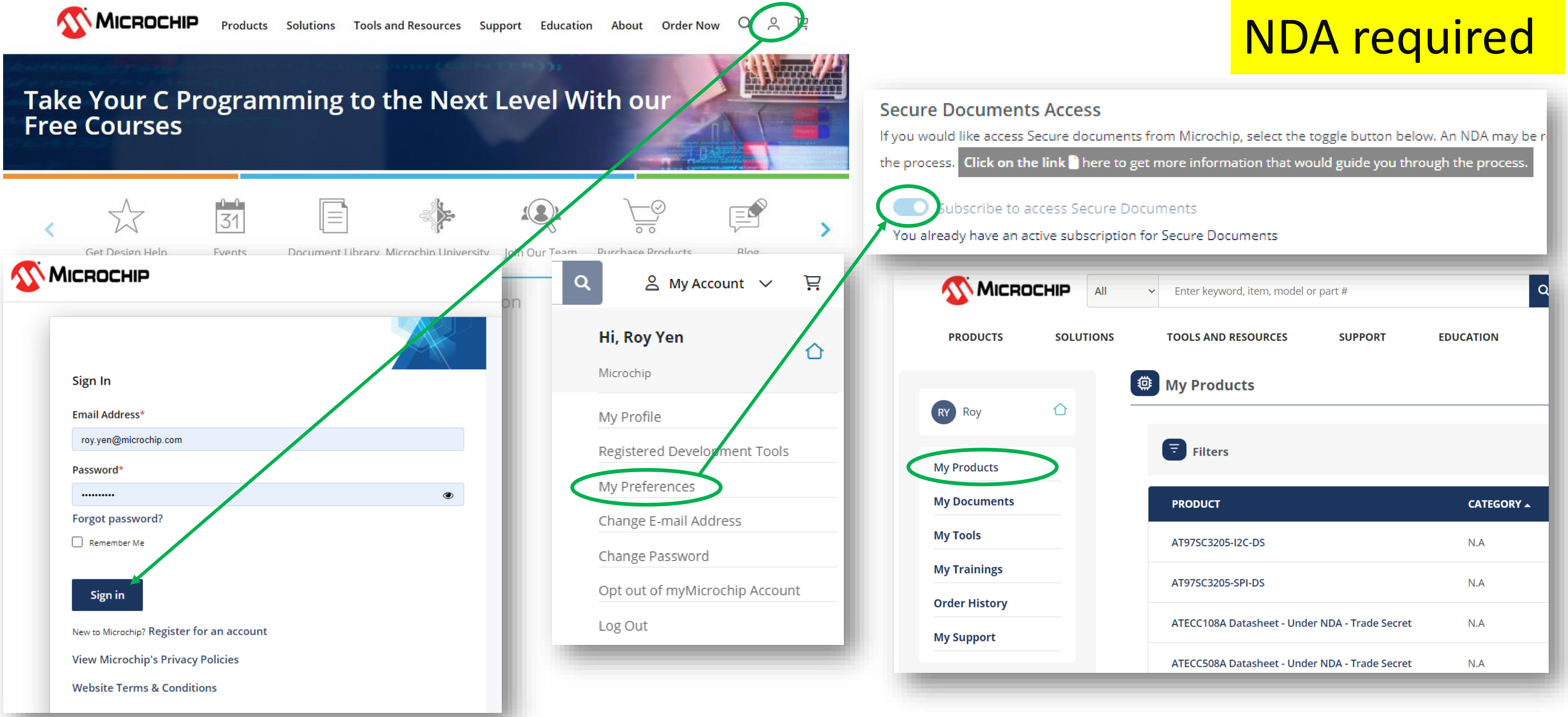
- **Documents and tools are available ONLY UNDER NDA**
  - Under MyMicrochip for MCHP and for Customers
- **TA100 Datasheet**
- **TA100 Programming Specification**



PRODUCT	CATEGORY
TA100 Documentation - Under NDA - Trade Secret	SDE Product
TA100-CAL CryptoAuthLib - Under NDA-Trade Secret	SDE Product
TA100-DEVSUITE Software Tool Suite - Under NDA-Trade Secret	SDE Product
TA100-TCSM TPDS configurator - Under NDA - Trade Secret	SDE Product

Download Documents from MyMicrochip SDE

NDA required



Take Your C Programming to the Next Level With our Free Courses

MICROCHIP

Sign In

Email Address*: roy.yen@microchip.com

Password*: ••••••••••

Forgot password?

□ Remember Me

Sign in

New to Microchip? Register for an account

View Microchip's Privacy Policies

Website Terms & Conditions

My Account

Hi, Roy Yen

Microchip

- My Profile
- Registered Development Tools
- My Preferences
- Change E-mail Address
- Change Password
- Opt out of myMicrochip Account
- Log Out

Secure Documents Access

If you would like access Secure documents from Microchip, select the toggle button below. An NDA may be r the process. Click on the link here to get more information that would guide you through the process.

Subscribe to access Secure Documents

You already have an active subscription for Secure Documents

- My Products
- My Documents
- My Tools
- My Trainings
- Order History
- My Support

My Products

Filters

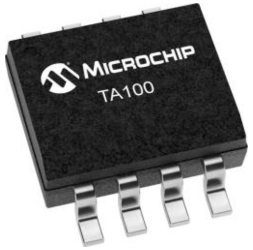
PRODUCT	CATEGORY ▲
AT97SC3205-I2C-DS	N.A
AT97SC3205-SPI-DS	N.A
ATECC108A Datasheet - Under NDA - Trade Secret	N.A
ATECC508A Datasheet - Under NDA - Trade Secret	N.A

Microchip Proprietary and Confidential

TA100 Hands-on Lab

HW Tools (RTC provides)

- **TA100 – TA100T-Y240C2X01-00B-VAO**
  - I2C communication

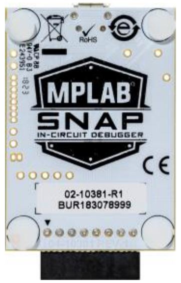


- **Socket Board - AC164167**
  - TA100 8-PIN SOIC CRYPTOAUTOMOTIVE(TM) SOCKET BOARD

- **Host MCU board – APP ALL MCU board**

  https://www.microchip.com.tw/uploads/tad_uploader/tmp/288/APP_All_MCU_2023_Dev_Resource.pdf
  - dsPIC33CH256MP505 + SNAP
  - or Pickit5

  https://www.microchip.com/en-us/development-tool/PG164150



Microchip Proprietary and STRICTLY Confidential

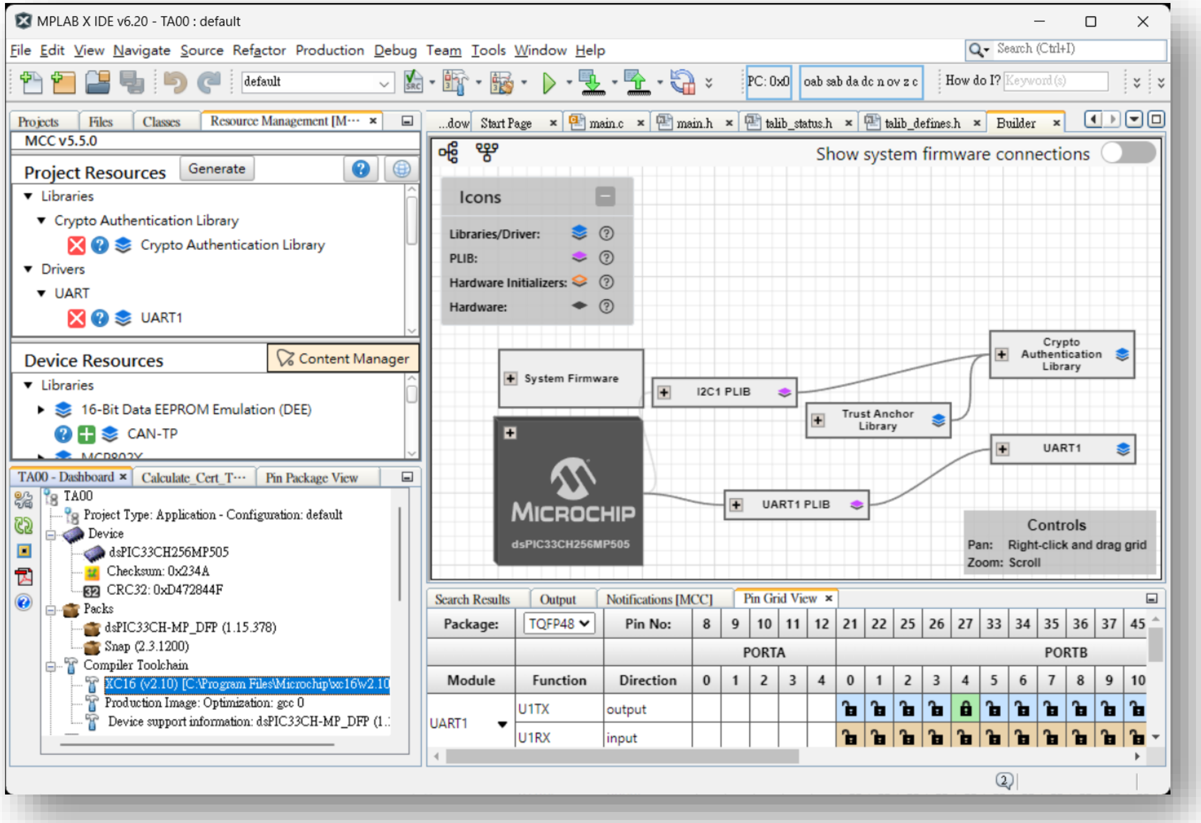
TA100 Hands-on Lab

SW Tools (Please install)

- **MPLABX IDE V6.20**
  - https://www.microchip.com/en-us/tools-resources/develop/mplab-x-ide
- **XC16 V2.10**
  - https://www.microchip.com/en-us/tools-resources/develop/mplab-xc-compilers/xc16
- **Cryptoauthlib** (refer to Lab2)
  - https://onlinedocs.microchip.com/pr/GUID-7F2639F3-1541-4BFC-A031-9A718BFFC502-en-US-16/index.html?GUID-B480AD4F-5342-4143-B7D9-76EED89D6045
- **RealTerm or TeraTerm**
- **Crypto Helper**

Tools_Share

You could download them here!
Password: MCHP



MCC v5.5.0

Project Resources — Generate

- Libraries
  - Crypto Authentication Library
    - Crypto Authentication Library
- Drivers
  - UART
    - UART1

Device Resources — Content Manager

- Libraries
  - 16-Bit Data EEPROM Emulation (DEE)
  - CAN-TP
  - MCP802X

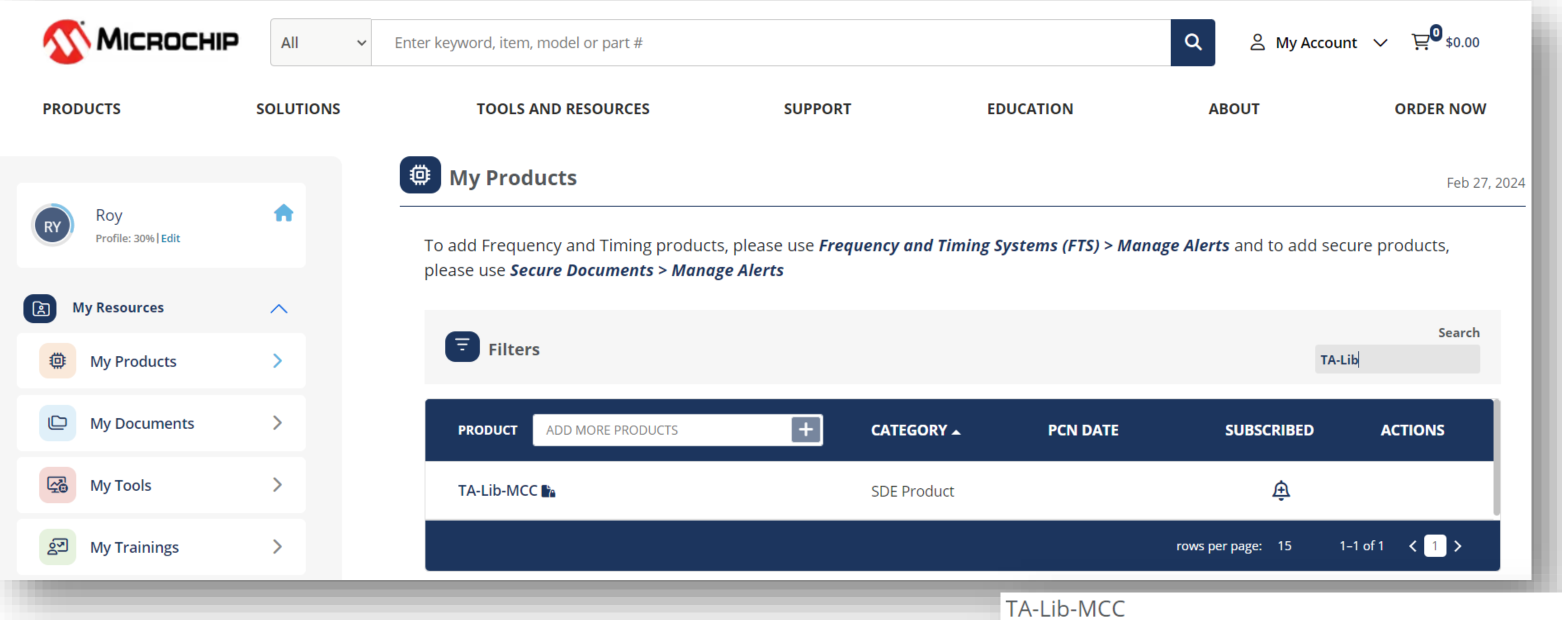
TA00 - Dashboard | Calculate_Cert_T··· | Pin Package View

```
TA00
  Project Type: Application - Configuration: default
  Device
    dsPIC33CH256MP505
    Checksum: 0x234A
    CRC32: 0xD472844F
  Packs
    dsPIC33CH-MP_DFP (1.15.378)
    Snap (2.3.1200)
  Compiler Toolchain
    XC16 (v2.10) [C:\Program Files\Microchip\xc16\v2.10
    Production Image: Optimization: gcc 0
    Device support information: dsPIC33CH-MP_DFP (1.:
```

Package:	TQFP48	Pin No:	8	9	10	11	12	21	22	25	26	27	33	34	35	36	37	45
			PORTA					PORTB										
Module	Function	Direction	0	1	2	3	4	0	1	2	3	4	5	6	7	8	9	10
UART1	U1TX	output						🔓	🔓	🔓	🔓	🔒	🔓	🔓	🔓	🔓	🔓	🔓
	U1RX	input						🔓	🔓	🔓	🔓	🔓	🔓	🔓	🔓	🔓	🔓	🔓

TA100 Hands-on Lab

TA100 lib for MCC download



My Products

Feb 27, 2024

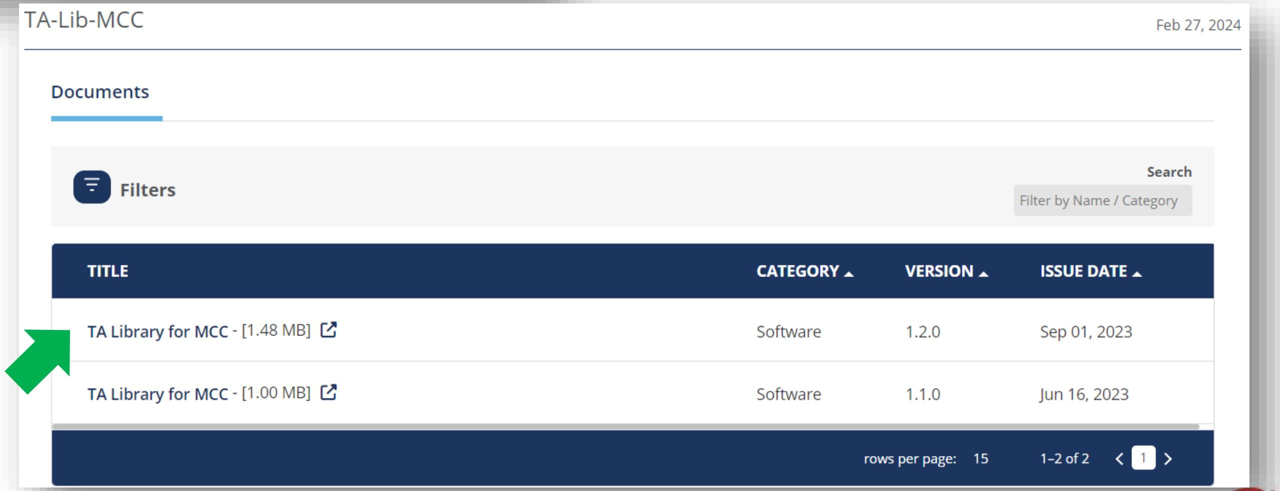
To add Frequency and Timing products, please use ***Frequency and Timing Systems (FTS) > Manage Alerts*** and to add secure products, please use ***Secure Documents > Manage Alerts***

Filters

Search: TA-Lib

PRODUCT	CATEGORY ▲	PCN DATE	SUBSCRIBED	ACTIONS
TA-Lib-MCC	SDE Product		🔔	

rows per page: 15 1-1 of 1 ‹ 1 ›



TA-Lib-MCC

Feb 27, 2024

Documents

Filters

Search: Filter by Name / Category

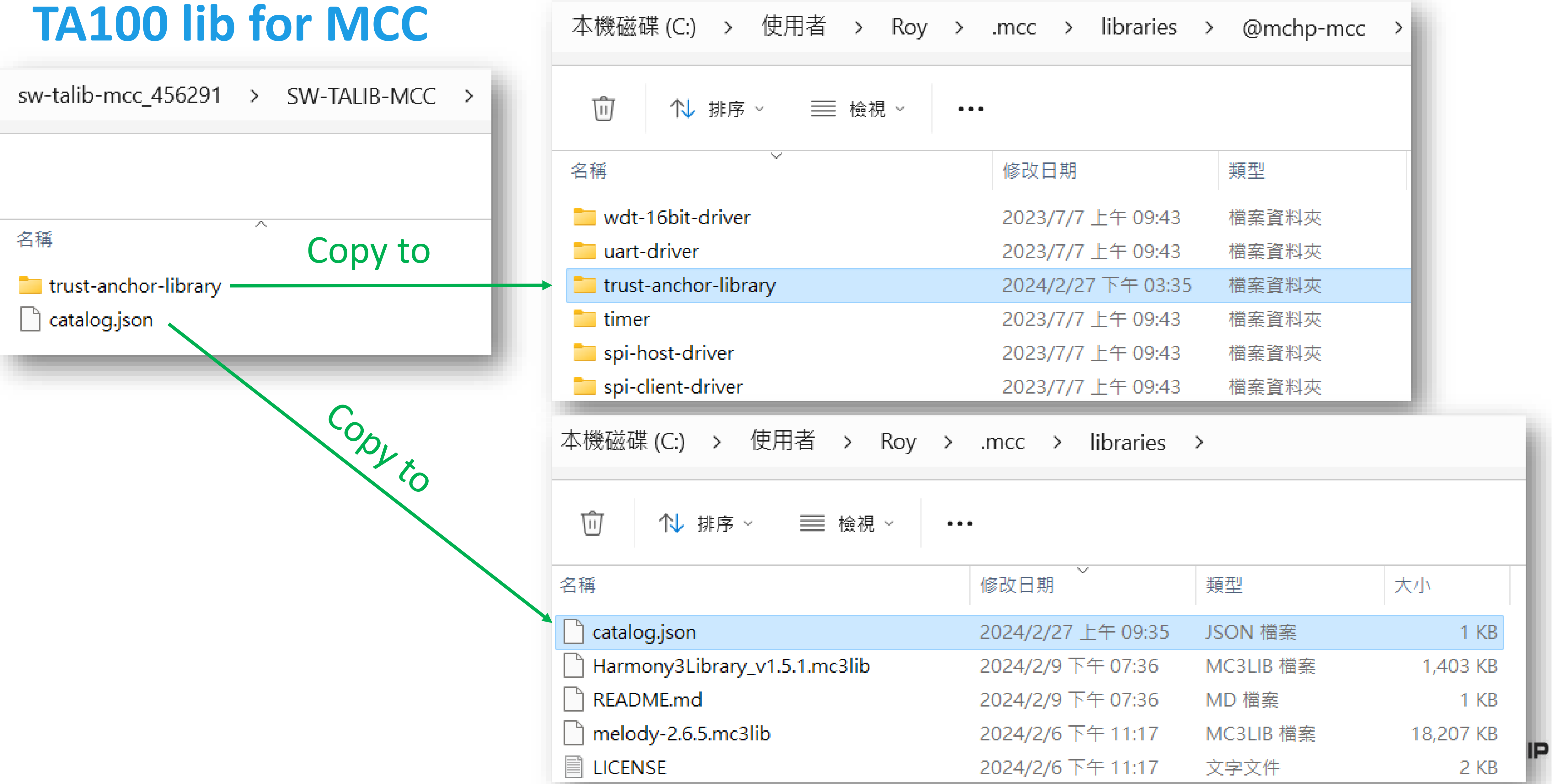
TITLE	CATEGORY ▲	VERSION ▲	ISSUE DATE ▲
TA Library for MCC - [1.48 MB]	Software	1.2.0	Sep 01, 2023
TA Library for MCC - [1.00 MB]	Software	1.1.0	Jun 16, 2023

rows per page: 15 1-2 of 2 ‹ 1 ›

Microchip Proprietary and STRICTLY Confidential

Copy TA100 lib Files to folder

TA100 lib for MCC



sw-talib-mcc_456291 > SW-TALIB-MCC >

名稱
trust-anchor-library
catalog.json

Copy to

Copy to

本機磁碟 (C:) > 使用者 > Roy > .mcc > libraries > @mchp-mcc >

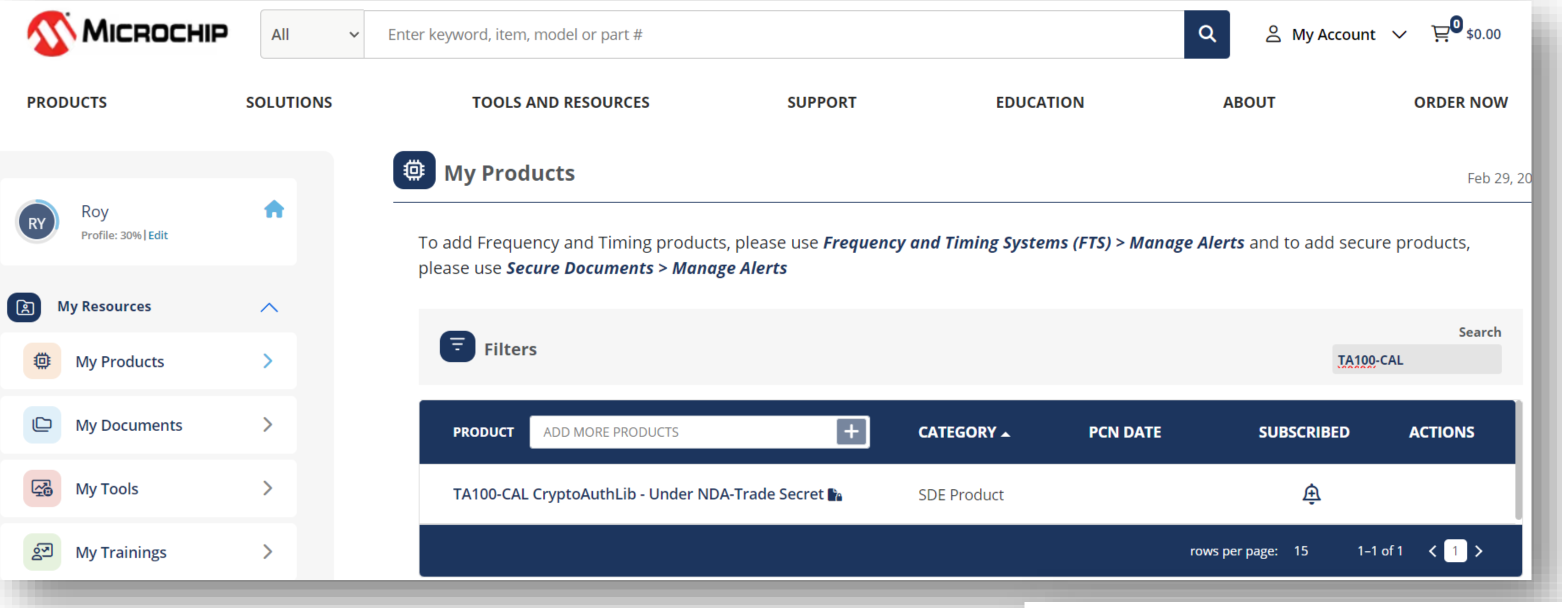
名稱	修改日期	類型
wdt-16bit-driver	2023/7/7 上午 09:43	檔案資料夾
uart-driver	2023/7/7 上午 09:43	檔案資料夾
trust-anchor-library	2024/2/27 下午 03:35	檔案資料夾
timer	2023/7/7 上午 09:43	檔案資料夾
spi-host-driver	2023/7/7 上午 09:43	檔案資料夾
spi-client-driver	2023/7/7 上午 09:43	檔案資料夾

本機磁碟 (C:) > 使用者 > Roy > .mcc > libraries >

名稱	修改日期	類型	大小
catalog.json	2024/2/27 上午 09:35	JSON 檔案	1 KB
Harmony3Library_v1.5.1.mc3lib	2024/2/9 下午 07:36	MC3LIB 檔案	1,403 KB
README.md	2024/2/9 下午 07:36	MD 檔案	1 KB
melody-2.6.5.mc3lib	2024/2/6 下午 11:17	MC3LIB 檔案	18,207 KB
LICENSE	2024/2/6 下午 11:17	文字文件	2 KB

TA100 Hands-on Lab (option)

TA100 lib for Harmony download



All | Enter keyword, item, model or part # | My Account | 0 $0.00

PRODUCTS SOLUTIONS TOOLS AND RESOURCES SUPPORT EDUCATION ABOUT ORDER NOW

Roy
Profile: 30% | Edit

My Resources
- My Products
- My Documents
- My Tools
- My Trainings

My Products

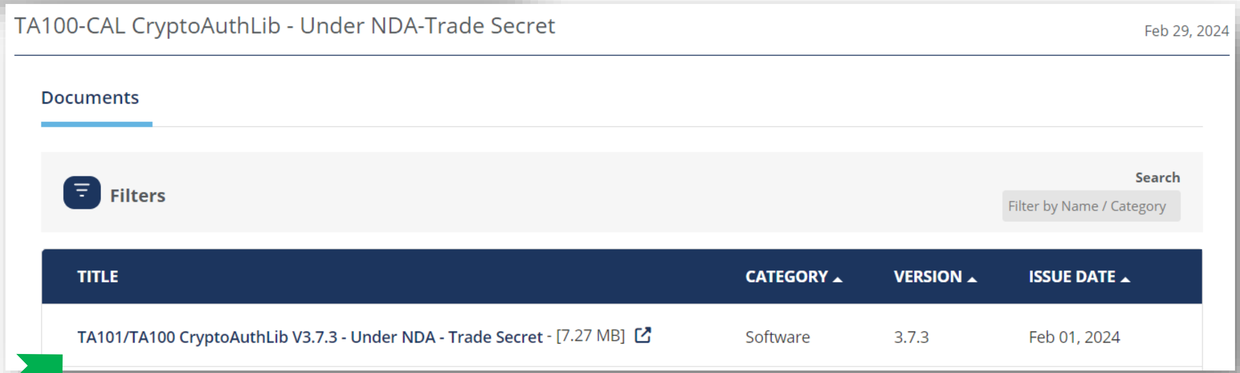
Feb 29, 20

To add Frequency and Timing products, please use ***Frequency and Timing Systems (FTS) > Manage Alerts*** and to add secure products, please use ***Secure Documents > Manage Alerts***

Filters — Search: TA100-CAL

PRODUCT (ADD MORE PRODUCTS)	CATEGORY ▲	PCN DATE	SUBSCRIBED	ACTIONS
TA100-CAL CryptoAuthLib - Under NDA-Trade Secret	SDE Product		🔔	

rows per page: 15 1–1 of 1 ‹ 1 ›



TA100-CAL CryptoAuthLib - Under NDA-Trade Secret

Feb 29, 2024

Documents

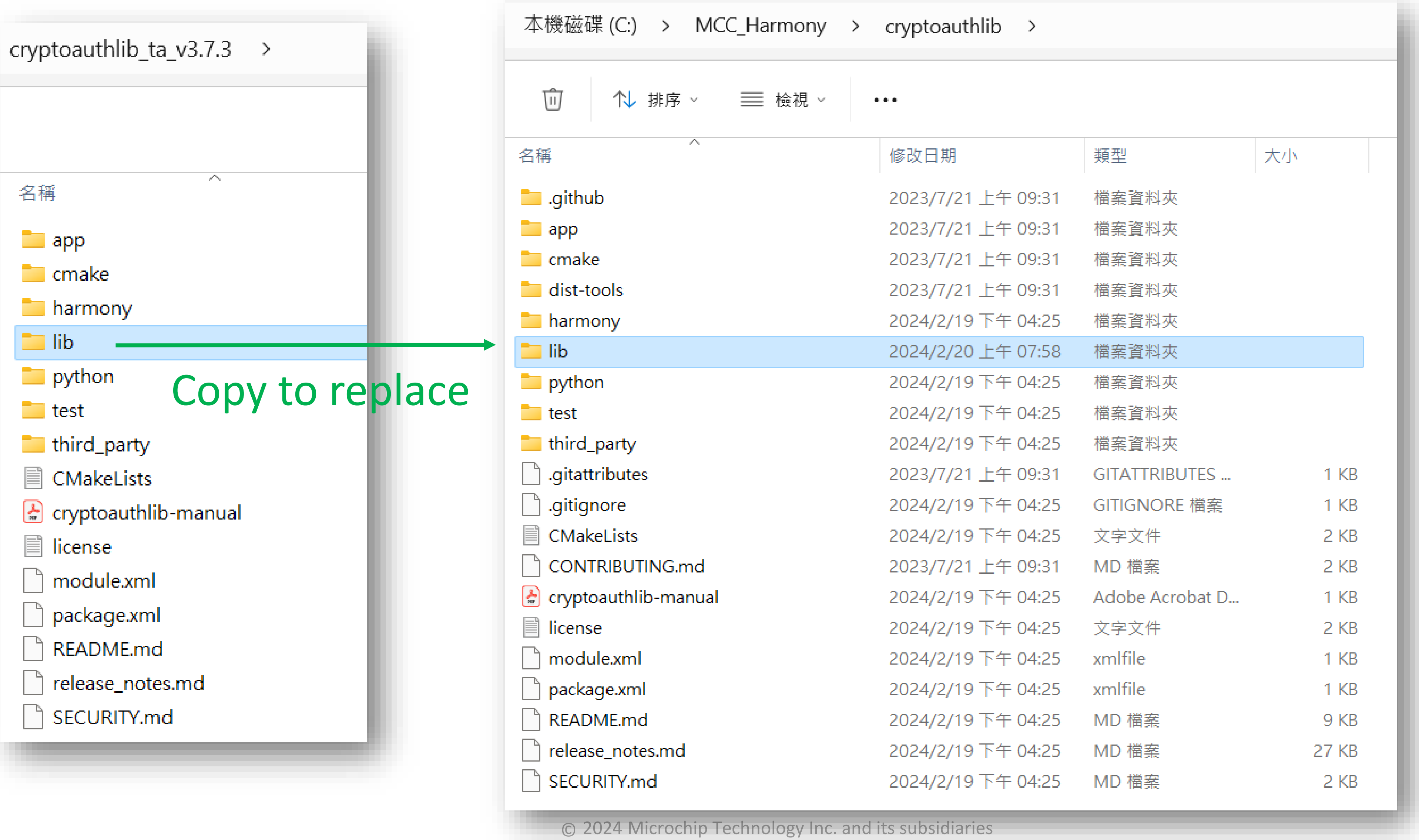
Filters — Search: Filter by Name / Category

TITLE	CATEGORY ▲	VERSION ▲	ISSUE DATE ▲
TA101/TA100 CryptoAuthLib V3.7.3 - Under NDA - Trade Secret - [7.27 MB]	Software	3.7.3	Feb 01, 2024

Microchip Proprietary and STRICTLY Confidential

Copy TA100 lib Files to folder (option)

TA100 lib for Harmony



cryptoauthlib_ta_v3.7.3 >

名稱
app
cmake
harmony
lib
python
test
third_party
CMakeLists
cryptoauthlib-manual
license
module.xml
package.xml
README.md
release_notes.md
SECURITY.md

Copy to replace

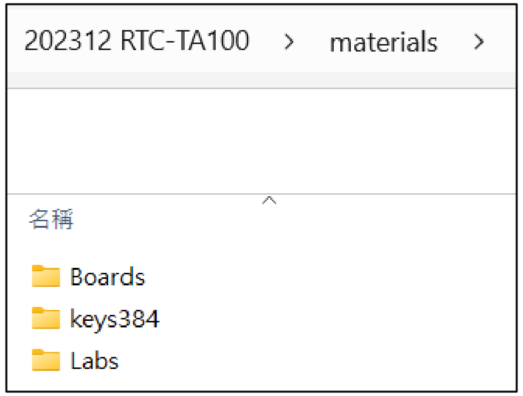
本機磁碟 (C:) > MCC_Harmony > cryptoauthlib >

名稱	修改日期	類型	大小
.github	2023/7/21 上午 09:31	檔案資料夾	
app	2023/7/21 上午 09:31	檔案資料夾	
cmake	2023/7/21 上午 09:31	檔案資料夾	
dist-tools	2023/7/21 上午 09:31	檔案資料夾	
harmony	2024/2/19 下午 04:25	檔案資料夾	
lib	2024/2/20 上午 07:58	檔案資料夾	
python	2024/2/19 下午 04:25	檔案資料夾	
test	2024/2/19 下午 04:25	檔案資料夾	
third_party	2024/2/19 下午 04:25	檔案資料夾	
.gitattributes	2023/7/21 上午 09:31	GITATTRIBUTES ...	1 KB
.gitignore	2024/2/19 下午 04:25	GITIGNORE 檔案	1 KB
CMakeLists	2024/2/19 下午 04:25	文字文件	2 KB
CONTRIBUTING.md	2023/7/21 上午 09:31	MD 檔案	2 KB
cryptoauthlib-manual	2024/2/19 下午 04:25	Adobe Acrobat D...	1 KB
license	2024/2/19 下午 04:25	文字文件	2 KB
module.xml	2024/2/19 下午 04:25	xmlfile	1 KB
package.xml	2024/2/19 下午 04:25	xmlfile	1 KB
README.md	2024/2/19 下午 04:25	MD 檔案	9 KB
release_notes.md	2024/2/19 下午 04:25	MD 檔案	27 KB
SECURITY.md	2024/2/19 下午 04:25	MD 檔案	2 KB

TA100 Hands-on Lab

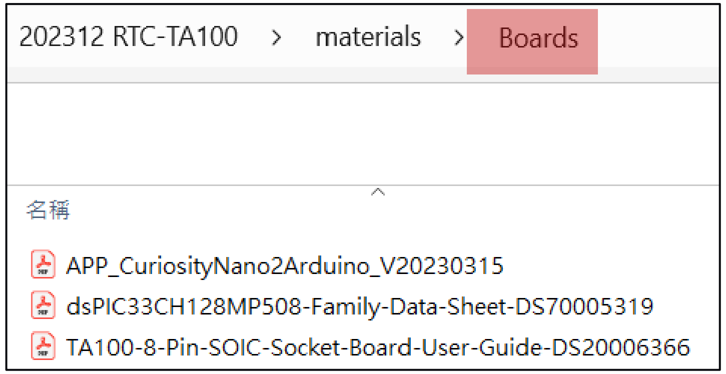
Lab files

- **Boards – APP All schematic (dsPIC33CH) & TA100 Socket UG**
- **Keys – Root/Signer/Device Certificates & Keys used in Labs**
- **Labs – All Labs/Practice Answers**


202312 RTC-TA100 > materials >

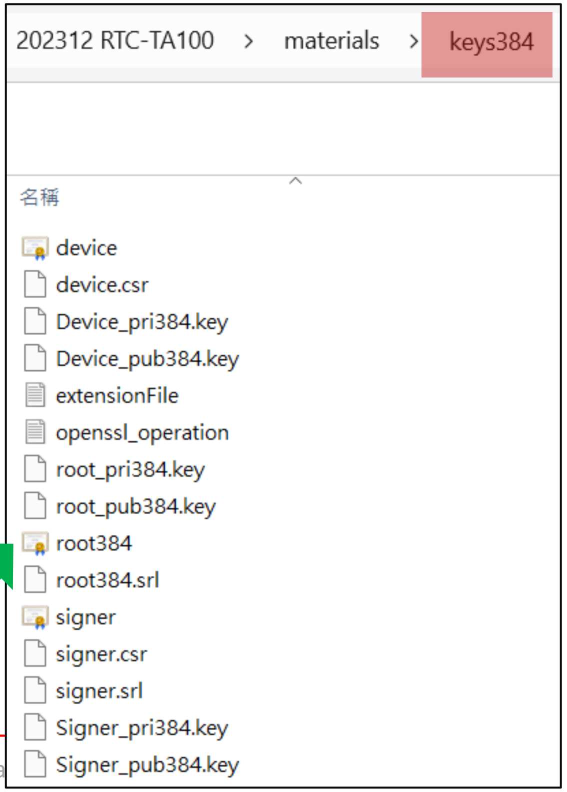
名稱

- Boards
- keys384
- Labs


202312 RTC-TA100 > materials > Boards

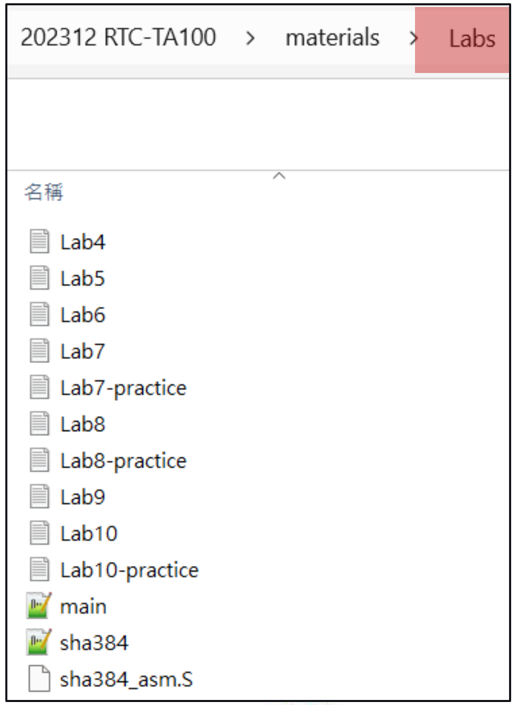
名稱

- APP_CuriosityNano2Arduino_V20230315
- dsPIC33CH128MP508-Family-Data-Sheet-DS70005319
- TA100-8-Pin-SOIC-Socket-Board-User-Guide-DS20006366


202312 RTC-TA100 > materials > keys384

名稱

- device
- device.csr
- Device_pri384.key
- Device_pub384.key
- extensionFile
- openssl_operation
- root_pri384.key
- root_pub384.key
- root384
- root384.srl
- signer
- signer.csr
- signer.srl
- Signer_pri384.key
- Signer_pub384.key


202312 RTC-TA100 > materials > Labs

名稱

- Lab4
- Lab5
- Lab6
- Lab7
- Lab7-practice
- Lab8
- Lab8-practice
- Lab9
- Lab10
- Lab10-practice
- main
- sha384
- sha384_asm.S

materials

You could download them here!
Password: MCHP

Use OpenSSL to pre-Generate Root/Signer/Device Keys&Certs for Lab usage

Microchip Proprietary and STRICT

TA100 basic Introduction

TA100

Command	Description
Cryptography	ECC-P224/P256/P384 and ECDSA sign/verify, ECDH-P256, ECBD-P224, SHA256 & HMAC AES128 encrypt/decrypt, Fast CMAC PRF/HKDF calculation for TLS1.2 & 1.3 RSA 2k KeyGen/Sign/Verify, RSA Verify (3k) RSA Encrypt/Decrypt (1k), bitcoin ECC curve, Brainpool
JIL resistance	High
EEPROM	11 kBytes, field upgradable
Counter	Yes
Serial Number	72 bits
RNG	NIST SP800-90 A/B/C
I/O	I2C, SPI
Supply Voltage	2.7V – 5.5V
Temperature	Automotive AECQ-100 grade 1 -40°C to 125°C
Certification	FIPS 140-2 module level 2, with physical protection level 3
Packages	SOIC8, VQFN-24

ECC608

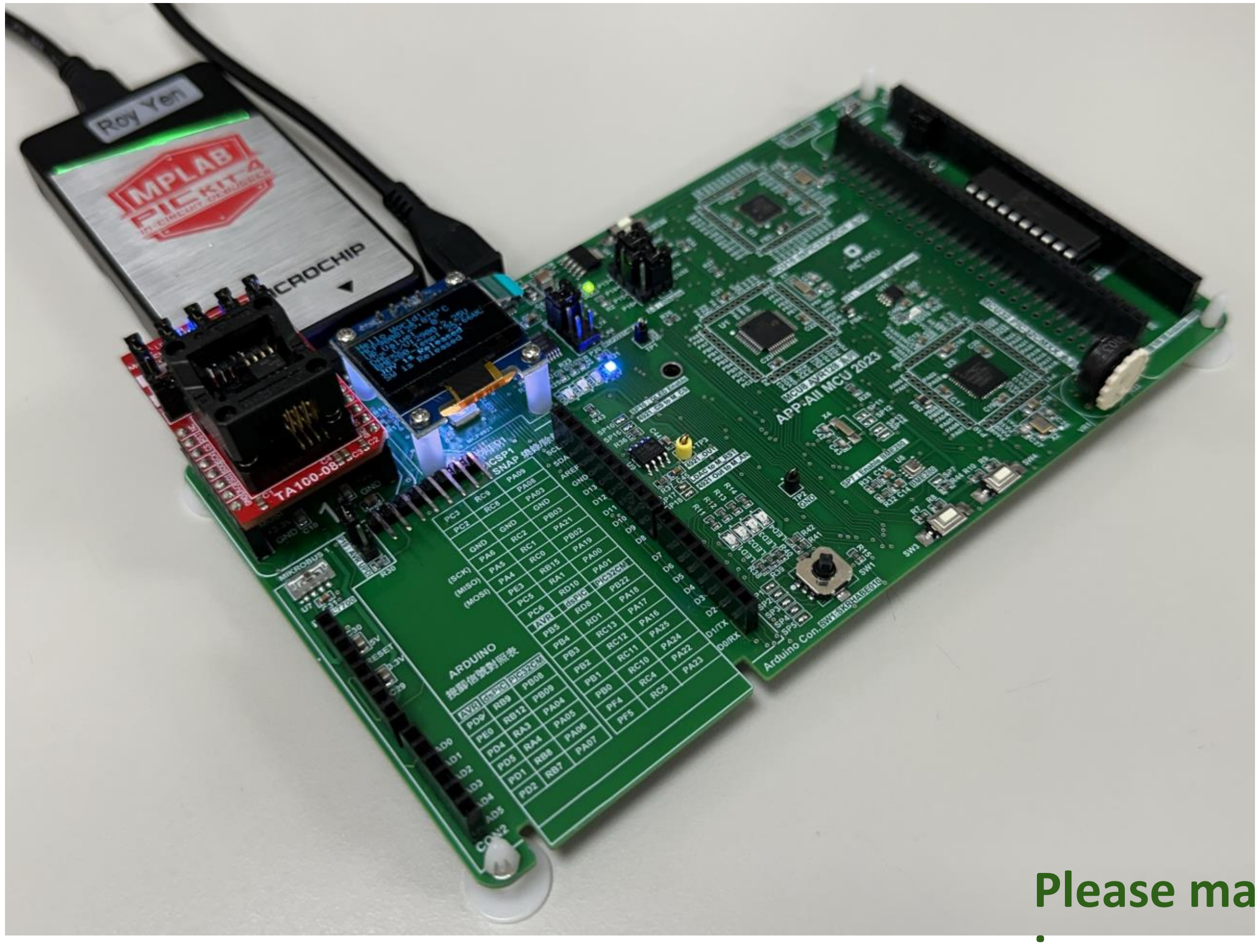
Command	Description
Cryptography	ECC-P256 and ECDSA sign/verify, ECDH SHA256 & HMAC AES128 encrypt/decrypt PRF/HKDF calculation for TLS1.2 & 1.3
JIL resistance	High
EEPROM	10kbits
Counter	Up to 2,000,000
Serial Number	72 bits
RNG	NIST SP800-90 A/B/C
I/O	I2C, SWI
Supply Voltage	2.0V – 5.5V
Temperature	-40°C to 85°C Extended temperature up to 100°C
Certification	FIPS 140-2 CAVP (algorithms) only
Packages	uDFN8, SOIC8, 3pin RBH, WCSP, die

Lab1 - Create TA100 project on dsPIC33CH

Using MCC Melody

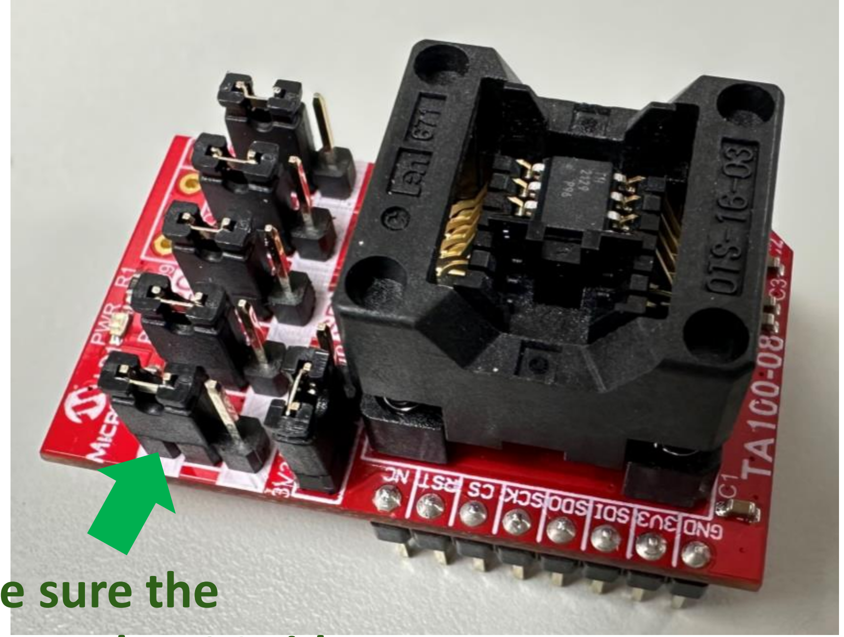
Make sure the connections (dsPIC33CH + TA100)

dsPIC33CH on APP ALL MCU + TA100 SOIC socket



APP ALL MCU - DSPIC33CH

AC164167 - TA100 8-PIN SOIC CRYPTOAUTOMOTIVE(TM) SOCKET BOARD



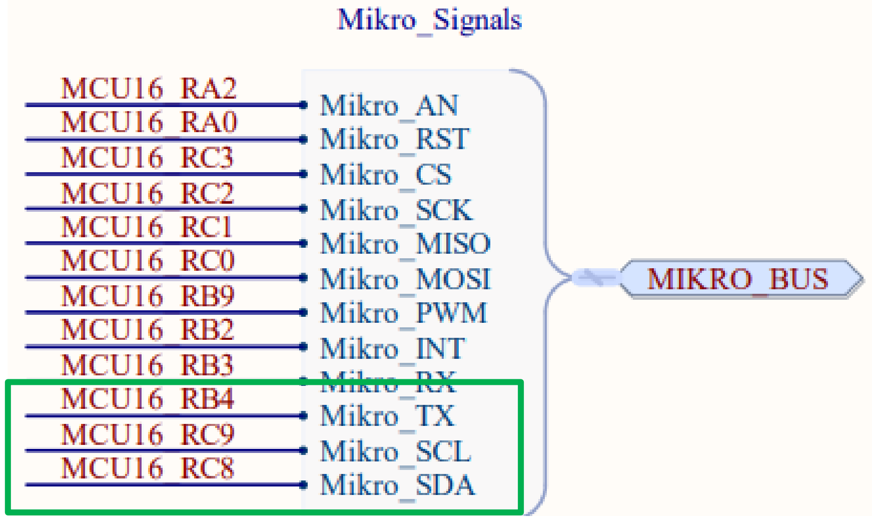
Please make sure the jumpers are on the I2C side

Board connections

Using APP All MCU - dsPIC33CH

- Make sure select using MCU16 ➔ **P15 jumper** on pin1 pin2
- Check I2C connections on schematic ➔ **RC8/RC9 (SDA/SCL)**
- Check UART coonections ➔ **RB4 (TX)**

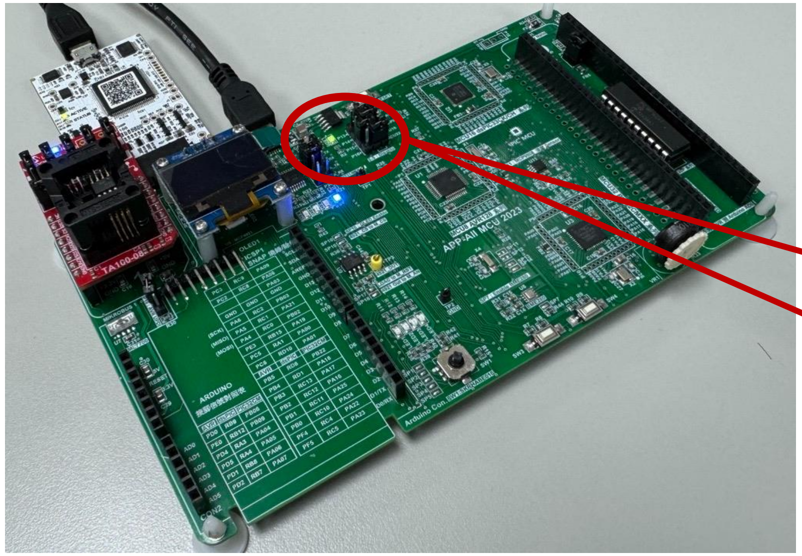
➔**P20 jumper** on M_TX & M_RX

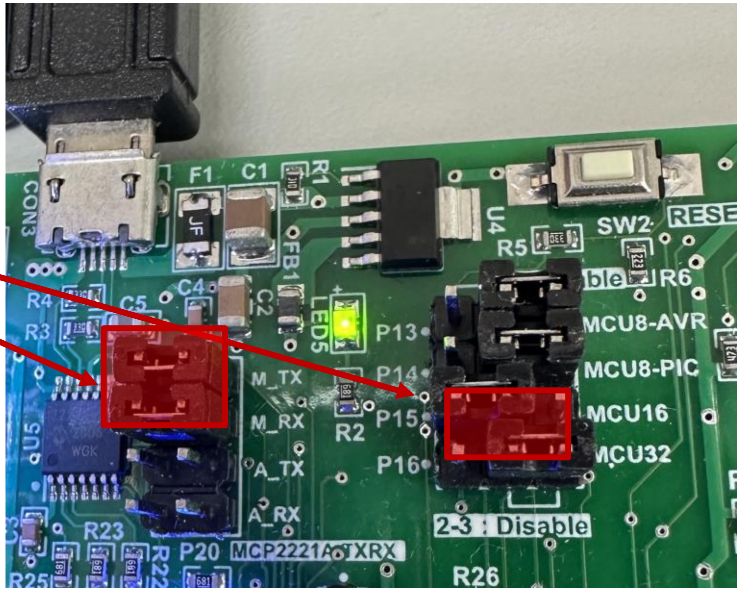


Mikro_Signals

MCU16_RA2	Mikro_AN
MCU16_RA0	Mikro_RST
MCU16_RC3	Mikro_CS
MCU16_RC2	Mikro_SCK
MCU16_RC1	Mikro_MISO
MCU16_RC0	Mikro_MOSI
MCU16_RB9	Mikro_PWM
MCU16_RB2	Mikro_INT
MCU16_RB3	Mikro_RX
MCU16_RB4	Mikro_TX
MCU16_RC9	Mikro_SCL
MCU16_RC8	Mikro_SDA

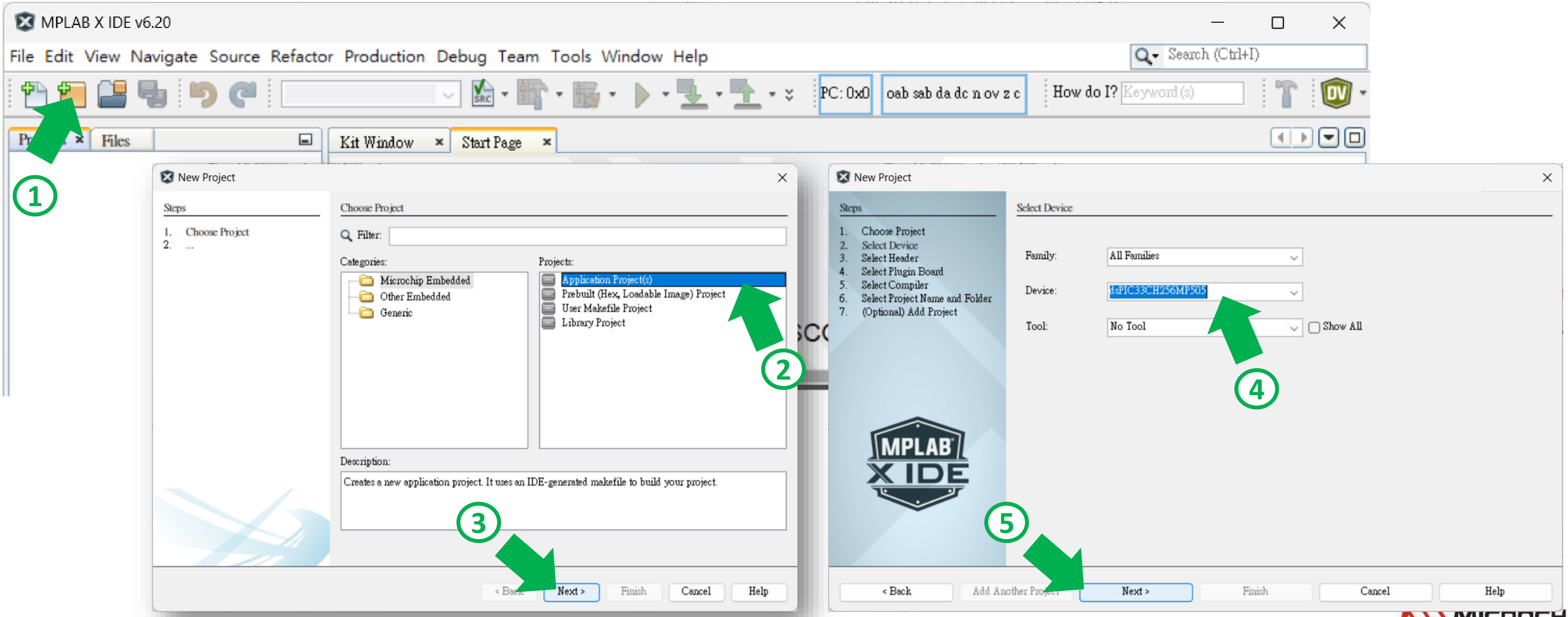
MIKRO_BUS





Step 1-1

- **Open MPLAB X IDE**
- **Create a new project using the dsPIC33CH256MP505 as the device**



New Project

Steps
1. Choose Project
2. ...

Choose Project

Filter:

Categories:
- Microchip Embedded
- Other Embedded
- Generic

Projects:
- Application Project(s)
- Prebuilt (Hex, Loadable Image) Project
- User Makefile Project
- Library Project

Description:
Creates a new application project. It uses an IDE-generated makefile to build your project.

New Project

Steps
1. Choose Project
2. Select Device
3. Select Header
4. Select Plugin Board
5. Select Compiler
6. Select Project Name and Folder
7. (Optional) Add Project

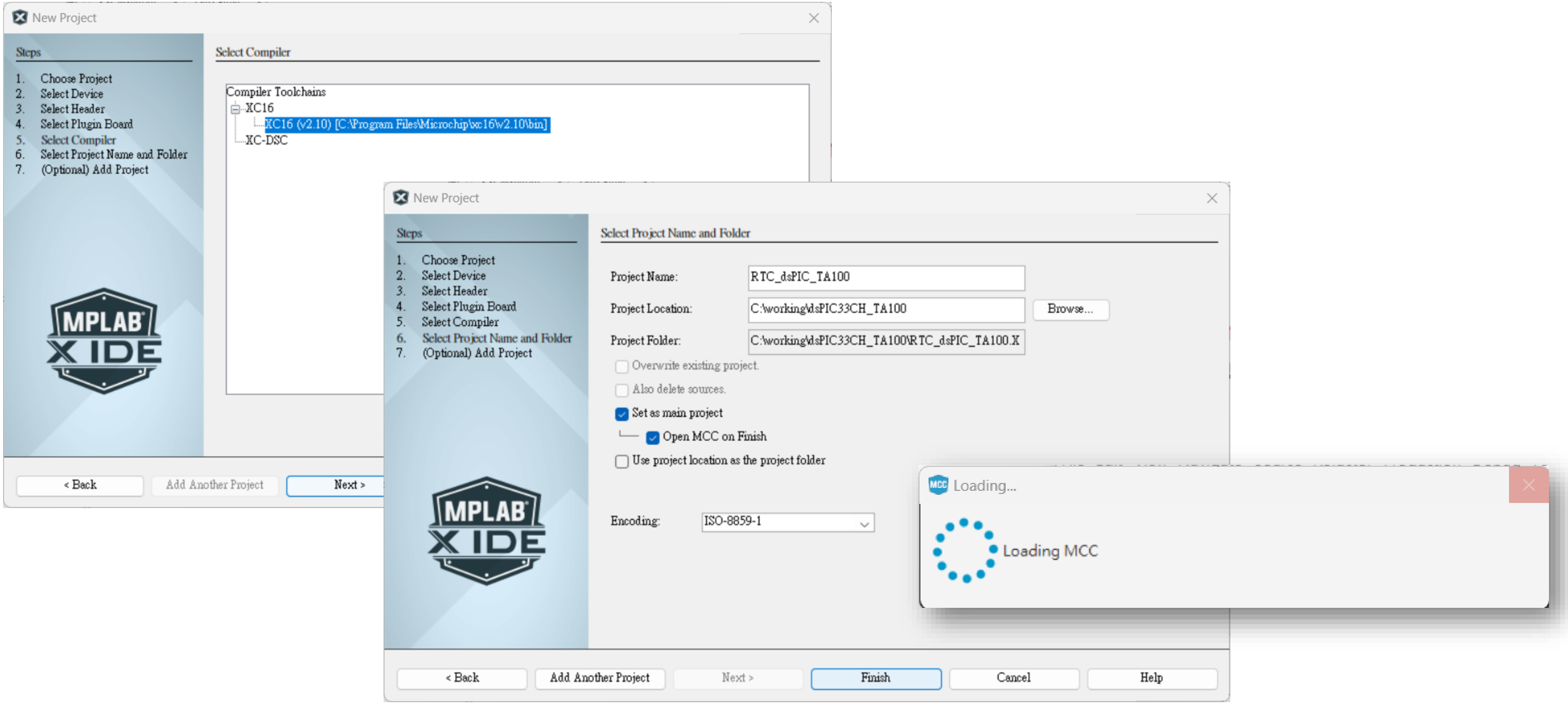
Select Device

Family: All Families

Device: dsPIC33CH256MP505

Tool: No Tool □ Show All

Step 1-1-2



New Project

Steps

1. Choose Project
2. Select Device
3. Select Header
4. Select Plugin Board
5. Select Compiler
6. Select Project Name and Folder
7. (Optional) Add Project

Select Compiler

Compiler Toolchains
- XC16
  - XC16 (v2.10) [C:\Program Files\Microchip\xc16\v2.10\bin]
- XC-DSC

New Project

Steps

1. Choose Project
2. Select Device
3. Select Header
4. Select Plugin Board
5. Select Compiler
6. Select Project Name and Folder
7. (Optional) Add Project

Select Project Name and Folder

Project Name: RTC_dsPIC_TA100

Project Location: C:\working\dsPIC33CH_TA100

Project Folder: C:\working\dsPIC33CH_TA100\RTC_dsPIC_TA100.X

□ Overwrite existing project.

□ Also delete sources.

☑ Set as main project

☑ Open MCC on Finish

□ Use project location as the project folder

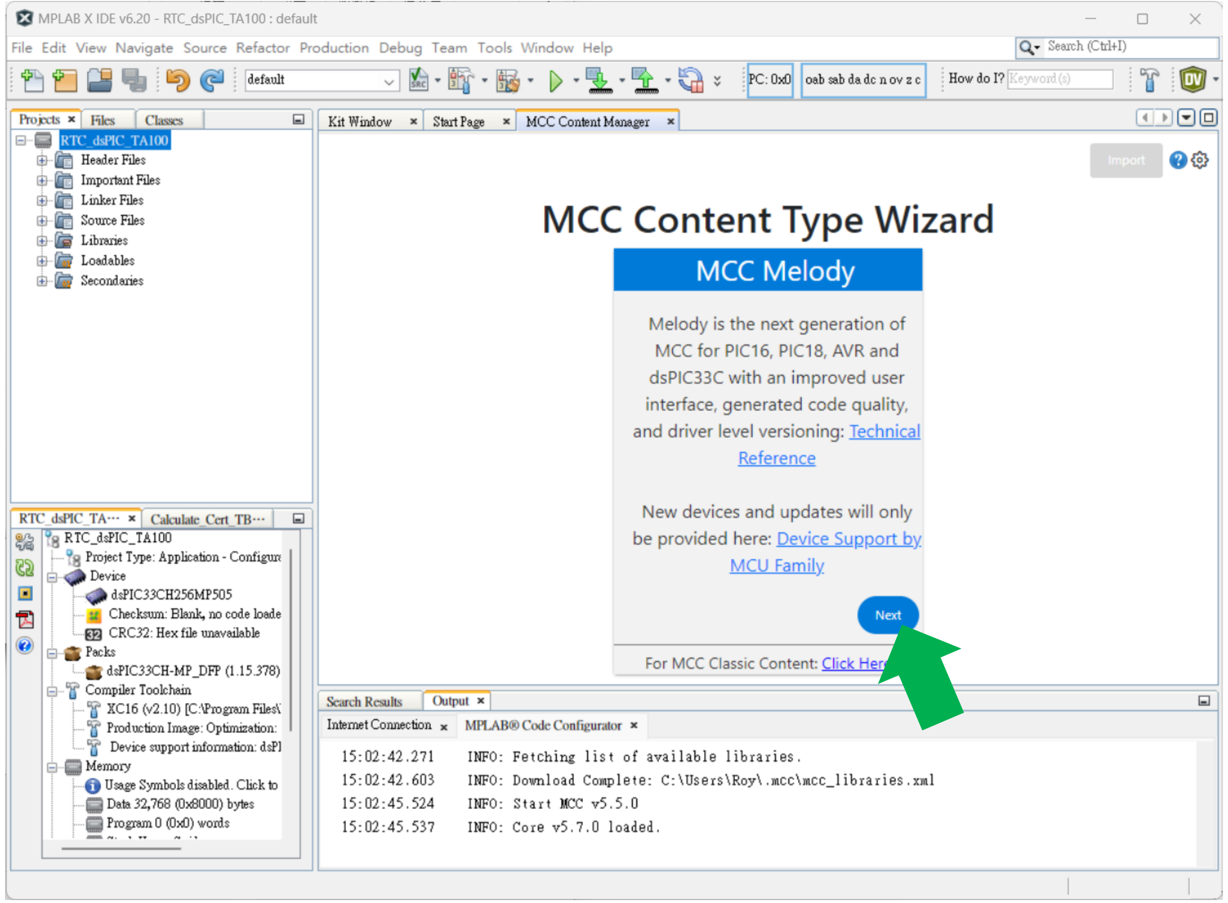
Encoding: ISO-8859-1

Loading...

Loading MCC

Step 1-1-3

- Open MCC by clicking the MCC button. (May Auto)
- Select Next to Run MCC Melody



RTC_dsPIC_TA100
- Header Files
- Important Files
- Linker Files
- Source Files
- Libraries
- Loadables
- Secondaries

RTC_dsPIC_TA100
- Project Type: Application - Configur
- Device
  - dsPIC33CH256MP505
  - Checksum: Blank, no code loade
  - CRC32: Hex file unavailable
- Packs
  - dsPIC33CH-MP_DFP (1.15.378)
- Compiler Toolchain
  - XC16 (v2.10) [C:\Program Files\
  - Production Image: Optimization:
  - Device support information: dsPI
- Memory
  - Usage Symbols disabled. Click to
  - Data 32,768 (0x8000) bytes
  - Program 0 (0x0) words

MCC Content Type Wizard

MCC Melody

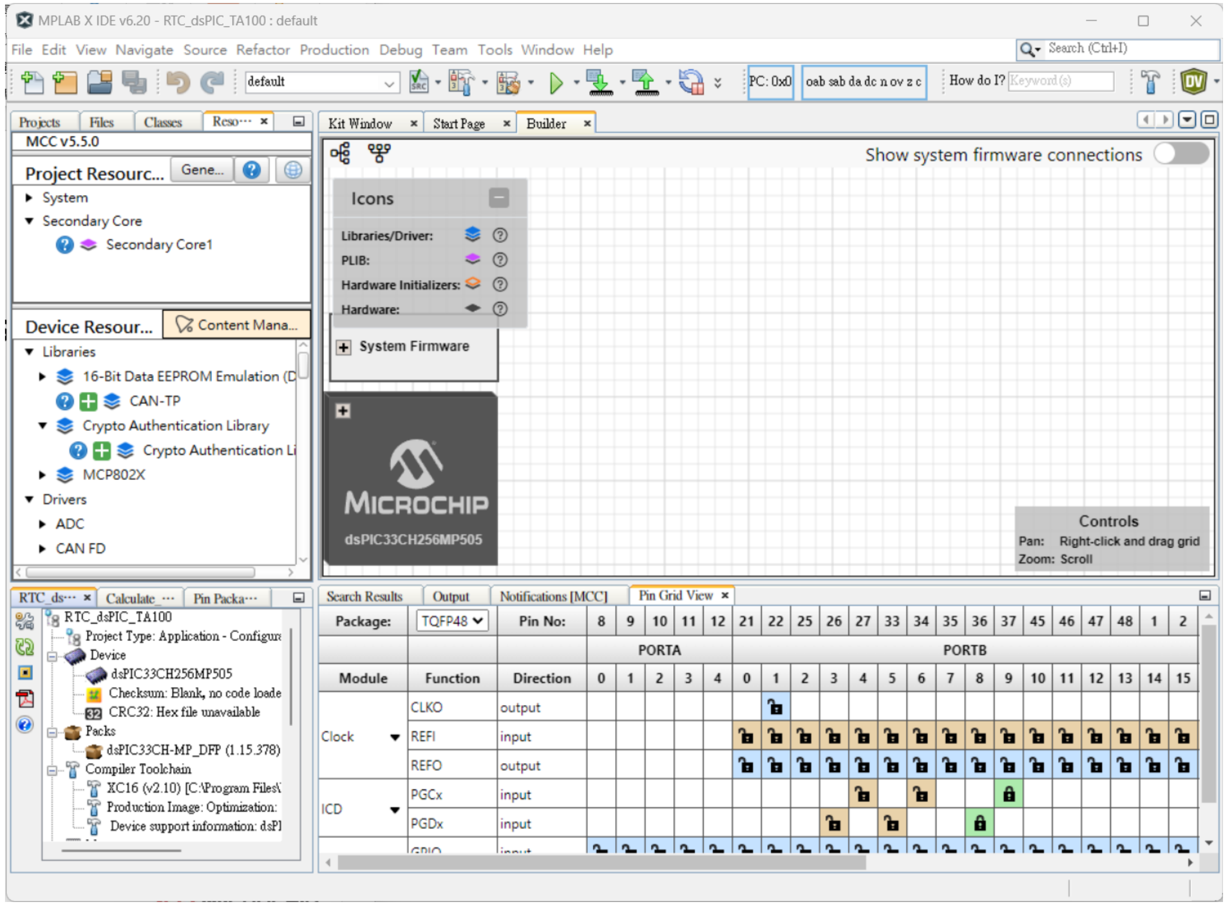
Melody is the next generation of MCC for PIC16, PIC18, AVR and dsPIC33C with an improved user interface, generated code quality, and driver level versioning: Technical Reference

New devices and updates will only be provided here: Device Support by MCU Family

Next

For MCC Classic Content: Click Here

```
15:02:42.271    INFO: Fetching list of available libraries.
15:02:42.603    INFO: Download Complete: C:\Users\Roy\.mcc\mcc_libraries.xml
15:02:45.524    INFO: Start MCC v5.5.0
15:02:45.537    INFO: Core v5.7.0 loaded.
```



MCC v5.5.0

Project Resourc... Gene...
- System
- Secondary Core
  - Secondary Core1

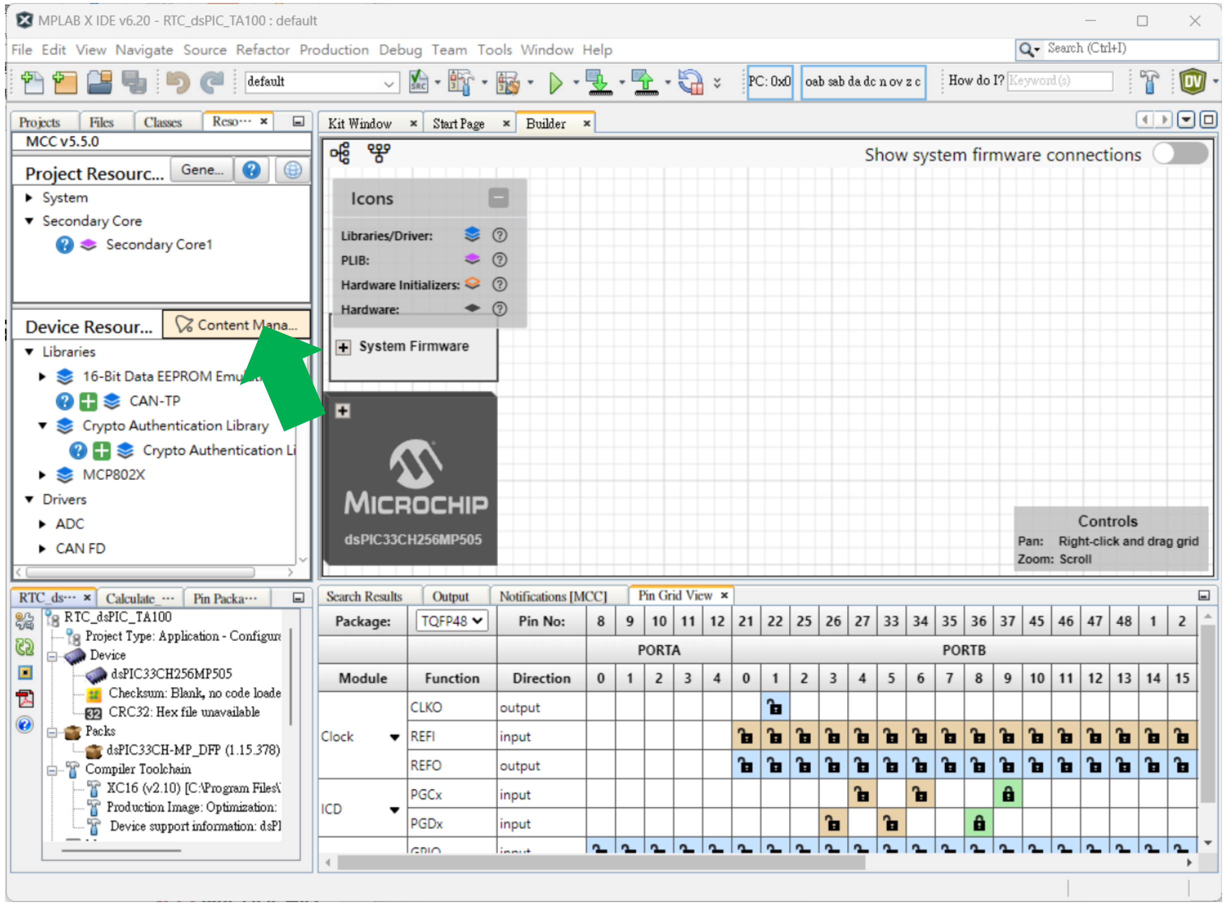
Device Resour... Content Mana...
- Libraries
  - 16-Bit Data EEPROM Emulation (D
  - CAN-TP
  - Crypto Authentication Library
    - Crypto Authentication Li
  - MCP802X
- Drivers
  - ADC
  - CAN FD

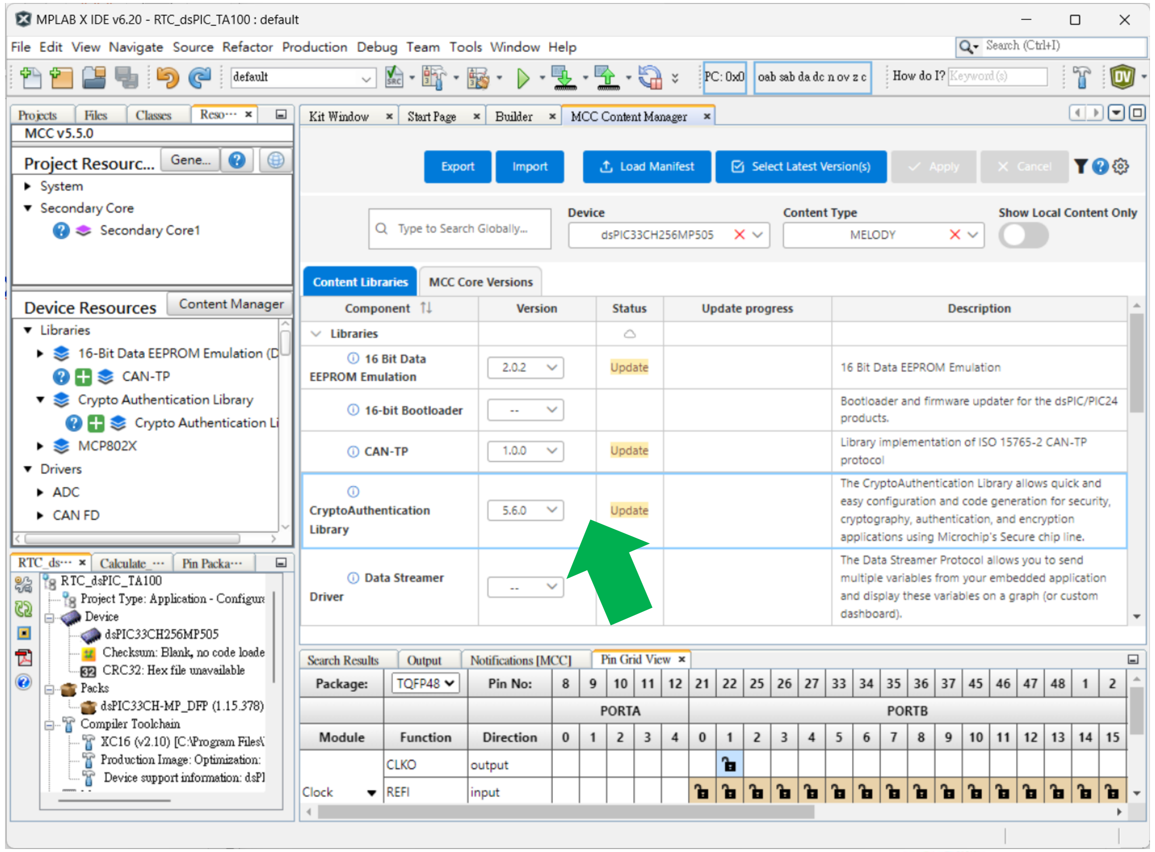
RTC_dsPIC_TA100
- Project Type: Application - Configur
- Device
  - dsPIC33CH256MP505
  - Checksum: Blank, no code loade
  - CRC32: Hex file unavailable
- Packs
  - dsPIC33CH-MP_DFP (1.15.378)
- Compiler Toolchain
  - XC16 (v2.10) [C:\Program Files\
  - Production Image: Optimization:
  - Device support information: dsPI

Package:	TQFP48	Pin No:	8	9	10	11	12	21	22	25	26	27	33	34	35	36	37	45	46	47	48	1	2
			PORTA					PORTB															
Module	Function	Direction	0	1	2	3	4	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Clock	CLKO	output							🔓														
	REFI	input						🔓	🔓	🔓	🔓	🔓	🔓	🔓	🔓	🔓	🔓	🔓	🔓	🔓	🔓	🔓	🔓
	REFO	output						🔓	🔓	🔓	🔓	🔓	🔓	🔓	🔓	🔓	🔓	🔓	🔓	🔓	🔓	🔓	🔓
ICD	PGCx	input										🔓		🔓			🔒						
	PGDx	input									🔓		🔓			🔒							
	GPIO	input	🔓	🔓	🔓	🔓	🔓	🔓	🔓	🔓	🔓	🔓	🔓	🔓	🔓	🔓	🔓	🔓	🔓	🔓	🔓	🔓	🔓

Step 1-2

- Click Content Manager to doble check the installed Libraries.
- Expand the “Libraries” category. Make sure “CryptoAuthentication Library” version is 5.6.0 or later. Click the “apply” button.

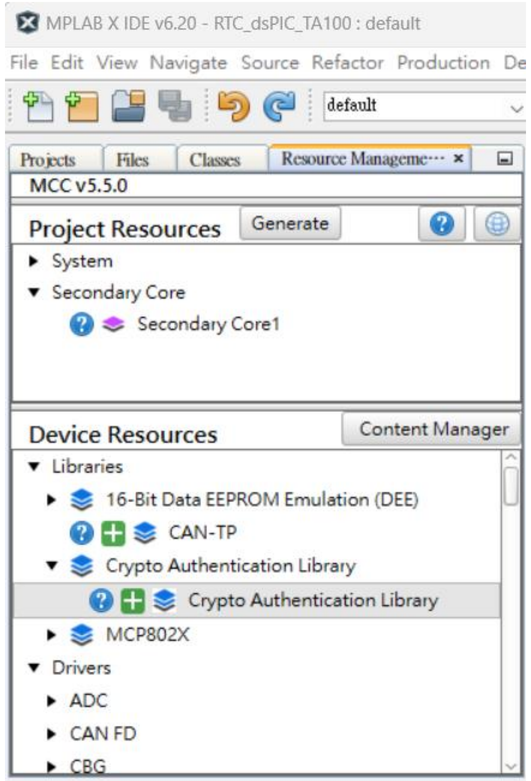


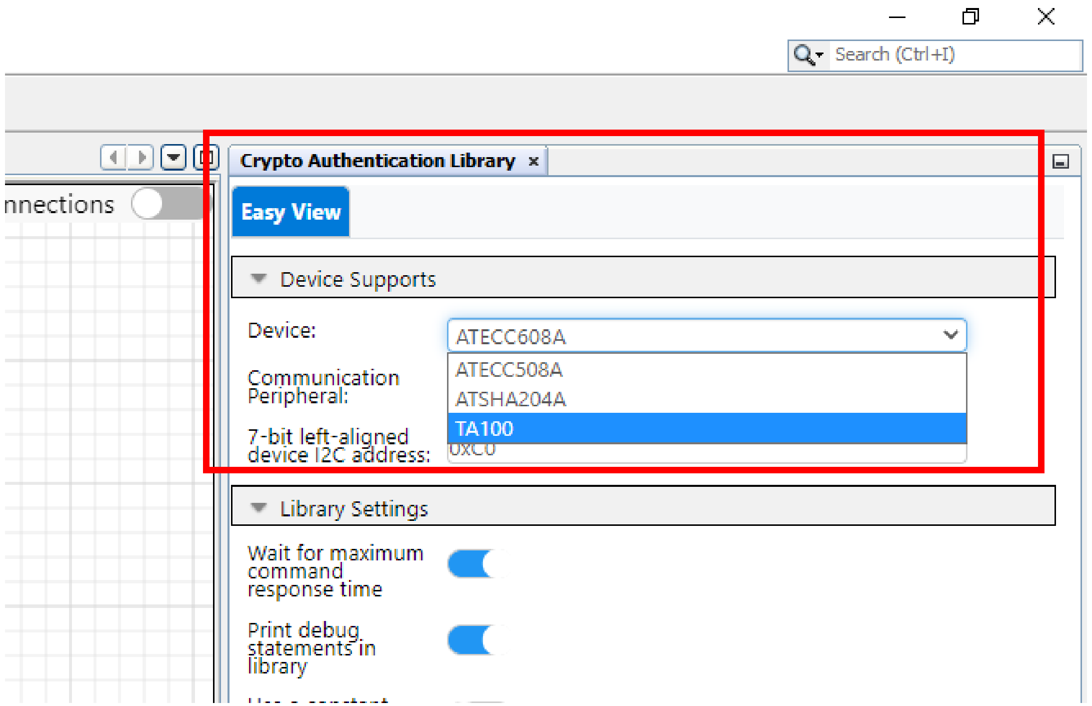


Component	Version	Status	Update progress	Description
Libraries				
16 Bit Data EEPROM Emulation	2.0.2	Update		16 Bit Data EEPROM Emulation
16-bit Bootloader	--			Bootloader and firmware updater for the dsPIC/PIC24 products.
CAN-TP	1.0.0	Update		Library implementation of ISO 15765-2 CAN-TP protocol
CryptoAuthentication Library	5.6.0	Update		The CryptoAuthentication Library allows quick and easy configuration and code generation for security, cryptography, authentication, and encryption applications using Microchip's Secure chip line.
Data Streamer Driver	--			The Data Streamer Protocol allows you to send multiple variables from your embedded application and display these variables on a graph (or custom dashboard).

Step 1-3

- In the “Device Resource” panel, add the CryptoAuthentication Library to the project by clicking the “+” sign next to the module.
- In the Crypto Authentication Library configuration easy view, select the TA100 from the device selection drop down:


MPLAB X IDE v6.20 - RTC_dsPIC_TA100 : default
File Edit View Navigate Source Refactor Production De
default
Projects Files Classes Resource Manageme···
MCC v5.5.0
Project Resources
Generate
System
Secondary Core
Secondary Core1
Device Resources
Content Manager
Libraries
16-Bit Data EEPROM Emulation (DEE)
CAN-TP
Crypto Authentication Library
Crypto Authentication Library
MCP802X
Drivers
ADC
CAN FD
CBG



Device: ATECC608A

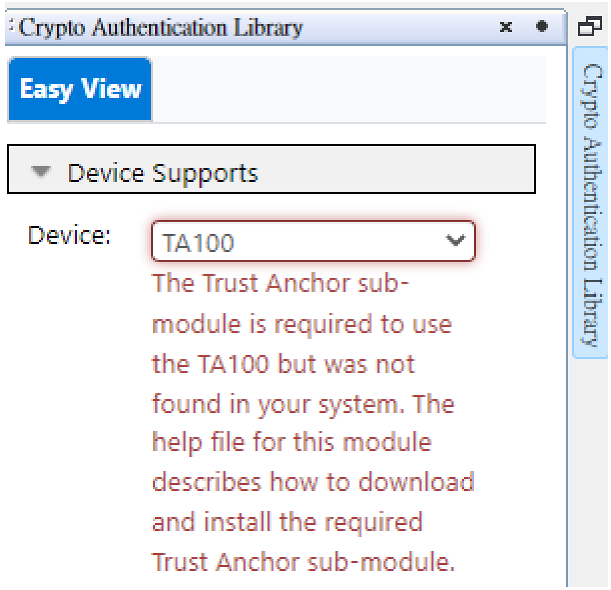
ATECC508A
ATSHA204A
TA100

Communication Peripheral:

7-bit left-aligned device I2C address: 0xC0

Wait for maximum command response time

Print debug statements in library



Device: TA100

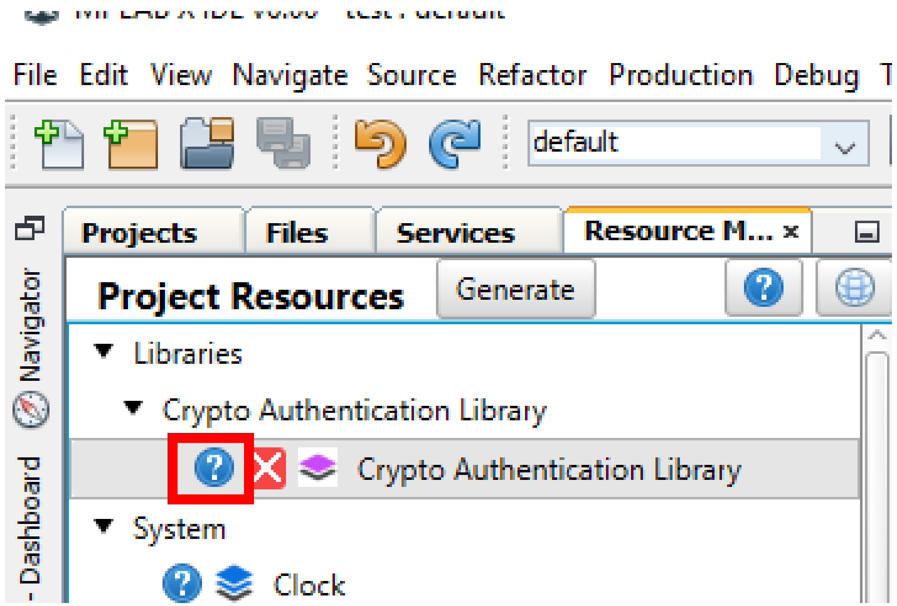
The Trust Anchor sub-module is required to use the TA100 but was not found in your system. The help file for this module describes how to download and install the required Trust Anchor sub-module.

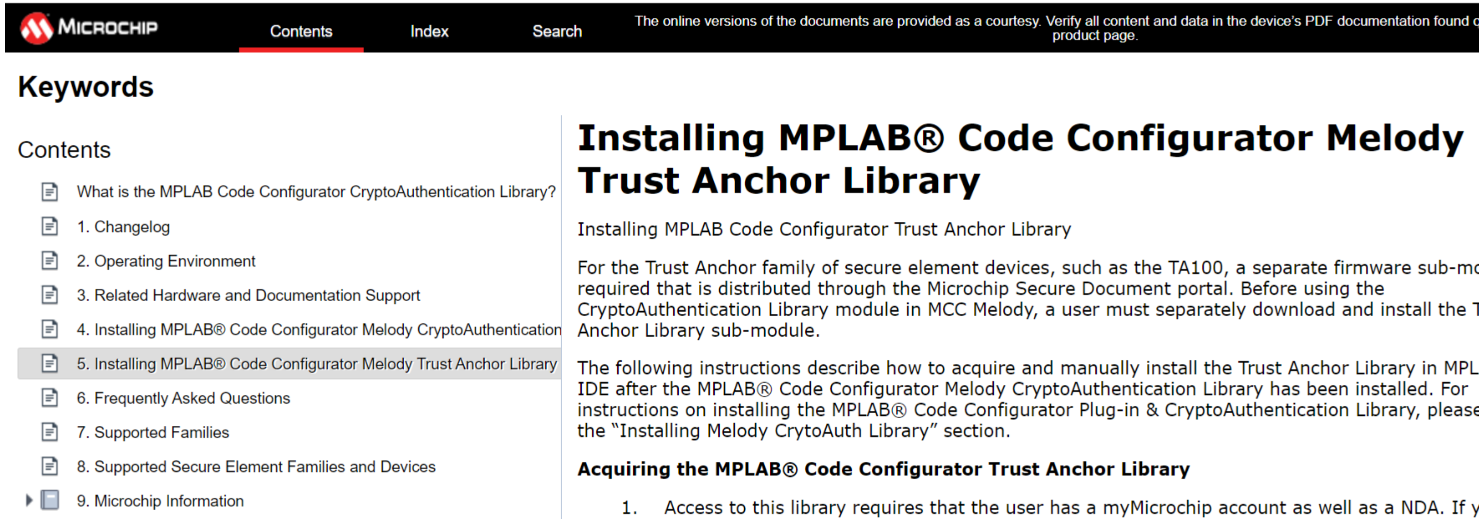
Lab2 - Installing Trust Anchor MCC SW Module

NDA & SDE is required

Step 2-1

- Open the help documentation for the Crypto Authentication module by clicking the “?” mark next to the module
- Click on the section “Installing MPLAB® Code Configurator Melody Trust Anchor Library” from the contents
- Follow the instruction on this page


File Edit View Navigate Source Refactor Production Debug T
default
Projects
Files
Services
Resource M...
Navigator
Dashboard
Project Resources
Generate
Libraries
Crypto Authentication Library
Crypto Authentication Library
System
Clock



The online versions of the documents are provided as a courtesy. Verify all content and data in the device's PDF documentation found o
product page.

Keywords

Contents

Installing MPLAB® Code Configurator Melody Trust Anchor Library

Installing MPLAB Code Configurator Trust Anchor Library

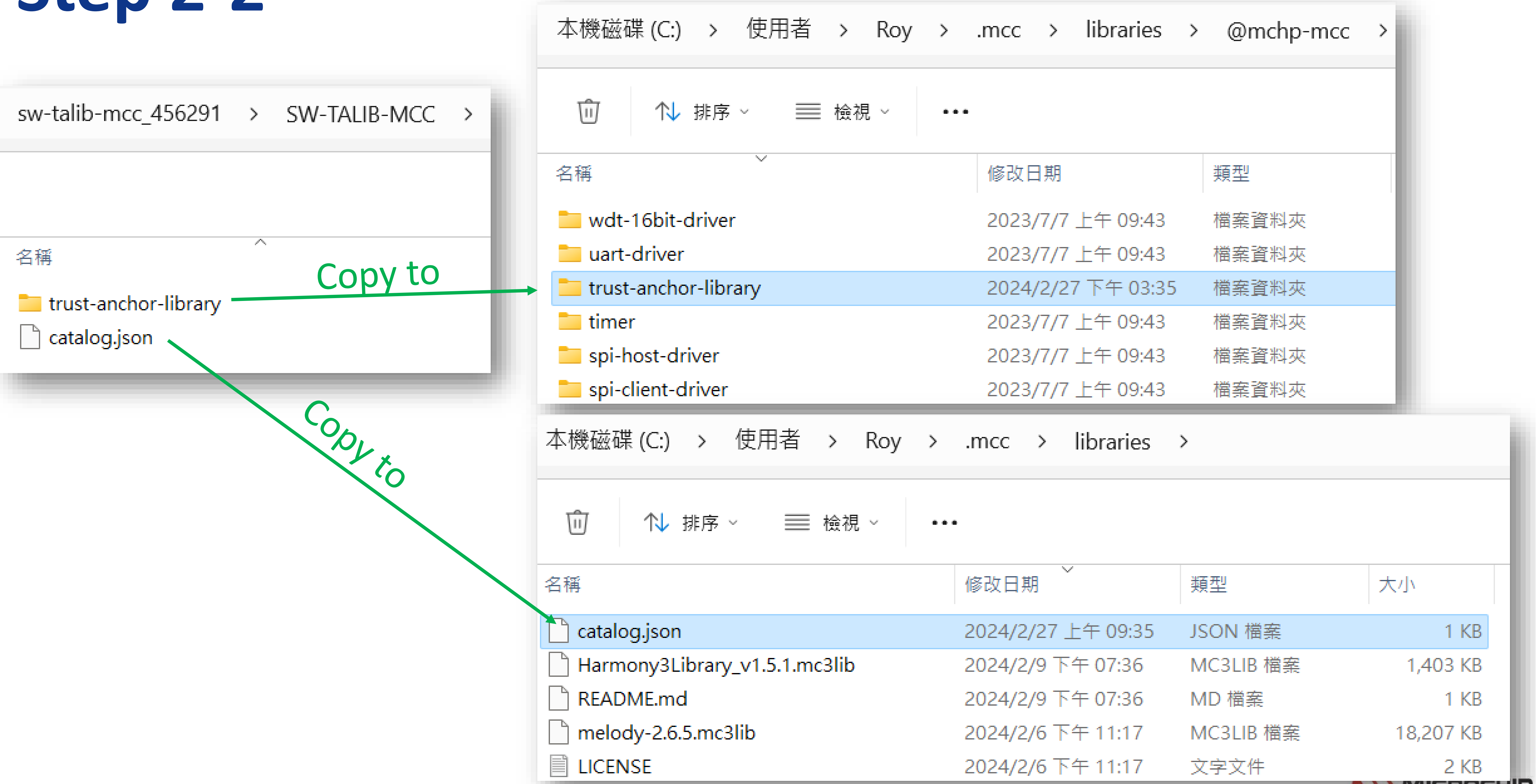
For the Trust Anchor family of secure element devices, such as the TA100, a separate firmware sub-mo
required that is distributed through the Microchip Secure Document portal. Before using the
CryptoAuthentication Library module in MCC Melody, a user must separately download and install the T
Anchor Library sub-module.

The following instructions describe how to acquire and manually install the Trust Anchor Library in MPL
IDE after the MPLAB® Code Configurator Melody CryptoAuthentication Library has been installed. For
instructions on installing the MPLAB® Code Configurator Plug-in & CryptoAuthentication Library, please
the “Installing Melody CrytoAuth Library” section.

Acquiring the MPLAB® Code Configurator Trust Anchor Library

1. Access to this library requires that the user has a myMicrochip account as well as a NDA. If y

Step 2-2



sw-talib-mcc_456291 > SW-TALIB-MCC >

名稱
trust-anchor-library
catalog.json

本機磁碟 (C:) > 使用者 > Roy > .mcc > libraries > @mchp-mcc >

名稱	修改日期	類型
wdt-16bit-driver	2023/7/7 上午 09:43	檔案資料夾
uart-driver	2023/7/7 上午 09:43	檔案資料夾
trust-anchor-library	2024/2/27 下午 03:35	檔案資料夾
timer	2023/7/7 上午 09:43	檔案資料夾
spi-host-driver	2023/7/7 上午 09:43	檔案資料夾
spi-client-driver	2023/7/7 上午 09:43	檔案資料夾

本機磁碟 (C:) > 使用者 > Roy > .mcc > libraries >

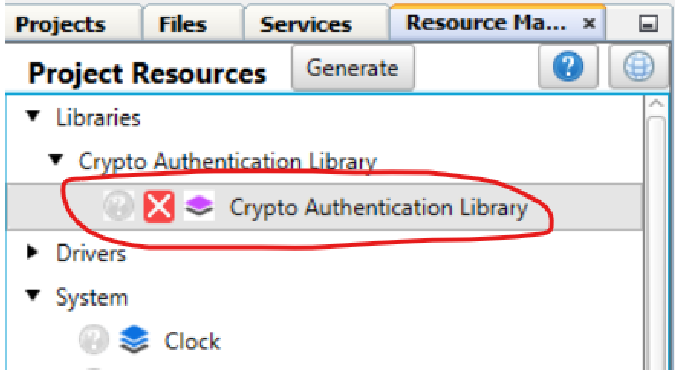
名稱	修改日期	類型	大小
catalog.json	2024/2/27 上午 09:35	JSON 檔案	1 KB
Harmony3Library_v1.5.1.mc3lib	2024/2/9 下午 07:36	MC3LIB 檔案	1,403 KB
README.md	2024/2/9 下午 07:36	MD 檔案	1 KB
melody-2.6.5.mc3lib	2024/2/6 下午 11:17	MC3LIB 檔案	18,207 KB
LICENSE	2024/2/6 下午 11:17	文字文件	2 KB

Lab3 - Generate dsPIC33CH code base

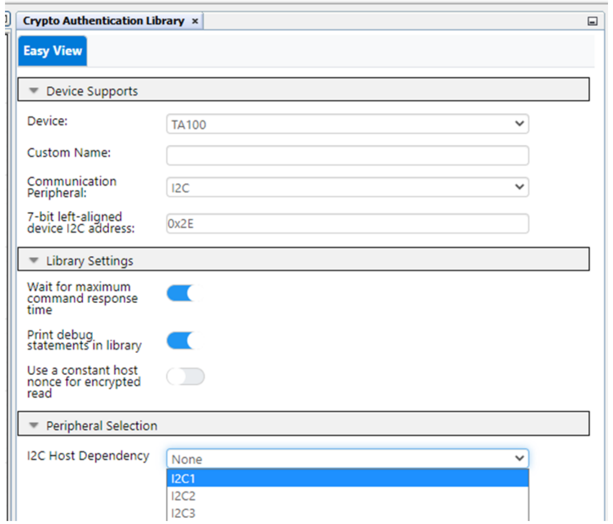
in MCC Melody

Step 3-1

- **Close & Re-open the project created in last step**
- **Open up MCC by clicking the MCC button**
- **Open the Crypto Authentication Library easy view by clicking the module in the project resources window:**
- **In the Crypto Authentication Easy view, select I2C1 from the selector.**
- **Choose RC8/RC9 as the I2C1 output pins**



- Libraries
  - Crypto Authentication Library
    - Crypto Authentication Library
- Drivers
- System
  - Clock



Device Supports

Device: TA100

Custom Name:

Communication Peripheral: I2C

7-bit left-aligned device I2C address: 0x2E

Library Settings

Wait for maximum command response time ●

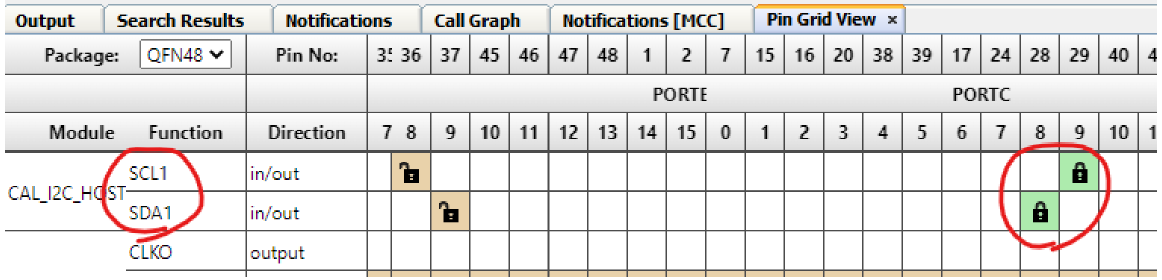
Print debug statements in library ●

Use a constant host nonce for encrypted read ○

Peripheral Selection

I2C Host Dependency: None

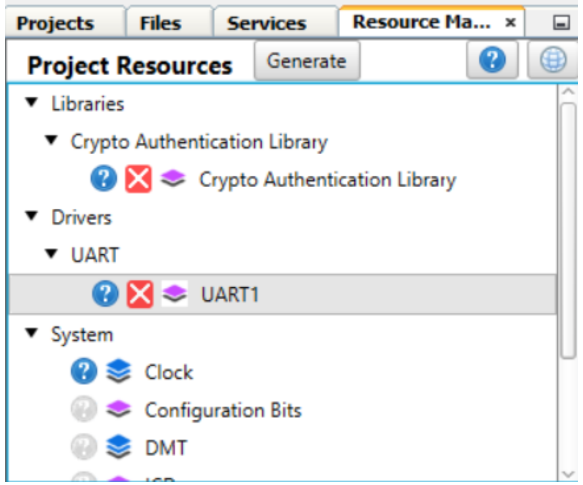
I2C1
I2C2
I2C3



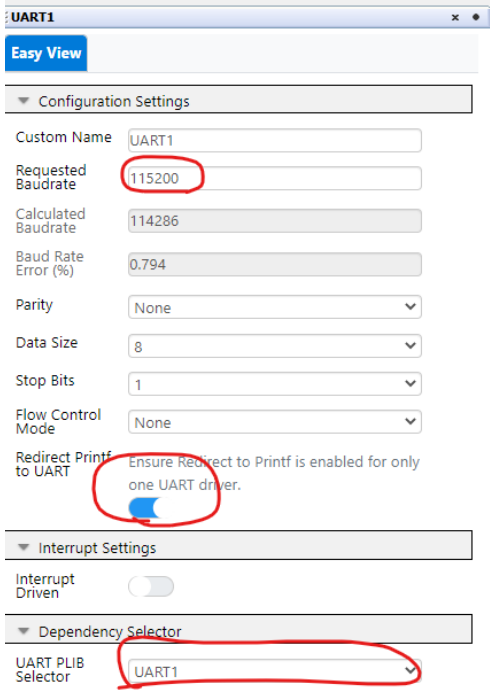
Package: QFN48		Pin No:	3:	36	37	45	46	47	48	1	2	7	15	16	20	38	39	17	24	28	29	40	4
			PORTE									PORTC											
Module	Function	Direction	7	8	9	10	11	12	13	14	15	0	1	2	3	4	5	6	7	8	9	10	1
CAL_I2C_HOST	SCL1	in/out		🔓																	🔒		
	SDA1	in/out			🔓															🔒			
	CLKO	output																					

Step 3-2

- In the “Device Resource” panel, add the Driver/UART to the project by clicking the “+” sign next to the module.
- In the UART Easy view, select UART1 from the dependency selector and config its Baudrate to 115200
- Choose RB4 as the UART1 TX output pin



- Libraries
  - Crypto Authentication Library
    - Crypto Authentication Library
- Drivers
  - UART
    - UART1
- System
  - Clock
  - Configuration Bits
  - DMT



Configuration Settings

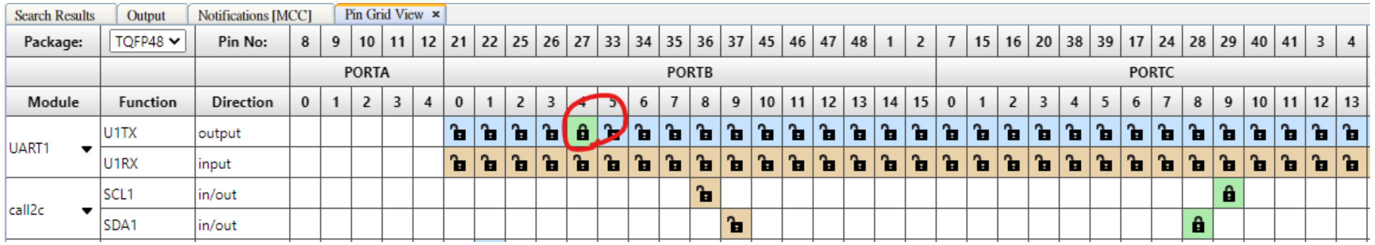
Custom Name	UART1
Requested Baudrate	115200
Calculated Baudrate	114286
Baud Rate Error (%)	0.794
Parity	None
Data Size	8
Stop Bits	1
Flow Control Mode	None
Redirect Printf to UART	Ensure Redirect to Printf is enabled for only one UART driver.

Interrupt Settings

Interrupt Driven

Dependency Selector

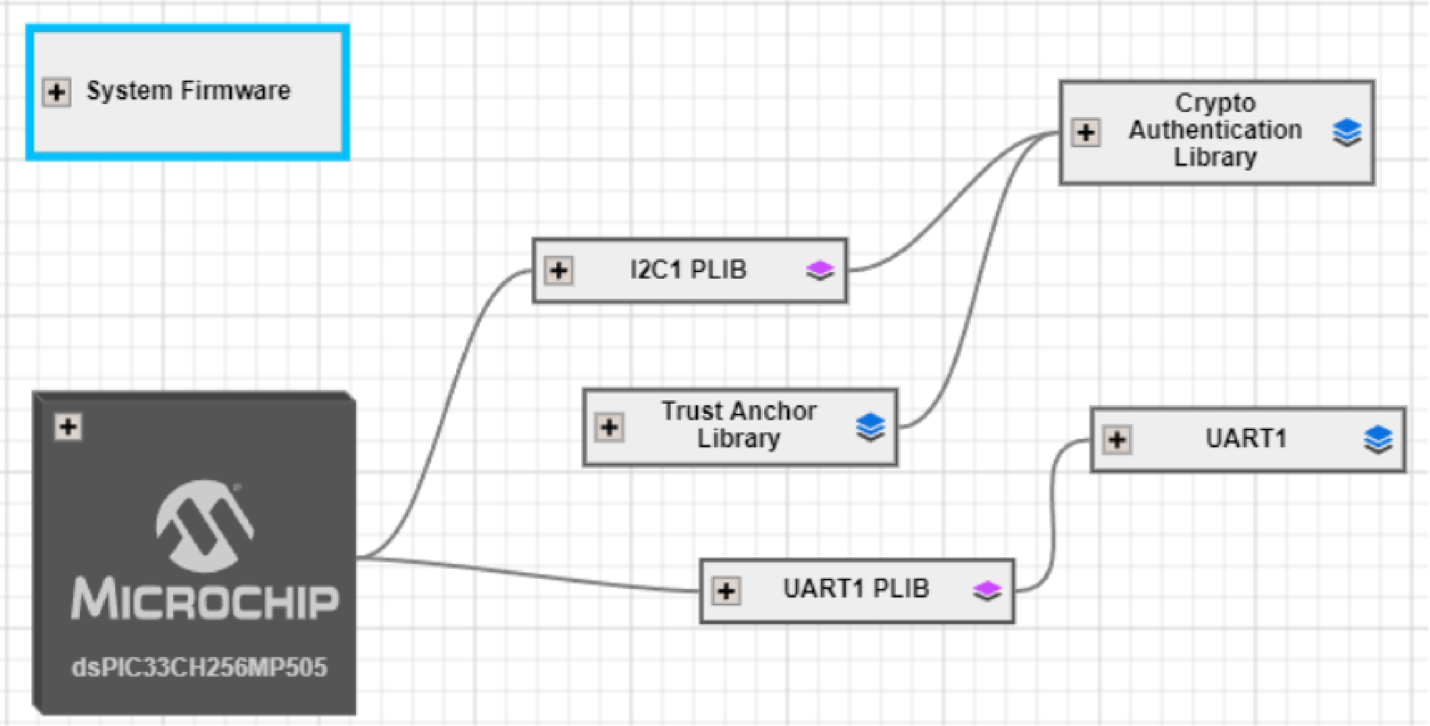
UART PLIB Selector	UART1

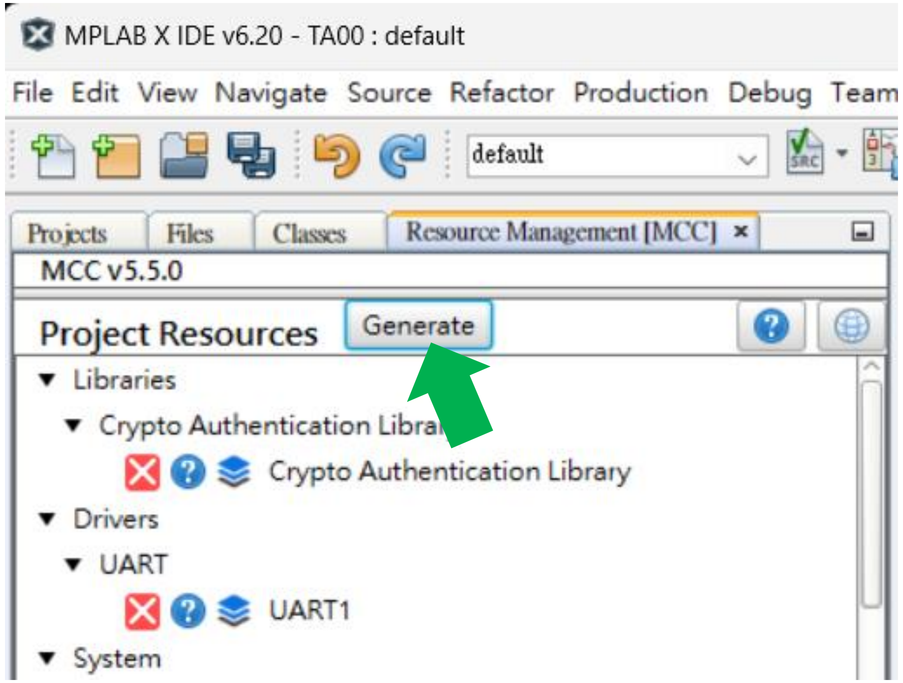


Package:	TQFP48	Pin No:	8	9	10	11	12	21	22	25	26	27	33	34	35	36	37	45	46	47	48	1	2	7	15	16	20	38	39	17	24	28	29	40	41	3	4
			PORTA					PORTB																PORTC													
Module	Function	Direction	0	1	2	3	4	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	0	1	2	3	4	5	6	7	8	9	10	11	12	13
UART1	U1TX	output						🔓	🔓	🔓	🔓	🔒	🔓	🔓	🔓	🔓	🔓	🔓	🔓	🔓	🔓	🔓	🔓	🔓	🔓	🔓	🔓	🔓	🔓	🔓	🔓	🔓	🔓	🔓	🔓	🔓	🔓
	U1RX	input						🔓	🔓	🔓	🔓	🔓	🔓	🔓	🔓	🔓	🔓	🔓	🔓	🔓	🔓	🔓	🔓	🔓	🔓	🔓	🔓	🔓	🔓	🔓	🔓	🔓	🔓	🔓	🔓	🔓	🔓
call2c	SCL1	in/out														🔓																	🔒				
	SDA1	in/out															🔓															🔒					

Step 3-3

- You can see the blocks in main screen as below
- Click the MCC “Generate” button in the project resourced panel

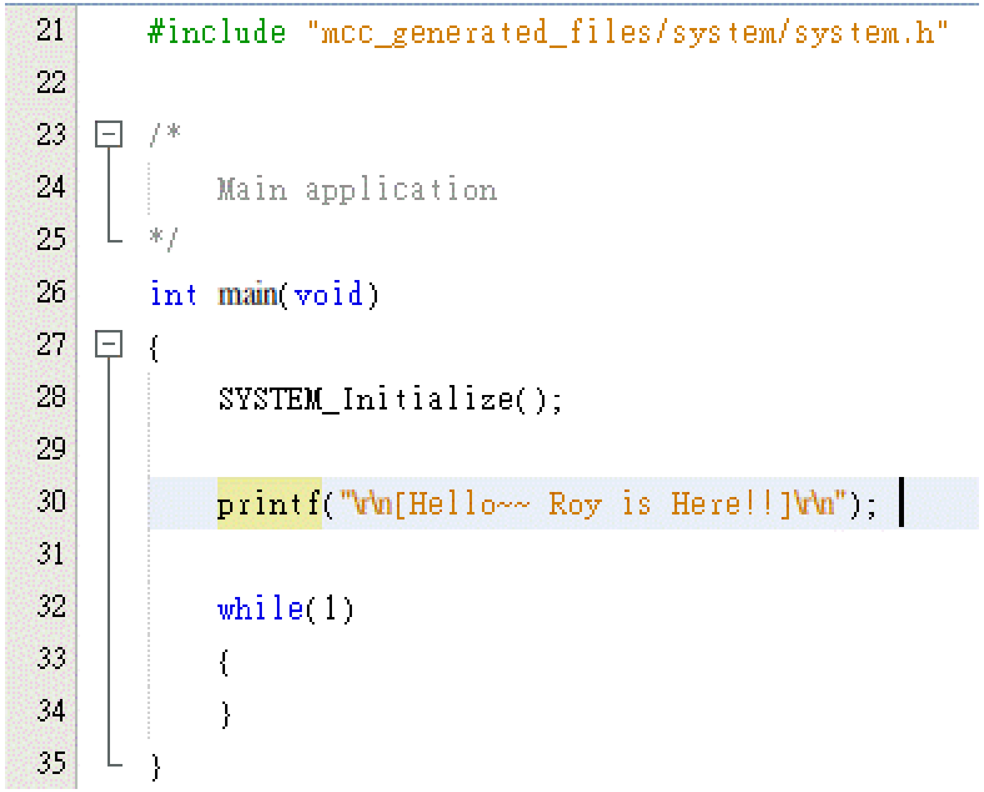



MPLAB X IDE v6.20 - TA00 : default
File Edit View Navigate Source Refactor Production Debug Team
default
Projects
Files
Classes
Resource Management [MCC] ×
MCC v5.5.0
Project Resources
Generate

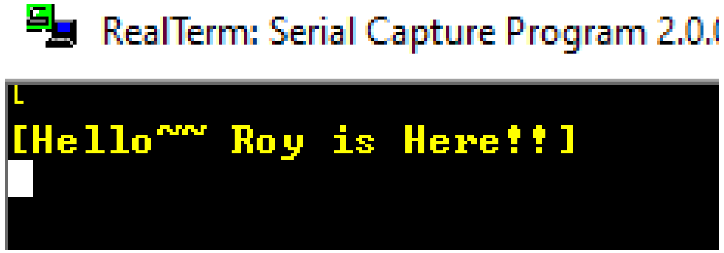
- Libraries
  - Crypto Authentication Librai
    - Crypto Authentication Library
- Drivers
  - UART
    - UART1
- System

Step 3-4

- You can Add some printf function to check UART is workable.
- Build & Program it.
- Open Terminal to check if the generated code base is workable



```
#include "mcc_generated_files/system/system.h"

/*
    Main application
*/
int main(void)
{
    SYSTEM_Initialize();

    printf("\r\n[Hello~~ Roy is Here!!]\r\n");

    while(1)
    {
    }
}
```



RealTerm: Serial Capture Program 2.0.(

```
[Hello~~ Roy is Here!!]
```

Lab4 - Try running TA100

Running Example code

Use: Lab4.txt

Main.h

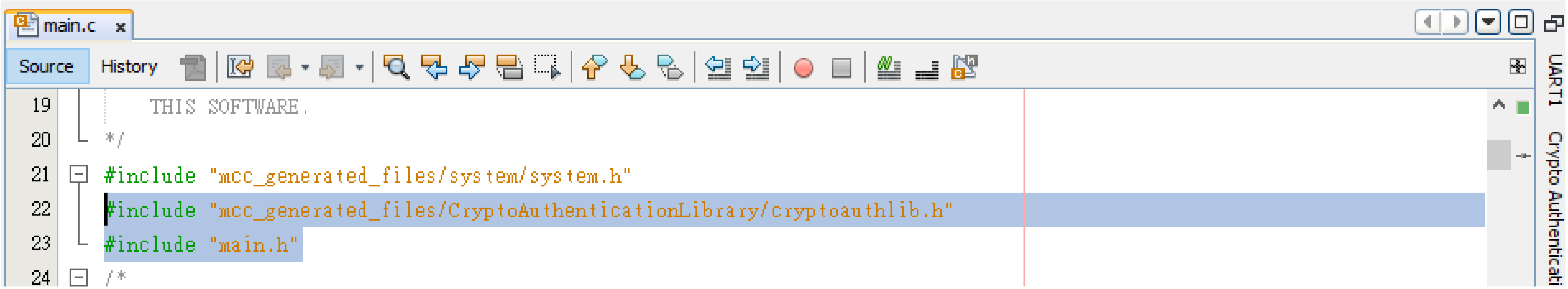
Make sure HW/SW are all good

Step 4-1

- Go to the project source code and find main.c.
- Include Cryptoauthlib.h

  #include "mcc_generated_files/CryptoAuthenticationLibrary/cryptoauthlib.h"
- Copy main.h file into project folder, and add it into project

  #include "main.h"

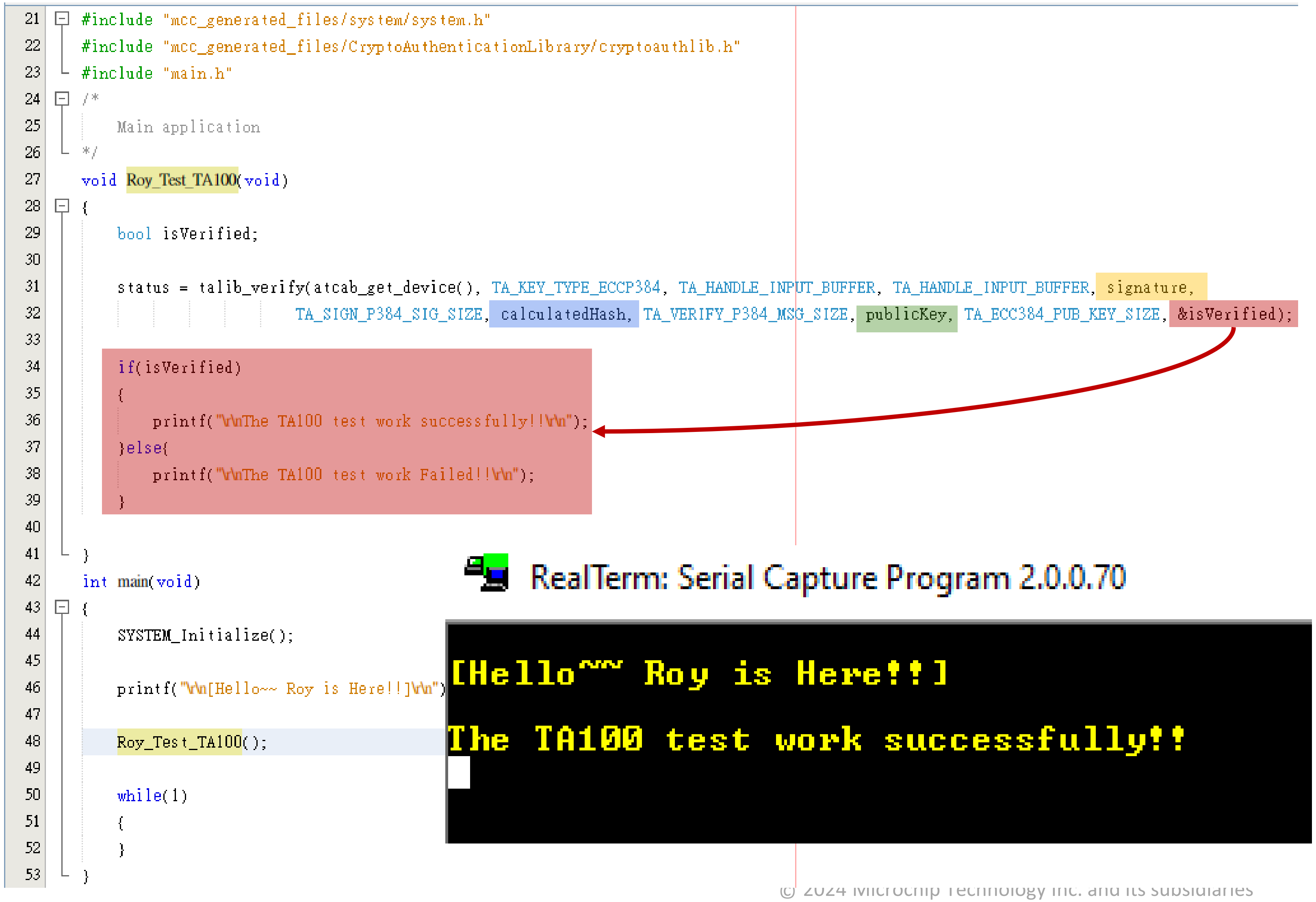


```
    THIS SOFTWARE.
 */
#include "mcc_generated_files/system/system.h"
#include "mcc_generated_files/CryptoAuthenticationLibrary/cryptoauthlib.h"
#include "main.h"
/*
```

- Copy Roy_Test_TA100() function from Lab4.txt to main.c
- Call Roy_Test_TA100() from main(), Build & Program it. (Step 4-2)
- Modify the “calculatedHash” value, check if the result changes. (Step 4-3)

Step 4-2

- **Check the result**

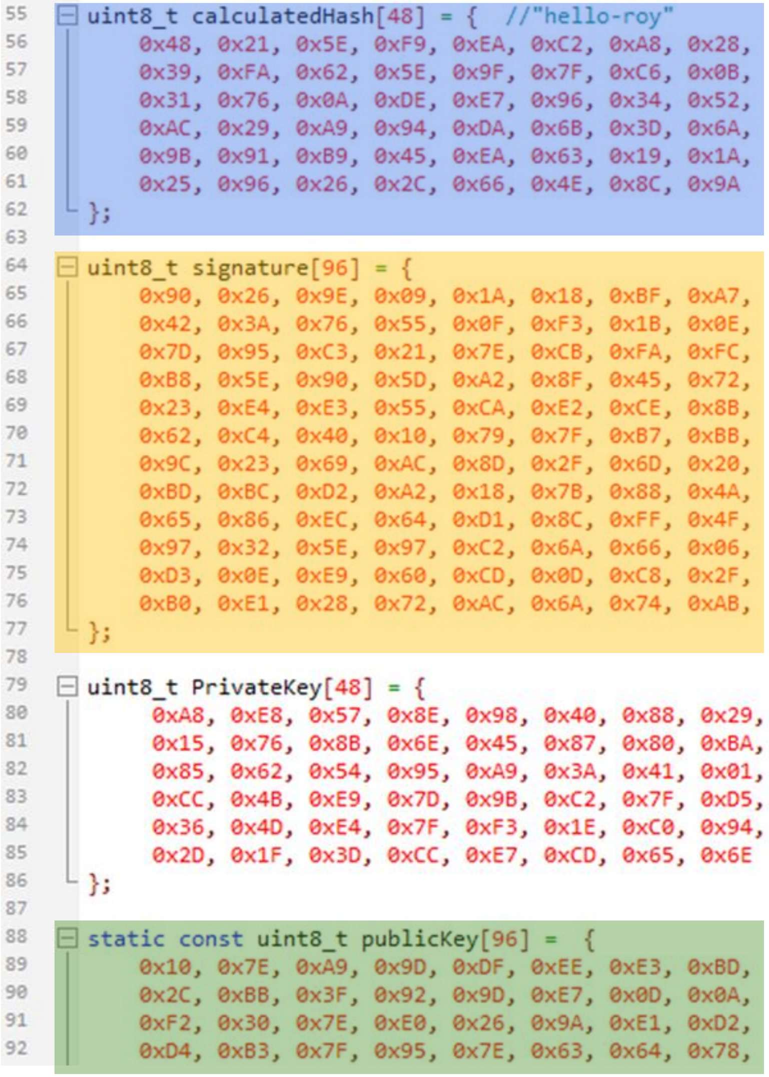


```c
#include "mcc_generated_files/system/system.h"
#include "mcc_generated_files/CryptoAuthenticationLibrary/cryptoauthlib.h"
#include "main.h"
/*
    Main application
*/
void Roy_Test_TA100(void)
{
    bool isVerified;

    status = talib_verify(atcab_get_device(), TA_KEY_TYPE_ECCP384, TA_HANDLE_INPUT_BUFFER, TA_HANDLE_INPUT_BUFFER, signature,
                          TA_SIGN_P384_SIG_SIZE, calculatedHash, TA_VERIFY_P384_MSG_SIZE, publicKey, TA_ECC384_PUB_KEY_SIZE, &isVerified);

    if(isVerified)
    {
        printf("\r\nThe TA100 test work successfully!!\r\n");
    }else{
        printf("\r\nThe TA100 test work Failed!!\r\n");
    }

}
int main(void)
{
    SYSTEM_Initialize();

    printf("\r\n[Hello~~ Roy is Here!!]\r\n")

    Roy_Test_TA100();

    while(1)
    {
    }
}
```

RealTerm: Serial Capture Program 2.0.0.70

```
[Hello~~ Roy is Here!!]

The TA100 test work successfully!!
```

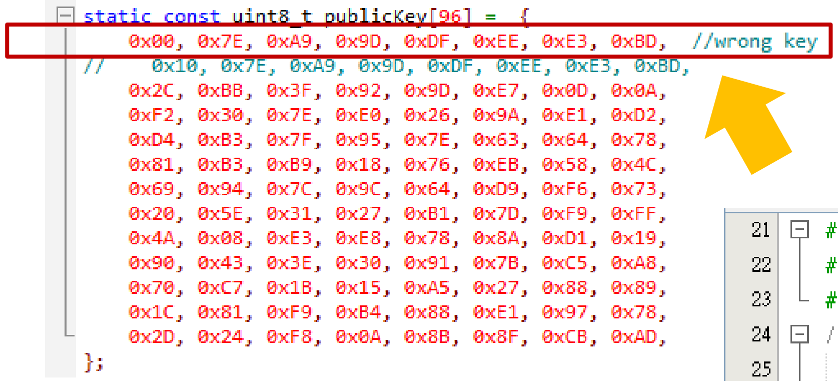
From main.h



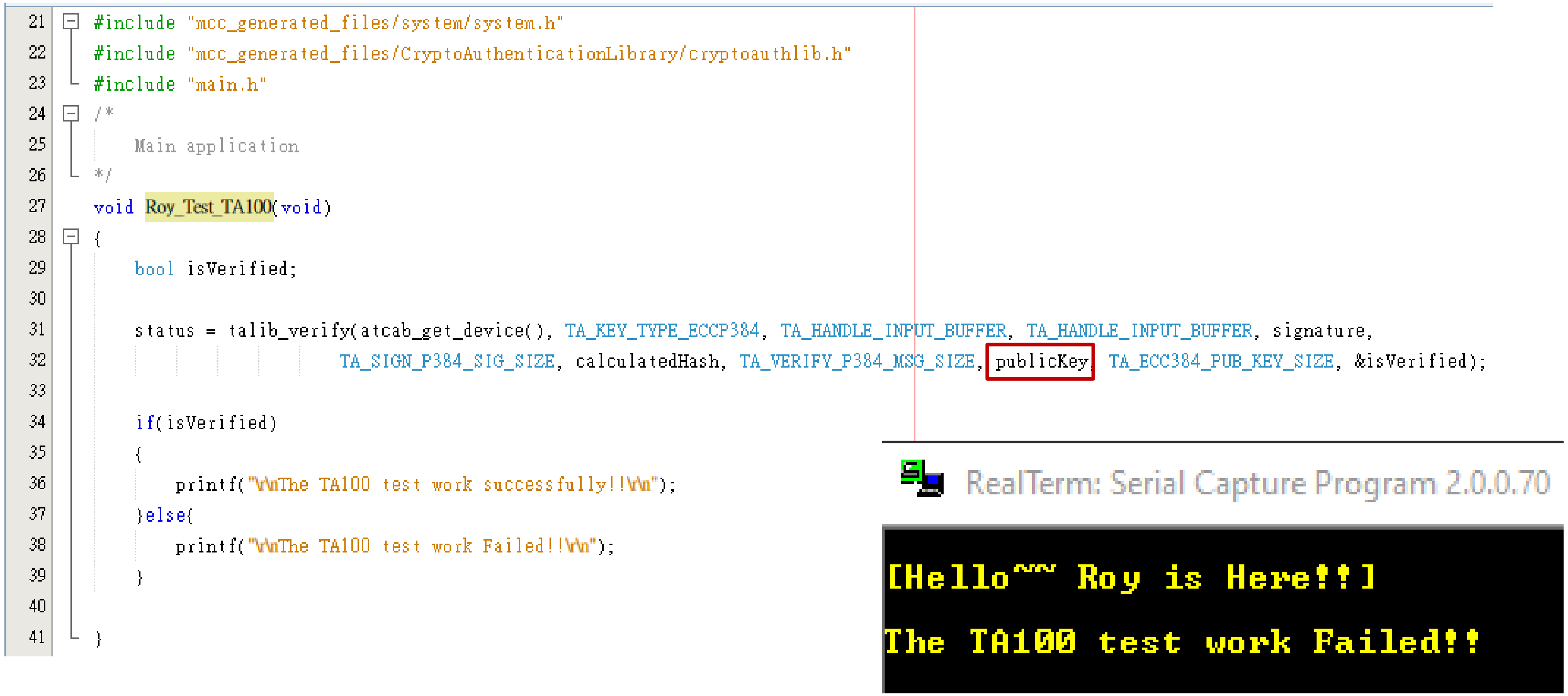
```c
uint8_t calculatedHash[48] = {  //"hello-roy"
    0x48, 0x21, 0x5E, 0xF9, 0xEA, 0xC2, 0xA8, 0x28,
    0x39, 0xFA, 0x62, 0x5E, 0x9F, 0x7F, 0xC6, 0x0B,
    0x31, 0x76, 0x0A, 0xDE, 0xE7, 0x96, 0x34, 0x52,
    0xAC, 0x29, 0xA9, 0x94, 0xDA, 0x6B, 0x3D, 0x6A,
    0x9B, 0x91, 0xB9, 0x45, 0xEA, 0x63, 0x19, 0x1A,
    0x25, 0x96, 0x26, 0x2C, 0x66, 0x4E, 0x8C, 0x9A
};

uint8_t signature[96] = {
    0x90, 0x26, 0x9E, 0x09, 0x1A, 0x18, 0xBF, 0xA7,
    0x42, 0x3A, 0x76, 0x55, 0x0F, 0xF3, 0x1B, 0x0E,
    0x7D, 0x95, 0xC3, 0x21, 0x7E, 0xCB, 0xFA, 0xFC,
    0xB8, 0x5E, 0x90, 0x5D, 0xA2, 0x8F, 0x45, 0x72,
    0x23, 0xE4, 0xE3, 0x55, 0xCA, 0xE2, 0xCE, 0x8B,
    0x62, 0xC4, 0x40, 0x10, 0x79, 0x7F, 0xB7, 0xBB,
    0x9C, 0x23, 0x69, 0xAC, 0x8D, 0x2F, 0x6D, 0x20,
    0xBD, 0xBC, 0xD2, 0xA2, 0x18, 0x7B, 0x88, 0x4A,
    0x65, 0x86, 0xEC, 0x64, 0xD1, 0x8C, 0xFF, 0x4F,
    0x97, 0x32, 0x5E, 0x97, 0xC2, 0x6A, 0x66, 0x06,
    0xD3, 0x0E, 0xE9, 0x60, 0xCD, 0x0D, 0xC8, 0x2F,
    0xB0, 0xE1, 0x28, 0x72, 0xAC, 0x6A, 0x74, 0xAB,
};

uint8_t PrivateKey[48] = {
     0xA8, 0xE8, 0x57, 0x8E, 0x98, 0x40, 0x88, 0x29,
     0x15, 0x76, 0x8B, 0x6E, 0x45, 0x87, 0x80, 0xBA,
     0x85, 0x62, 0x54, 0x95, 0xA9, 0x3A, 0x41, 0x01,
     0xCC, 0x4B, 0xE9, 0x7D, 0x9B, 0xC2, 0x7F, 0xD5,
     0x36, 0x4D, 0xE4, 0x7F, 0xF3, 0x1E, 0xC0, 0x94,
     0x2D, 0x1F, 0x3D, 0xCC, 0xE7, 0xCD, 0x65, 0x6E
};

static const uint8_t publicKey[96] =  {
    0x10, 0x7E, 0xA9, 0x9D, 0xDF, 0xEE, 0xE3, 0xBD,
    0x2C, 0xBB, 0x3F, 0x92, 0x9D, 0xE7, 0x0D, 0x0A,
    0xF2, 0x30, 0x7E, 0xE0, 0x26, 0x9A, 0xE1, 0xD2,
    0xD4, 0xB3, 0x7F, 0x95, 0x7E, 0x63, 0x64, 0x78,
```

Step 4-3

- **Modify the public key & run Lab4 again**



```
static const uint8_t publicKey[96] =  {
    0x00, 0x7E, 0xA9, 0x9D, 0xDF, 0xEE, 0xE3, 0xBD,  //wrong key
//    0x10, 0x7E, 0xA9, 0x9D, 0xDF, 0xEE, 0xE3, 0xBD,
    0x2C, 0xBB, 0x3F, 0x92, 0x9D, 0xE7, 0x0D, 0x0A,
    0xF2, 0x30, 0x7E, 0xE0, 0x26, 0x9A, 0xE1, 0xD2,
    0xD4, 0xB3, 0x7F, 0x95, 0x7E, 0x63, 0x64, 0x78,
    0x81, 0xB3, 0xB9, 0x18, 0x76, 0xEB, 0x58, 0x4C,
    0x69, 0x94, 0x7C, 0x9C, 0x64, 0xD9, 0xF6, 0x73,
    0x20, 0x5E, 0x31, 0x27, 0xB1, 0x7D, 0xF9, 0xFF,
    0x4A, 0x08, 0xE3, 0xE8, 0x78, 0x8A, 0xD1, 0x19,
    0x90, 0x43, 0x3E, 0x30, 0x91, 0x7B, 0xC5, 0xA8,
    0x70, 0xC7, 0x1B, 0x15, 0xA5, 0x27, 0x88, 0x89,
    0x1C, 0x81, 0xF9, 0xB4, 0x88, 0xE1, 0x97, 0x78,
    0x2D, 0x24, 0xF8, 0x0A, 0x8B, 0x8F, 0xCB, 0xAD,
};
```



```
#include "mcc_generated_files/system/system.h"
#include "mcc_generated_files/CryptoAuthenticationLibrary/cryptoauthlib.h"
#include "main.h"
/*
    Main application
*/
void Roy_Test_TA100(void)
{
    bool isVerified;

    status = talib_verify(atcab_get_device(), TA_KEY_TYPE_ECCP384, TA_HANDLE_INPUT_BUFFER, TA_HANDLE_INPUT_BUFFER, signature,
                        TA_SIGN_P384_SIG_SIZE, calculatedHash, TA_VERIFY_P384_MSG_SIZE, publicKey, TA_ECC384_PUB_KEY_SIZE, &isVerified);

    if(isVerified)
    {
        printf("\r\nThe TA100 test work successfully!!\r\n");
    }else{
        printf("\r\nThe TA100 test work Failed!!\r\n");
    }

}
```

```
[Hello~~ Roy is Here!!]
The TA100 test work Failed!!
```

Microchip Proprietary and STRICTLY Confidential

Lab5 – Try your 1st TA100 function

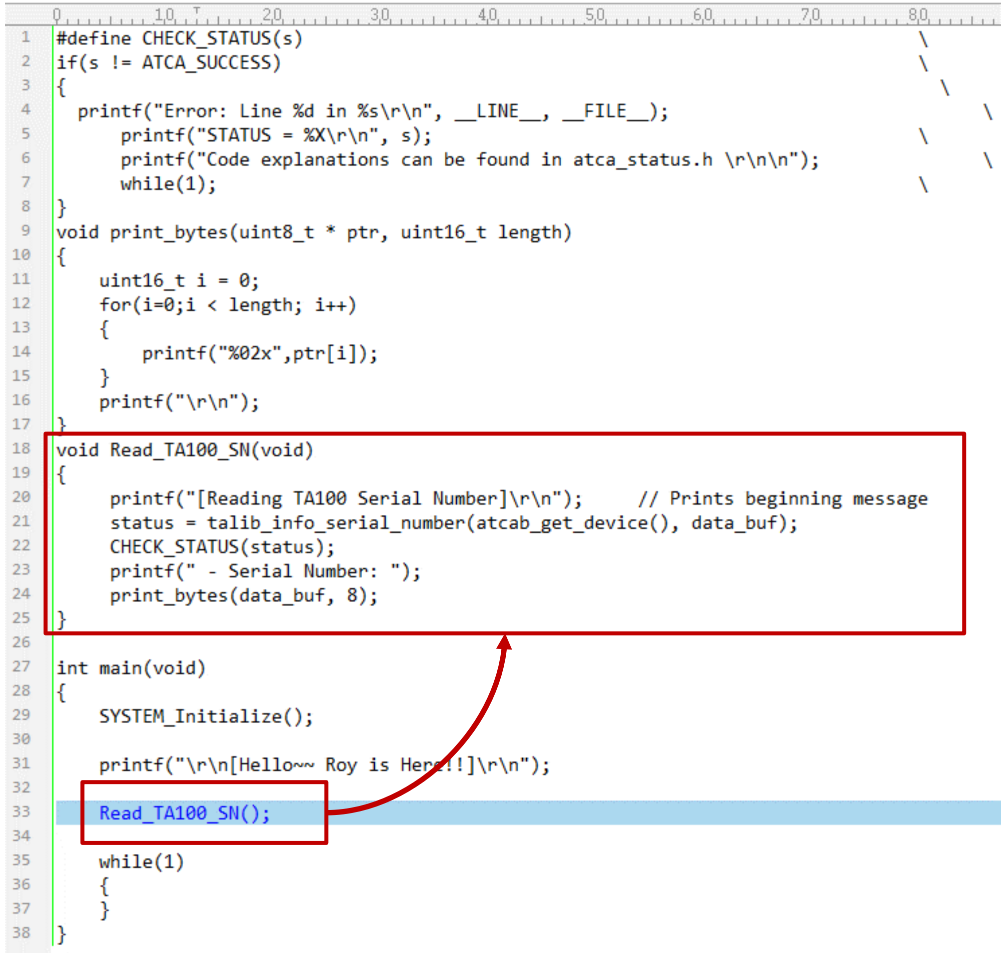
Read Serial Number

Use: Lab5.txt

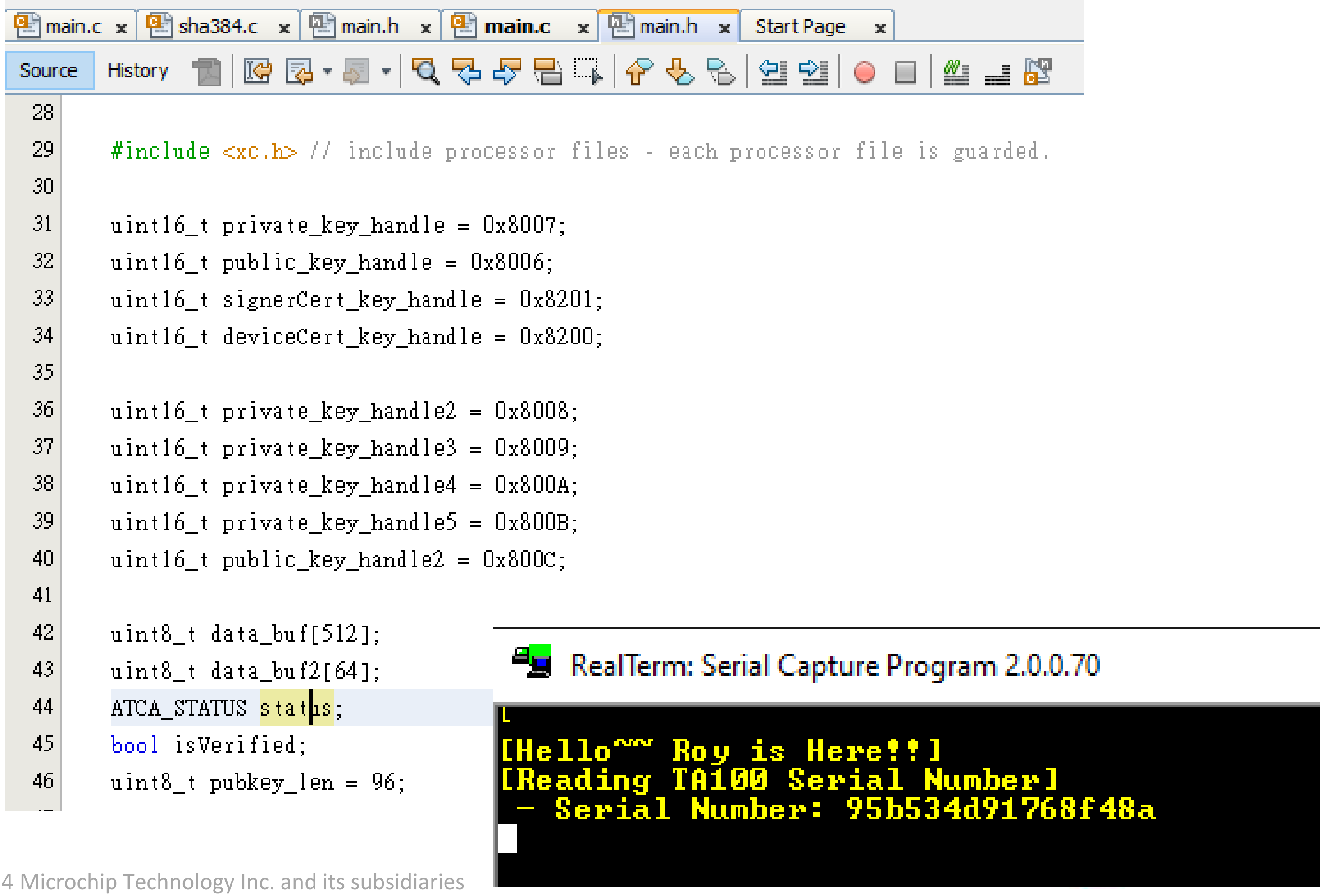
Everyone should get different result per TA100

Step 5-1

- **Copy functions from Lab5.txt to main.c , Program & Run!**
- **Check the initial processes from SYSTEM_Initialize();**



```
#define CHECK_STATUS(s)                                                  \
if(s != ATCA_SUCCESS)                                                    \
{                                                                          \
  printf("Error: Line %d in %s\r\n", __LINE__, __FILE__);                    \
      printf("STATUS = %X\r\n", s);                                      \
      printf("Code explanations can be found in atca_status.h \r\n\n");      \
      while(1);                                                          \
}
void print_bytes(uint8_t * ptr, uint16_t length)
{
    uint16_t i = 0;
    for(i=0;i < length; i++)
    {
        printf("%02x",ptr[i]);
    }
    printf("\r\n");
}
void Read_TA100_SN(void)
{
     printf("[Reading TA100 Serial Number]\r\n");     // Prints beginning message
     status = talib_info_serial_number(atcab_get_device(), data_buf);
     CHECK_STATUS(status);
     printf(" - Serial Number: ");
     print_bytes(data_buf, 8);
}

int main(void)
{
    SYSTEM_Initialize();

    printf("\r\n[Hello~~ Roy is Here!!]\r\n");

    Read_TA100_SN();

    while(1)
    {
    }
}
```



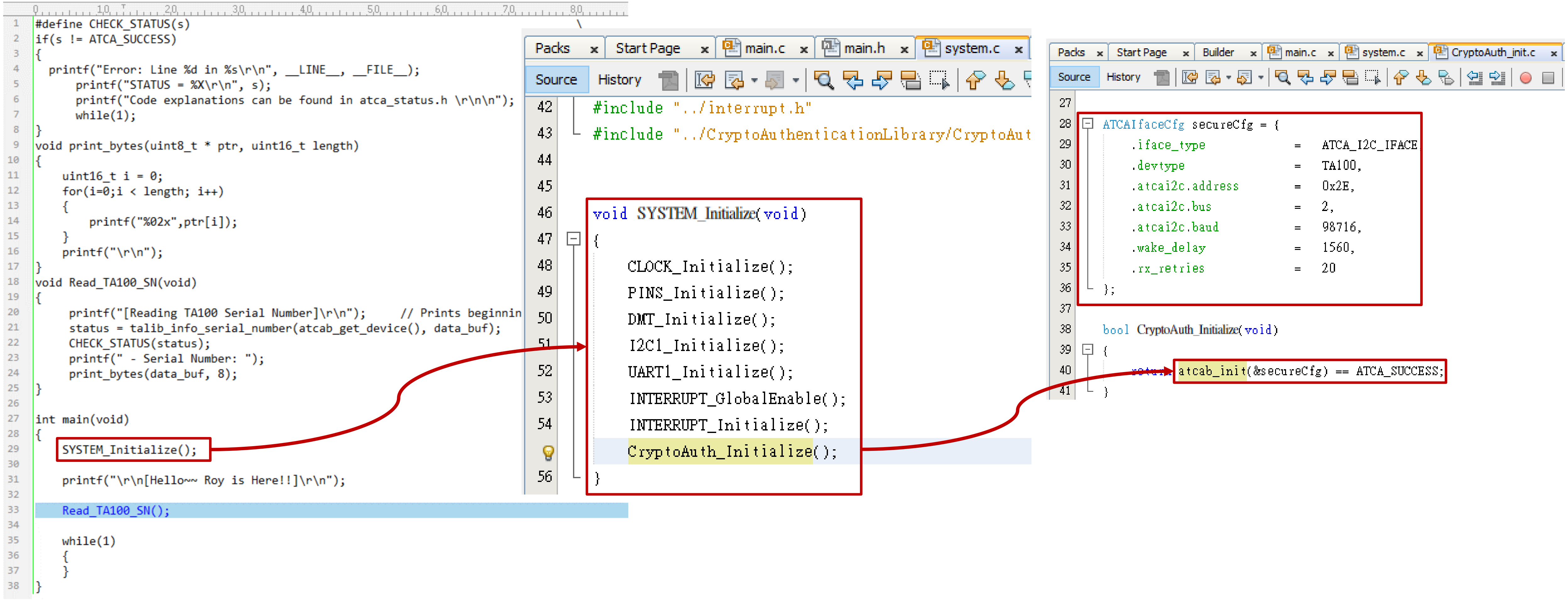
```
#include <xc.h> // include processor files - each processor file is guarded.

uint16_t private_key_handle = 0x8007;
uint16_t public_key_handle = 0x8006;
uint16_t signerCert_key_handle = 0x8201;
uint16_t deviceCert_key_handle = 0x8200;

uint16_t private_key_handle2 = 0x8008;
uint16_t private_key_handle3 = 0x8009;
uint16_t private_key_handle4 = 0x800A;
uint16_t private_key_handle5 = 0x800B;
uint16_t public_key_handle2 = 0x800C;

uint8_t data_buf[512];
uint8_t data_buf2[64];
ATCA_STATUS status;
bool isVerified;
uint8_t pubkey_len = 96;
```

RealTerm: Serial Capture Program 2.0.0.70

```
[Hello~~ Roy is Here!!]
[Reading TA100 Serial Number]
 - Serial Number: 95b534d91768f48a
```

Step 5-2

- **Check the initial processes from SYSTEM_Initialize();**



```
#define CHECK_STATUS(s)
if(s != ATCA_SUCCESS)
{
  printf("Error: Line %d in %s\r\n", __LINE__, __FILE__);
      printf("STATUS = %X\r\n", s);
      printf("Code explanations can be found in atca_status.h \r\n\n");
      while(1);
}
void print_bytes(uint8_t * ptr, uint16_t length)
{
    uint16_t i = 0;
    for(i=0;i < length; i++)
    {
        printf("%02x",ptr[i]);
    }
    printf("\r\n");
}
void Read_TA100_SN(void)
{
     printf("[Reading TA100 Serial Number]\r\n");     // Prints beginnin
     status = talib_info_serial_number(atcab_get_device(), data_buf);
     CHECK_STATUS(status);
     printf(" - Serial Number: ");
     print_bytes(data_buf, 8);
}

int main(void)
{
    SYSTEM_Initialize();

    printf("\r\n[Hello~~ Roy is Here!!]\r\n");

    Read_TA100_SN();

    while(1)
    {
    }
}
```

```
#include "../interrupt.h"
#include "../CryptoAuthenticationLibrary/CryptoAut


void SYSTEM_Initialize(void)
{
    CLOCK_Initialize();
    PINS_Initialize();
    DMT_Initialize();
    I2C1_Initialize();
    UART1_Initialize();
    INTERRUPT_GlobalEnable();
    INTERRUPT_Initialize();
    CryptoAuth_Initialize();
}
```

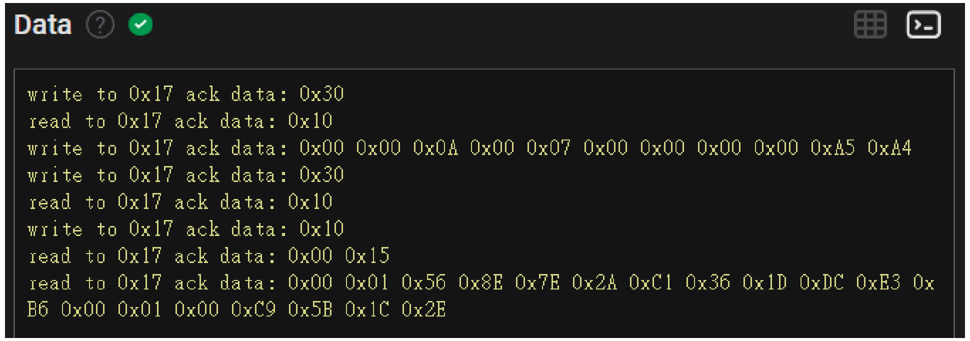
```

ATCAIfaceCfg secureCfg = {
        .iface_type             =    ATCA_I2C_IFACE
        .devtype                =    TA100,
        .atcai2c.address        =    0x2E,
        .atcai2c.bus            =    2,
        .atcai2c.baud           =    98716,
        .wake_delay             =    1560,
        .rx_retries             =    20
};

bool CryptoAuth_Initialize(void)
{
        return atcab_init(&secureCfg) == ATCA_SUCCESS;
}
```

Let’s start the Lab1~ Lab5

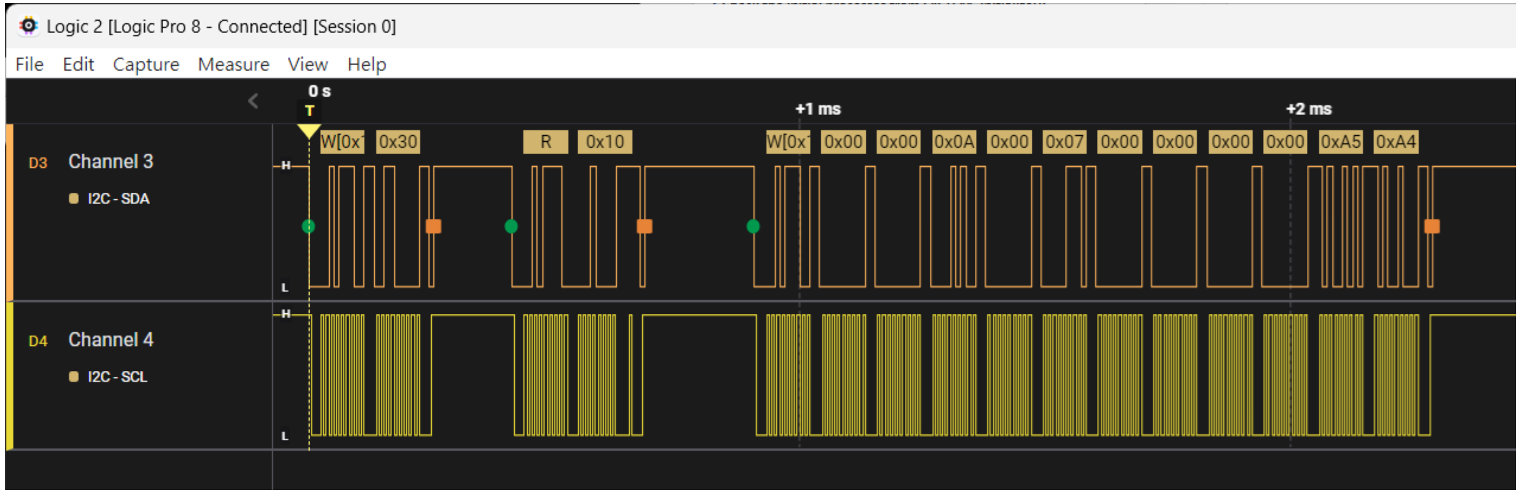
Try to finish the Labs before 12:00

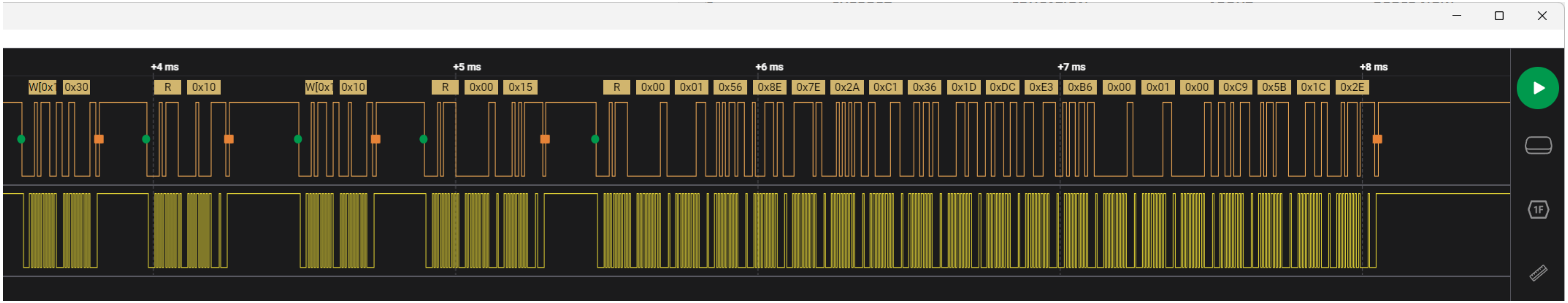
look into TA100 I2C communication

SDA/SCL = RC8/RC9, I2C address = 0x17



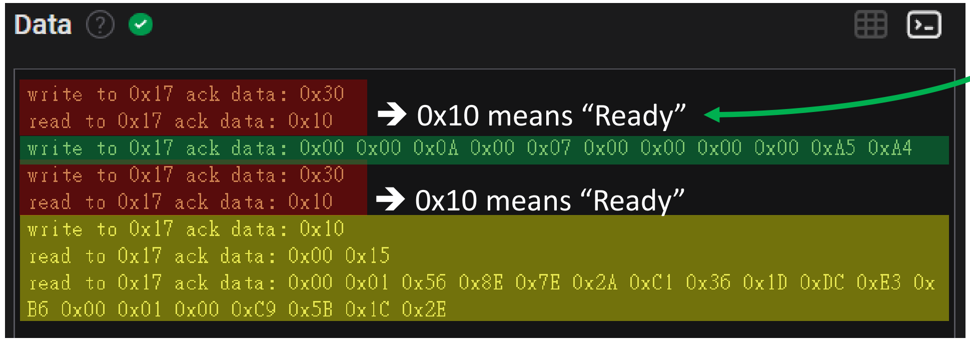
```
write to 0x17 ack data: 0x30
read to 0x17 ack data: 0x10
write to 0x17 ack data: 0x00 0x00 0x0A 0x00 0x07 0x00 0x00 0x00 0x00 0xA5 0xA4
write to 0x17 ack data: 0x30
read to 0x17 ack data: 0x10
write to 0x17 ack data: 0x10
read to 0x17 ack data: 0x00 0x15
read to 0x17 ack data: 0x00 0x01 0x56 0x8E 0x7E 0x2A 0xC1 0x36 0x1D 0xDC 0xE3 0x
B6 0x00 0x01 0x00 0xC9 0x5B 0x1C 0x2E
```





look into TA100 I2C communication

SDA/SCL = RC8/RC9, I2C address = 0x17



```
write to 0x17 ack data: 0x30
read to 0x17 ack data: 0x10
write to 0x17 ack data: 0x00 0x00 0x0A 0x00 0x07 0x00 0x00 0x00 0x00 0xA5 0xA4
write to 0x17 ack data: 0x30
read to 0x17 ack data: 0x10
write to 0x17 ack data: 0x10
read to 0x17 ack data: 0x00 0x15
read to 0x17 ack data: 0x00 0x01 0x56 0x8E 0x7E 0x2A 0xC1 0x36 0x1D 0xDC 0xE3 0x
B6 0x00 0x01 0x00 0xC9 0x5B 0x1C 0x2E
```

➔ 0x10 means “Ready”

➔ 0x10 means “Ready”

Bit 4 (RRDY)	0 = The command response buffer is empty. 1 = The command response buffer is ready to be read.

Table 10-1. Transaction Type Table

Name	Kind	Value[1, 2]	Description
RD_CSR	Status	0011 0000	Reads the Command Processor Status register (CSR)
WR_CCR	Control	0010 0000	Writes the Command Control register (CCR)
RD_CMD	Command	0001 0000	Reads the response from the command processor response buffer. First byte is MSB of length.
WR_CMD	Command	0000 0000	Writes the command/data to the command processor input buffer. First byte is MSB of length.

Table 11-1. Command Packet Formatting

Field Name	Bytes	Description
Length	2	Total number of bytes in the command, including this “Length” field. This length includes the “CRC” field (2 bytes).
Opcode	1	Command to be executed by TA100.
Param1	1	The first parameter of the command. In the descriptions below, if this parameter is used, it will have a descriptive name. This is often a mode modifier.
Param2	4	A second parameter of the command. In the descriptions below, if this parameter is used, it may have a descriptive name. If unused for a particular command, it must be sent to TA100 as 0x00 00 00 00.
Data	0-1024	Additional information for the command, it must be no more than 1024 bytes. If this parameter is not required for a given command, it is typically not shown in the input parameter table of the command or is listed as 0 bytes in length.
CRC	2	CRC verification of the length, opcode, parameters and data bytes. See 11.1.2. CRC Algorithm.

Info (0x00)

This command returns status or state information from the device.

Table 12-48. Info Input Parameters

Name	Size	Description
Opcode	1	0x00
Mode	1	Selection field for the return information, see Table 12-50.
Param2	4	Handle for modes 2 and 3, otherwise, it must be ‘0’.

Table 12-49. Info Output Parameters

Name	Size	Description
Resp_Code	1	‘0’ if successful. If not, there is an error code.
Data	1-258	The information appropriate for the mode parameter.

Command: Opcode Mode 0x07 means “Dedicated Memory”

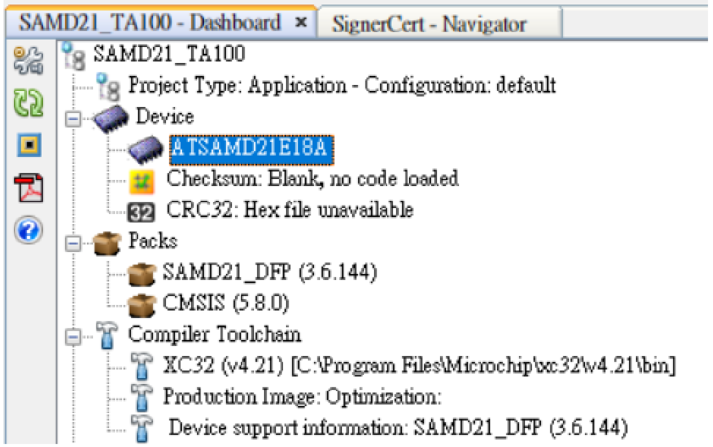
0x00 0x00 0x0A 0x00 0x07 0x00 0x00 0x00 0x00 0xA5 0xA4 CRC

Output: Byte count Successful Serial Number

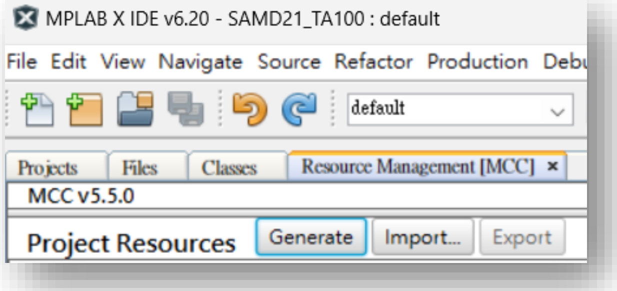
0x00 0x15 0x00 0x01 0x56 0x8E 0x7E 0x2A 0xC1 0x36 0x1D
0xDC 0xE3 0xB6 0x00 0x01 0x00 0xC9 0x5B 0x1C 0x2E

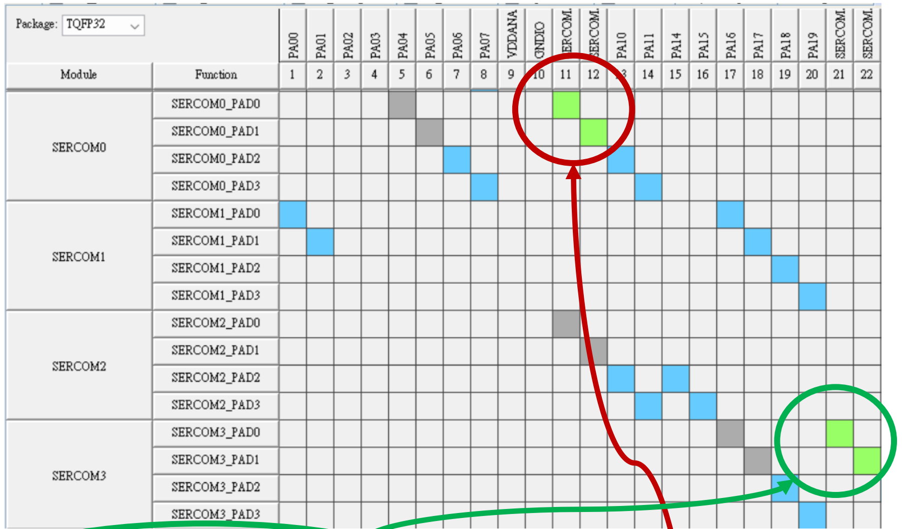
Run the same code using TPDS demo board

SAMD21(32bits) with MCC Harmony



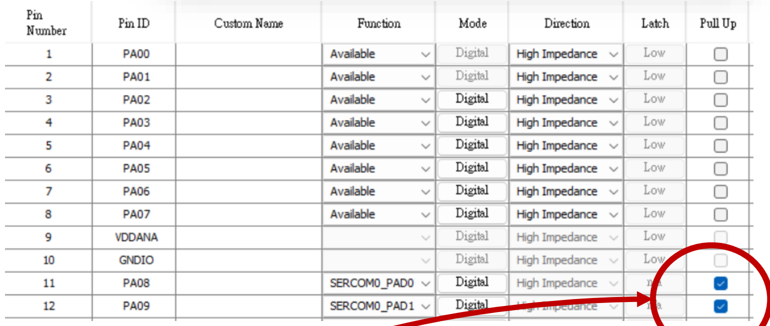
- SAMD21_TA100
  - Project Type: Application - Configuration: default
  - Device
    - ATSAMD21E18A
    - Checksum: Blank, no code loaded
    - CRC32: Hex file unavailable
  - Packs
    - SAMD21_DFP (3.6.144)
    - CMSIS (5.8.0)
  - Compiler Toolchain
    - XC32 (v4.21) [C:\Program Files\Microchip\xc32\v4.21\bin]
    - Production Image: Optimization:
    - Device support information: SAMD21_DFP (3.6.144)


MPLAB X IDE v6.20 - SAMD21_TA100 : default
File Edit View Navigate Source Refactor Production Debu
default
Projects Files Classes Resource Management [MCC] ×
MCC v5.5.0
Project Resources
Generate
Import...
Export

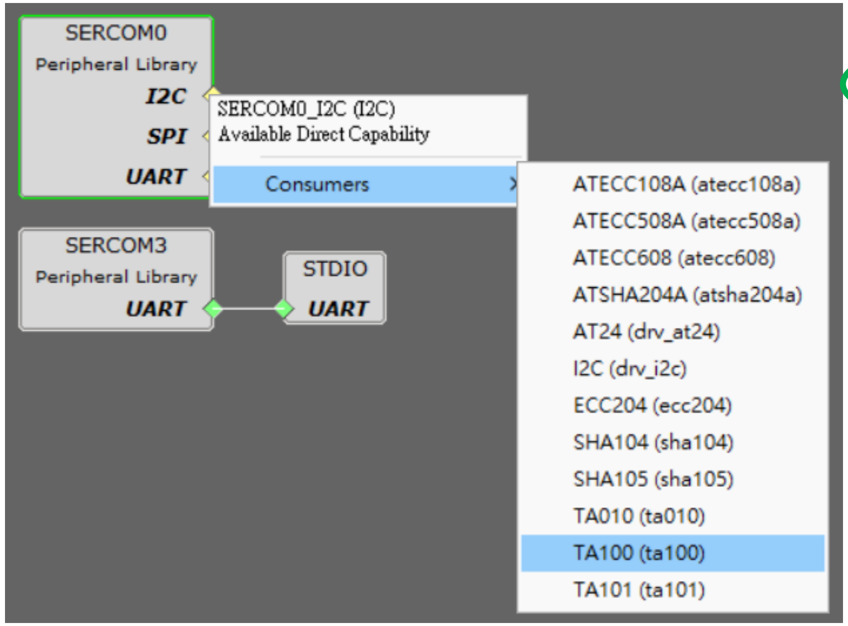


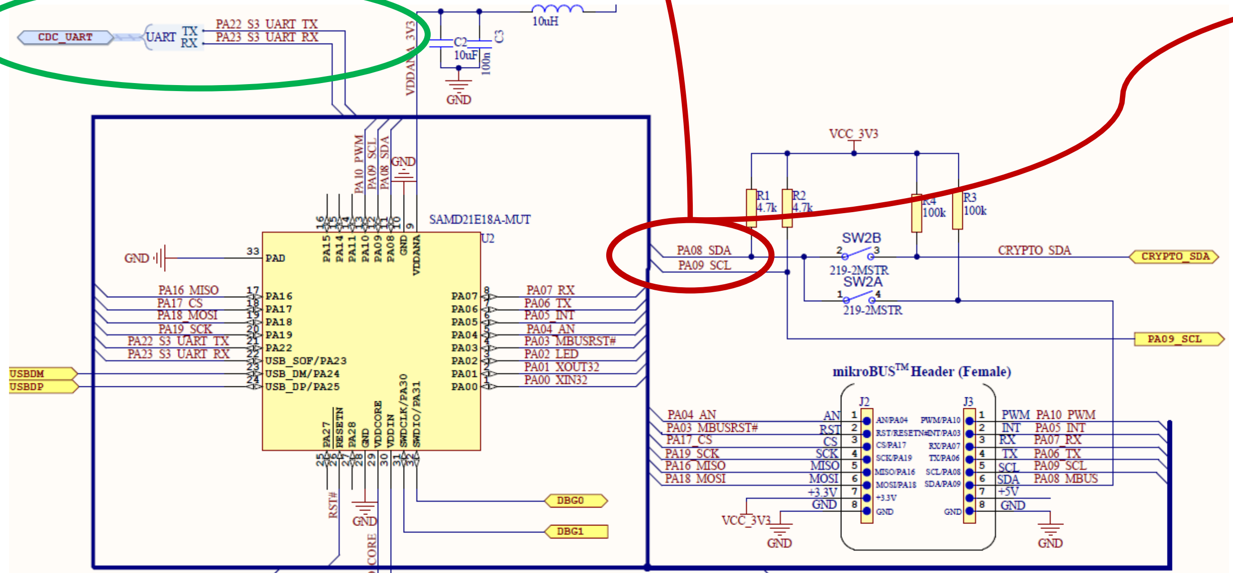
Package: TQFP32

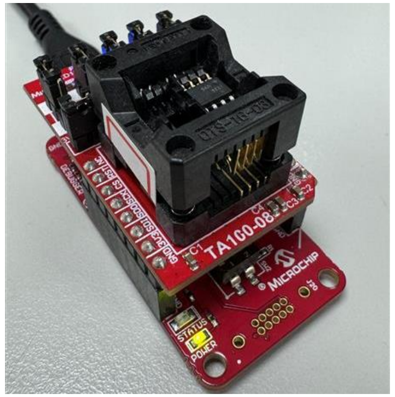
Module	Function	PA00 1	PA01 2	PA02 3	PA03 4	PA04 5	PA05 6	PA06 7	PA07 8	VDDANA 9	GNDIO 10	ERCOM 11	SERCOM 12	PA10 13	PA11 14	PA14 15	PA15 16	PA16 17	PA17 18	PA18 19	PA19 20	SERCOM 21	SERCOM 22
SERCOM0	SERCOM0_PAD0					▨						🟩											
	SERCOM0_PAD1						▨						🟩										
	SERCOM0_PAD2							🟦						🟦									
	SERCOM0_PAD3								🟦						🟦								
SERCOM1	SERCOM1_PAD0	🟦																🟦					
	SERCOM1_PAD1		🟦																🟦				
	SERCOM1_PAD2																			🟦			
	SERCOM1_PAD3																				🟦		
SERCOM2	SERCOM2_PAD0											▨											
	SERCOM2_PAD1												▨										
	SERCOM2_PAD2													🟦		🟦							
	SERCOM2_PAD3														🟦		🟦						
SERCOM3	SERCOM3_PAD0																	▨				🟩	
	SERCOM3_PAD1																		▨				🟩
	SERCOM3_PAD2																			🟦			
	SERCOM3_PAD3																				🟦		



Pin Number	Pin ID	Custom Name	Function	Mode	Direction	Latch	Pull Up
1	PA00		Available	Digital	High Impedance	Low	☐
2	PA01		Available	Digital	High Impedance	Low	☐
3	PA02		Available	Digital	High Impedance	Low	☐
4	PA03		Available	Digital	High Impedance	Low	☐
5	PA04		Available	Digital	High Impedance	Low	☐
6	PA05		Available	Digital	High Impedance	Low	☐
7	PA06		Available	Digital	High Impedance	Low	☐
8	PA07		Available	Digital	High Impedance	Low	☐
9	VDDANA			Digital	High Impedance	Low	☐
10	GNDIO			Digital	High Impedance	Low	☐
11	PA08		SERCOM0_PAD0	Digital	High Impedance	n/a	☑
12	PA09		SERCOM0_PAD1	Digital	High Impedance	n/a	☑

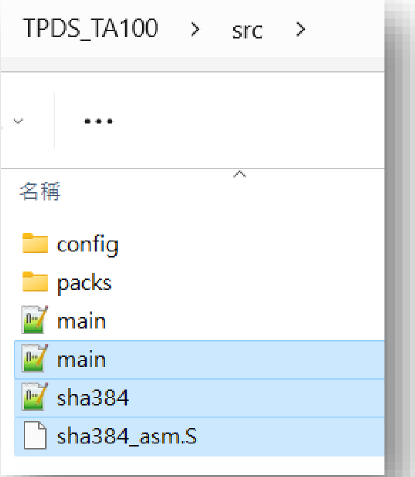


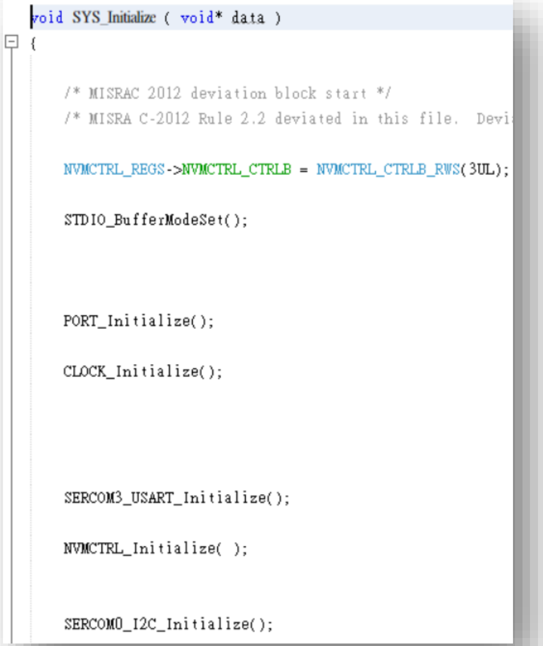




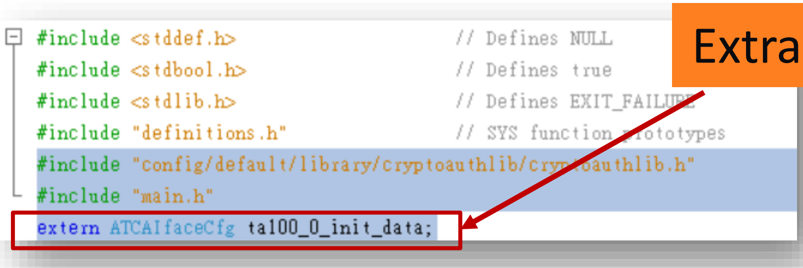
Run the same code using TPDS demo board

SAMD21(32bits) with MCC Harmony



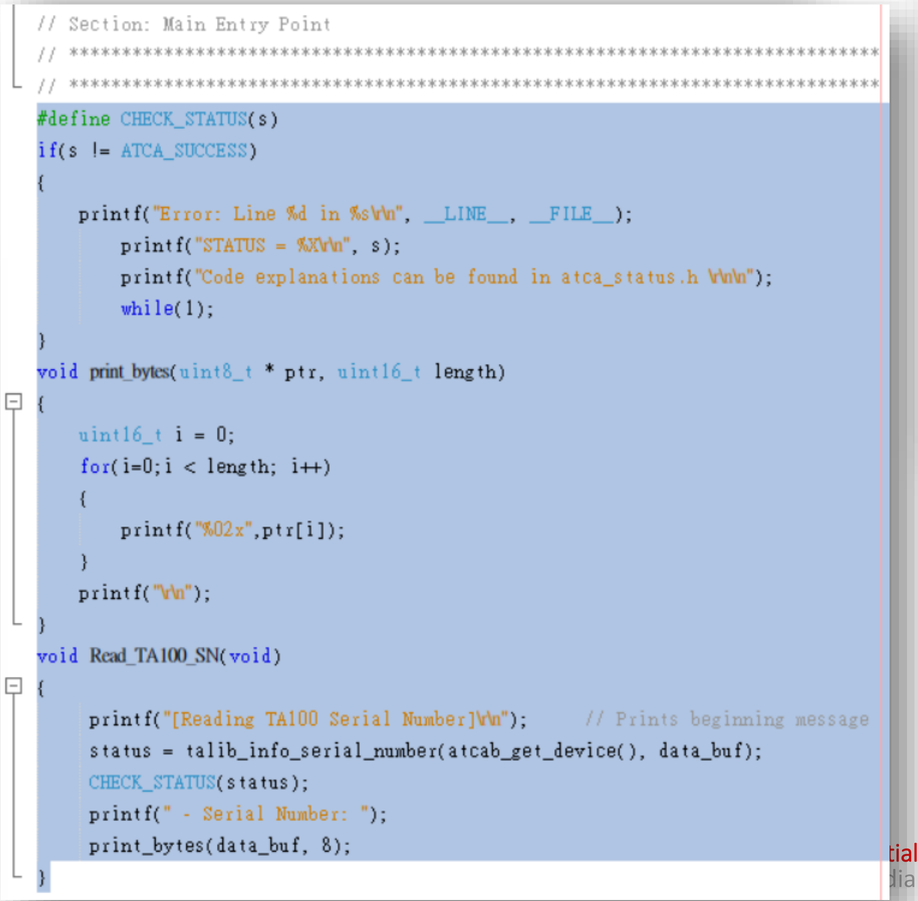


```
void SYS_Initialize ( void* data )
{

    /* MISRAC 2012 deviation block start */
    /* MISRA C-2012 Rule 2.2 deviated in this file.  Devi

    NVMCTRL_REGS->NVMCTRL_CTRLB = NVMCTRL_CTRLB_RWS(3UL);

    STDIO_BufferModeSet();



    PORT_Initialize();

    CLOCK_Initialize();




    SERCOM3_USART_Initialize();

    NVMCTRL_Initialize( );


    SERCOM0_I2C_Initialize();
```

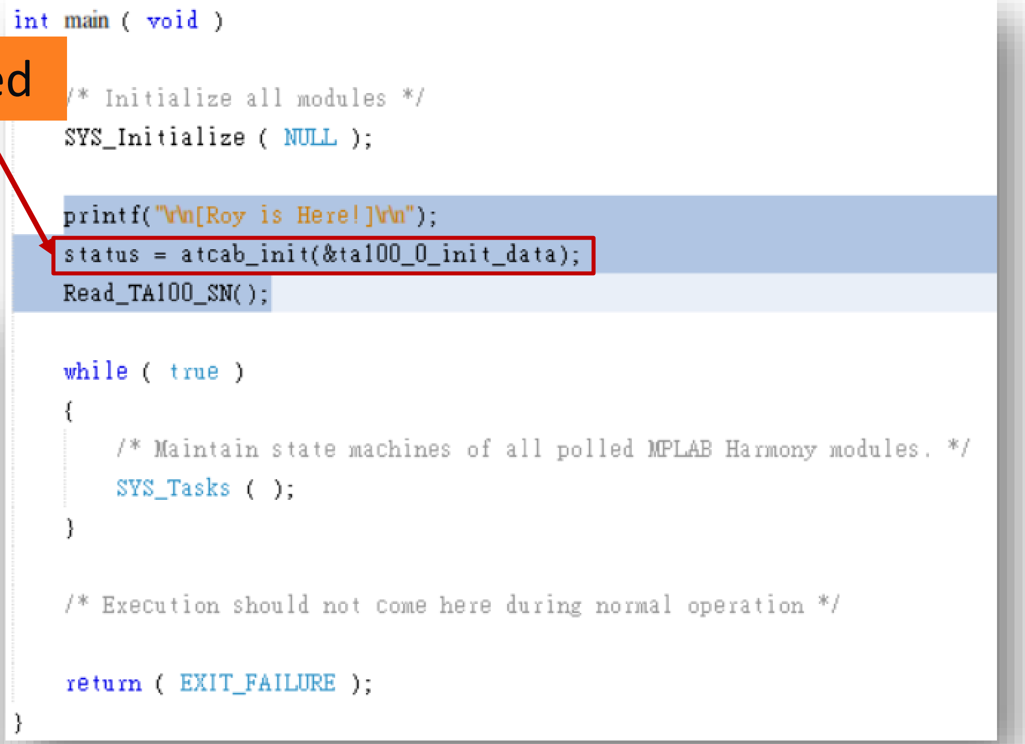


```
#include <stddef.h>                     // Defines NULL
#include <stdbool.h>                    // Defines true
#include <stdlib.h>                     // Defines EXIT_FAILURE
#include "definitions.h"                // SYS function prototypes
#include "config/default/library/cryptoauthlib/cryptoauthlib.h"
#include "main.h"
extern ATCAIfaceCfg ta100_0_init_data;
```

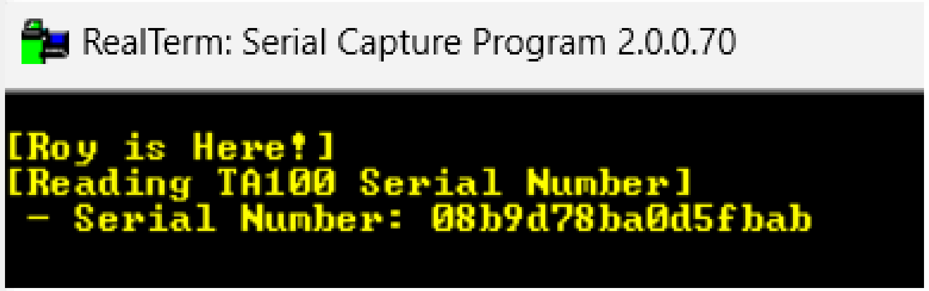
Extra added



```
// Section: Main Entry Point
// ******************************************************************************
// ******************************************************************************
#define CHECK_STATUS(s)
if(s != ATCA_SUCCESS)
{
    printf("Error: Line %d in %s\r\n", __LINE__, __FILE__);
        printf("STATUS = %X\r\n", s);
        printf("Code explanations can be found in atca_status.h \r\n");
        while(1);
}
void print_bytes(uint8_t * ptr, uint16_t length)
{
    uint16_t i = 0;
    for(i=0;i < length; i++)
    {
        printf("%02x",ptr[i]);
    }
    printf("\r\n");
}
void Read_TA100_SN(void)
{
     printf("[Reading TA100 Serial Number]\r\n");     // Prints beginning message
     status = talib_info_serial_number(atcab_get_device(), data_buf);
     CHECK_STATUS(status);
     printf(" - Serial Number: ");
     print_bytes(data_buf, 8);
}
```



```
int main ( void )
{
    /* Initialize all modules */
    SYS_Initialize ( NULL );

    printf("\r\n[Roy is Here!]\r\n");
    status = atcab_init(&ta100_0_init_data);
    Read_TA100_SN();

    while ( true )
    {
        /* Maintain state machines of all polled MPLAB Harmony modules. */
        SYS_Tasks ( );
    }

    /* Execution should not come here during normal operation */

    return ( EXIT_FAILURE );
}
```



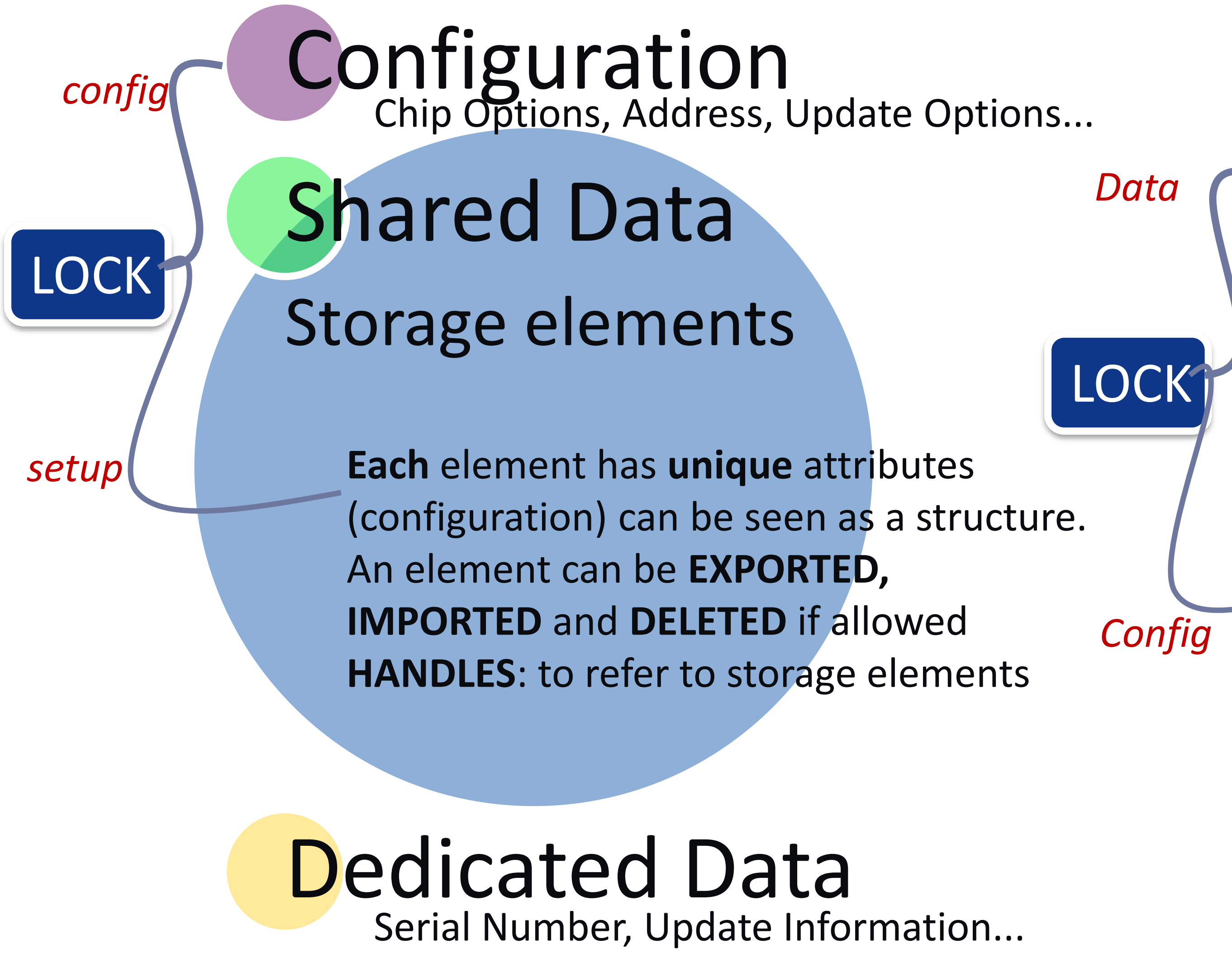
RealTerm: Serial Capture Program 2.0.0.70

```
[Roy is Here!]
[Reading TA100 Serial Number]
 - Serial Number: 08b9d78ba0d5fbab
```

TA100 Elements/Handles Introduction

Device Memory Organization and Configuration

TA100 vs 608



Configuration

Chip Options, Address, Update Options...

Shared Data

Storage elements

Each element has **unique** attributes (configuration) can be seen as a structure.
An element can be **EXPORTED, IMPORTED** and **DELETED** if allowed
HANDLES: to refer to storage elements

Dedicated Data

Serial Number, Update Information...

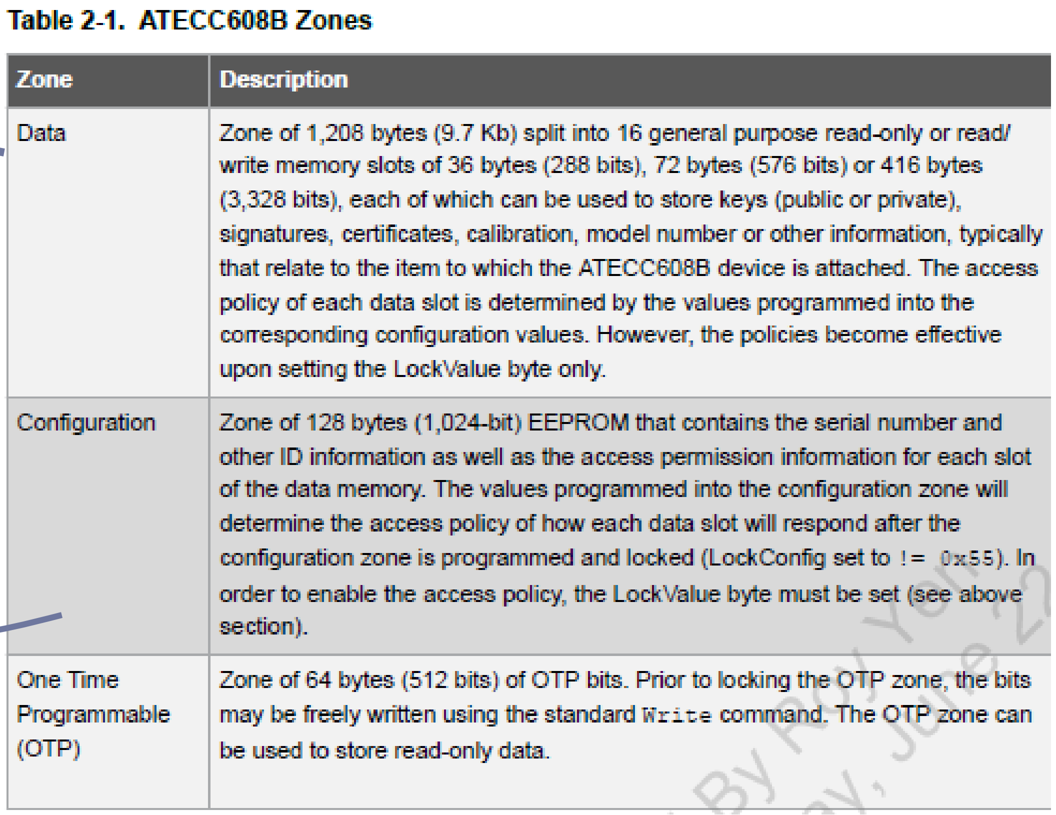
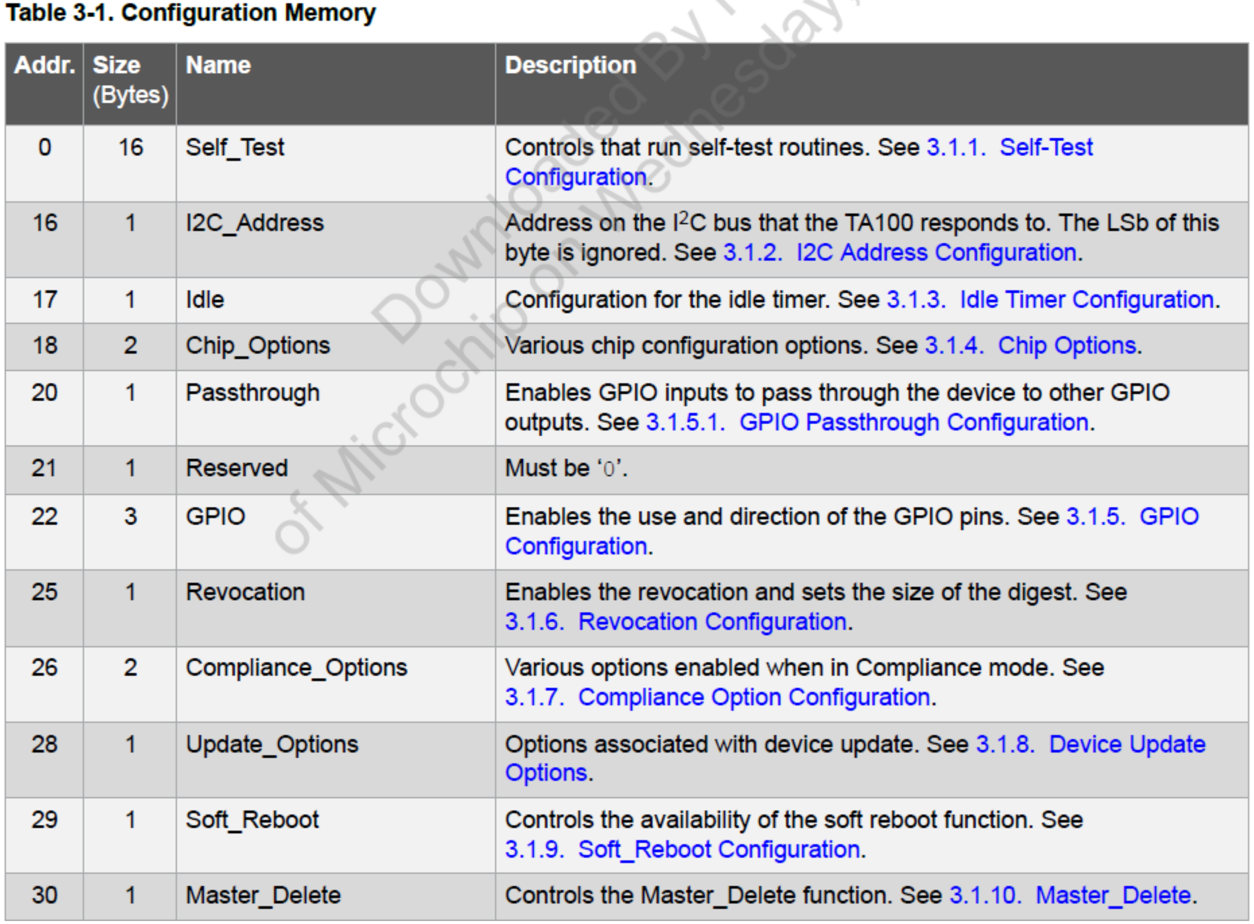


Table 2-1. ATECC608B Zones

Zone	Description
Data	Zone of 1,208 bytes (9.7 Kb) split into 16 general purpose read-only or read/write memory slots of 36 bytes (288 bits), 72 bytes (576 bits) or 416 bytes (3,328 bits), each of which can be used to store keys (public or private), signatures, certificates, calibration, model number or other information, typically that relate to the item to which the ATECC608B device is attached. The access policy of each data slot is determined by the values programmed into the corresponding configuration values. However, the policies become effective upon setting the LockValue byte only.
Configuration	Zone of 128 bytes (1,024-bit) EEPROM that contains the serial number and other ID information as well as the access permission information for each slot of the data memory. The values programmed into the configuration zone will determine the access policy of how each data slot will respond after the configuration zone is programmed and locked (LockConfig set to `!= 0x55`). In order to enable the access policy, the LockValue byte must be set (see above section).
One Time Programmable (OTP)	Zone of 64 bytes (512 bits) of OTP bits. Prior to locking the OTP zone, the bits may be freely written using the standard `Write` command. The OTP zone can be used to store read-only data.

Microchip Proprietary and STRICTLY Confidential

Configuration



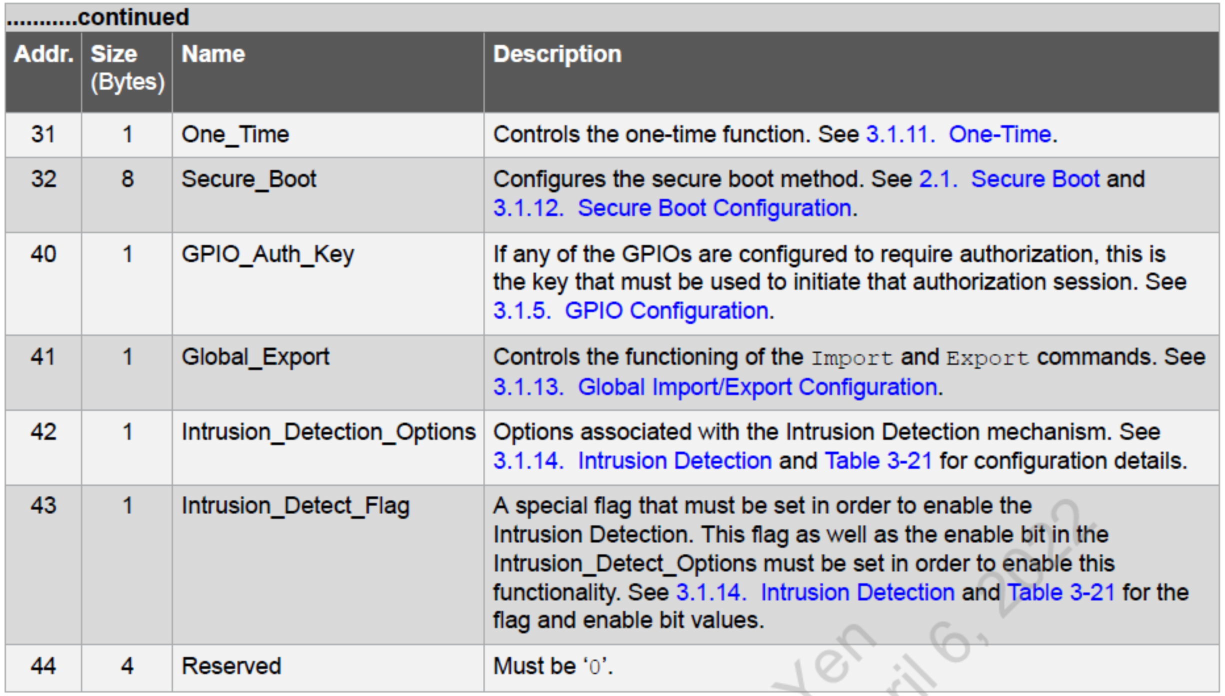
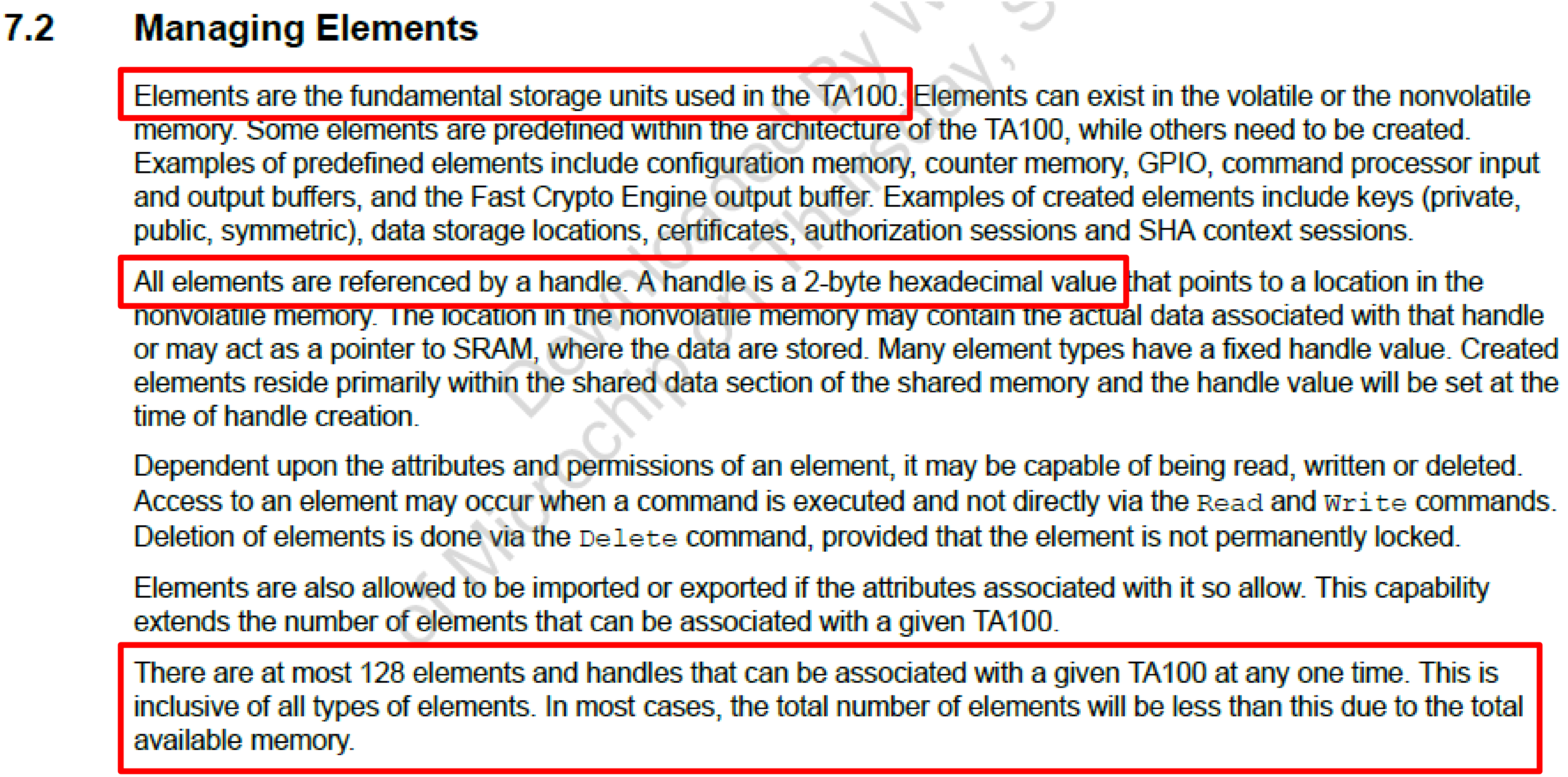


Table 3-1. Configuration Memory

Addr.	Size (Bytes)	Name	Description
0	16	Self_Test	Controls that run self-test routines. See 3.1.1. Self-Test Configuration.
16	1	I2C_Address	Address on the I^{2}C bus that the TA100 responds to. The LSb of this byte is ignored. See 3.1.2. I2C Address Configuration.
17	1	Idle	Configuration for the idle timer. See 3.1.3. Idle Timer Configuration.
18	2	Chip_Options	Various chip configuration options. See 3.1.4. Chip Options.
20	1	Passthrough	Enables GPIO inputs to pass through the device to other GPIO outputs. See 3.1.5.1. GPIO Passthrough Configuration.
21	1	Reserved	Must be '0'.
22	3	GPIO	Enables the use and direction of the GPIO pins. See 3.1.5. GPIO Configuration.
25	1	Revocation	Enables the revocation and sets the size of the digest. See 3.1.6. Revocation Configuration.
26	2	Compliance_Options	Various options enabled when in Compliance mode. See 3.1.7. Compliance Option Configuration.
28	1	Update_Options	Options associated with device update. See 3.1.8. Device Update Options.
29	1	Soft_Reboot	Controls the availability of the soft reboot function. See 3.1.9. Soft_Reboot Configuration.
30	1	Master_Delete	Controls the Master_Delete function. See 3.1.10. Master_Delete.
31	1	One_Time	Controls the one-time function. See 3.1.11. One-Time.
32	8	Secure_Boot	Configures the secure boot method. See 2.1. Secure Boot and 3.1.12. Secure Boot Configuration.
40	1	GPIO_Auth_Key	If any of the GPIOs are configured to require authorization, this is the key that must be used to initiate that authorization session. See 3.1.5. GPIO Configuration.
41	1	Global_Export	Controls the functioning of the `Import` and `Export` commands. See 3.1.13. Global Import/Export Configuration.
42	1	Intrusion_Detection_Options	Options associated with the Intrusion Detection mechanism. See 3.1.14. Intrusion Detection and Table 3-21 for configuration details.
43	1	Intrusion_Detect_Flag	A special flag that must be set in order to enable the Intrusion Detection. This flag as well as the enable bit in the Intrusion_Detect_Options must be set in order to enable this functionality. See 3.1.14. Intrusion Detection and Table 3-21 for the flag and enable bit values.
44	4	Reserved	Must be '0'.

```
//TA100 Configuration Bytes
static const uint8_t configuration[48] = {
    0x00, 0x00, 0xEB, 0x77, 0x00, 0x00, 0xB9, 0x7D, 0x00, 0x00, 0xB9, 0xD7, 0x00, 0x02, 0xAB, 0xDD,
    0x2E, 0x03, 0x00, 0x1F, 0x03, 0x00, 0x00, 0x00, 0x00, 0x07, 0x3F, 0x00, 0x1F, 0x23, 0x21, 0xF3,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x0F, 0x93, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00
};
```

Elements / Handles



7.2 Managing Elements

Elements are the fundamental storage units used in the TA100. Elements can exist in the volatile or the nonvolatile memory. Some elements are predefined within the architecture of the TA100, while others need to be created. Examples of predefined elements include configuration memory, counter memory, GPIO, command processor input and output buffers, and the Fast Crypto Engine output buffer. Examples of created elements include keys (private, public, symmetric), data storage locations, certificates, authorization sessions and SHA context sessions.

All elements are referenced by a handle. A handle is a 2-byte hexadecimal value that points to a location in the nonvolatile memory. The location in the nonvolatile memory may contain the actual data associated with that handle or may act as a pointer to SRAM, where the data are stored. Many element types have a fixed handle value. Created elements reside primarily within the shared data section of the shared memory and the handle value will be set at the time of handle creation.

Dependent upon the attributes and permissions of an element, it may be capable of being read, written or deleted. Access to an element may occur when a command is executed and not directly via the `Read` and `Write` commands. Deletion of elements is done via the `Delete` command, provided that the element is not permanently locked.

Elements are also allowed to be imported or exported if the attributes associated with it so allow. This capability extends the number of elements that can be associated with a given TA100.

There are at most 128 elements and handles that can be associated with a given TA100 at any one time. This is inclusive of all types of elements. In most cases, the total number of elements will be less than this due to the total available memory.

Handle settings

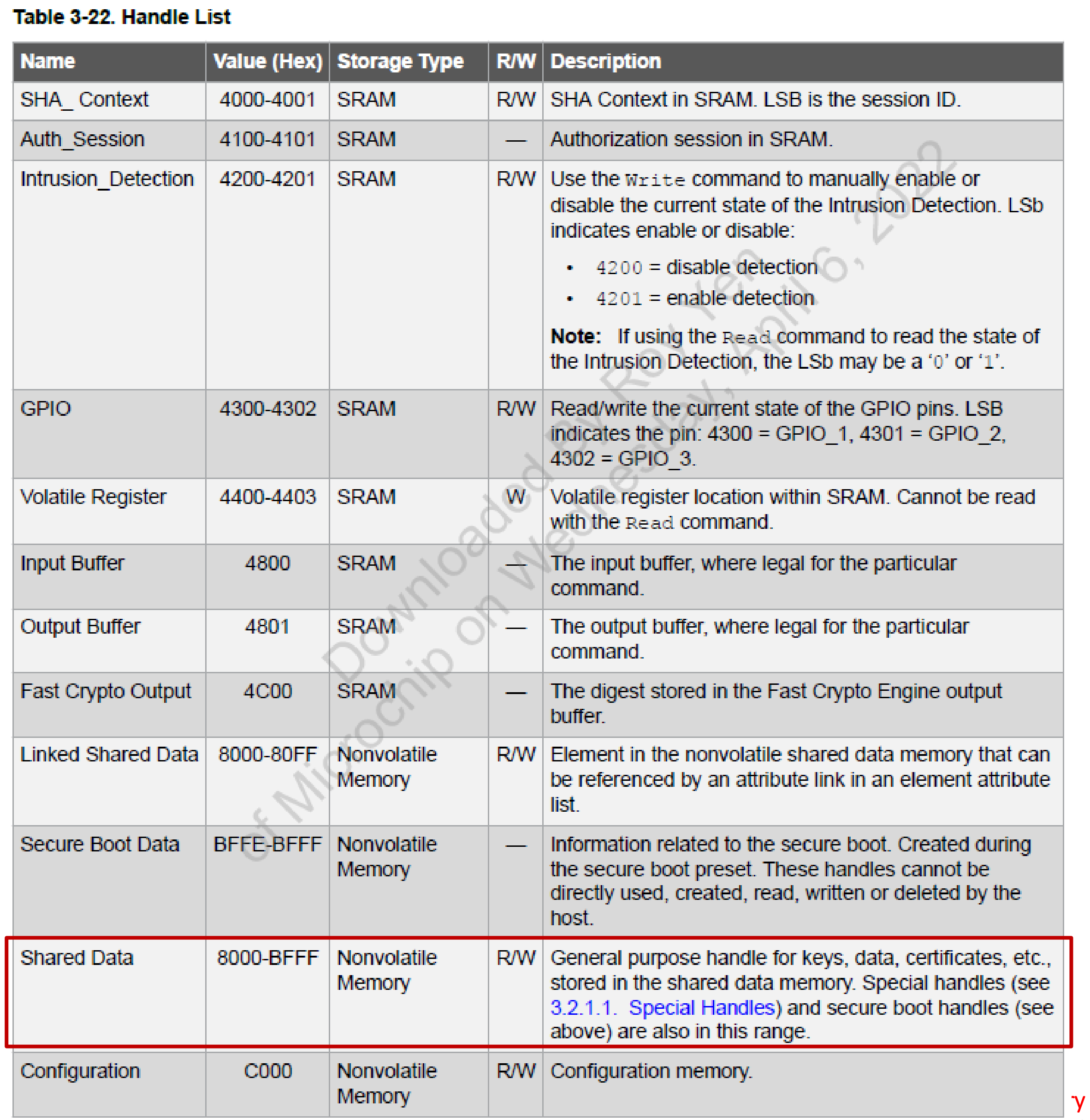


Table 3-22. Handle List

Name	Value (Hex)	Storage Type	R/W	Description
SHA_ Context	4000-4001	SRAM	R/W	SHA Context in SRAM. LSB is the session ID.
Auth_Session	4100-4101	SRAM	—	Authorization session in SRAM.
Intrusion_Detection	4200-4201	SRAM	R/W	Use the `Write` command to manually enable or disable the current state of the Intrusion Detection. LSb indicates enable or disable: • `4200` = disable detection • `4201` = enable detection **Note:** If using the `Read` command to read the state of the Intrusion Detection, the LSb may be a '`0`' or '`1`'.
GPIO	4300-4302	SRAM	R/W	Read/write the current state of the GPIO pins. LSB indicates the pin: 4300 = GPIO_1, 4301 = GPIO_2, 4302 = GPIO_3.
Volatile Register	4400-4403	SRAM	W	Volatile register location within SRAM. Cannot be read with the `Read` command.
Input Buffer	4800	SRAM	—	The input buffer, where legal for the particular command.
Output Buffer	4801	SRAM	—	The output buffer, where legal for the particular command.
Fast Crypto Output	4C00	SRAM	—	The digest stored in the Fast Crypto Engine output buffer.
Linked Shared Data	8000-80FF	Nonvolatile Memory	R/W	Element in the nonvolatile shared data memory that can be referenced by an attribute link in an element attribute list.
Secure Boot Data	BFFE-BFFF	Nonvolatile Memory	—	Information related to the secure boot. Created during the secure boot preset. These handles cannot be directly used, created, read, written or deleted by the host.
Shared Data	8000-BFFF	Nonvolatile Memory	R/W	General purpose handle for keys, data, certificates, etc., stored in the shared data memory. Special handles (see 3.2.1.1. Special Handles) and secure boot handles (see above) are also in this range.
Configuration	C000	Nonvolatile Memory	R/W	Configuration memory.

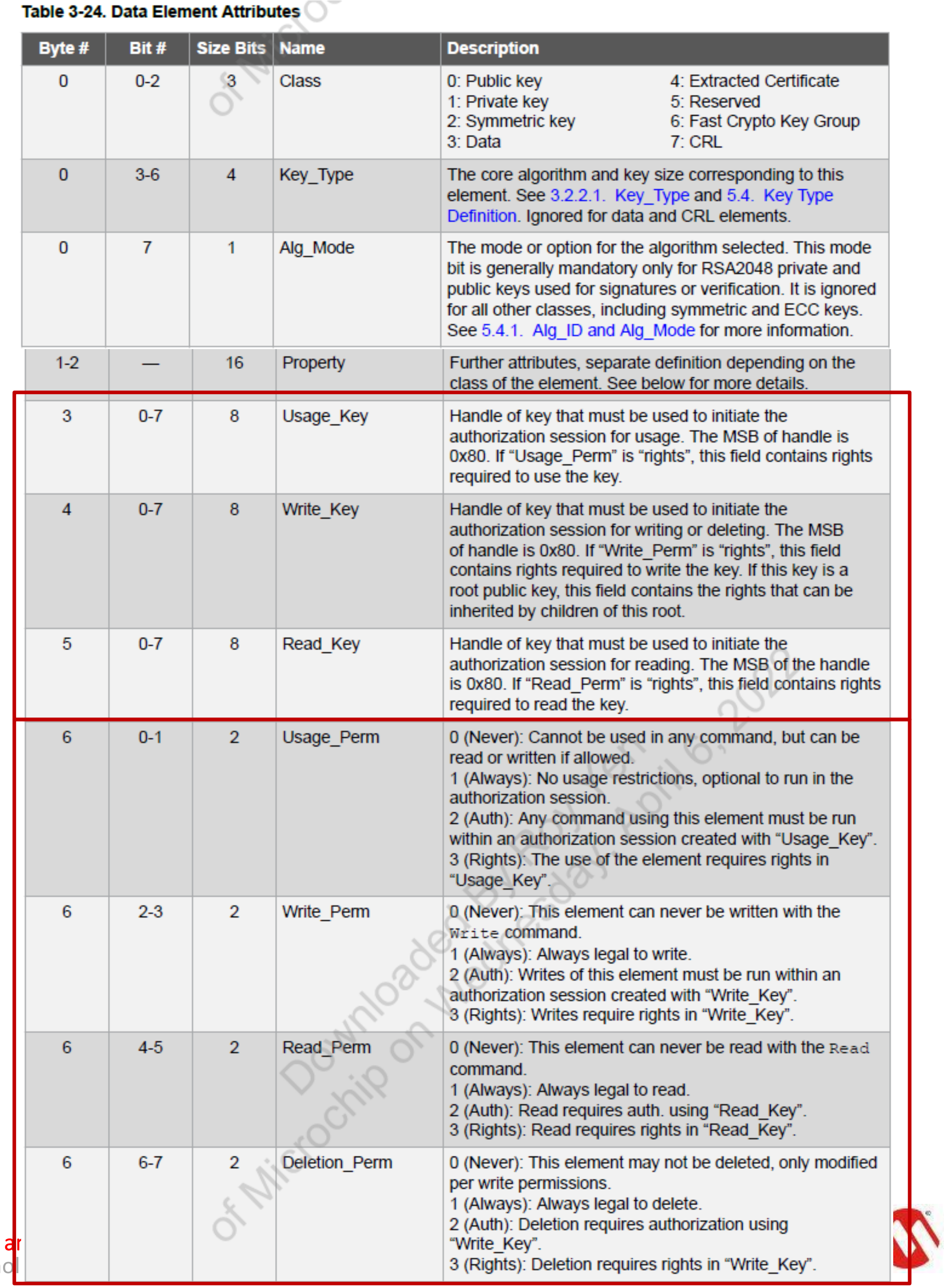


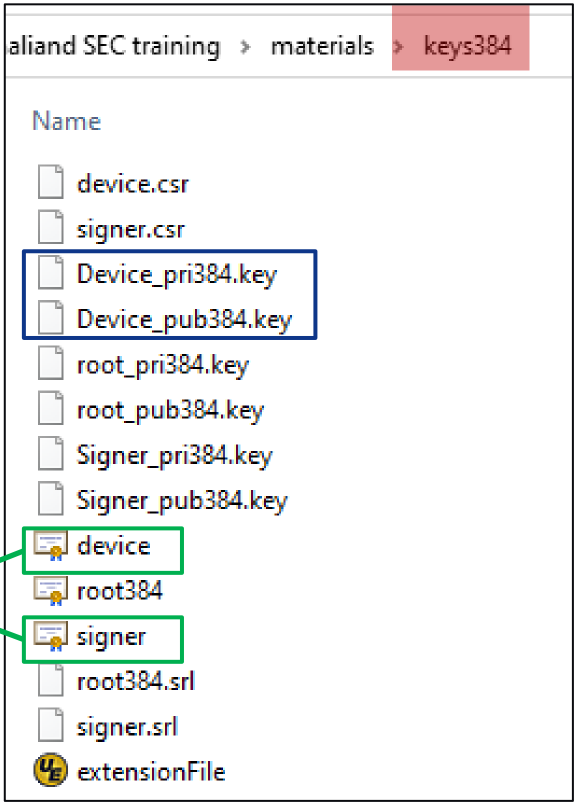
Table 3-24. Data Element Attributes

Byte #	Bit #	Size Bits	Name	Description
0	0-2	3	Class	0: Public key 1: Private key 2: Symmetric key 3: Data 4: Extracted Certificate 5: Reserved 6: Fast Crypto Key Group 7: CRL
0	3-6	4	Key_Type	The core algorithm and key size corresponding to this element. See 3.2.2.1. Key_Type and 5.4. Key Type Definition. Ignored for data and CRL elements.
0	7	1	Alg_Mode	The mode or option for the algorithm selected. This mode bit is generally mandatory only for RSA2048 private and public keys used for signatures or verification. It is ignored for all other classes, including symmetric and ECC keys. See 5.4.1. Alg_ID and Alg_Mode for more information.
1-2	—	16	Property	Further attributes, separate definition depending on the class of the element. See below for more details.
3	0-7	8	Usage_Key	Handle of key that must be used to initiate the authorization session for usage. The MSB of handle is 0x80. If "Usage_Perm" is "rights", this field contains rights required to use the key.
4	0-7	8	Write_Key	Handle of key that must be used to initiate the authorization session for writing or deleting. The MSB of handle is 0x80. If "Write_Perm" is "rights", this field contains rights required to write the key. If this key is a root public key, this field contains the rights that can be inherited by children of this root.
5	0-7	8	Read_Key	Handle of key that must be used to initiate the authorization session for reading. The MSB of the handle is 0x80. If "Read_Perm" is "rights", this field contains rights required to read the key.
6	0-1	2	Usage_Perm	0 (Never): Cannot be used in any command, but can be read or written if allowed. 1 (Always): No usage restrictions, optional to run in the authorization session. 2 (Auth): Any command using this element must be run within an authorization session created with "Usage_Key". 3 (Rights): The use of the element requires rights in "Usage_Key".
6	2-3	2	Write_Perm	0 (Never): This element can never be written with the `Write` command. 1 (Always): Always legal to write. 2 (Auth): Writes of this element must be run within an authorization session created with "Write_Key". 3 (Rights): Writes require rights in "Write_Key".
6	4-5	2	Read_Perm	0 (Never): This element can never be read with the `Read` command. 1 (Always): Always legal to read. 2 (Auth): Read requires auth. using "Read_Key". 3 (Rights): Read requires rights in "Read_Key".
6	6-7	2	Deletion_Perm	0 (Never): This element may not be deleted, only modified per write permissions. 1 (Always): Always legal to delete. 2 (Auth): Deletion requires authorization using "Write_Key". 3 (Rights): Deletion requires rights in "Write_Key".

Lab Elements for Labs

What we used in Lab4 test code

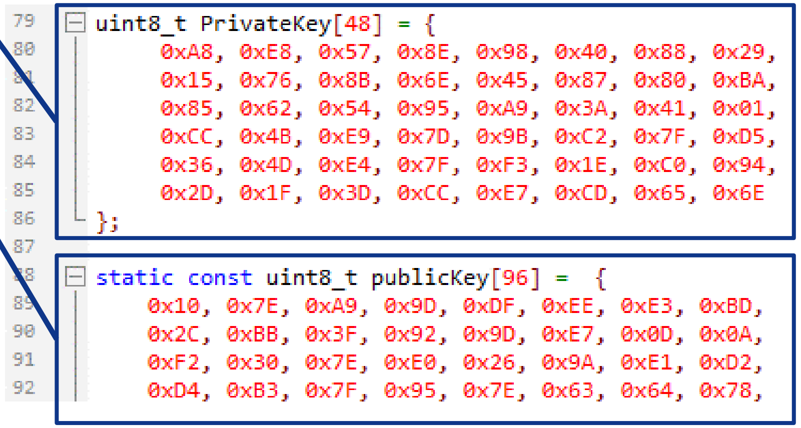
Element Name	Class	Type/Notes
Private Key @ 0x8007	Private Key	ECC P384 (Fixed Private Key)
Public Key @ 0x8006	Public Key	ECC P384 (Fixed paired Public key)
Private Key @ 0x8008	Private Key	ECC P384 (**Random Generated** Device Private Key)
Public Key @ 0x800C	Public Key	ECC P384 (the paired Device Public key)
signerCert @ 0x8201	Data	Example Signer Certificate (Hold signer Public key)
deviceCert @ 0x8200	Data	Example Device Certificate (Hold Device Public key)



aliand SEC training › materials › keys384

Name

- device.csr
- signer.csr
- Device_pri384.key
- Device_pub384.key
- root_pri384.key
- root_pub384.key
- Signer_pri384.key
- Signer_pub384.key
- device
- root384
- signer
- root384.srl
- signer.srl
- extensionFile



```
uint8_t PrivateKey[48] = {
        0xA8, 0xE8, 0x57, 0x8E, 0x98, 0x40, 0x88, 0x29,
        0x15, 0x76, 0x8B, 0x6E, 0x45, 0x87, 0x80, 0xBA,
        0x85, 0x62, 0x54, 0x95, 0xA9, 0x3A, 0x41, 0x01,
        0xCC, 0x4B, 0xE9, 0x7D, 0x9B, 0xC2, 0x7F, 0xD5,
        0x36, 0x4D, 0xE4, 0x7F, 0xF3, 0x1E, 0xC0, 0x94,
        0x2D, 0x1F, 0x3D, 0xCC, 0xE7, 0xCD, 0x65, 0x6E
};

static const uint8_t publicKey[96] =  {
       0x10, 0x7E, 0xA9, 0x9D, 0xDF, 0xEE, 0xE3, 0xBD,
       0x2C, 0xBB, 0x3F, 0x92, 0x9D, 0xE7, 0x0D, 0x0A,
       0xF2, 0x30, 0x7E, 0xE0, 0x26, 0x9A, 0xE1, 0xD2,
       0xD4, 0xB3, 0x7F, 0x95, 0x7E, 0x63, 0x64, 0x78,
```

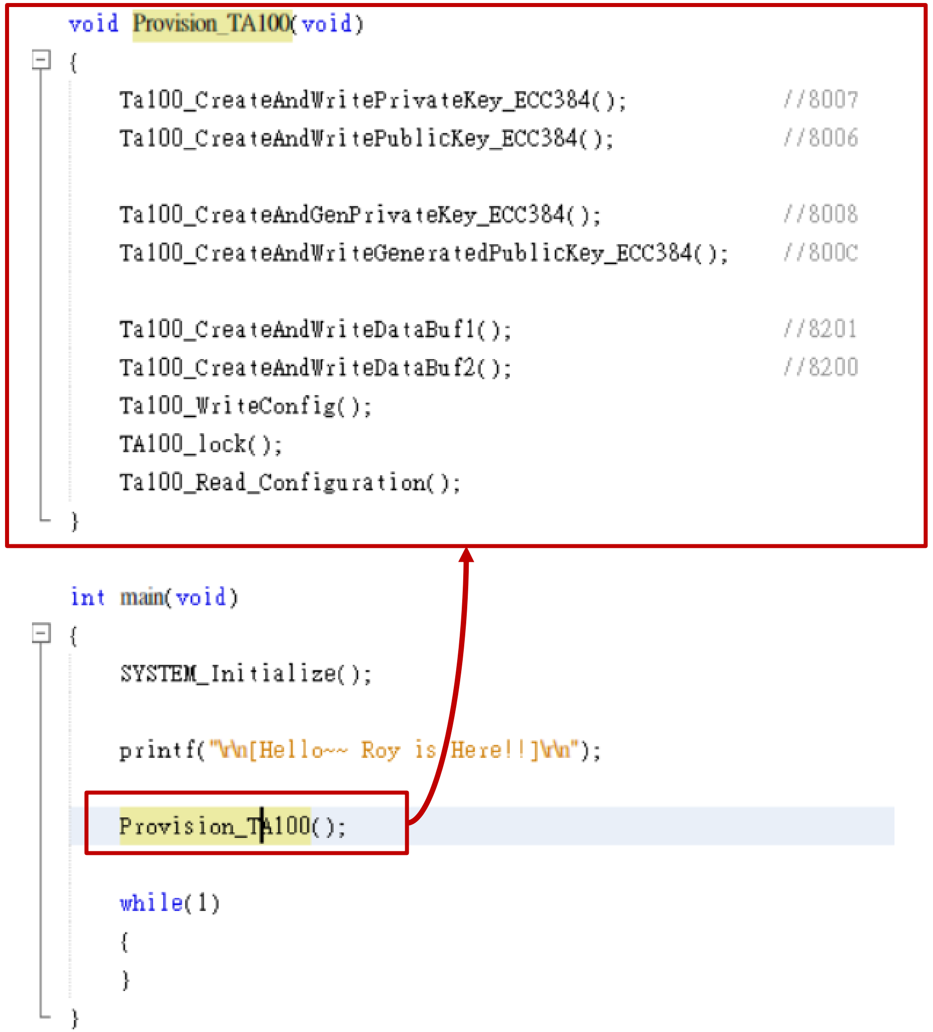
Microchip Proprietary and STRICTLY Confidential

Lab6 - Provision TA100

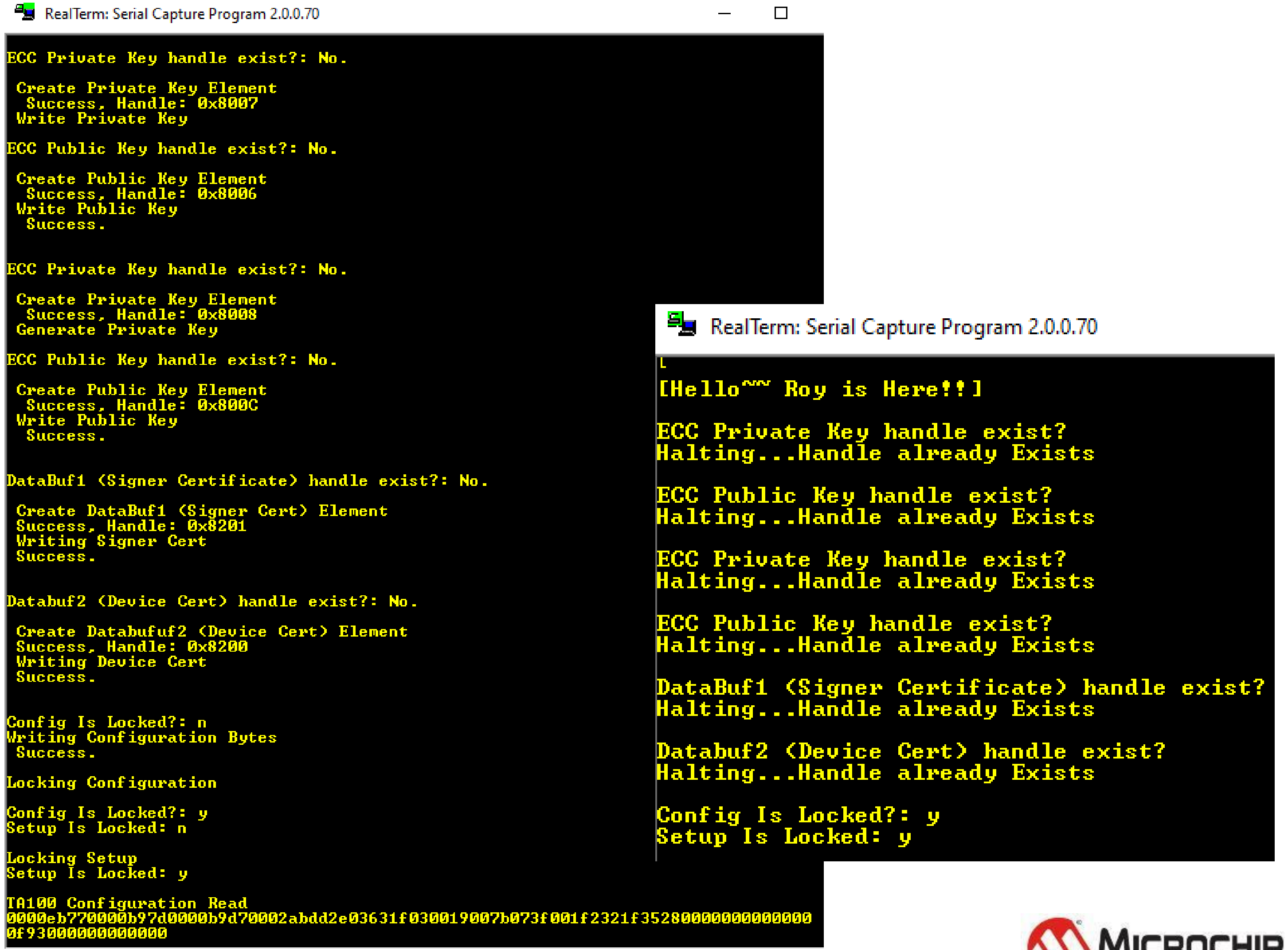
Use: Lab6.txt

Step 6-1

- **Copy functions from Lab6.txt to main.c , Program & Run!**



```c
void Provision_TA100(void)
{
    Ta100_CreateAndWritePrivateKey_ECC384();            //8007
    Ta100_CreateAndWritePublicKey_ECC384();             //8006

    Ta100_CreateAndGenPrivateKey_ECC384();              //8008
    Ta100_CreateAndWriteGeneratedPublicKey_ECC384();    //800C

    Ta100_CreateAndWriteDataBuf1();                     //8201
    Ta100_CreateAndWriteDataBuf2();                     //8200
    Ta100_WriteConfig();
    TA100_lock();
    Ta100_Read_Configuration();
}

int main(void)
{
    SYSTEM_Initialize();

    printf("\r\n[Hello~~ Roy is Here!!]\r\n");

    Provision_TA100();

    while(1)
    {
    }
}
```

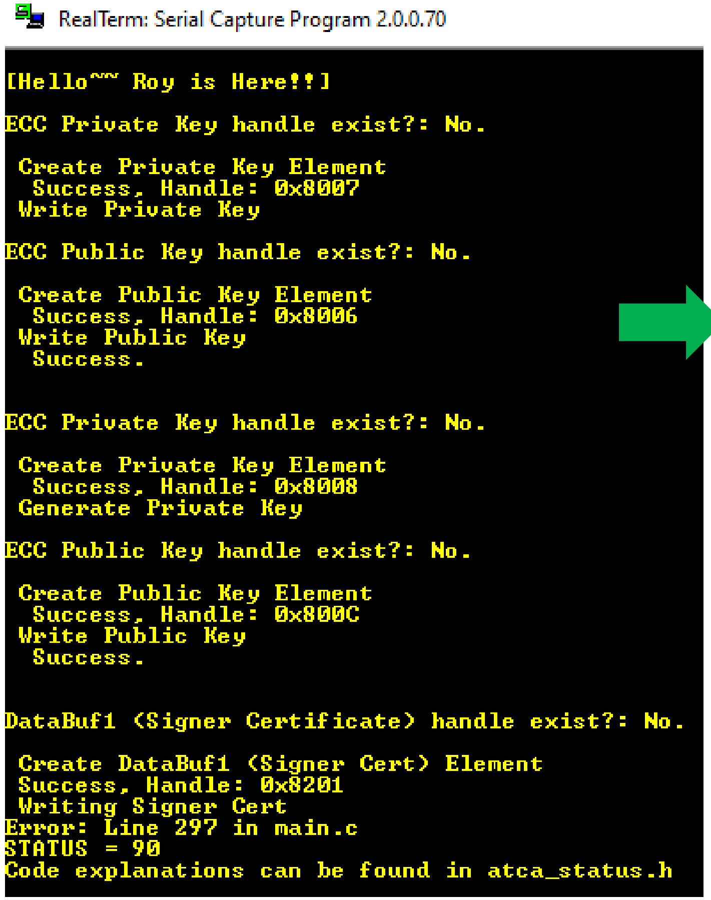


```
ECC Private Key handle exist?: No.
 Create Private Key Element
  Success, Handle: 0x8007
 Write Private Key
ECC Public Key handle exist?: No.
 Create Public Key Element
  Success, Handle: 0x8006
 Write Public Key
  Success.

ECC Private Key handle exist?: No.
 Create Private Key Element
  Success, Handle: 0x8008
 Generate Private Key
ECC Public Key handle exist?: No.
 Create Public Key Element
  Success, Handle: 0x800C
 Write Public Key
  Success.

DataBuf1 (Signer Certificate) handle exist?: No.
 Create DataBuf1 (Signer Cert) Element
 Success, Handle: 0x8201
 Writing Signer Cert
 Success.

Databuf2 (Device Cert) handle exist?: No.
 Create Databufuf2 (Device Cert) Element
 Success, Handle: 0x8200
 Writing Device Cert
 Success.

Config Is Locked?: n
Writing Configuration Bytes
 Success.
Locking Configuration

Config Is Locked?: y
Setup Is Locked: n

Locking Setup
Setup Is Locked: y

TA100 Configuration Read
0000eb770000b97d0000b9d70002abdd2e03631f030019007b073f001f2321f35280000000000000
0f93000000000000
```

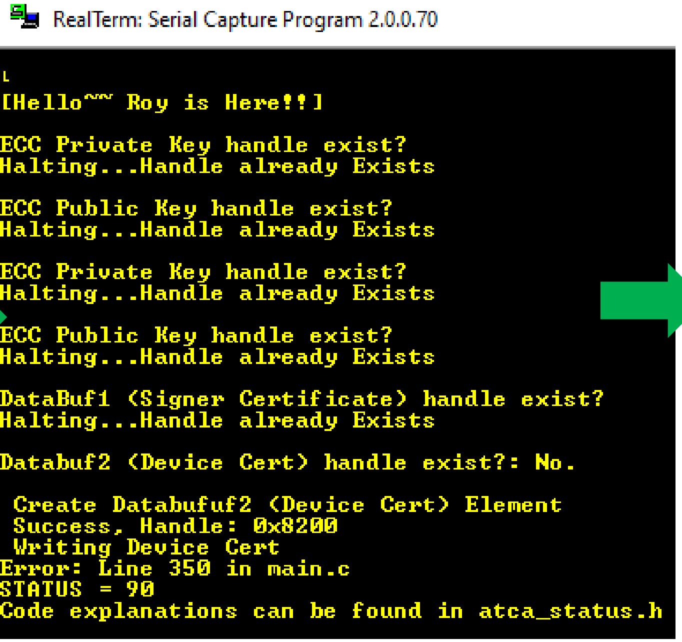
```
[Hello~~ Roy is Here!!]

ECC Private Key handle exist?
Halting...Handle already Exists

ECC Public Key handle exist?
Halting...Handle already Exists

ECC Private Key handle exist?
Halting...Handle already Exists

ECC Public Key handle exist?
Halting...Handle already Exists

DataBuf1 (Signer Certificate) handle exist?
Halting...Handle already Exists

Databuf2 (Device Cert) handle exist?
Halting...Handle already Exists

Config Is Locked?: y
Setup Is Locked: y
```

Once Failed message show up?

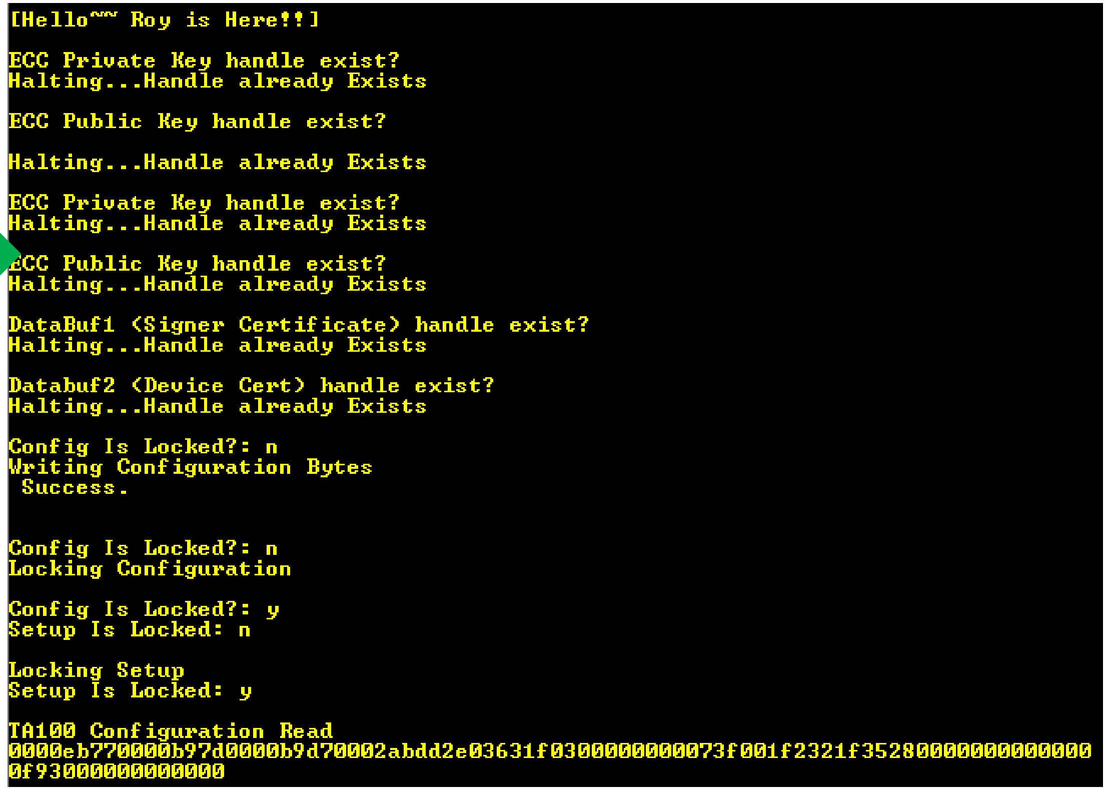
- Just press “Reset” Button to run the provision again.
- The provision just detects & passes “completed steps”



```
[Hello~~ Roy is Here!!]
ECC Private Key handle exist?: No.

 Create Private Key Element
  Success, Handle: 0x8007
 Write Private Key

ECC Public Key handle exist?: No.

 Create Public Key Element
  Success, Handle: 0x8006
 Write Public Key
  Success.


ECC Private Key handle exist?: No.

 Create Private Key Element
  Success, Handle: 0x8008
 Generate Private Key

ECC Public Key handle exist?: No.

 Create Public Key Element
  Success, Handle: 0x800C
 Write Public Key
  Success.


DataBuf1 (Signer Certificate) handle exist?: No.

 Create DataBuf1 (Signer Cert) Element
 Success, Handle: 0x8201
 Writing Signer Cert
Error: Line 297 in main.c
STATUS = 90
Code explanations can be found in atca_status.h
```



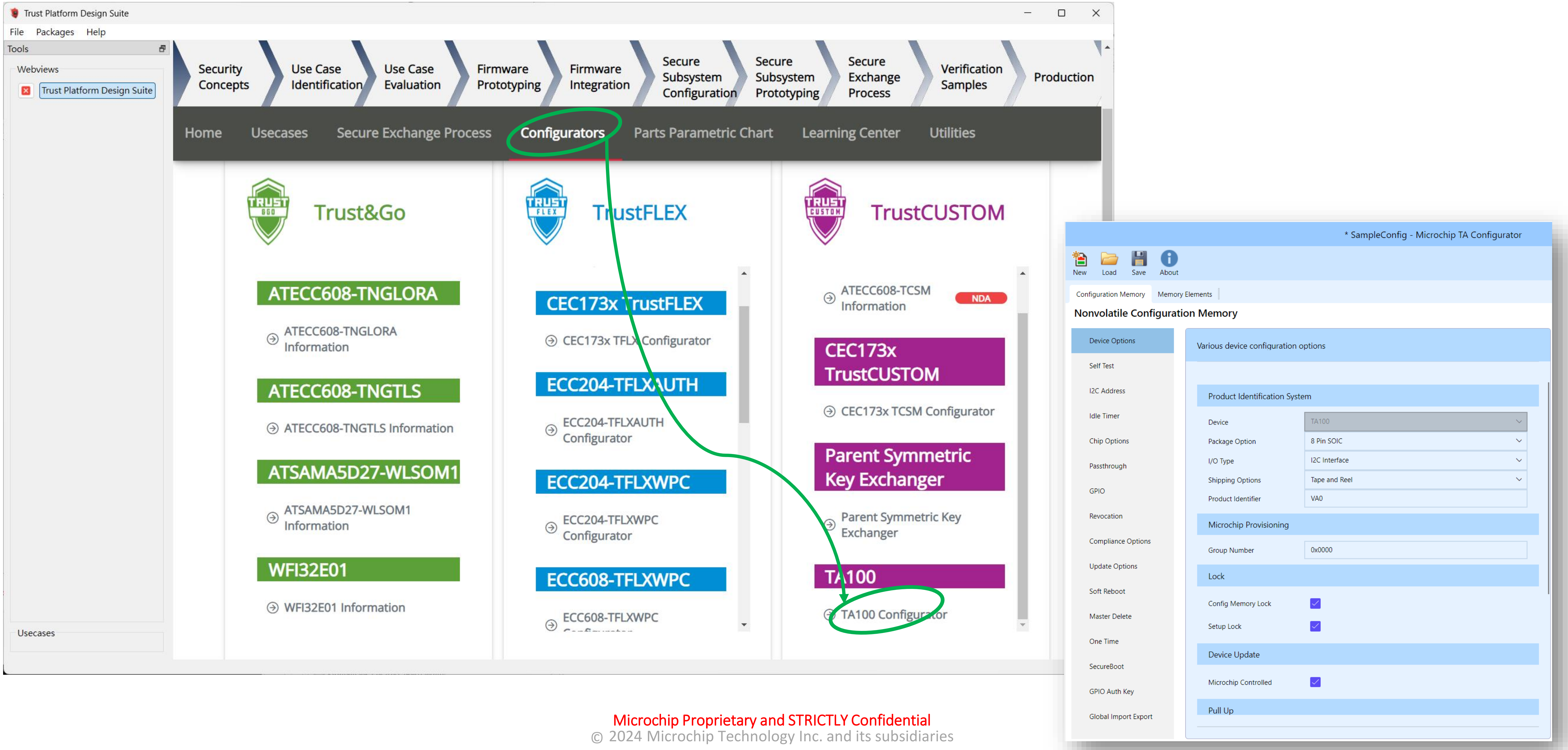
```
L
[Hello~~ Roy is Here!!]

ECC Private Key handle exist?
Halting...Handle already Exists

ECC Public Key handle exist?
Halting...Handle already Exists

ECC Private Key handle exist?
Halting...Handle already Exists

ECC Public Key handle exist?
Halting...Handle already Exists

DataBuf1 (Signer Certificate) handle exist?
Halting...Handle already Exists

Databuf2 (Device Cert) handle exist?: No.

 Create Databufuf2 (Device Cert) Element
 Success, Handle: 0x8200
 Writing Device Cert
Error: Line 350 in main.c
STATUS = 90
Code explanations can be found in atca_status.h
```



```
[Hello~~ Roy is Here!!]

ECC Private Key handle exist?
Halting...Handle already Exists

ECC Public Key handle exist?

Halting...Handle already Exists

ECC Private Key handle exist?
Halting...Handle already Exists

ECC Public Key handle exist?
Halting...Handle already Exists

DataBuf1 (Signer Certificate) handle exist?
Halting...Handle already Exists

Databuf2 (Device Cert) handle exist?
Halting...Handle already Exists

Config Is Locked?: n
Writing Configuration Bytes
 Success.


Config Is Locked?: n
Locking Configuration

Config Is Locked?: y
Setup Is Locked: n

Locking Setup
Setup Is Locked: y

TA100 Configuration Read
0000eb770000b97d0000b9d70002abdd2e03631f03000000000073f001f2321f35280000000000000
0f93000000000000
```

Microchip Proprietary and STRICTLY Confidential

TA100 Configurator - TPDS



Nonvolatile Configuration Memory

Various device configuration options

Product Identification System

Device	TA100
Package Option	8 Pin SOIC
I/O Type	I2C Interface
Shipping Options	Tape and Reel
Product Identifier	VA0

Microchip Provisioning

Group Number	0x0000

Lock

Config Memory Lock	☑
Setup Lock	☑

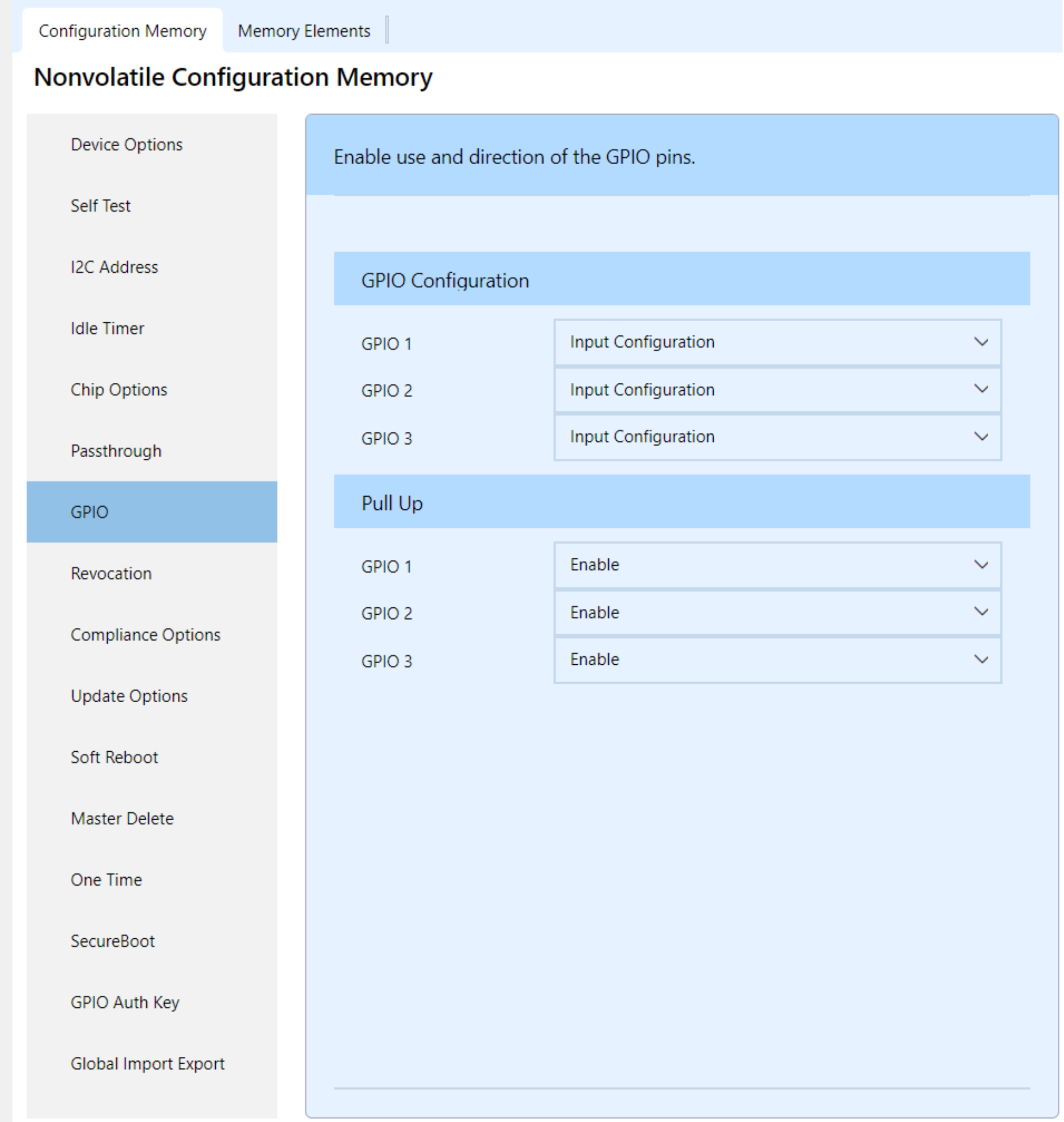
Device Update

Microchip Controlled	☑

Pull Up

Microchip Proprietary and STRICTLY Confidential

TA100 Configurator - TPDS



Nonvolatile Configuration Memory

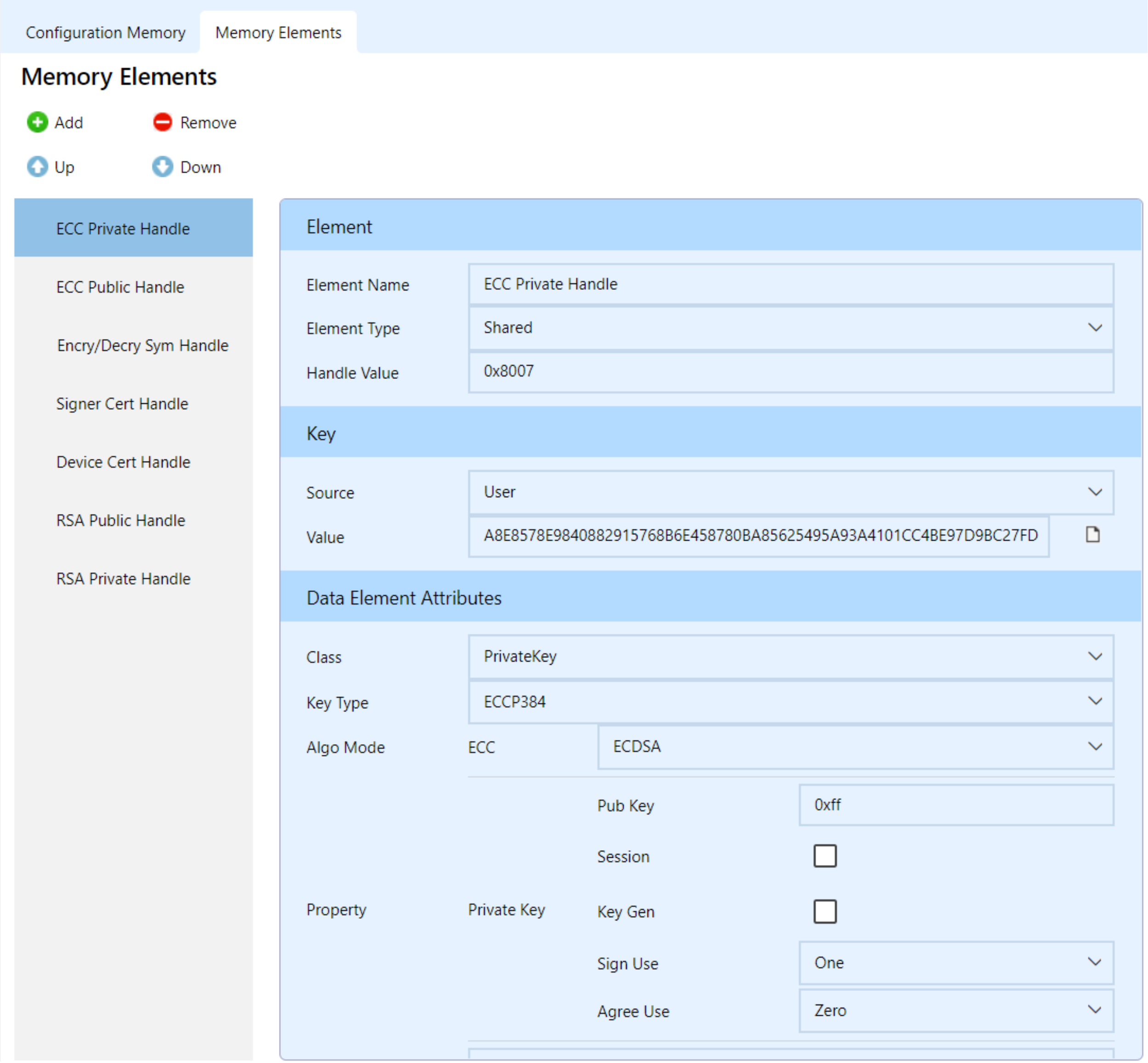
Enable use and direction of the GPIO pins.

GPIO Configuration

GPIO 1	Input Configuration
GPIO 2	Input Configuration
GPIO 3	Input Configuration

Pull Up

GPIO 1	Enable
GPIO 2	Enable
GPIO 3	Enable



Memory Elements

Element

Element Name	ECC Private Handle
Element Type	Shared
Handle Value	0x8007

Key

Source	User
Value	A8E8578E9840882915768B6E458780BA85625495A93A4101CC4BE97D9BC27FD

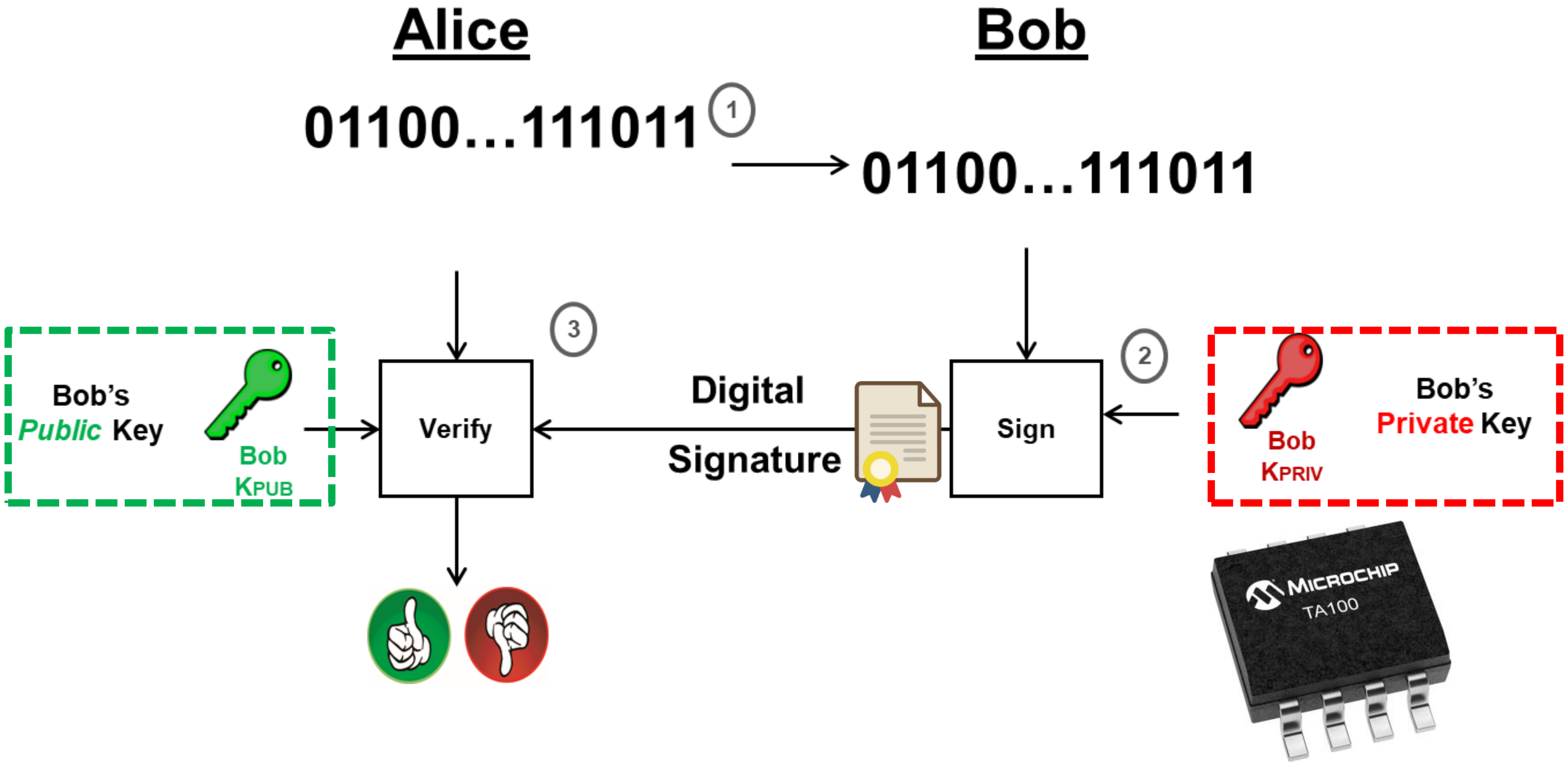
Data Element Attributes

Class	PrivateKey		
Key Type	ECCP384		
Algo Mode	ECC	ECDSA	
Property	Private Key	Pub Key	0xff
		Session	□
		Key Gen	□
		Sign Use	One
		Agree Use	Zero

Asymmetric Authentication

ECDSA P384 Sign & Verify

Asymmetric cipher



Alice

Bob

01100…111011 1

01100…111011

Bob's *Public* Key

Bob KPUB

Verify

3

Digital Signature

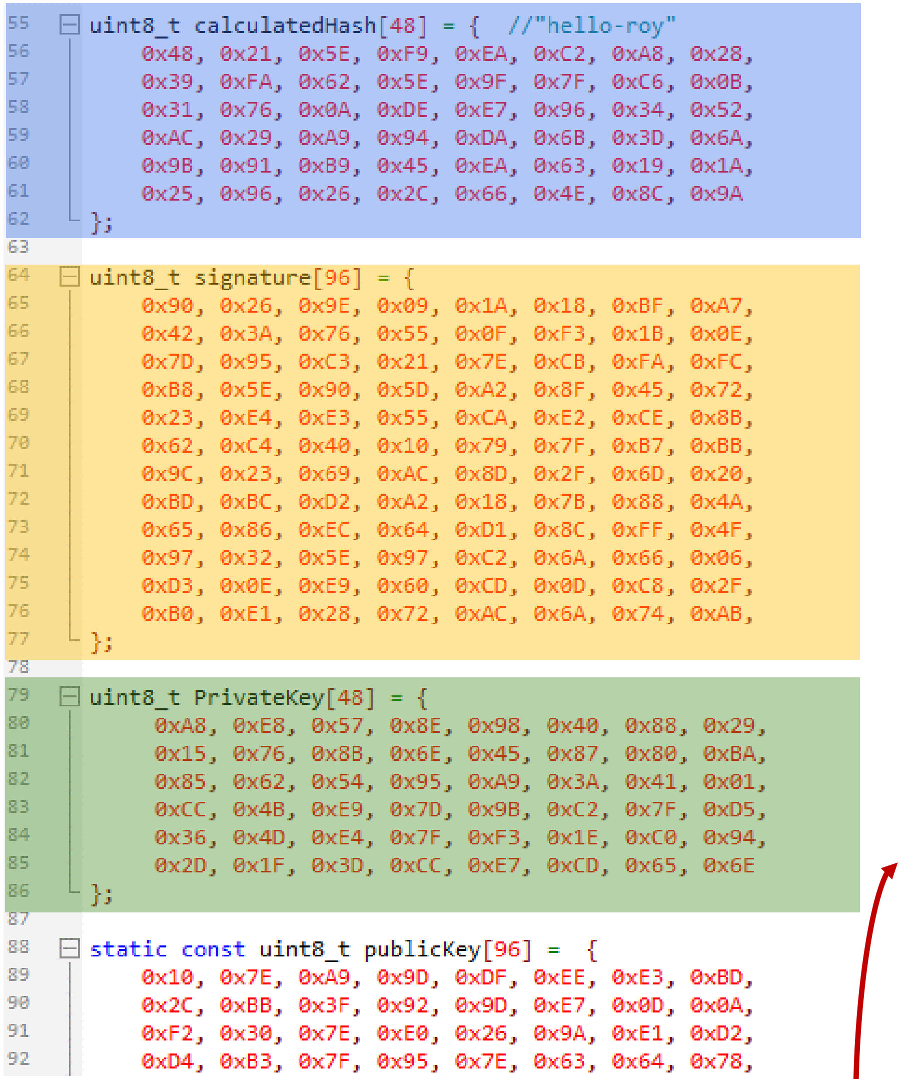
Sign

2

Bob's Private Key

Bob KPRIV

ECDSA P384 Sign



```
uint8_t calculatedHash[48] = {  //"hello-roy"
     0x48, 0x21, 0x5E, 0xF9, 0xEA, 0xC2, 0xA8, 0x28,
     0x39, 0xFA, 0x62, 0x5E, 0x9F, 0x7F, 0xC6, 0x0B,
     0x31, 0x76, 0x0A, 0xDE, 0xE7, 0x96, 0x34, 0x52,
     0xAC, 0x29, 0xA9, 0x94, 0xDA, 0x6B, 0x3D, 0x6A,
     0x9B, 0x91, 0xB9, 0x45, 0xEA, 0x63, 0x19, 0x1A,
     0x25, 0x96, 0x26, 0x2C, 0x66, 0x4E, 0x8C, 0x9A
};

uint8_t signature[96] = {
     0x90, 0x26, 0x9E, 0x09, 0x1A, 0x18, 0xBF, 0xA7,
     0x42, 0x3A, 0x76, 0x55, 0x0F, 0xF3, 0x1B, 0x0E,
     0x7D, 0x95, 0xC3, 0x21, 0x7E, 0xCB, 0xFA, 0xFC,
     0xB8, 0x5E, 0x90, 0x5D, 0xA2, 0x8F, 0x45, 0x72,
     0x23, 0xE4, 0xE3, 0x55, 0xCA, 0xE2, 0xCE, 0x8B,
     0x62, 0xC4, 0x40, 0x10, 0x79, 0x7F, 0xB7, 0xBB,
     0x9C, 0x23, 0x69, 0xAC, 0x8D, 0x2F, 0x6D, 0x20,
     0xBD, 0xBC, 0xD2, 0xA2, 0x18, 0x7B, 0x88, 0x4A,
     0x65, 0x86, 0xEC, 0x64, 0xD1, 0x8C, 0xFF, 0x4F,
     0x97, 0x32, 0x5E, 0x97, 0xC2, 0x6A, 0x66, 0x06,
     0xD3, 0x0E, 0xE9, 0x60, 0xCD, 0x0D, 0xC8, 0x2F,
     0xB0, 0xE1, 0x28, 0x72, 0xAC, 0x6A, 0x74, 0xAB,
};

uint8_t PrivateKey[48] = {
      0xA8, 0xE8, 0x57, 0x8E, 0x98, 0x40, 0x88, 0x29,
      0x15, 0x76, 0x8B, 0x6E, 0x45, 0x87, 0x80, 0xBA,
      0x85, 0x62, 0x54, 0x95, 0xA9, 0x3A, 0x41, 0x01,
      0xCC, 0x4B, 0xE9, 0x7D, 0x9B, 0xC2, 0x7F, 0xD5,
      0x36, 0x4D, 0xE4, 0x7F, 0xF3, 0x1E, 0xC0, 0x94,
      0x2D, 0x1F, 0x3D, 0xCC, 0xE7, 0xCD, 0x65, 0x6E
};

static const uint8_t publicKey[96] =  {
     0x10, 0x7E, 0xA9, 0x9D, 0xDF, 0xEE, 0xE3, 0xBD,
     0x2C, 0xBB, 0x3F, 0x92, 0x9D, 0xE7, 0x0D, 0x0A,
     0xF2, 0x30, 0x7E, 0xE0, 0x26, 0x9A, 0xE1, 0xD2,
     0xD4, 0xB3, 0x7F, 0x95, 0x7E, 0x63, 0x64, 0x78,
```

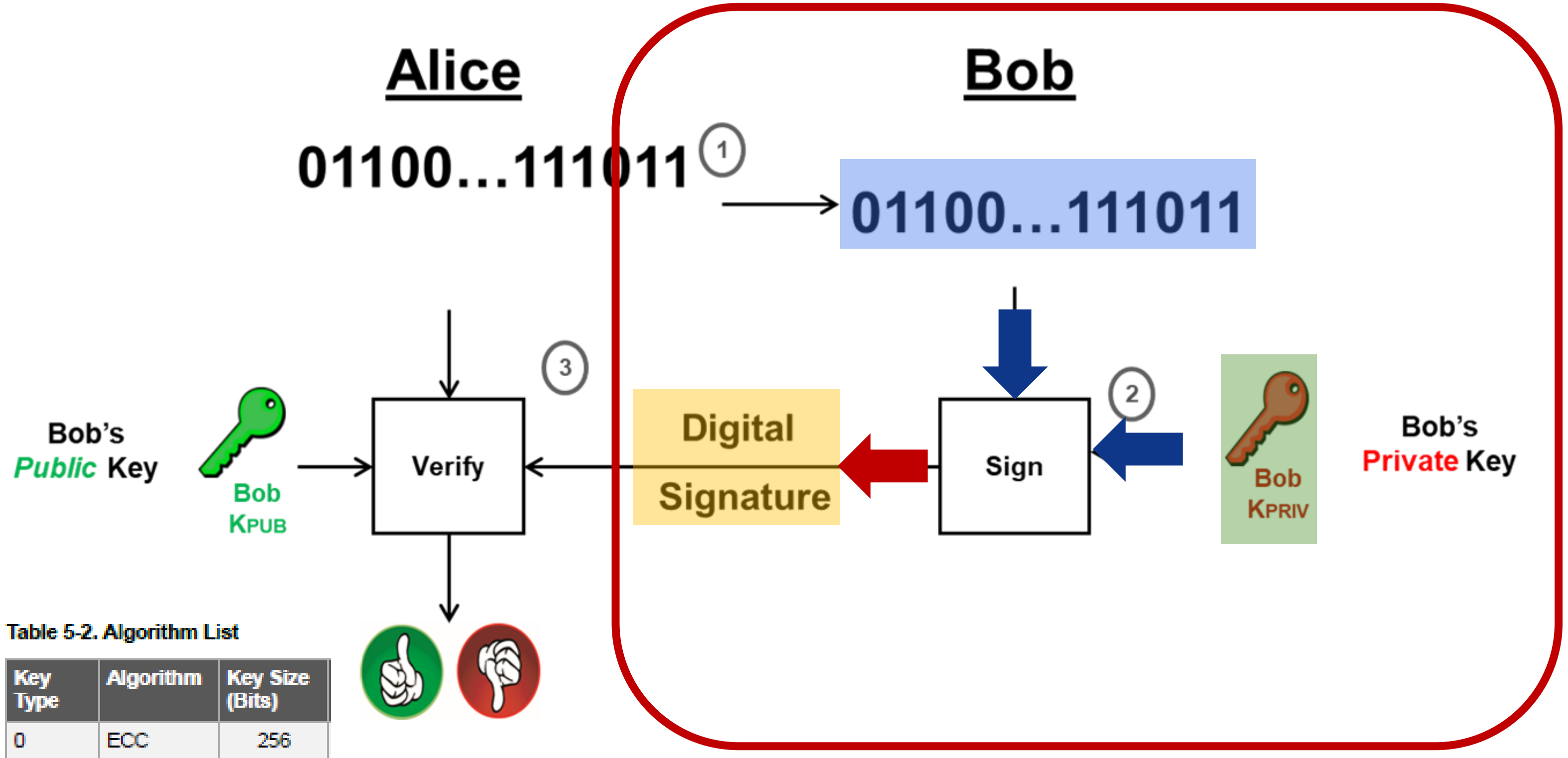


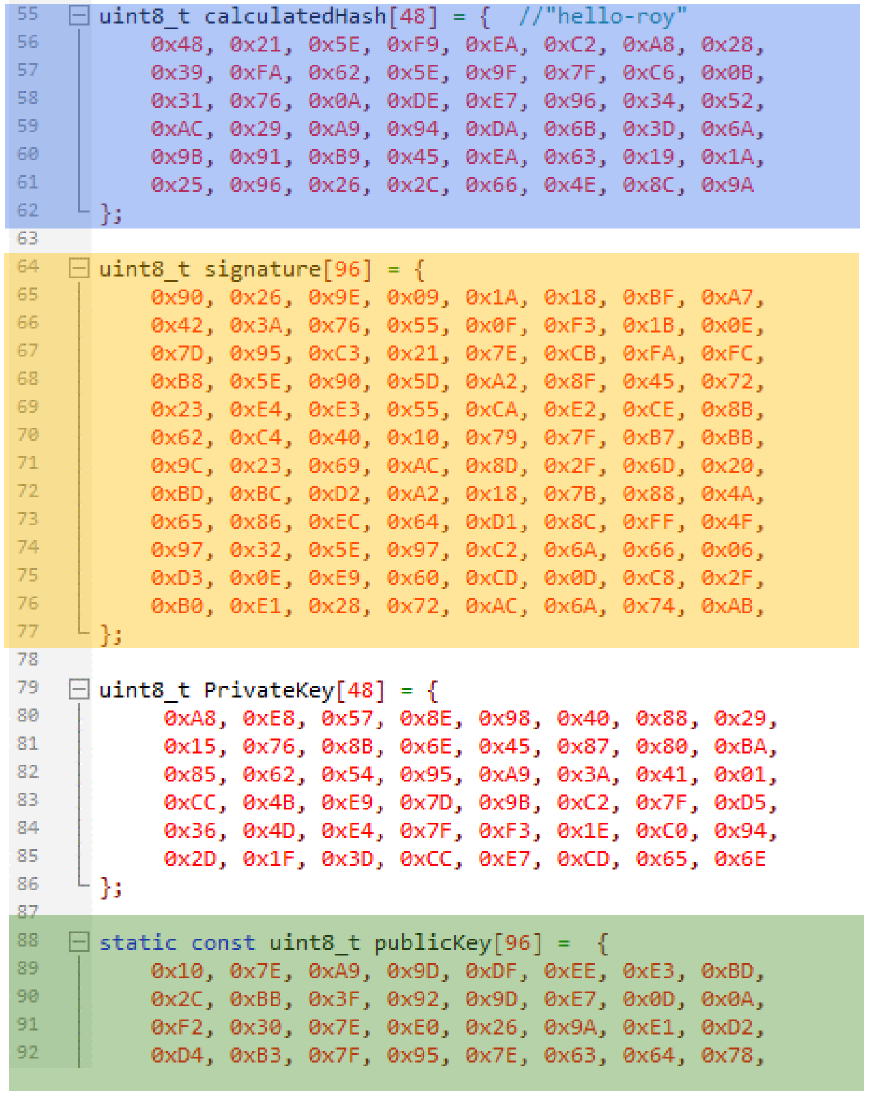
Table 5-2. Algorithm List

Key Type	Algorithm	Key Size (Bits)
0	ECC	256
1	ECC	224
2	ECC	384

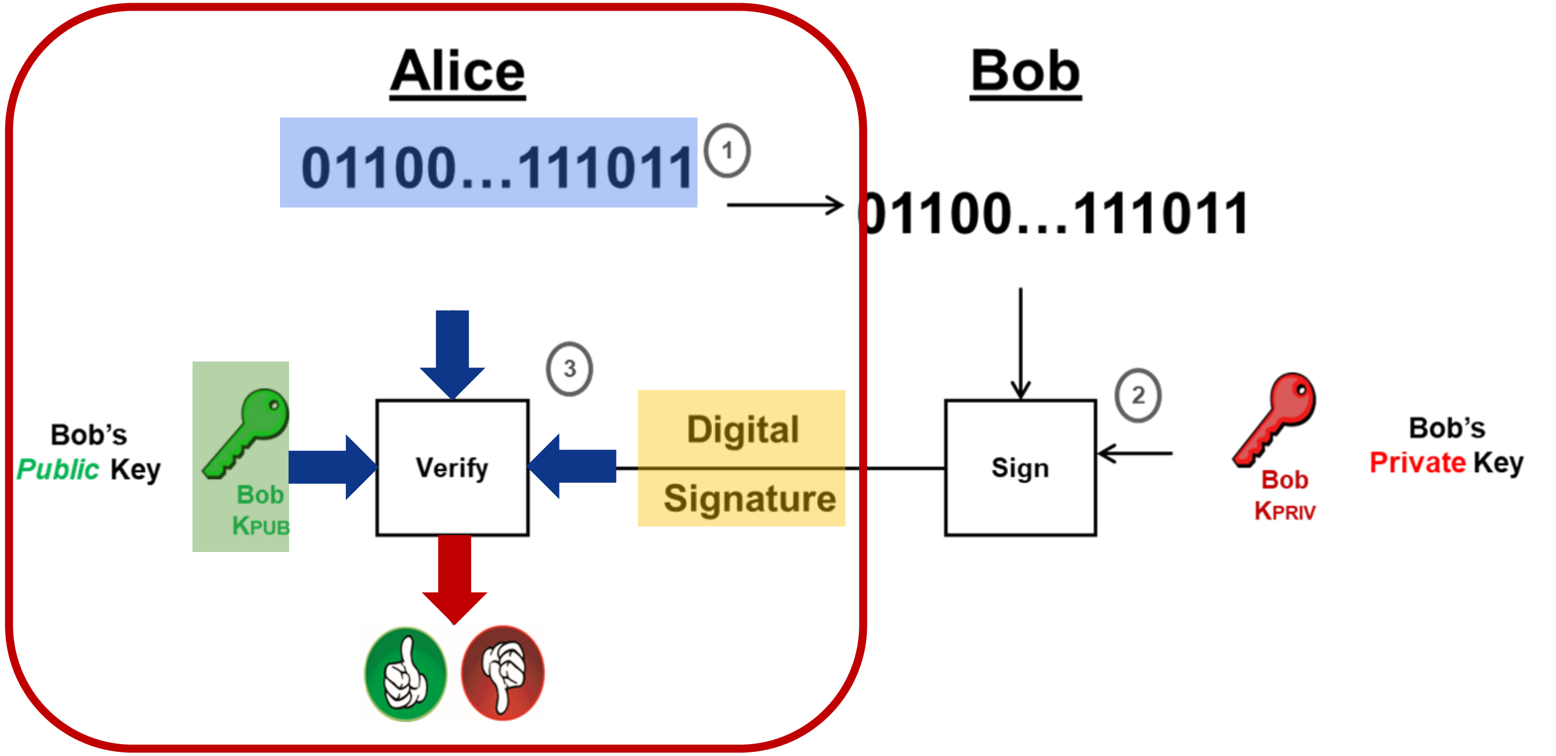
output

```
status =  talib_sign_external(atcab_get_device(), 0x02, private_key_handle, TA_HANDLE_INPUT_BUFFER, calculatedHash, 48, signature, &size_signature_384);
```

ECDSA P384 Verify



```
uint8_t calculatedHash[48] = {  //"hello-roy"
    0x48, 0x21, 0x5E, 0xF9, 0xEA, 0xC2, 0xA8, 0x28,
    0x39, 0xFA, 0x62, 0x5E, 0x9F, 0x7F, 0xC6, 0x0B,
    0x31, 0x76, 0x0A, 0xDE, 0xE7, 0x96, 0x34, 0x52,
    0xAC, 0x29, 0xA9, 0x94, 0xDA, 0x6B, 0x3D, 0x6A,
    0x9B, 0x91, 0xB9, 0x45, 0xEA, 0x63, 0x19, 0x1A,
    0x25, 0x96, 0x26, 0x2C, 0x66, 0x4E, 0x8C, 0x9A
};

uint8_t signature[96] = {
    0x90, 0x26, 0x9E, 0x09, 0x1A, 0x18, 0xBF, 0xA7,
    0x42, 0x3A, 0x76, 0x55, 0x0F, 0xF3, 0x1B, 0x0E,
    0x7D, 0x95, 0xC3, 0x21, 0x7E, 0xCB, 0xFA, 0xFC,
    0xB8, 0x5E, 0x90, 0x5D, 0xA2, 0x8F, 0x45, 0x72,
    0x23, 0xE4, 0xE3, 0x55, 0xCA, 0xE2, 0xCE, 0x8B,
    0x62, 0xC4, 0x40, 0x10, 0x79, 0x7F, 0xB7, 0xBB,
    0x9C, 0x23, 0x69, 0xAC, 0x8D, 0x2F, 0x6D, 0x20,
    0xBD, 0xBC, 0xD2, 0xA2, 0x18, 0x7B, 0x88, 0x4A,
    0x65, 0x86, 0xEC, 0x64, 0xD1, 0x8C, 0xFF, 0x4F,
    0x97, 0x32, 0x5E, 0x97, 0xC2, 0x6A, 0x66, 0x06,
    0xD3, 0x0E, 0xE9, 0x60, 0xCD, 0x0D, 0xC8, 0x2F,
    0xB0, 0xE1, 0x28, 0x72, 0xAC, 0x6A, 0x74, 0xAB,
};

uint8_t PrivateKey[48] = {
     0xA8, 0xE8, 0x57, 0x8E, 0x98, 0x40, 0x88, 0x29,
     0x15, 0x76, 0x8B, 0x6E, 0x45, 0x87, 0x80, 0xBA,
     0x85, 0x62, 0x54, 0x95, 0xA9, 0x3A, 0x41, 0x01,
     0xCC, 0x4B, 0xE9, 0x7D, 0x9B, 0xC2, 0x7F, 0xD5,
     0x36, 0x4D, 0xE4, 0x7F, 0xF3, 0x1E, 0xC0, 0x94,
     0x2D, 0x1F, 0x3D, 0xCC, 0xE7, 0xCD, 0x65, 0x6E
};

static const uint8_t publicKey[96] =  {
    0x10, 0x7E, 0xA9, 0x9D, 0xDF, 0xEE, 0xE3, 0xBD,
    0x2C, 0xBB, 0x3F, 0x92, 0x9D, 0xE7, 0x0D, 0x0A,
    0xF2, 0x30, 0x7E, 0xE0, 0x26, 0x9A, 0xE1, 0xD2,
    0xD4, 0xB3, 0x7F, 0x95, 0x7E, 0x63, 0x64, 0x78,
```



Yes/No

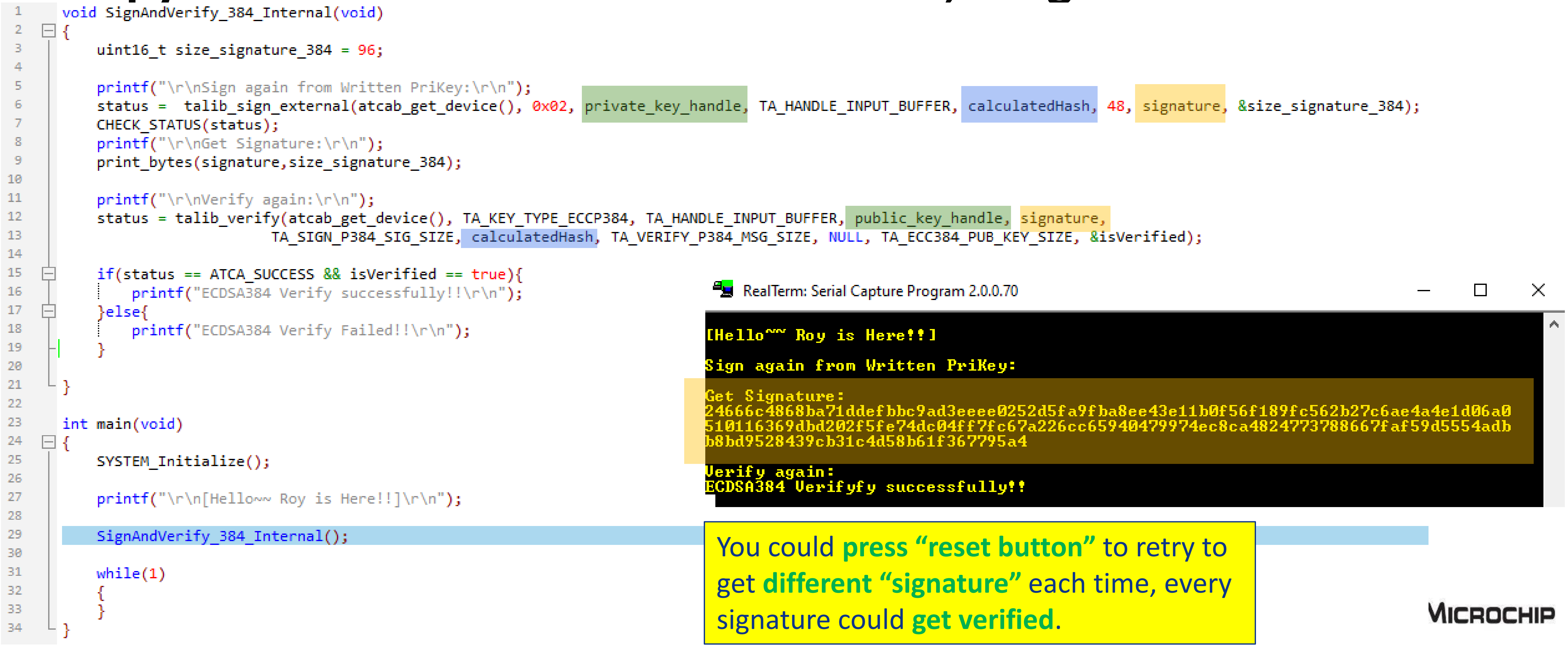
```
status = talib_verify(atcab_get_device(), TA_KEY_TYPE_ECCP384, TA_HANDLE_INPUT_BUFFER, TA_HANDLE_INPUT_BUFFER, signature,
                TA_SIGN_P384_SIG_SIZE, calculatedHash, TA_VERIFY_P384_MSG_SIZE, publicKey, TA_ECC384_PUB_KEY_SIZE, &isVerified);
```

Lab7 – ECDSA Sign & Verify using TA100

Use: Lab7.txt

Step 7-1

- **Use the pre-provisioned Device keys to run Sign & Verify**
- **Copy functions from Lab7.txt to main.c , Program & Run!**



```c
void SignAndVerify_384_Internal(void)
{
    uint16_t size_signature_384 = 96;

    printf("\r\nSign again from Written PriKey:\r\n");
    status =  talib_sign_external(atcab_get_device(), 0x02, private_key_handle, TA_HANDLE_INPUT_BUFFER, calculatedHash, 48, signature, &size_signature_384);
    CHECK_STATUS(status);
    printf("\r\nGet Signature:\r\n");
    print_bytes(signature,size_signature_384);

    printf("\r\nVerify again:\r\n");
    status = talib_verify(atcab_get_device(), TA_KEY_TYPE_ECCP384, TA_HANDLE_INPUT_BUFFER, public_key_handle, signature,
                    TA_SIGN_P384_SIG_SIZE, calculatedHash, TA_VERIFY_P384_MSG_SIZE, NULL, TA_ECC384_PUB_KEY_SIZE, &isVerified);

    if(status == ATCA_SUCCESS && isVerified == true){
        printf("ECDSA384 Verify successfully!!\r\n");
    }else{
        printf("ECDSA384 Verify Failed!!\r\n");
    }

}

int main(void)
{
    SYSTEM_Initialize();

    printf("\r\n[Hello~~ Roy is Here!!]\r\n");

    SignAndVerify_384_Internal();

    while(1)
    {
    }
}
```

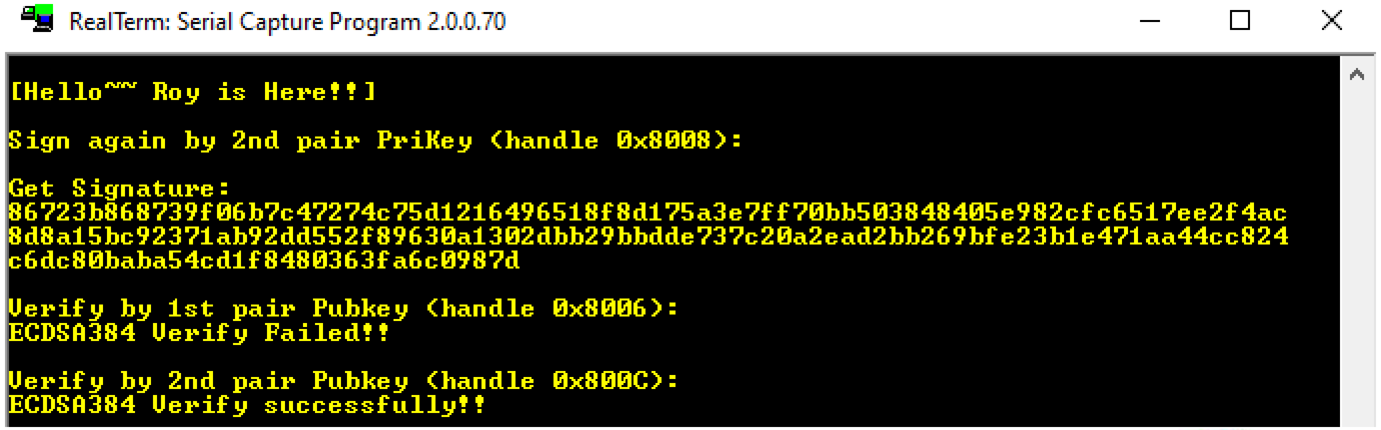
```
[Hello~~ Roy is Here!!]

Sign again from Written PriKey:

Get Signature:
24666c4868ba71ddefbbc9ad3eeee0252d5fa9fba8ee43e11b0f56f189fc562b27c6ae4a4e1d06a0
510116369dbd202f5fe74dc04ff7fc67a226cc65940479974ec8ca482477378 8667faf59d5554adb
b8bd9528439cb31c4d58b61f367795a4

Verify again:
ECDSA384 Verifyfy successfully!!
```

You could **press "reset button"** to retry to get **different "signature"** each time, every signature could **get verified.**

Lab 7 - Practice

1. Use 2nd pair Private key to do Sign, then
2. Use 1st pair Public key to verify, should be failed
3. Use 2nd pair Public key to verify, should be Successful

Element Name	Class	Type/Notes
Private Key @ 0x8007	Private Key	ECC P384 (Fixed Private Key)
Public Key @ 0x8006	Public Key	ECC P384 (Fixed paired Public key)
Private Key @ 0x8008	Private Key	ECC P384 (Random Device Private Key)
Public Key @ 0x800C	Public Key	ECC P384 (paired Device Public key)
signerCert @ 0x8201	Data	Example Signer Certificate (Hold signer Public key)
deviceCert @ 0x8200	Data	Example Device Certificate (Hold Device Public key)



```
[Hello~~ Roy is Here!!]

Sign again by 2nd pair PriKey (handle 0x8008):

Get Signature:
86723b868739f06b7c47274c75d1216496518f8d175a3e7ff70bb503848405e982cfc6517ee2f4ac
8d8a15bc92371ab92dd552f89630a1302dbb29bbdde737c20a2ead2bb269bfe23b1e471aa44cc824
c6dc80baba54cd1f8480363fa6c0987d

Verify by 1st pair Pubkey (handle 0x8006):
ECDSA384 Verify Failed!!

Verify by 2nd pair Pubkey (handle 0x800C):
ECDSA384 Verify successfully!!
```

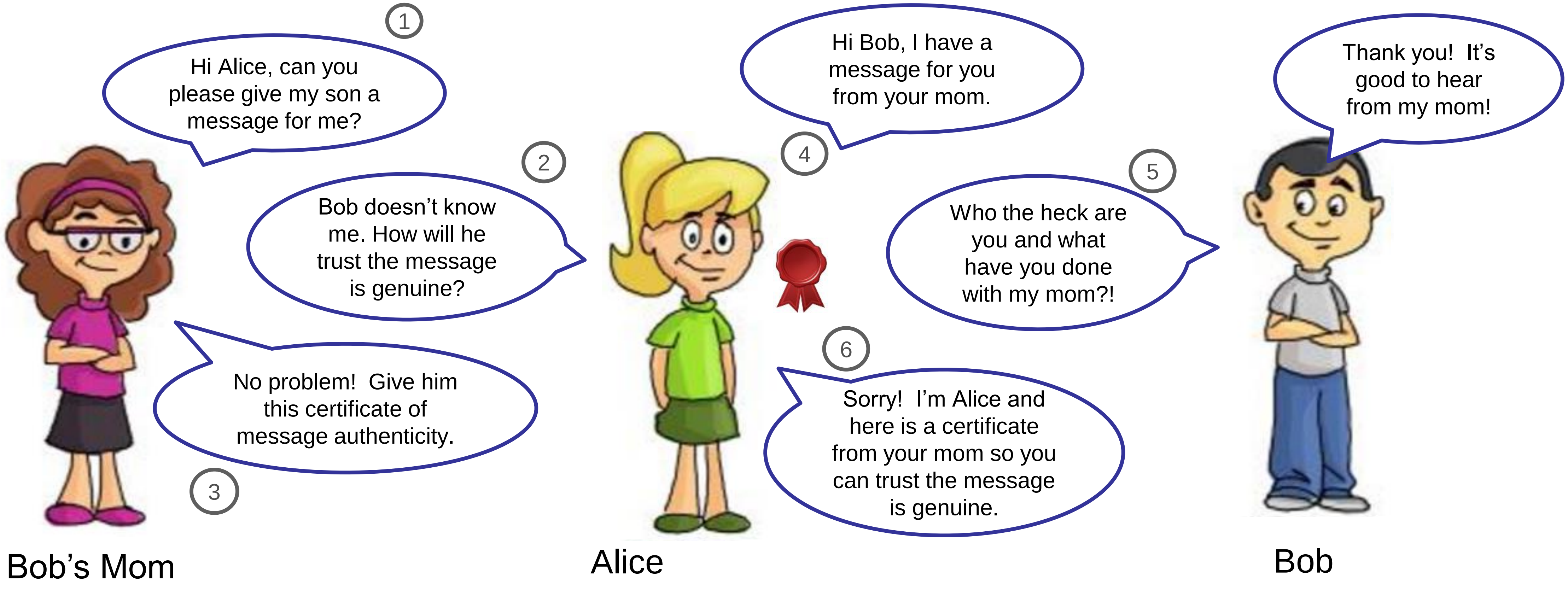
Let’s start the Lab7 & practice

Message Authentication

Certificates

The example of Message Authentication

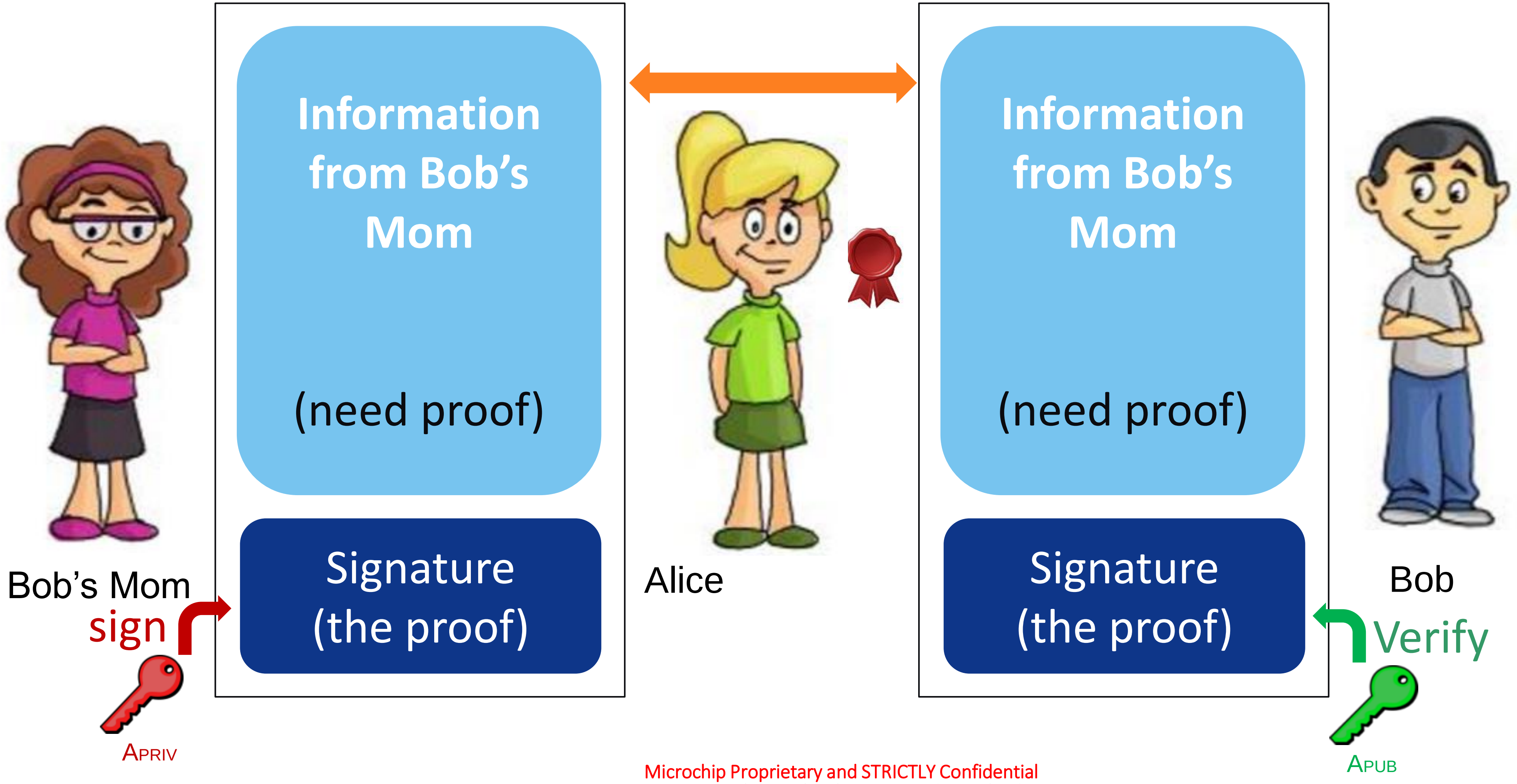
Alice knows and trusts Bob’s mom but has never met Bob


1
Hi Alice, can you please give my son a message for me?
2
Bob doesn’t know me. How will he trust the message is genuine?
3
No problem! Give him this certificate of message authenticity.
4
Hi Bob, I have a message for you from your mom.
5
Who the heck are you and what have you done with my mom?!
6
Sorry! I’m Alice and here is a certificate from your mom so you can trust the message is genuine.
Thank you! It’s good to hear from my mom!
Bob’s Mom
Alice
Bob

This is an example of ***message authentication***. You could imagine Bob’s mom could have given Alice a certificate directing Bob to trust whatever Alice says, essentially making the introduction and removing herself.

Certificate is used to prove some information

Use Public key to verify a certificate



Information from Bob's Mom

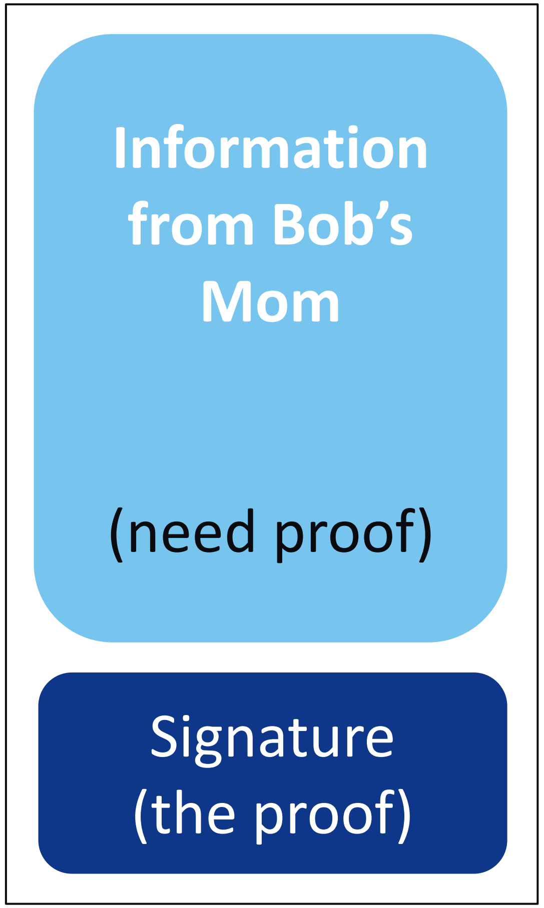
(need proof)

Signature (the proof)

Information from Bob's Mom

(need proof)

Signature (the proof)



Information from Bob's Mom

(need proof)

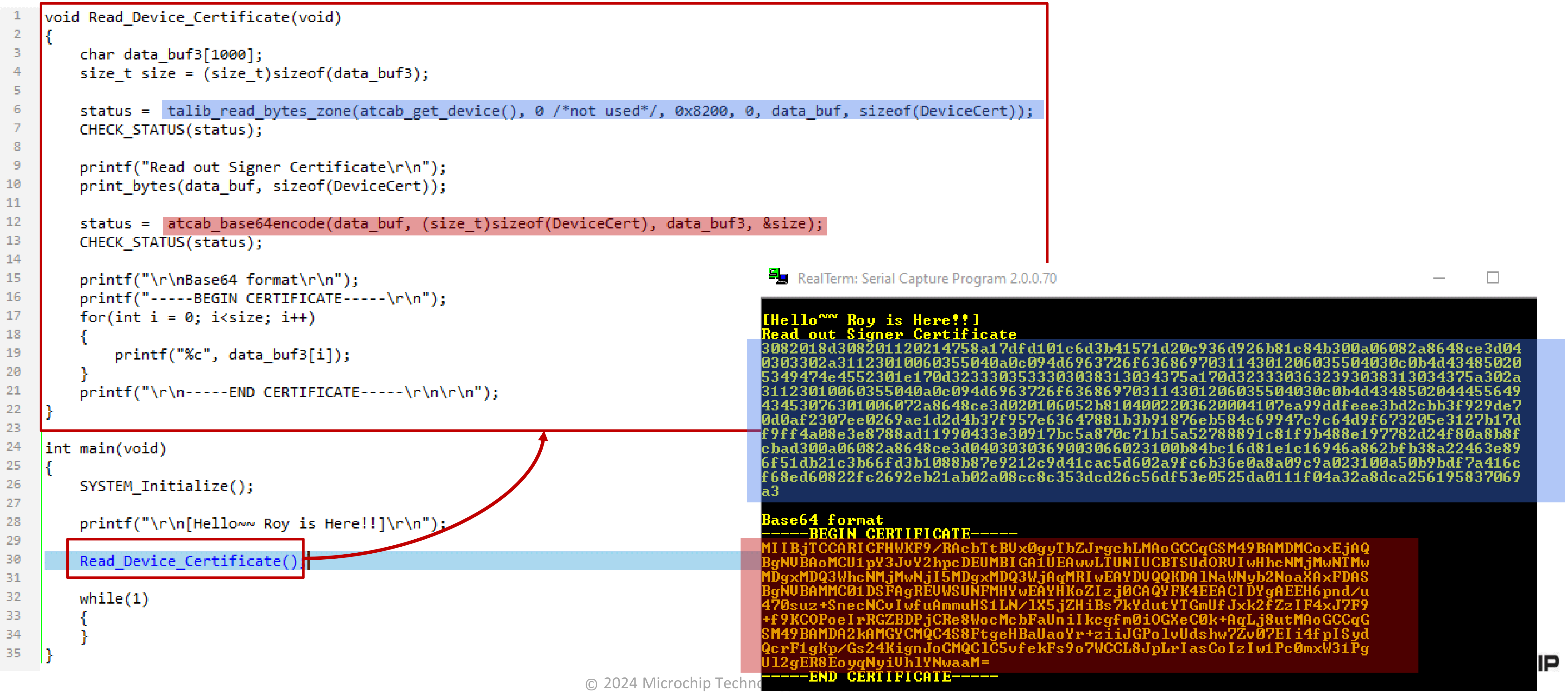
Signature (the proof)

Lab8 – Read out Device Certificate from TA100

Use:Lab8.txt

Step 8-1

- **Copy functions from Lab8.txt to main.c , Program & Run!**

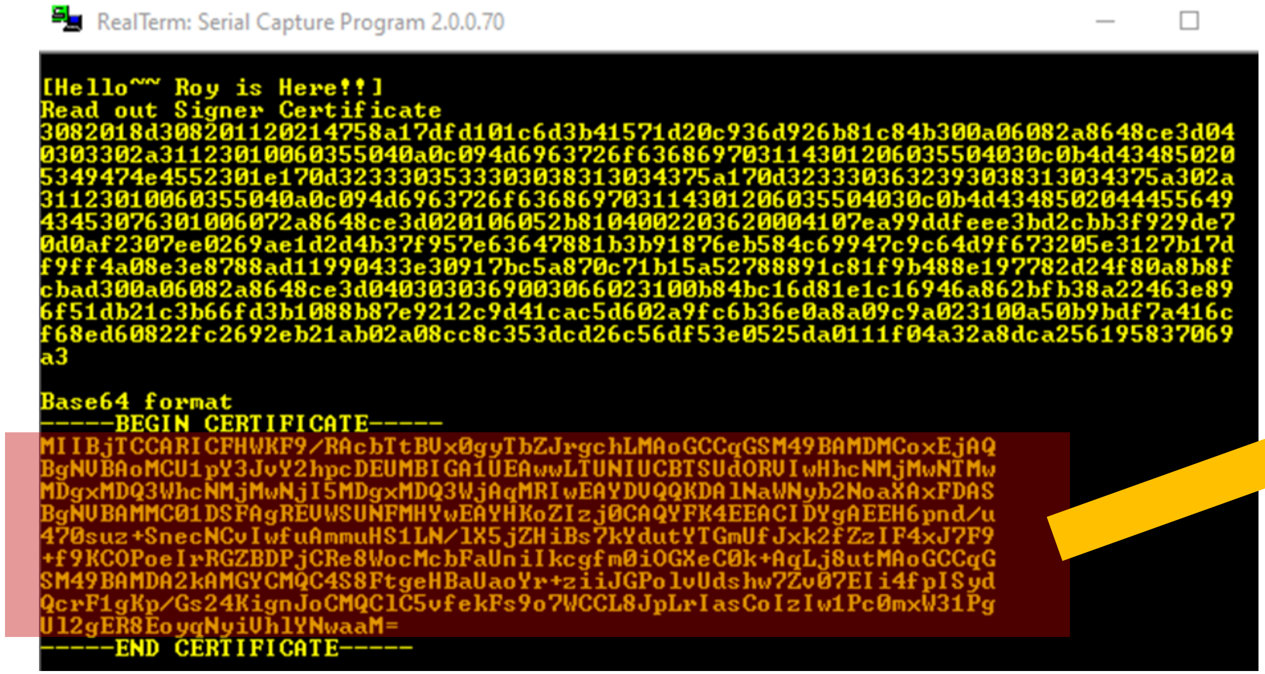


```c
void Read_Device_Certificate(void)
{
    char data_buf3[1000];
    size_t size = (size_t)sizeof(data_buf3);

    status = talib_read_bytes_zone(atcab_get_device(), 0 /*not used*/, 0x8200, 0, data_buf, sizeof(DeviceCert));
    CHECK_STATUS(status);

    printf("Read out Signer Certificate\r\n");
    print_bytes(data_buf, sizeof(DeviceCert));

    status = atcab_base64encode(data_buf, (size_t)sizeof(DeviceCert), data_buf3, &size);
    CHECK_STATUS(status);

    printf("\r\nBase64 format\r\n");
    printf("-----BEGIN CERTIFICATE-----\r\n");
    for(int i = 0; i<size; i++)
    {
        printf("%c", data_buf3[i]);
    }
    printf("\r\n-----END CERTIFICATE-----\r\n\r\n");
}

int main(void)
{
    SYSTEM_Initialize();

    printf("\r\n[Hello~~ Roy is Here!!]\r\n");

    Read_Device_Certificate();

    while(1)
    {
    }
}
```

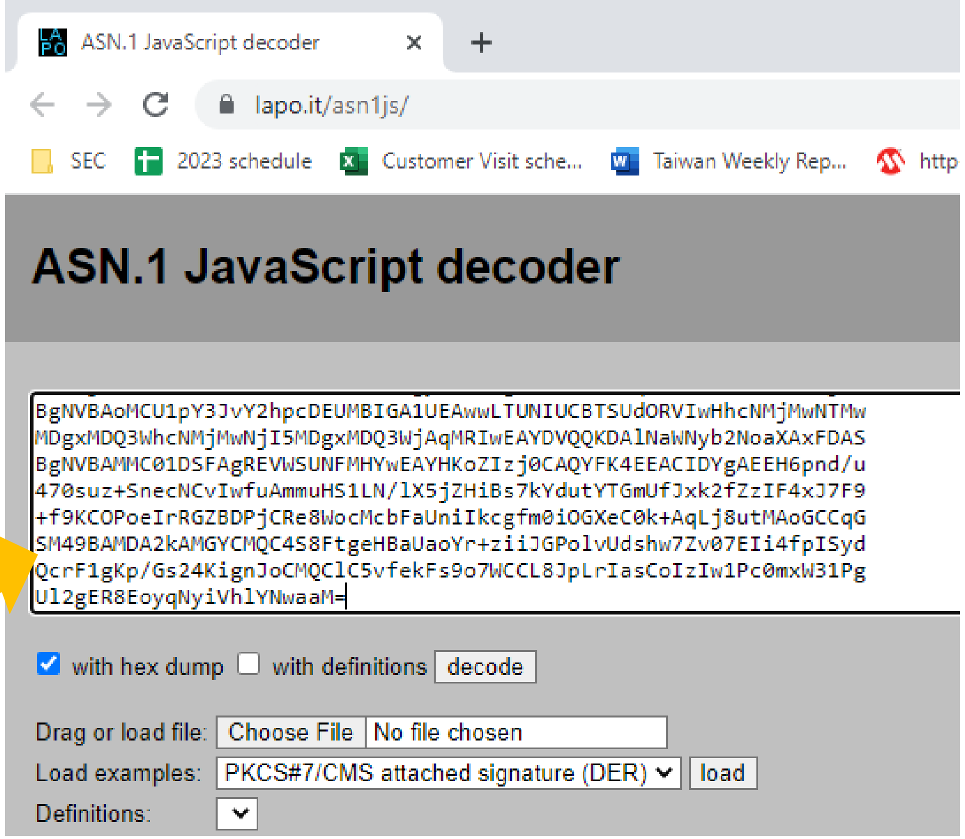
```
[Hello~~ Roy is Here!!]
Read out Signer Certificate
3082018d308201120214758a17dfd101c6d3b41571d20c936d926b81c84b300a06082a8648ce3d04
0303302a311230100603550040a0c094d6963726f636869703114301206035504030c0b4d43485020
5349474e4552301e170d32333035333030383130343 75a170d3233303632393038313034375a302a
31123010060355040a0c094d6963726f6368697031143012060355040 30c0b4d4348502044455649
43453076301006072a8648ce3d020106052b8104002203620004107ea99ddfeee3bd2cbb3f929de7
0d0af2307ee0269ae1d2d4b37f957e63647881b3b91876eb584c69947c9c64d9f673205e3127b17d
f9ff4a08e3e8788ad11990433e30917bc5a870c71b15a52788891c81f9b488e197782d24f80a8b8f
cbad300a06082a8648ce3d0403030369003066023100b84bc16d81e1c16946a862bfb38a22463e89
6f51db21c3b66fd3b1088b87e9212c9d41cac5d602a9fc6b36e0a8a09c9a023100a50b9bdf7a416c
f68ed60822fc2692eb21ab02a08cc8c353dcd26c56df53e0525da0111f04a32a8dca256195837069
a3

Base64 format
-----BEGIN CERTIFICATE-----
MIIBjTCCARICFHWKF9/RAcbTtBVx0gyTbZJrgchLMAoGCCqGSM49BAMDMCoxEjAQ
BgNVBAoMCU1pY3JvY2hpcDEUMBIGA1UEAwwLTUNIUCBTSUdORVIwHhcNMjMwNTMw
MDgxMDQ3WhcNMjMwNjI5MDgxMDQ3WjAqMRIwEAYDVQQKDAlNaWNyb2NoaXAxFDAS
BgNVBAMMC01DSFAgREVWSUNFMHYwEAYHKoZIzj0CAQYFK4EEACIDYgAEEH6pnd/u
470suz+SnecNCvIwfuAmmuHS1LN/lX5jZHiBs7kYdutYTGmUfJxk2fZzIF4xJ7F9
+f9KCOPoeIrRGZBDPjCRe8WocMcbFaUnilkcgfm0iOGXeC0k+AqLj8utMAoGCCqG
SM49BAMDA2kAMGYCMQC4S8FtgeHBaUaoYr+ziiJGPolvUdshw7Zv07EIi4fpISyd
QcrF1gKp/Gs24KignJoCMQClC5vfekFs9o7WCCL8JpLrIasCoIzIw1Pc0mxW31Pg
Ul2gER8EoygNyiVhlYNwaaM=
-----END CERTIFICATE-----
```

Step 8-2

- **Decode Certificate to see its details**



```
[Hello~~ Roy is Here!!]
Read out Signer Certificate
3082018d3082011202014758a17dfd101c6d3b41571d20c936d926b81c84b300a06082a8648ce3d04
0303302a3112301006035504 0a0c094d6963726f6368697031143012060355040 30c0b4d43485020
5349474e4552301e170d3233303533303038313034375a170d3233303632393038313034375a302a
311230100603550400a0c094d6963726f63686970311430120603550403 0c0b4d434850204445564 9
434530763010060 72a8648ce3d020106052b8104002203620004107ea99ddfeee3bd2cbb3f929de7
0d0af2307ee0269ae1d2d4b37f957e63647881b3b91876eb584c69947c9c64d9f673205e3127b17d
f9ff4a08e3e8788ad11990433e30917bc5a870c71b15a52788891c81f9b488e197782d24f80a8b8f
cbad300a06082a8648ce3d040303036900306602310 0b84bc16d81e1c16946a862bfb38a22463e89
6f51db21c3b66fd3b1088b87e9212c9d41cac5d602a9fc6b36e0a8a09c9a023100a50b9bdf7a416c
f68ed60822fc2692eb21ab02a08cc8c353dcd26c56df53e0525da0111f04a32a8dca256195837069
a3

Base64 format
-----BEGIN CERTIFICATE-----
MIIBjTCCARICFHWKF9/RAcbTtBVx0gyTbZJrgchLMAoGCCqGSM49BAMDMCoxEjAQ
BgNVBAoMCU1pY3JvY2hpcDEUMBIGA1UEAwwLTUNIUCBTSUdORVIwHhcNMjMwNTMw
MDgxMDQ3WhcNMjMwNjI5MDgxMDQ3WjAqMRIwEAYDVQQKDAlNaWNyb2NoaXAxFDAS
BgNVBAMMC01DSFAgREVWSUNFMHYwEAYHKoZIzj0CAQYFK4EEACIDYgAEEH6pnd/u
470suz+SnecNCvIwfuAmmuHS1LN/lX5jZHiBs7kYdutYTGmUfJxk2fZzIF4xJ7F9
+f9KCOPoeIrRGZBDPjCRe8WocMcbFaUniIkcgfm0iOGXeC0k+AqLj8utMAoGCCqG
SM49BAMDA2kAMGYCMQC4S8FtgeHBaUaoYr+ziiJGPolvUdshw7Zv07EIi4fpISyd
QcrF1gKp/Gs24KignJoCMQClC5vfekFs9o7WCCL8JpLrIasCoIzIw1Pc0mxW31Pg
Ul2gER8EoyqNyiVhlYNwaaM=
-----END CERTIFICATE-----
```



ASN.1 JavaScript decoder

```
BgNVBAoMCU1pY3JvY2hpcDEUMBIGA1UEAwwLTUNIUCBTSUdORVIwHhcNMjMwNTMw
MDgxMDQ3WhcNMjMwNjI5MDgxMDQ3WjAqMRIwEAYDVQQKDAlNaWNyb2NoaXAxFDAS
BgNVBAMMC01DSFAgREVWSUNFMHYwEAYHKoZIzj0CAQYFK4EEACIDYgAEEH6pnd/u
470suz+SnecNCvIwfuAmmuHS1LN/lX5jZHiBs7kYdutYTGmUfJxk2fZzIF4xJ7F9
+f9KCOPoeIrRGZBDPjCRe8WocMcbFaUniIkcgfm0iOGXeC0k+AqLj8utMAoGCCqG
SM49BAMDA2kAMGYCMQC4S8FtgeHBaUaoYr+ziiJGPolvUdshw7Zv07EIi4fpISyd
QcrF1gKp/Gs24KignJoCMQClC5vfekFs9o7WCCL8JpLrIasCoIzIw1Pc0mxW31Pg
Ul2gER8EoyqNyiVhlYNwaaM=
```

☑ with hex dump ☐ with definitions [decode]

Drag or load file: [Choose File] No file chosen

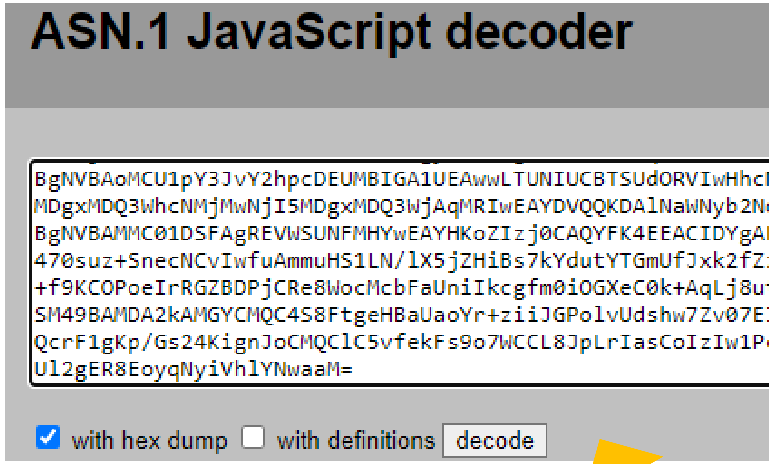
Load examples: PKCS#7/CMS attached signature (DER) [load]

Definitions:

https://lapo.it/asn1js/#

Step 8-3

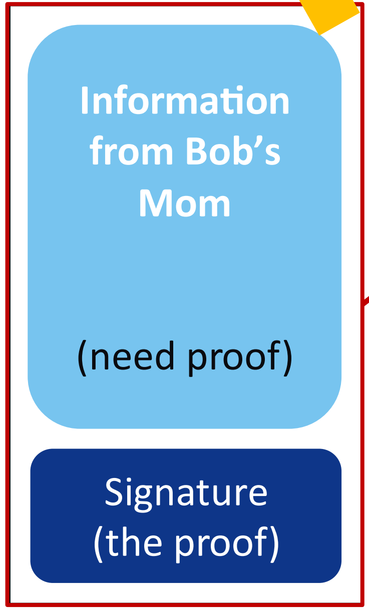
- **Decode Certificate to see details**



ASN.1 JavaScript decoder

```
BgNVBAoMCU1pY3JvY2hpcDEUMBIGA1UEAwwLTUNIUCBTSUdORVIwHhcl
MDgxMDQ3WhcNMjMwNjI5MDgxMDQ3WjAqMRIwEAYDVQQKDAlNaWNyb2N
BgNVBAMMC01DSFAgREVWSUNFMHYwEAYHKoZIzj0CAQYFK4EEACIDYgAl
470suz+SnecNCvIwfuAmmuHS1LN/lX5jZHiBs7kYdutYTGmUfJxk2fZ
+f9KCOPoeIrRGZBDPjCRe8WocMcbFaUniIkcgfm0iOGXeC0k+AqLj8u
SM49BAMDA2kAMGYCMQC4S8FtgeHBaUaoYr+ziiJGPolvUdshw7Zv07E
QcrF1gKp/Gs24KignJoCMQClC5vfekFs9o7WCCL8JpLrIasCoIzIw1P
Ul2gER8EoyqNyiVhlYNwaaM=
```

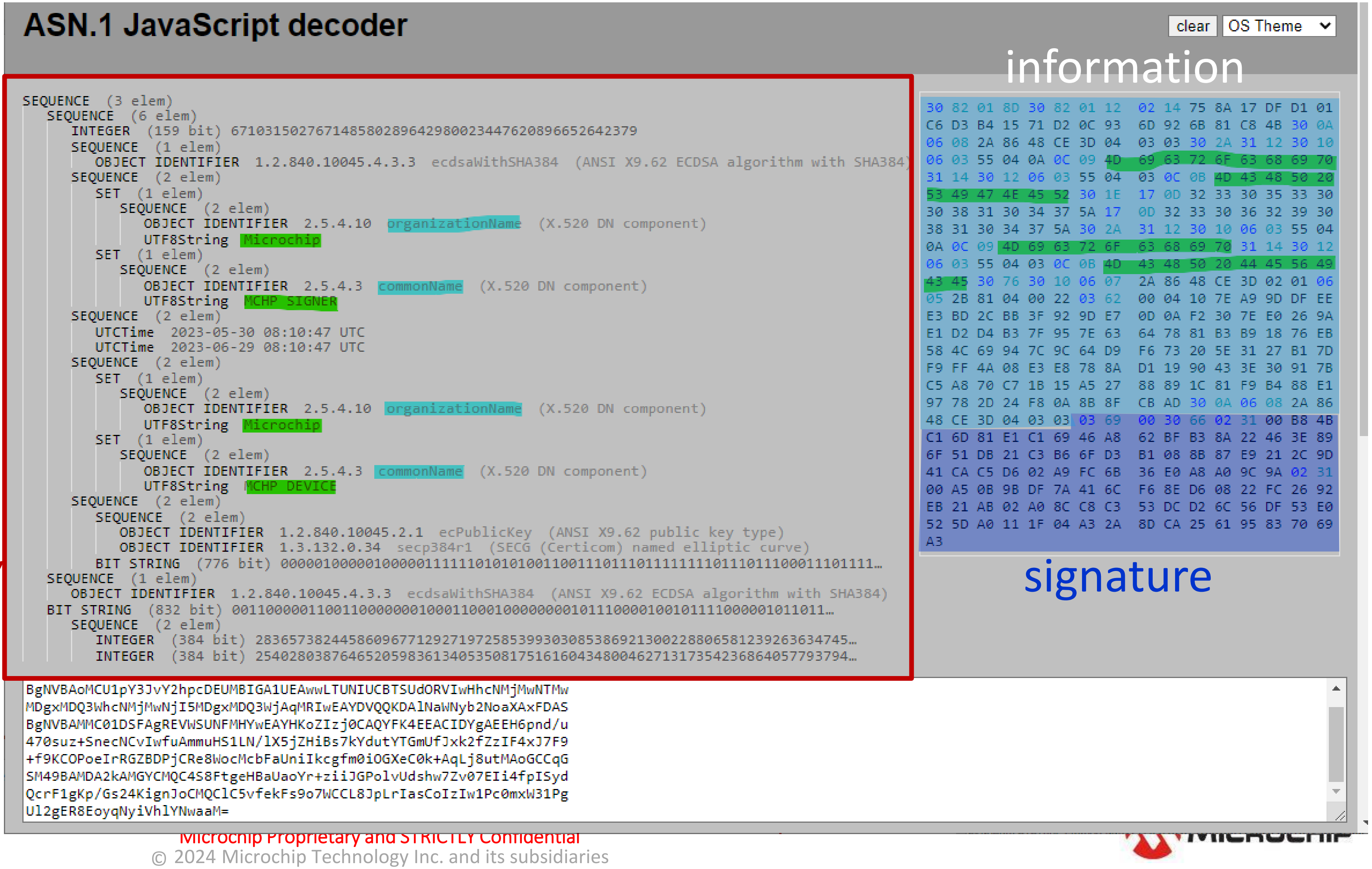
☑ with hex dump ☐ with definitions decode



Information from Bob's Mom

(need proof)

Signature (the proof)



ASN.1 JavaScript decoder

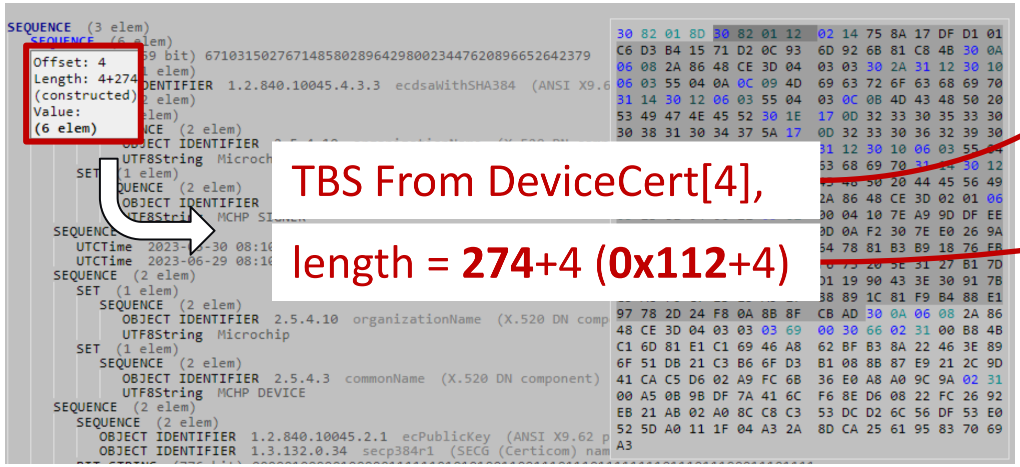
```
SEQUENCE  (3 elem)
  SEQUENCE  (6 elem)
    INTEGER  (159 bit) 671031502767148580289642980023447620896652642379
    SEQUENCE  (1 elem)
      OBJECT IDENTIFIER  1.2.840.10045.4.3.3  ecdsaWithSHA384  (ANSI X9.62 ECDSA algorithm with SHA384)
    SEQUENCE  (2 elem)
      SET  (1 elem)
        SEQUENCE  (2 elem)
          OBJECT IDENTIFIER  2.5.4.10  organizationName  (X.520 DN component)
          UTF8String  Microchip
      SET  (1 elem)
        SEQUENCE  (2 elem)
          OBJECT IDENTIFIER  2.5.4.3  commonName  (X.520 DN component)
          UTF8String  MCHP SIGNER
    SEQUENCE  (2 elem)
      UTCTime  2023-05-30 08:10:47 UTC
      UTCTime  2023-06-29 08:10:47 UTC
    SEQUENCE  (2 elem)
      SET  (1 elem)
        SEQUENCE  (2 elem)
          OBJECT IDENTIFIER  2.5.4.10  organizationName  (X.520 DN component)
          UTF8String  Microchip
      SET  (1 elem)
        SEQUENCE  (2 elem)
          OBJECT IDENTIFIER  2.5.4.3  commonName  (X.520 DN component)
          UTF8String  MCHP DEVICE
    SEQUENCE  (2 elem)
      SEQUENCE  (2 elem)
        OBJECT IDENTIFIER  1.2.840.10045.2.1  ecPublicKey  (ANSI X9.62 public key type)
        OBJECT IDENTIFIER  1.3.132.0.34  secp384r1  (SECG (Certicom) named elliptic curve)
      BIT STRING  (776 bit) 0000010000010000011111101010100110011101110111111110111011100011101111…
  SEQUENCE  (1 elem)
    OBJECT IDENTIFIER  1.2.840.10045.4.3.3  ecdsaWithSHA384  (ANSI X9.62 ECDSA algorithm with SHA384)
  BIT STRING  (832 bit) 0011000001100110000000100011000100000000101110000100101111000001011011…
    SEQUENCE  (2 elem)
      INTEGER  (384 bit) 2836573824458609677129271972585399303085386921300228806581239263634745…
      INTEGER  (384 bit) 2540280387646520598361340535081751616043480046271317354236864057793794…
```

```
30 82 01 8D 30 82 01 12  02 14 75 8A 17 DF D1 01
C6 D3 B4 15 71 D2 0C 93  6D 92 6B 81 C8 4B 30 0A
06 08 2A 86 48 CE 3D 04  03 03 30 2A 31 12 30 10
06 03 55 04 0A 0C 09 4D  69 63 72 6F 63 68 69 70
31 14 30 12 06 03 55 04  03 0C 0B 4D 43 48 50 20
53 49 47 4E 45 52 30 1E  17 0D 32 33 30 35 33 30
30 38 31 30 34 37 5A 17  0D 32 33 30 36 32 39 30
38 31 30 34 37 5A 30 2A  31 12 30 10 06 03 55 04
0A 0C 09 4D 69 63 72 6F  63 68 69 70 31 14 30 12
06 03 55 04 03 0C 0B 4D  43 48 50 20 44 45 56 49
43 45 30 76 30 10 06 07  2A 86 48 CE 3D 02 01 06
05 2B 81 04 00 22 03 62  00 04 10 7E A9 9D DF EE
E3 BD 2C BB 3F 92 9D E7  0D 0A F2 30 7E E0 26 9A
E1 D2 D4 B3 7F 95 7E 63  64 78 81 B3 B9 18 76 EB
58 4C 69 94 7C 9C 64 D9  F6 73 20 5E 31 27 B1 7D
F9 FF 4A 08 E3 E8 78 8A  D1 19 90 43 3E 30 91 7B
C5 A8 70 C7 1B 15 A5 27  88 89 1C 81 F9 B4 88 E1
97 78 2D 24 F8 0A 8B 8F  CB AD 30 0A 06 08 2A 86
48 CE 3D 04 03 03 03 69  00 30 66 02 31 00 B8 4B
C1 6D 81 E1 C1 69 46 A8  62 BF B3 8A 22 46 3E 89
6F 51 DB 21 C3 B6 6F D3  B1 08 8B 87 E9 21 2C 9D
41 CA C5 D6 02 A9 FC 6B  36 E0 A8 A0 9C 9A 02 31
00 A5 0B 9B DF 7A 41 6C  F6 8E D6 08 22 FC 26 92
EB 21 AB 02 A0 8C C8 C3  53 DC D2 6C 56 DF 53 E0
52 5D A0 11 1F 04 A3 2A  8D CA 25 61 95 83 70 69
A3
```

```
BgNVBAoMCU1pY3JvY2hpcDEUMBIGA1UEAwwLTUNIUCBTSUdORVIwHhcNMjMwNTMw
MDgxMDQ3WhcNMjMwNjI5MDgxMDQ3WjAqMRIwEAYDVQQKDAlNaWNyb2NoaXAxFDAS
BgNVBAMMC01DSFAgREVWSUNFMHYwEAYHKoZIzj0CAQYFK4EEACIDYgAEEH6pnd/u
470suz+SnecNCvIwfuAmmuHS1LN/lX5jZHiBs7kYdutYTGmUfJxk2fZzIF4xJ7F9
+f9KCOPoeIrRGZBDPjCRe8WocMcbFaUniIkcgfm0iOGXeC0k+AqLj8utMAoGCCqG
SM49BAMDA2kAMGYCMQC4S8FtgeHBaUaoYr+ziiJGPolvUdshw7Zv07EIi4fpISyd
QcrF1gKp/Gs24KignJoCMQClC5vfekFs9o7WCCL8JpLrIasCoIzIw1Pc0mxW31Pg
Ul2gER8EoyqNyiVhlYNwaaM=
```

Step 8-4

- **Look into the certificate**



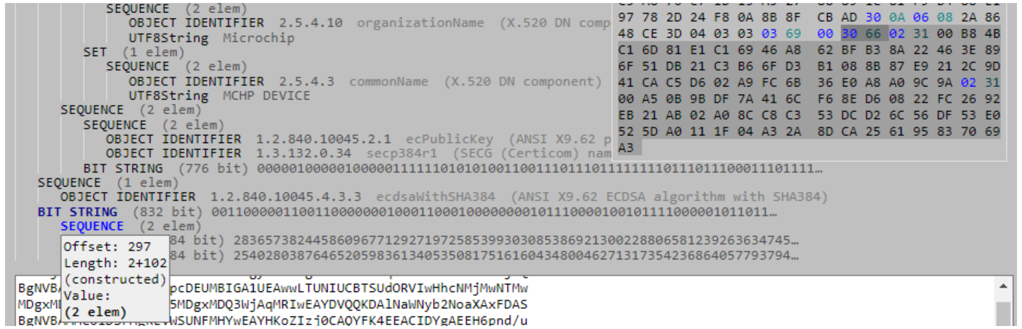
```
SEQUENCE  (3 elem)
  SEQUENCE  (6 elem)
  Offset: 4
  Length: 4+274
  (constructed)
  Value:
  (6 elem)
              9 bit) 671031502767148580289642980023447620896652642379
              1 elem)
              DENTIFIER  1.2.840.10045.4.3.3  ecdsaWithSHA384  (ANSI X9.6
              2 elem)
              elem)
          NCE  (2 elem)
          OBJECT IDENTIFIER
          UTF8String  Microch
      SET  (1 elem)
          QUENCE  (2 elem)
          OBJECT IDENTIFIER
          UTF8Stri    MCHP SI
    SEQUENCE
      UTCTime  2023-  -30 08:10
      UTCTime  2023-06-29 08:10
    SEQUENCE  (2 elem)
      SET  (1 elem)
        SEQUENCE  (2 elem)
          OBJECT IDENTIFIER  2.5.4.10  organizationName  (X.520 DN comp
          UTF8String  Microchip
      SET  (1 elem)
        SEQUENCE  (2 elem)
          OBJECT IDENTIFIER  2.5.4.3  commonName  (X.520 DN component)
          UTF8String  MCHP DEVICE
    SEQUENCE  (2 elem)
      SEQUENCE  (2 elem)
        OBJECT IDENTIFIER  1.2.840.10045.2.1  ecPublicKey  (ANSI X9.62 p
        OBJECT IDENTIFIER  1.3.132.0.34  secp384r1  (SECG (Certicom) nam
```

```
30 82 01 8D 30 82 01 12  02 14 75 8A 17 DF D1 01
C6 D3 B4 15 71 D2 0C 93  6D 92 6B 81 C8 4B 30 0A
06 08 2A 86 48 CE 3D 04  03 03 30 2A 31 12 30 10
06 03 55 04 0A 0C 09 4D  69 63 72 6F 63 68 69 70
31 14 30 12 06 03 55 04  03 0C 0B 4D 43 48 50 20
53 49 47 4E 45 52 30 1E  17 0D 32 33 30 35 33 30
30 38 31 30 34 37 5A 17  0D 32 33 30 36 32 39 30
                         31 12 30 10 06 03 55 04
                         63 68 69 70 31 14 30 12
                         43 48 50 20 44 45 56 49
                         2A 86 48 CE 3D 02 01 06
                         00 04 10 7E A9 9D DF EE
                         0D 0A F2 30 7E E0 26 9A
                         64 78 81 B3 B9 18 76 EB
                         F6 73 20 5E 31 27 B1 7D
                         D1 19 90 43 3E 30 91 7B
                         88 89 1C 81 F9 B4 88 E1
97 78 2D 24 F8 0A 8B 8F  CB AD 30 0A 06 08 2A 86
48 CE 3D 04 03 03 03 69  00 30 66 02 31 00 B8 4B
C1 6D 81 E1 C1 69 46 A8  62 BF B3 8A 22 46 3E 89
6F 51 DB 21 C3 B6 6F D3  B1 08 8B 87 E9 21 2C 9D
41 CA C5 D6 02 A9 FC 6B  36 E0 A8 A0 9C 9A 02 31
00 A5 0B 9B DF 7A 41 6C  F6 8E D6 08 22 FC 26 92
EB 21 AB 02 A0 8C C8 C3  53 DC D2 6C 56 DF 53 E0
52 5D A0 11 1F 04 A3 2A  8D CA 25 61 95 83 70 69
A3
```

TBS From DeviceCert[4],

length = **274**+4 (**0x112**+4)

To Be Signed Area



```
          SEQUENCE  (2 elem)
            OBJECT IDENTIFIER  2.5.4.10  organizationName  (X.520 DN comp
            UTF8String  Microchip
        SET  (1 elem)
          SEQUENCE  (2 elem)
            OBJECT IDENTIFIER  2.5.4.3  commonName  (X.520 DN component)
            UTF8String  MCHP DEVICE
      SEQUENCE  (2 elem)
        SEQUENCE  (2 elem)
          OBJECT IDENTIFIER  1.2.840.10045.2.1  ecPublicKey  (ANSI X9.62 p
          OBJECT IDENTIFIER  1.3.132.0.34  secp384r1  (SECG (Certicom) nam
        BIT STRING  (776 bit) 0000010000010000011111101010100110011101110111111110111011100011101111…
    SEQUENCE  (1 elem)
      OBJECT IDENTIFIER  1.2.840.10045.4.3.3  ecdsaWithSHA384  (ANSI X9.62 ECDSA algorithm with SHA384)
    BIT STRING  (832 bit) 0011000001100110000000100011000100000000101110000100101111000001011011…
      SEQUENCE  (2 elem)
      Offset: 297      84 bit) 2836573824458609677129271972585399303085386921300228806581239263634745…
      Length: 2+102    84 bit) 2540280387646520598361340535081751616043480046271317354236864057793794…
      (constructed)
      Value:
      (2 elem)
BgNVBA             pcDEUMBIGA1UEAwwLTUNIUCBTSUdORVIwHhcNMjMwNTMw
MDgxMI             5MDgxMDQ3WjAqMRIwEAYDVQQKDAlNaWNyb2NoaXAxFDAS
BgNVBA             WSUNFMHYwEAYHKoZIzj0CAQYFK4EEACIDYgAEEH6pnd/u
```

```
97 78 2D 24 F8 0A 8B 8F  CB AD 30 0A 06 08 2A 86
48 CE 3D 04 03 03 03 69  00 30 66 02 31 00 B8 4B
C1 6D 81 E1 C1 69 46 A8  62 BF B3 8A 22 46 3E 89
6F 51 DB 21 C3 B6 6F D3  B1 08 8B 87 E9 21 2C 9D
41 CA C5 D6 02 A9 FC 6B  36 E0 A8 A0 9C 9A 02 31
00 A5 0B 9B DF 7A 41 6C  F6 8E D6 08 22 FC 26 92
EB 21 AB 02 A0 8C C8 C3  53 DC D2 6C 56 DF 53 E0
52 5D A0 11 1F 04 A3 2A  8D CA 25 61 95 83 70 69
A3
```

Signature R & S

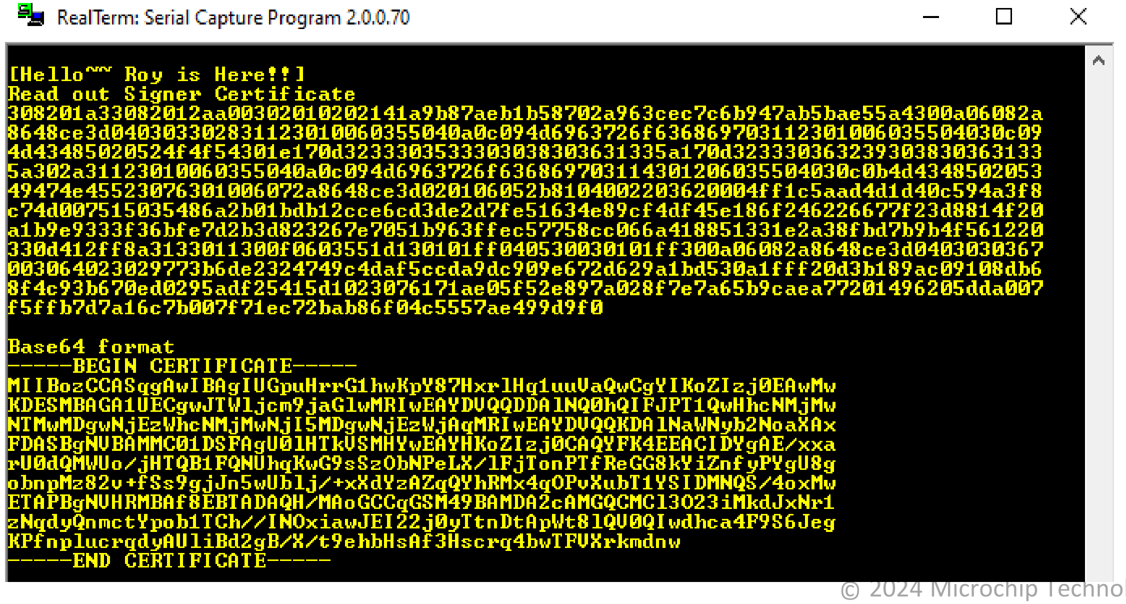
```
static const uint8_t DeviceCert[] = {
0x30, 0x82, 0x01, 0x8D, 0x30, 0x82, 0x01, 0x12, 0x02, 0x14, 0x75, 0x8A, 0x17, 0xDF, 0xD1, 0x01,
0xC6, 0xD3, 0xB4, 0x15, 0x71, 0xD2, 0x0C, 0x93, 0x6D, 0x92, 0x6B, 0x81, 0xC8, 0x4B, 0x30, 0x0A,
0x06, 0x08, 0x2A, 0x86, 0x48, 0xCE, 0x3D, 0x04, 0x03, 0x03, 0x30, 0x2A, 0x31, 0x12, 0x30, 0x10,
0x06, 0x03, 0x55, 0x04, 0x0A, 0x0C, 0x09, 0x4D, 0x69, 0x63, 0x72, 0x6F, 0x63, 0x68, 0x69, 0x70,
0x31, 0x14, 0x30, 0x12, 0x06, 0x03, 0x55, 0x04, 0x03, 0x0C, 0x0B, 0x4D, 0x43, 0x48, 0x50, 0x20,
0x53, 0x49, 0x47, 0x4E, 0x45, 0x52, 0x30, 0x1E, 0x17, 0x0D, 0x32, 0x33, 0x30, 0x35, 0x33, 0x30,
0x30, 0x38, 0x31, 0x30, 0x34, 0x37, 0x5A, 0x17, 0x0D, 0x32, 0x33, 0x30, 0x36, 0x32, 0x39, 0x30,
0x38, 0x31, 0x30, 0x34, 0x37, 0x5A, 0x30, 0x2A, 0x31, 0x12, 0x30, 0x10, 0x06, 0x03, 0x55, 0x04,
0x0A, 0x0C, 0x09, 0x4D, 0x69, 0x63, 0x72, 0x6F, 0x63, 0x68, 0x69, 0x70, 0x31, 0x14, 0x30, 0x12,
0x06, 0x03, 0x55, 0x04, 0x03, 0x0C, 0x0B, 0x4D, 0x43, 0x48, 0x50, 0x20, 0x44, 0x45, 0x56, 0x49,
0x43, 0x45, 0x30, 0x76, 0x30, 0x10, 0x06, 0x07, 0x2A, 0x86, 0x48, 0xCE, 0x3D, 0x02, 0x01, 0x06,
0x05, 0x2B, 0x81, 0x04, 0x00, 0x22, 0x03, 0x62, 0x00, 0x04, 0x10, 0x7E, 0xA9, 0x9D, 0xDF, 0xEE,
0xE3, 0xBD, 0x2C, 0xBB, 0x3F, 0x92, 0x9D, 0xE7, 0x0D, 0x0A, 0xF2, 0x30, 0x7E, 0xE0, 0x26, 0x9A,
0xE1, 0xD2, 0xD4, 0xB3, 0x7F, 0x95, 0x7E, 0x63, 0x64, 0x78, 0x81, 0xB3, 0xB9, 0x18, 0x76, 0xEB,
0x58, 0x4C, 0x69, 0x94, 0x7C, 0x9C, 0x64, 0xD9, 0xF6, 0x73, 0x20, 0x5E, 0x31, 0x27, 0xB1, 0x7D,
0xF9, 0xFF, 0x4A, 0x08, 0xE3, 0xE8, 0x78, 0x8A, 0xD1, 0x19, 0x90, 0x43, 0x3E, 0x30, 0x91, 0x7B,
0xC5, 0xA8, 0x70, 0xC7, 0x1B, 0x15, 0xA5, 0x27, 0x88, 0x89, 0x1C, 0x81, 0xF9, 0xB4, 0x88, 0xE1,
0x97, 0x78, 0x2D, 0x24, 0xF8, 0x0A, 0x8B, 0x8F, 0xCB, 0xAD, 0x30, 0x0A, 0x06, 0x08, 0x2A, 0x86,
0x48, 0xCE, 0x3D, 0x04, 0x03, 0x03, 0x03, 0x69, 0x00, 0x30, 0x66, 0x02, 0x31, 0x00, 0xB8, 0x4B,
0xC1, 0x6D, 0x81, 0xE1, 0xC1, 0x69, 0x46, 0xA8, 0x62, 0xBF, 0xB3, 0x8A, 0x22, 0x46, 0x3E, 0x89,
0x6F, 0x51, 0xDB, 0x21, 0xC3, 0xB6, 0x6F, 0xD3, 0xB1, 0x08, 0x8B, 0x87, 0xE9, 0x21, 0x2C, 0x9D,
0x41, 0xCA, 0xC5, 0xD6, 0x02, 0xA9, 0xFC, 0x6B, 0x36, 0xE0, 0xA8, 0xA0, 0x9C, 0x9A, 0x02, 0x31,
0x00, 0xA5, 0x0B, 0x9B, 0xDF, 0x7A, 0x41, 0x6C, 0xF6, 0x8E, 0xD6, 0x08, 0x22, 0xFC, 0x26, 0x92,
0xEB, 0x21, 0xAB, 0x02, 0xA0, 0x8C, 0xC8, 0xC3, 0x53, 0xDC, 0xD2, 0x6C, 0x56, 0xDF, 0x53, 0xE0,
0x52, 0x5D, 0xA0, 0x11, 0x1F, 0x04, 0xA3, 0x2A, 0x8D, 0xCA, 0x25, 0x61, 0x95, 0x83, 0x70, 0x69,
0xA3
};
```

Microchip Proprietary and STRICTLY Confidential

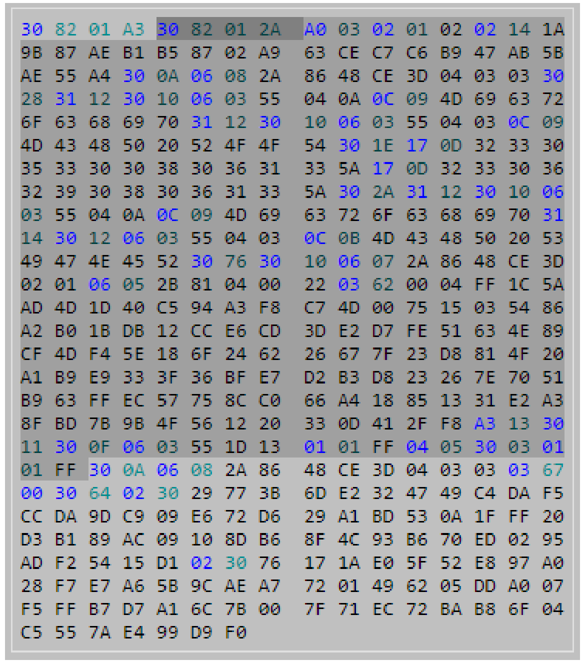
Lab 8 - Practice

1. Try to Read out the Signer Certificate
2. Use ASN.1 to decode the TBS area & Signature

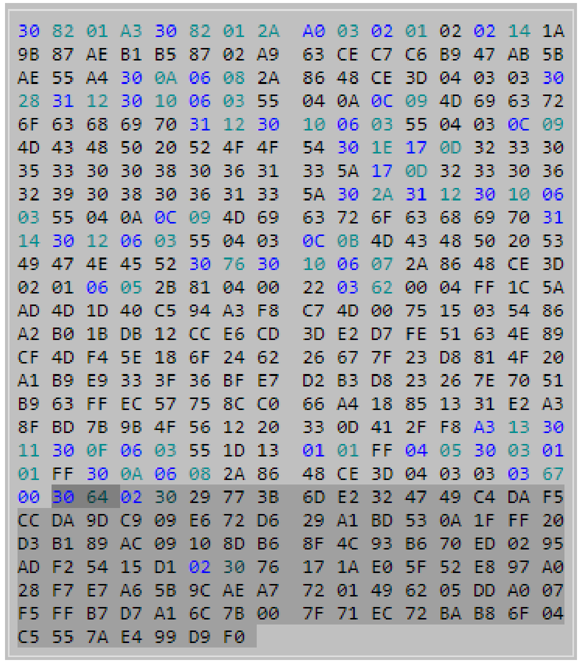
Element Name	Class	Type/Notes
Private Key @ 0x8007	Private Key	ECC P384 (Fixed Private Key)
Public Key @ 0x8006	Public Key	ECC P384 (Fixed paired Public key)
Private Key @ 0x8008	Private Key	ECC P384 (Random Device Private Key)
Public Key @ 0x800C	Public Key	ECC P384 (paired Device Public key)
signerCert @ 0x8201	Data	Example Signer Certificate (Hold signer Public key)
deviceCert @ 0x8200	Data	Example Device Certificate (Hold Device Public key)



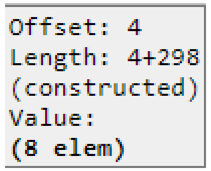
```
[Hello~~ Roy is Here!!]
Read out Signer Certificate
308201a33082012aa00302010202141a9b87aeb1b58702a963cec7c6b947ab5bae55a4300a06082a
8648ce3d04030330283112301006035504 0a0c094d6963726f636869703112301006035504030c09
4d43485020524f4f54301e170d3233303533303038303631335a170d323330363239303830363133
5a302a31123010060355040a0c094d6963726f63686970311430120603550403 0c0b4d434850 2053
49474e4552307630100 6072a8648ce3d020106052b8104002203620004ff1c5aad4d1d40c594a3f8
c74d007515035486a2b01bdb12cce6cd3de2d7fe51634e89cf4df45e186f246226677f23d8814f20
a1b9e9333f36bfe7d2b3d823267e7051b963ffec57758cc066a418851331e2a38fbd7b9b4f561220
330d412ff8a3133011300f0603551d130101ff04053003 0101ff300a06082a8648ce3d040303 0367
003064023029773b6de2324749c4daf5ccda9dc909e672d629a1bd530a1fff20d3b189ac09108db6
8f4c93b670ed0295adf25415d1023076171ae05f52e897a028f7e7a65b9caea77201496205dda007
f5ffb7d7a16c7b007f71ec72bab86f04c5557ae499d9f0

Base64 format
-----BEGIN CERTIFICATE-----
MIIBozCCASqgAwIBAgIUGpuHrrG1hwKpY87HxrlHq1uuVaQwCgYIKoZIzj0EAwMw
KDESMBAGA1UECgwJTWljcm9jaGlwMRIwEAYDVQQDDA1NQ0hQIFJPT1QwHhcNMjMw
NTMwMDgwNjEzWhcNMjMwNjI5MDgwNjEzWjAqMRIwEAYDVQQKDA1NaWNyb2NoaXAx
FDASBgNVBAMMC01DSFAgU01HTkVSMHYwEAYHKoZIzj0CAQYFK4EEACIDYgAE/xxa
rU0dQMWUo/jHTQB1FQNUhqKwG9sSzObNPeLX/lFjTonPTfReGG8kYiZnfyPYgU8g
obnpMz82v+fSs9gjJn5wUblj/+xXdYzAZqQYhRMx4qOPvXubT1YSIDMNQS/4oxMw
ETAPBgNVHRMBAf8EBTADAQH/MAoGCCqGSM49BAMDA2cAMGQCMC13O23iMkdJxNr1
zNqdyQnmctYpob1TCh//INOxiawJEI22j0yTtnDtApWt81QV0QIwdhca4F9S6Jeg
KPfnplucrqdyAUliBd2gB/X/t9ehbHsAf3Hscrq4bwTFVXrkmdnw
-----END CERTIFICATE-----
```



```
30 82 01 A3 30 82 01 2A  A0 03 02 01 02 02 14 1A
9B 87 AE B1 B5 87 02 A9  63 CE C7 C6 B9 47 AB 5B
AE 55 A4 30 0A 06 08 2A  86 48 CE 3D 04 03 03 30
28 31 12 30 10 06 03 55  04 0A 0C 09 4D 69 63 72
6F 63 68 69 70 31 12 30  10 06 03 55 04 03 0C 09
4D 43 48 50 20 52 4F 4F  54 30 1E 17 0D 32 33 30
35 33 30 30 38 30 36 31  33 5A 17 0D 32 33 30 36
32 39 30 38 30 36 31 33  5A 30 2A 31 12 30 10 06
03 55 04 0A 0C 09 4D 69  63 72 6F 63 68 69 70 31
14 30 12 06 03 55 04 03  0C 0B 4D 43 48 50 20 53
49 47 4E 45 52 30 76 30  10 06 07 2A 86 48 CE 3D
02 01 06 05 2B 81 04 00  22 03 62 00 04 FF 1C 5A
AD 4D 1D 40 C5 94 A3 F8  C7 4D 00 75 15 03 54 86
A2 B0 1B DB 12 CC E6 CD  3D E2 D7 FE 51 63 4E 89
CF 4D F4 5E 18 6F 24 62  26 67 7F 23 D8 81 4F 20
A1 B9 E9 33 3F 36 BF E7  D2 B3 D8 23 26 7E 70 51
B9 63 FF EC 57 75 8C C0  66 A4 18 85 13 31 E2 A3
8F BD 7B 9B 4F 56 12 20  33 0D 41 2F F8 A3 13 30
11 30 0F 06 03 55 1D 13  01 01 FF 04 05 30 03 01
01 FF 30 0A 06 08 2A 86  48 CE 3D 04 03 03 03 67
00 30 64 02 30 29 77 3B  6D E2 32 47 49 C4 DA F5
CC DA 9D C9 09 E6 72 D6  29 A1 BD 53 0A 1F FF 20
D3 B1 89 AC 09 10 8D B6  8F 4C 93 B6 70 ED 02 95
AD F2 54 15 D1 02 30 76  17 1A E0 5F 52 E8 97 A0
28 F7 E7 A6 5B 9C AE A7  72 01 49 62 05 DD A0 07
F5 FF B7 D7 A1 6C 7B 00  7F 71 EC 72 BA B8 6F 04
C5 55 7A E4 99 D9 F0
```

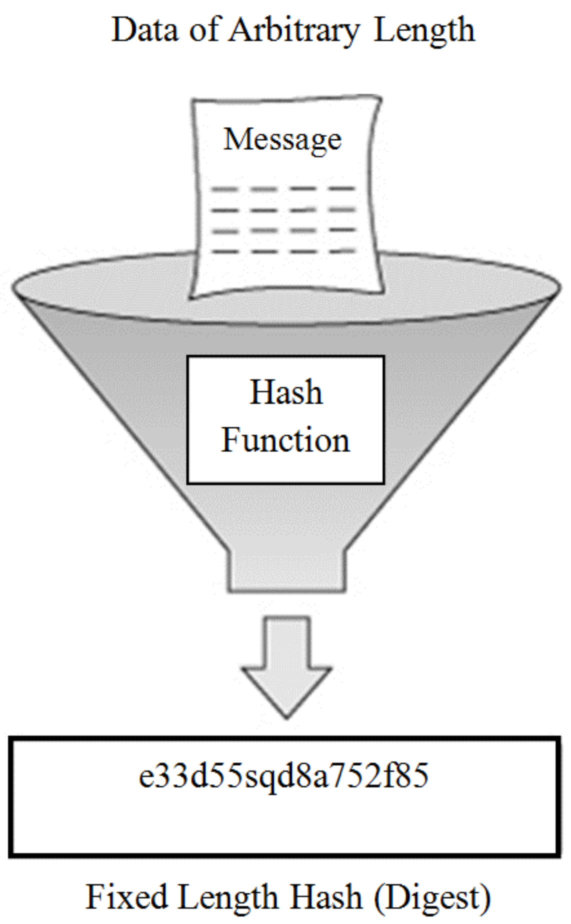


```
30 82 01 A3 30 82 01 2A  A0 03 02 01 02 02 14 1A
9B 87 AE B1 B5 87 02 A9  63 CE C7 C6 B9 47 AB 5B
AE 55 A4 30 0A 06 08 2A  86 48 CE 3D 04 03 03 30
28 31 12 30 10 06 03 55  04 0A 0C 09 4D 69 63 72
6F 63 68 69 70 31 12 30  10 06 03 55 04 03 0C 09
4D 43 48 50 20 52 4F 4F  54 30 1E 17 0D 32 33 30
35 33 30 30 38 30 36 31  33 5A 17 0D 32 33 30 36
32 39 30 38 30 36 31 33  5A 30 2A 31 12 30 10 06
03 55 04 0A 0C 09 4D 69  63 72 6F 63 68 69 70 31
14 30 12 06 03 55 04 03  0C 0B 4D 43 48 50 20 53
49 47 4E 45 52 30 76 30  10 06 07 2A 86 48 CE 3D
02 01 06 05 2B 81 04 00  22 03 62 00 04 FF 1C 5A
AD 4D 1D 40 C5 94 A3 F8  C7 4D 00 75 15 03 54 86
A2 B0 1B DB 12 CC E6 CD  3D E2 D7 FE 51 63 4E 89
CF 4D F4 5E 18 6F 24 62  26 67 7F 23 D8 81 4F 20
A1 B9 E9 33 3F 36 BF E7  D2 B3 D8 23 26 7E 70 51
B9 63 FF EC 57 75 8C C0  66 A4 18 85 13 31 E2 A3
8F BD 7B 9B 4F 56 12 20  33 0D 41 2F F8 A3 13 30
11 30 0F 06 03 55 1D 13  01 01 FF 04 05 30 03 01
01 FF 30 0A 06 08 2A 86  48 CE 3D 04 03 03 03 67
00 30 64 02 30 29 77 3B  6D E2 32 47 49 C4 DA F5
CC DA 9D C9 09 E6 72 D6  29 A1 BD 53 0A 1F FF 20
D3 B1 89 AC 09 10 8D B6  8F 4C 93 B6 70 ED 02 95
AD F2 54 15 D1 02 30 76  17 1A E0 5F 52 E8 97 A0
28 F7 E7 A6 5B 9C AE A7  72 01 49 62 05 DD A0 07
F5 FF B7 D7 A1 6C 7B 00  7F 71 EC 72 BA B8 6F 04
C5 55 7A E4 99 D9 F0
```



```
Offset: 4
Length: 4+298
(constructed)
Value:
(8 elem)
```

Let’s start the Lab8 & practice

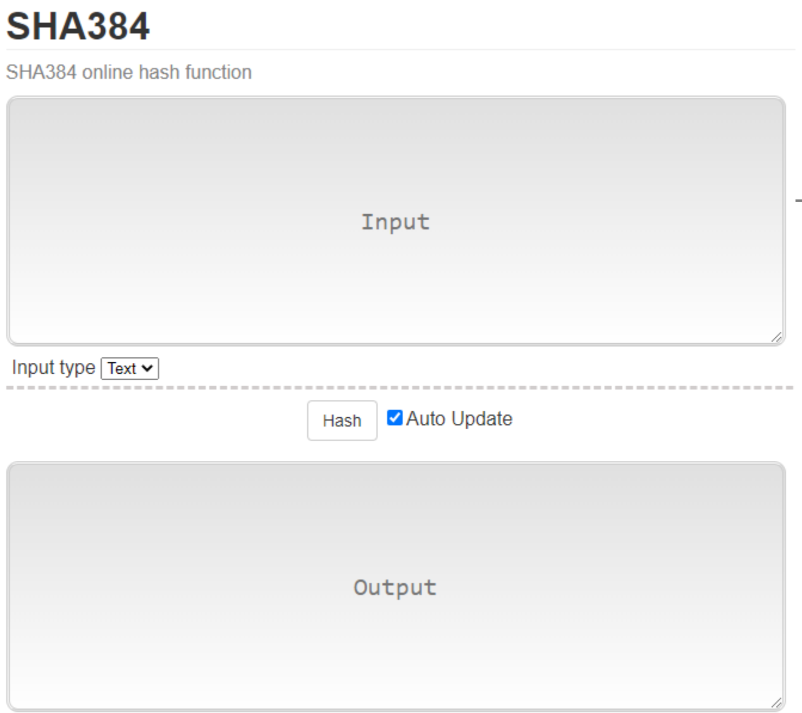


Hash Function

SHA384

Lab 9-1 Calculate digest using Online SHA384

Web Tool based



SHA384

SHA384 online hash function

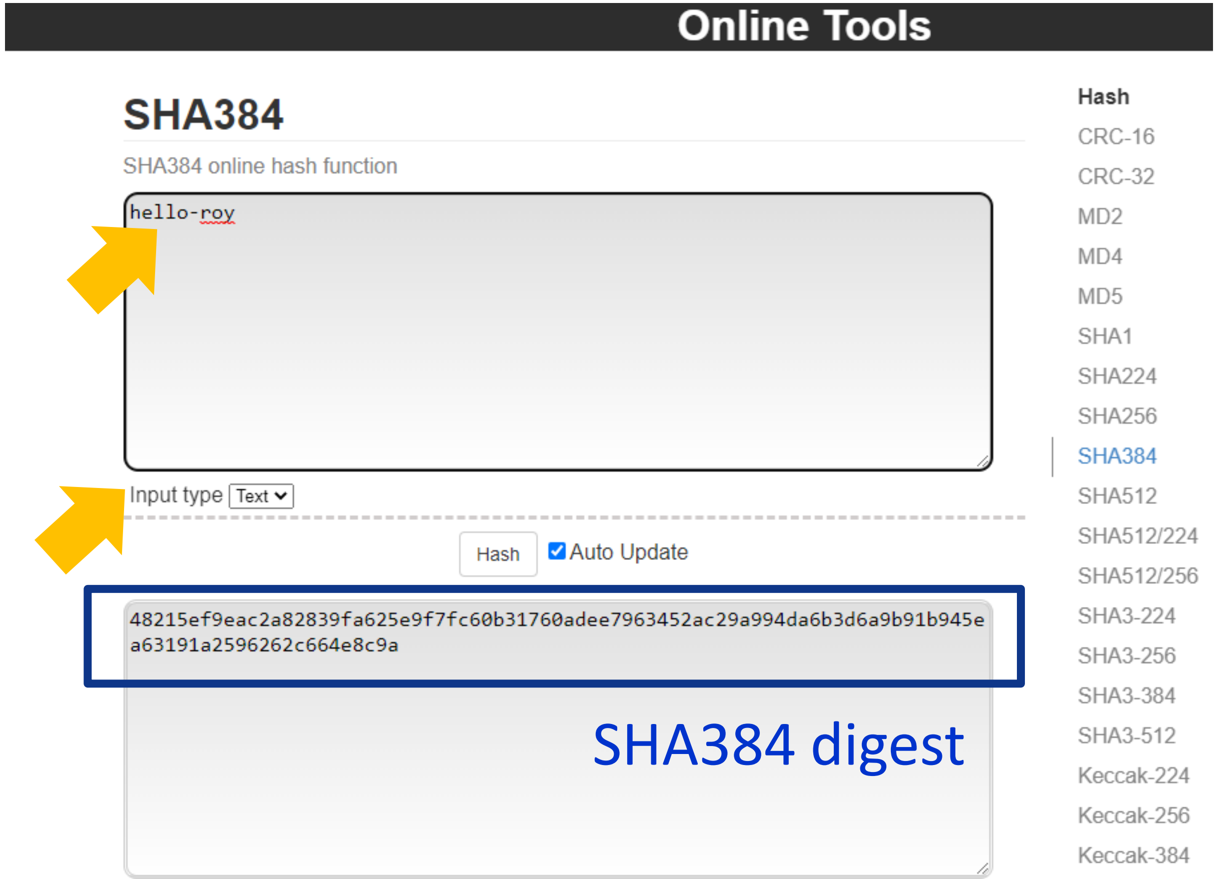
Input type Text

☑ Auto Update

https://emn178.github.io/online-tools/sha384.html

Step 9-1-1

- Fill “hello-roy” into upper frame (input side)
- Change the input type to “Text”
- Get the digest of “hello-roy” from below frame (output side)



```
hello-roy
```

Input type: Text

```
48215ef9eac2a82839fa625e9f7fc60b31760adee7963452ac29a994da6b3d6a9b91b945e
a63191a2596262c664e8c9a
```

SHA384 digest

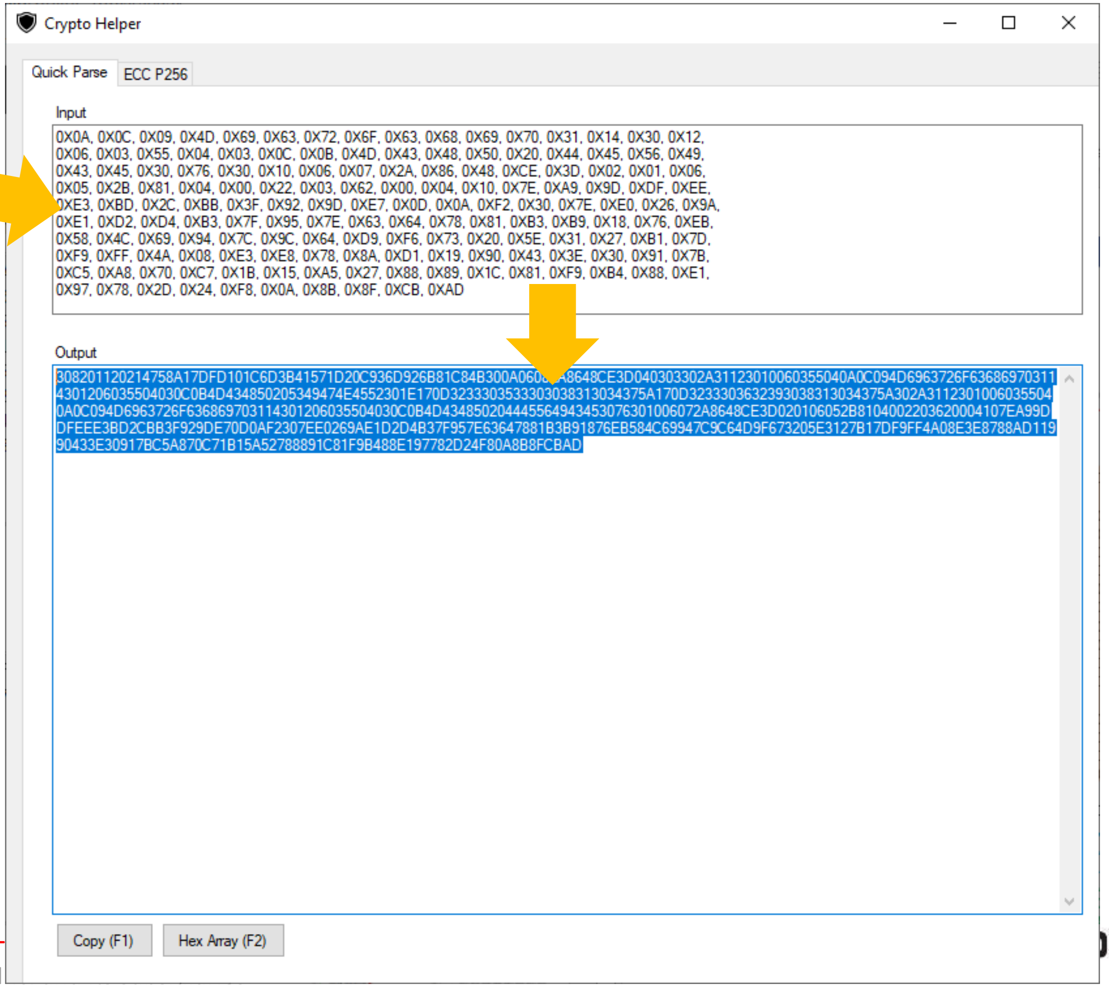
Lab9-2 – Calculate Device Certificate TBS digest

Web Tool based

Step 9-2-1

- Copy the Device Certificate TBS area hex (as Lab8)
- Use tool – CryptoTools to extract binaries

```
static const uint8_t DeviceCert[] = {
0x30, 0x82, 0x01, 0x8D, 0x30, 0x82, 0x01, 0x12, 0x02, 0x14, 0x75, 0x8A, 0x17, 0xDF, 0xD1, 0x01,
0xC6, 0xD3, 0xB4, 0x15, 0x71, 0xD2, 0x0C, 0x93, 0x6D, 0x92, 0x6B, 0x81, 0xC8, 0x4B, 0x30, 0x0A,
0x06, 0x08, 0x2A, 0x86, 0x48, 0xCE, 0x3D, 0x04, 0x03, 0x03, 0x30, 0x2A, 0x31, 0x12, 0x30, 0x10,
0x06, 0x03, 0x55, 0x04, 0x0A, 0x0C, 0x09, 0x4D, 0x69, 0x63, 0x72, 0x6F, 0x63, 0x68, 0x69, 0x70,
0x31, 0x14, 0x30, 0x12, 0x06, 0x03, 0x55, 0x04, 0x03, 0x0C, 0x0B, 0x4D, 0x43, 0x48, 0x50, 0x20,
0x53, 0x49, 0x47, 0x4E, 0x45, 0x52, 0x30, 0x1E, 0x17, 0x0D, 0x32, 0x33, 0x30, 0x35, 0x33, 0x30,
0x30, 0x38, 0x31, 0x30, 0x34, 0x37, 0x5A, 0x17, 0x0D, 0x32, 0x33, 0x30, 0x36, 0x32, 0x39, 0x30,
0x38, 0x31, 0x30, 0x34, 0x37, 0x5A, 0x30, 0x2A, 0x31, 0x12, 0x30, 0x10, 0x06, 0x03, 0x55, 0x04,
0x0A, 0x0C, 0x09, 0x4D, 0x69, 0x63, 0x72, 0x6F, 0x63, 0x68, 0x69, 0x70, 0x31, 0x14, 0x30, 0x12,
0x06, 0x03, 0x55, 0x04, 0x03, 0x0C, 0x0B, 0x4D, 0x43, 0x48, 0x50, 0x20, 0x44, 0x45, 0x56, 0x49,
0x43, 0x45, 0x30, 0x76, 0x30, 0x10, 0x06, 0x07, 0x2A, 0x86, 0x48, 0xCE, 0x3D, 0x02, 0x01, 0x06,
0x05, 0x2B, 0x81, 0x04, 0x00, 0x22, 0x03, 0x62, 0x00, 0x04, 0x10, 0x7E, 0xA9, 0x9D, 0xDF, 0xEE,
0xE3, 0xBD, 0x2C, 0xBB, 0x3F, 0x92, 0x9D, 0xE7, 0x0D, 0x0A, 0xF2, 0x30, 0x7E, 0xE0, 0x26, 0x9A,
0xE1, 0xD2, 0xD4, 0xB3, 0x7F, 0x95, 0x7E, 0x63, 0x64, 0x78, 0x81, 0xB3, 0xB9, 0x18, 0x76, 0xEB,
0x58, 0x4C, 0x69, 0x94, 0x7C, 0x9C, 0x64, 0xD9, 0xF6, 0x73, 0x20, 0x5E, 0x31, 0x27, 0xB1, 0x7D,
0xF9, 0xFF, 0x4A, 0x08, 0xE3, 0xE8, 0x78, 0x8A, 0xD1, 0x19, 0x90, 0x43, 0x3E, 0x30, 0x91, 0x7B,
0xC5, 0xA8, 0x70, 0xC7, 0x1B, 0x15, 0xA5, 0x27, 0x88, 0x89, 0x1C, 0x81, 0xF9, 0xB4, 0x88, 0xE1,
0x97, 0x78, 0x2D, 0x24, 0xF8, 0x0A, 0x8B, 0x8F, 0xCB, 0xAD, 0x30, 0x0A, 0x06, 0x08, 0x2A, 0x86,
0x48, 0xCE, 0x3D, 0x04, 0x03, 0x03, 0x03, 0x69, 0x00, 0x30, 0x66, 0x02, 0x31, 0x00, 0xB8, 0x4B,
0xC1, 0x6D, 0x81, 0xE1, 0xC1, 0x69, 0x46, 0xA8, 0x62, 0xBF, 0xB3, 0x8A, 0x22, 0x46, 0x3E, 0x89,
0x6F, 0x51, 0xDB, 0x21, 0xC3, 0xB6, 0x6F, 0xD3, 0xB1, 0x08, 0x8B, 0x87, 0xE9, 0x21, 0x2C, 0x9D,
0x41, 0xCA, 0xC5, 0xD6, 0x02, 0xA9, 0xFC, 0x6B, 0x36, 0xE0, 0xA8, 0xA0, 0x9C, 0x9A, 0x02, 0x31,
0x00, 0xA5, 0x0B, 0x9B, 0xDF, 0x7A, 0x41, 0x6C, 0xF6, 0x8E, 0xD6, 0x08, 0x22, 0xFC, 0x26, 0x92,
0xEB, 0x21, 0xAB, 0x02, 0xA0, 0x8C, 0xC8, 0xC3, 0x53, 0xDC, 0xD2, 0x6C, 0x56, 0xDF, 0x53, 0xE0,
0x52, 0x5D, 0xA0, 0x11, 0x1F, 0x04, 0xA3, 0x2A, 0x8D, 0xCA, 0x25, 0x61, 0x95, 0x83, 0x70, 0x69,
0xA3
};
```



Input

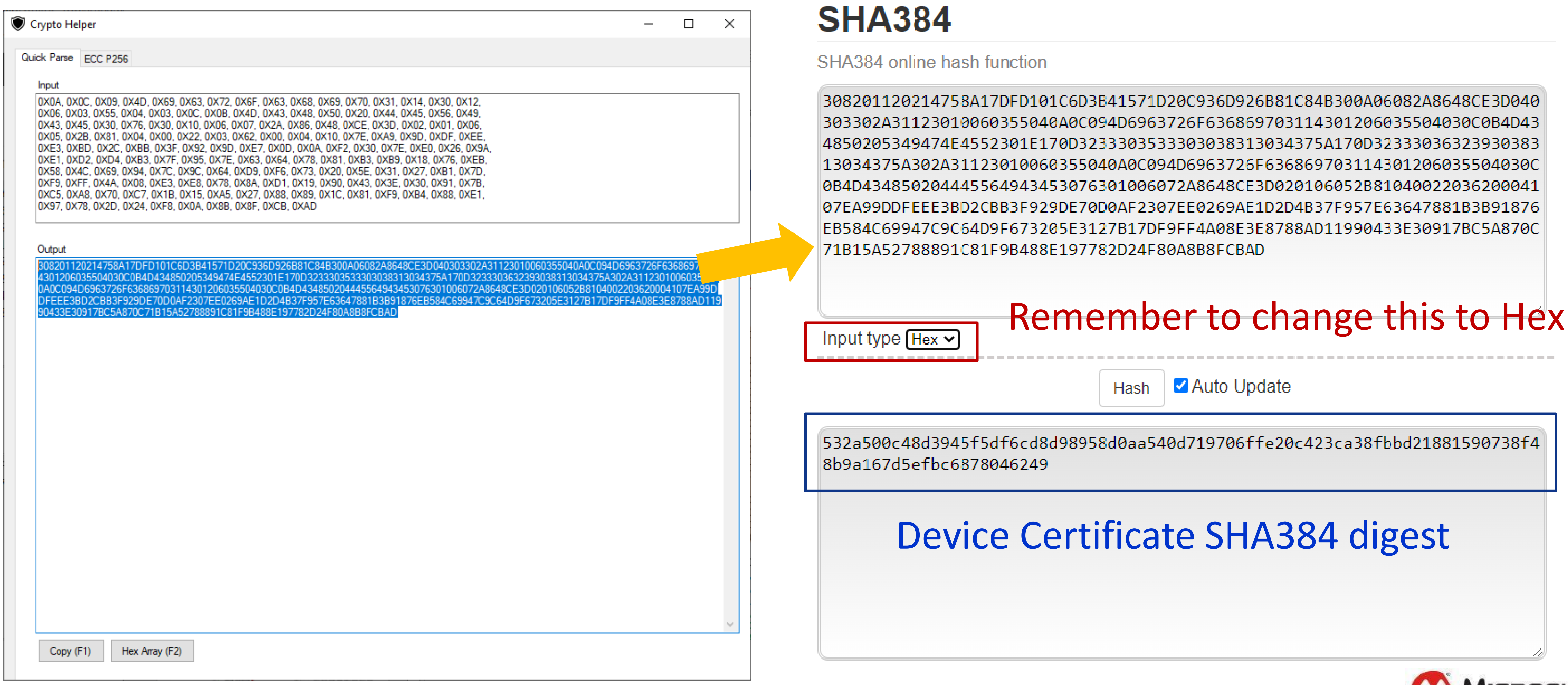
```
0X0A, 0X0C, 0X09, 0X4D, 0X69, 0X63, 0X72, 0X6F, 0X63, 0X68, 0X69, 0X70, 0X31, 0X14, 0X30, 0X12,
0X06, 0X03, 0X55, 0X04, 0X03, 0X0C, 0X0B, 0X4D, 0X43, 0X48, 0X50, 0X20, 0X44, 0X45, 0X56, 0X49,
0X43, 0X45, 0X30, 0X76, 0X30, 0X10, 0X06, 0X07, 0X2A, 0X86, 0X48, 0XCE, 0X3D, 0X02, 0X01, 0X06,
0X05, 0X2B, 0X81, 0X04, 0X00, 0X22, 0X03, 0X62, 0X00, 0X04, 0X10, 0X7E, 0XA9, 0X9D, 0XDF, 0XEE,
0XE3, 0XBD, 0X2C, 0XBB, 0X3F, 0X92, 0X9D, 0XE7, 0X0D, 0X0A, 0XF2, 0X30, 0X7E, 0XE0, 0X26, 0X9A,
0XE1, 0XD2, 0XD4, 0XB3, 0X7F, 0X95, 0X7E, 0X63, 0X64, 0X78, 0X81, 0XB3, 0XB9, 0X18, 0X76, 0XEB,
0X58, 0X4C, 0X69, 0X94, 0X7C, 0X9C, 0X64, 0XD9, 0XF6, 0X73, 0X20, 0X5E, 0X31, 0X27, 0XB1, 0X7D,
0XF9, 0XFF, 0X4A, 0X08, 0XE3, 0XE8, 0X78, 0X8A, 0XD1, 0X19, 0X90, 0X43, 0X3E, 0X30, 0X91, 0X7B,
0XC5, 0XA8, 0X70, 0XC7, 0X1B, 0X15, 0XA5, 0X27, 0X88, 0X89, 0X1C, 0X81, 0XF9, 0XB4, 0X88, 0XE1,
0X97, 0X78, 0X2D, 0X24, 0XF8, 0X0A, 0X8B, 0X8F, 0XCB, 0XAD
```

Output

```
308201120214758A17DFD101C6D3B41571D20C936D926B81C84B300A06082A8648CE3D040303302A31123010060355040A0C094D6963726F63686970311
4301206035504030C0B4D434850205349474E4552301E170D3233303533303038313034375A170D3233303632393038313034375A302A3112301006035504
0A0C094D6963726F636869703114301206035504030C0B4D434850204445564943453076301006072A8648CE3D020106052B8104002203620004107EA99D
DFEEE3BD2CBB3F929DE70D0AF2307EE0269AE1D2D4B37F957E63647881B3B91876EB584C69947C9C64D9F673205E3127B17DF9FF4A08E3E8788AD119
90433E30917BC5A870C71B15A52788891C81F9B488E197782D24F80A8B8FCBAD
```

Step 9-2-2

- **Paste binaries to online SHA384 to calculate digest**



Input

```
0X0A, 0X0C, 0X09, 0X4D, 0X69, 0X63, 0X72, 0X6F, 0X63, 0X68, 0X69, 0X70, 0X31, 0X14, 0X30, 0X12,
0X06, 0X03, 0X55, 0X04, 0X03, 0X0C, 0X0B, 0X4D, 0X43, 0X48, 0X50, 0X20, 0X44, 0X45, 0X56, 0X49,
0X43, 0X45, 0X30, 0X76, 0X30, 0X10, 0X06, 0X07, 0X2A, 0X86, 0X48, 0XCE, 0X3D, 0X02, 0X01, 0X06,
0X05, 0X2B, 0X81, 0X04, 0X00, 0X22, 0X03, 0X62, 0X00, 0X04, 0X10, 0X7E, 0XA9, 0X9D, 0XDF, 0XEE,
0XE3, 0XBD, 0X2C, 0XBB, 0X3F, 0X92, 0X9D, 0XE7, 0X0D, 0X0A, 0XF2, 0X30, 0X7E, 0XE0, 0X26, 0X9A,
0XE1, 0XD2, 0XD4, 0XB3, 0X7F, 0X95, 0X7E, 0X63, 0X64, 0X78, 0X81, 0XB3, 0XB9, 0X18, 0X76, 0XEB,
0X58, 0X4C, 0X69, 0X94, 0X7C, 0X9C, 0X64, 0XD9, 0XF6, 0X73, 0X20, 0X5E, 0X31, 0X27, 0XB1, 0X7D,
0XF9, 0XFF, 0X4A, 0X08, 0XE3, 0XE8, 0X78, 0X8A, 0XD1, 0X19, 0X90, 0X43, 0X3E, 0X30, 0X91, 0X7B,
0XC5, 0XA8, 0X70, 0XC7, 0X1B, 0X15, 0XA5, 0X27, 0X88, 0X89, 0X1C, 0X81, 0XF9, 0XB4, 0X88, 0XE1,
0X97, 0X78, 0X2D, 0X24, 0XF8, 0X0A, 0X8B, 0X8F, 0XCB, 0XAD
```

Output

```
308201120214758A17DFD101C6D3B41571D20C936D926B81C84B300A06082A8648CE3D040303302A3112301006035504 0A0C094D6963726F6368697
4301206035504030C0B4D434850205349474E4552301E170D3233303533303038313034375A170D3233303632393038313034375A302A3112301006035
0A0C094D6963726F6368697031143012060355040 30C0B4D434850204445564943453076301006072A8648CE3D020106052B8104002203620004107EA99D
DFEEE3BD2CBB3F929DE70D0AF2307EE0269AE1D2D4B37F957E63647881B3B91876EB584C69947C9C64D9F673205E3127B17DF9FF4A08E3E8788AD119
90433E30917BC5A870C71B15A52788891C81F9B488E197782D24F80A8B8FCBAD
```

SHA384

SHA384 online hash function

```
308201120214758A17DFD101C6D3B41571D20C936D926B81C84B300A06082A8648CE3D040
303302A3112301006035504 0A0C094D6963726F636869703114301206035504030C0B4D43
4850205349474E4552301E170D3233303533303038313034375A170D3233303632393038 3
13034375A302A3112301006035504 0A0C094D6963726F636869703114301206035504030C
0B4D434850204445564943453076301006072A8648CE3D020106052B8104002203620004 1
07EA99DDFEEE3BD2CBB3F929DE70D0AF2307EE0269AE1D2D4B37F957E63647881B3B91876
EB584C69947C9C64D9F673205E3127B17DF9FF4A08E3E8788AD11990433E30917BC5A870C
71B15A52788891C81F9B488E197782D24F80A8B8FCBAD
```

Remember to change this to Hex!

Input type: Hex

☑ Auto Update

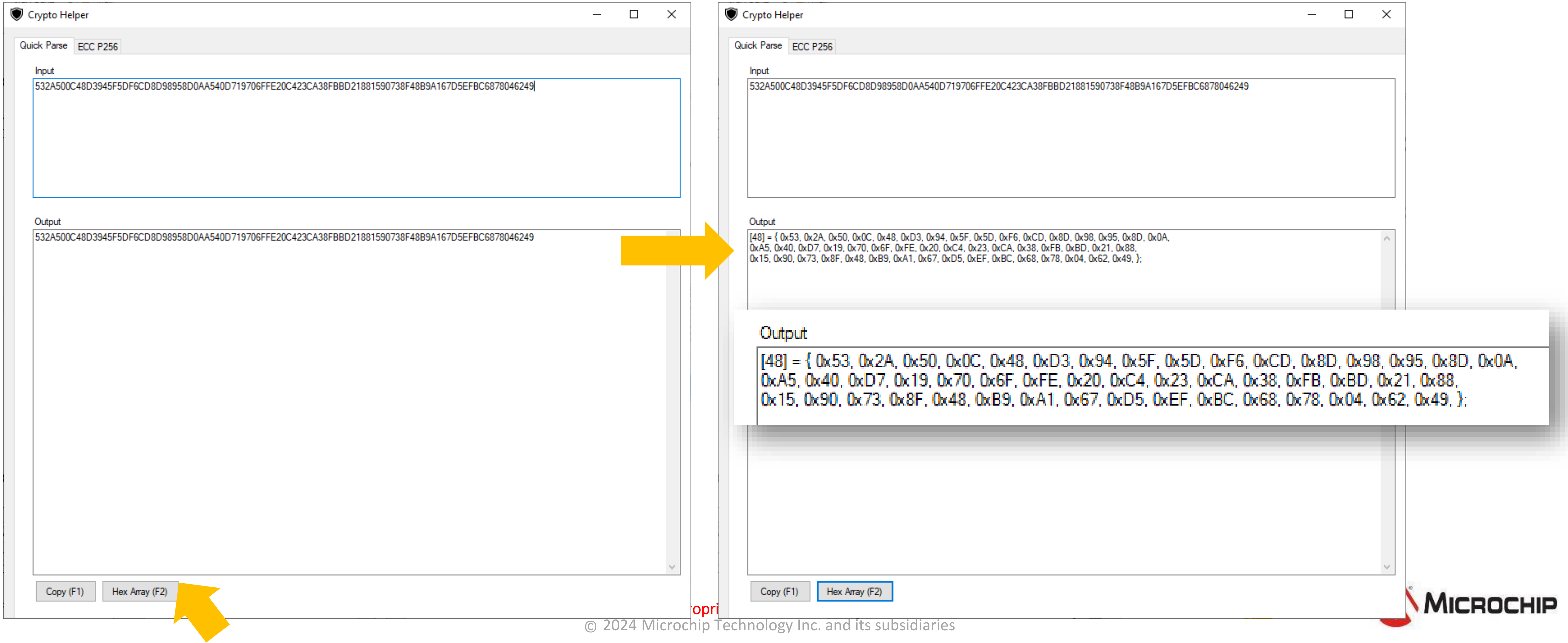
```
532a500c48d3945f5df6cd8d98958d0aa540d719706ffe20c423ca38fbbd21881590738f4
8b9a167d5efbc6878046249
```

Device Certificate SHA384 digest

Microchip Proprietary and STRICTLY Confidential

CryptoTools - further function (option)

1. **Copy the binaries to its input side**
2. **Press the “Hex Array” Button to get hex array, easy to use in code**



Input

```
532A500C48D3945F5DF6CD8D98958D0AA540D719706FFE20C423CA38FBBD21881590738F48B9A167D5EFBC6878046249
```

Output

```
532A500C48D3945F5DF6CD8D98958D0AA540D719706FFE20C423CA38FBBD21881590738F48B9A167D5EFBC6878046249
```

Input

```
532A500C48D3945F5DF6CD8D98958D0AA540D719706FFE20C423CA38FBBD21881590738F48B9A167D5EFBC6878046249
```

Output

```
[48] = { 0x53, 0x2A, 0x50, 0x0C, 0x48, 0xD3, 0x94, 0x5F, 0x5D, 0xF6, 0xCD, 0x8D, 0x98, 0x95, 0x8D, 0x0A,
0xA5, 0x40, 0xD7, 0x19, 0x70, 0x6F, 0xFE, 0x20, 0xC4, 0x23, 0xCA, 0x38, 0xFB, 0xBD, 0x21, 0x88,
0x15, 0x90, 0x73, 0x8F, 0x48, 0xB9, 0xA1, 0x67, 0xD5, 0xEF, 0xBC, 0x68, 0x78, 0x04, 0x62, 0x49, };
```

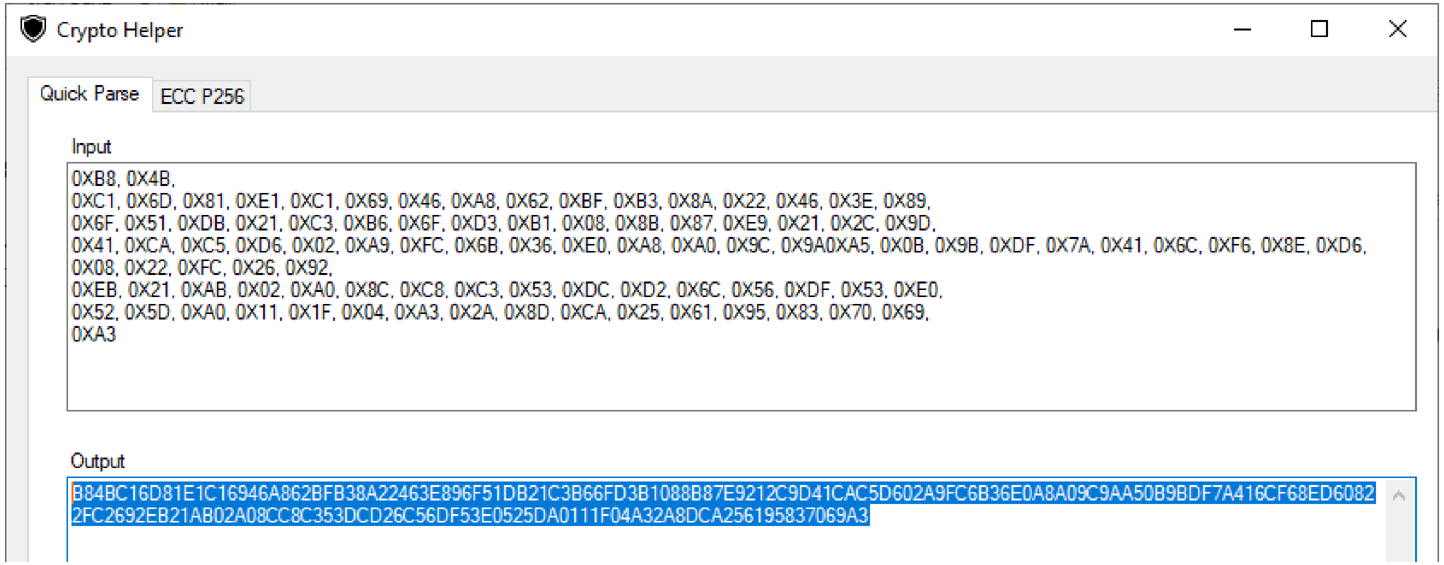
Output

```
[48] = { 0x53, 0x2A, 0x50, 0x0C, 0x48, 0xD3, 0x94, 0x5F, 0x5D, 0xF6, 0xCD, 0x8D, 0x98, 0x95, 0x8D, 0x0A,
0xA5, 0x40, 0xD7, 0x19, 0x70, 0x6F, 0xFE, 0x20, 0xC4, 0x23, 0xCA, 0x38, 0xFB, 0xBD, 0x21, 0x88,
0x15, 0x90, 0x73, 0x8F, 0x48, 0xB9, 0xA1, 0x67, 0xD5, 0xEF, 0xBC, 0x68, 0x78, 0x04, 0x62, 0x49, };
```

Lab 9 Practice – extract the Device signature

- **Copy the Device Certificate signature hex (as Lab 8)**
- **Paste the signature R into CryptoTools**
- **Paste the signature S into CryptoTools**

```
static const uint8_t DeviceCert[] = {
0x30, 0x82, 0x01, 0x8D, 0x30, 0x82, 0x01, 0x12, 0x02, 0x14, 0x75, 0x8A, 0x17, 0xDF, 0xD1, 0x01,
0xC6, 0xD3, 0xB4, 0x15, 0x71, 0xD2, 0x0C, 0x93, 0x6D, 0x92, 0x6B, 0x81, 0xC8, 0x4B, 0x30, 0x0A,
0x06, 0x08, 0x2A, 0x86, 0x48, 0xCE, 0x3D, 0x04, 0x03, 0x03, 0x30, 0x2A, 0x31, 0x12, 0x30, 0x10,
0x06, 0x03, 0x55, 0x04, 0x0A, 0x0C, 0x09, 0x4D, 0x69, 0x63, 0x72, 0x6F, 0x63, 0x68, 0x69, 0x70,
0x31, 0x14, 0x30, 0x12, 0x06, 0x03, 0x55, 0x04, 0x03, 0x0C, 0x0B, 0x4D, 0x43, 0x48, 0x50, 0x20,
0x53, 0x49, 0x47, 0x4E, 0x45, 0x52, 0x30, 0x1E, 0x17, 0x0D, 0x32, 0x33, 0x30, 0x35, 0x33, 0x30,
0x30, 0x38, 0x31, 0x30, 0x34, 0x37, 0x5A, 0x17, 0x0D, 0x32, 0x33, 0x30, 0x36, 0x32, 0x39, 0x30,
0x38, 0x31, 0x30, 0x34, 0x37, 0x5A, 0x30, 0x2A, 0x31, 0x12, 0x30, 0x10, 0x06, 0x03, 0x55, 0x04,
0x0A, 0x0C, 0x09, 0x4D, 0x69, 0x63, 0x72, 0x6F, 0x63, 0x68, 0x69, 0x70, 0x31, 0x14, 0x30, 0x12,
0x06, 0x03, 0x55, 0x04, 0x03, 0x0C, 0x0B, 0x4D, 0x43, 0x48, 0x50, 0x20, 0x44, 0x45, 0x56, 0x49,
0x43, 0x45, 0x30, 0x76, 0x30, 0x10, 0x06, 0x07, 0x2A, 0x86, 0x48, 0xCE, 0x3D, 0x02, 0x01, 0x06,
0x05, 0x2B, 0x81, 0x04, 0x00, 0x22, 0x03, 0x62, 0x00, 0x04, 0x10, 0x7E, 0xA9, 0x9D, 0xDF, 0xEE,
0xE3, 0xBD, 0x2C, 0xBB, 0x3F, 0x92, 0x9D, 0xE7, 0x0D, 0x0A, 0xF2, 0x30, 0x7E, 0xE0, 0x26, 0x9A,
0xE1, 0xD2, 0xD4, 0xB3, 0x7F, 0x95, 0x7E, 0x63, 0x64, 0x78, 0x81, 0xB3, 0xB9, 0x18, 0x76, 0xEB,
0x58, 0x4C, 0x69, 0x94, 0x7C, 0x9C, 0x64, 0xD9, 0xF6, 0x73, 0x20, 0x5E, 0x31, 0x27, 0xB1, 0x7D,
0xF9, 0xFF, 0x4A, 0x08, 0xE3, 0xE8, 0x78, 0x8A, 0xD1, 0x19, 0x90, 0x43, 0x3E, 0x30, 0x91, 0x7B,
0xC5, 0xA8, 0x70, 0xC7, 0x1B, 0x15, 0xA5, 0x27, 0x88, 0x89, 0x1C, 0x81, 0xF9, 0xB4, 0x88, 0xE1,
0x97, 0x78, 0x2D, 0x24, 0xF8, 0x0A, 0x8B, 0x8F, 0xCB, 0xAD, 0x30, 0x0A, 0x06, 0x08, 0x2A, 0x86,
0x48, 0xCE, 0x3D, 0x04, 0x03, 0x03, 0x03, 0x69, 0x00, 0x30, 0x66, 0x02, 0x31, 0x00, 0xB8, 0x4B,
0xC1, 0x6D, 0x81, 0xE1, 0xC1, 0x69, 0x46, 0xA8, 0x62, 0xBF, 0xB3, 0x8A, 0x22, 0x46, 0x3E, 0x89,
0x6F, 0x51, 0xDB, 0x21, 0xC3, 0xB6, 0x6F, 0xD3, 0xB1, 0x08, 0x8B, 0x87, 0xE9, 0x21, 0x2C, 0x9D,
0x41, 0xCA, 0xC5, 0xD6, 0x02, 0xA9, 0xFC, 0x6B, 0x36, 0xE0, 0xA8, 0xA0, 0x9C, 0x9A, 0x02, 0x31,
0x00, 0xA5, 0x0B, 0x9B, 0xDF, 0x7A, 0x41, 0x6C, 0xF6, 0x8E, 0xD6, 0x08, 0x22, 0xFC, 0x26, 0x92,
0xEB, 0x21, 0xAB, 0x02, 0xA0, 0x8C, 0xC8, 0xC3, 0x53, 0xDC, 0xD2, 0x6C, 0x56, 0xDF, 0x53, 0xE0,
0x52, 0x5D, 0xA0, 0x11, 0x1F, 0x04, 0xA3, 0x2A, 0x8D, 0xCA, 0x25, 0x61, 0x95, 0x83, 0x70, 0x69,
0xA3
};
```



Input

```
0XB8, 0X4B,
0XC1, 0X6D, 0X81, 0XE1, 0XC1, 0X69, 0X46, 0XA8, 0X62, 0XBF, 0XB3, 0X8A, 0X22, 0X46, 0X3E, 0X89,
0X6F, 0X51, 0XDB, 0X21, 0XC3, 0XB6, 0X6F, 0XD3, 0XB1, 0X08, 0X8B, 0X87, 0XE9, 0X21, 0X2C, 0X9D,
0X41, 0XCA, 0XC5, 0XD6, 0X02, 0XA9, 0XFC, 0X6B, 0X36, 0XE0, 0XA8, 0XA0, 0X9C, 0X9A0XA5, 0X0B, 0X9B, 0XDF, 0X7A, 0X41, 0X6C, 0XF6, 0X8E, 0XD6,
0X08, 0X22, 0XFC, 0X26, 0X92,
0XEB, 0X21, 0XAB, 0X02, 0XA0, 0X8C, 0XC8, 0XC3, 0X53, 0XDC, 0XD2, 0X6C, 0X56, 0XDF, 0X53, 0XE0,
0X52, 0X5D, 0XA0, 0X11, 0X1F, 0X04, 0XA3, 0X2A, 0X8D, 0XCA, 0X25, 0X61, 0X95, 0X83, 0X70, 0X69,
0XA3
```

Output

```
B84BC16D81E1C16946A862BFB38A22463E896F51DB21C3B66FD3B1088B87E9212C9D41CAC5D602A9FC6B36E0A8A09C9AA50B9BDF7A416CF68ED6082
2FC2692EB21AB02A08CC8C353DCD26C56DF53E0525DA0111F04A32A8DCA256195837069A3
```

Lab9-3 – Calculate SHA384 using dsPIC33CK

Assembly based

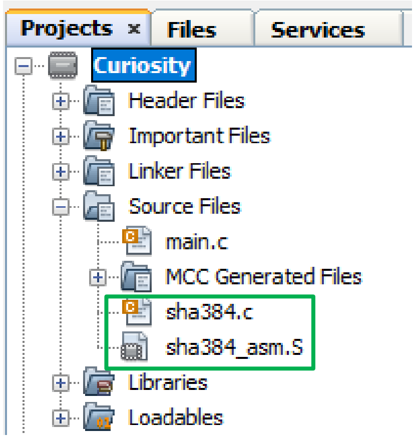
Use: Lab9.txt

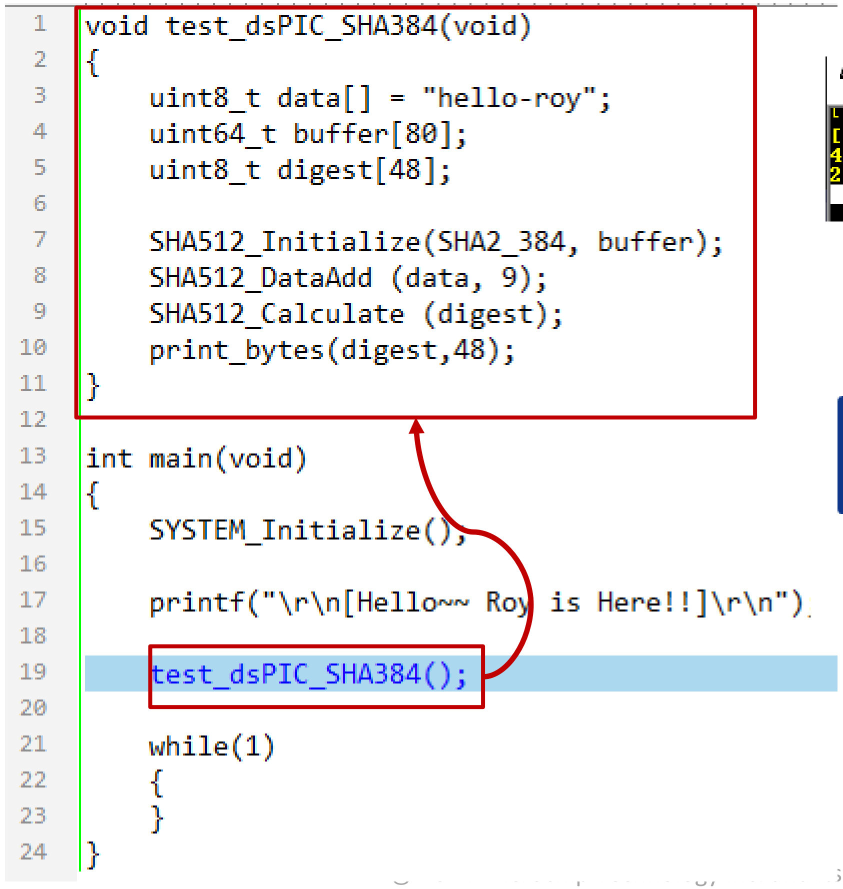
sha384.c

sha384_asm.s

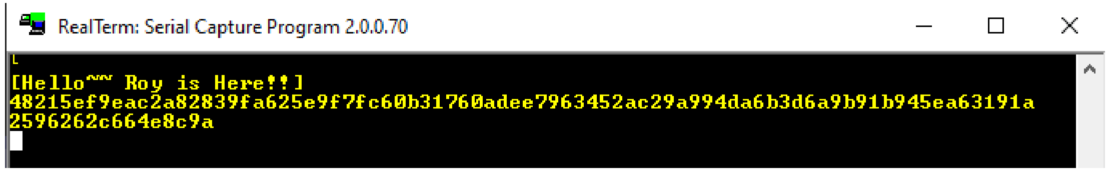
Step 9-3-1

- Add sha384.c & sha384_asm.s into project
- Copy functions from Lab9.txt to main.c , Program & Run!
- Compare the result with Lab 9-1.

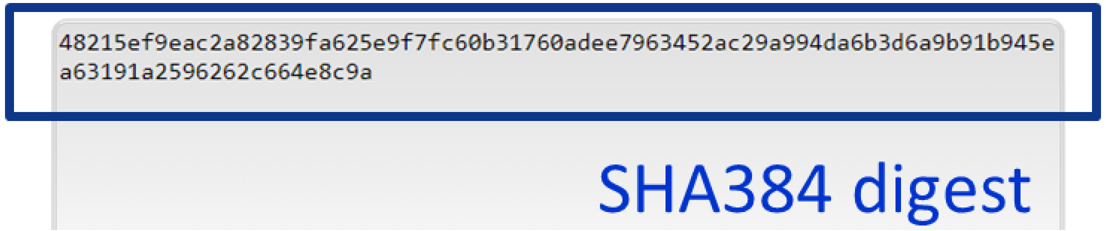




```c
void test_dsPIC_SHA384(void)
{
    uint8_t data[] = "hello-roy";
    uint64_t buffer[80];
    uint8_t digest[48];

    SHA512_Initialize(SHA2_384, buffer);
    SHA512_DataAdd (data, 9);
    SHA512_Calculate (digest);
    print_bytes(digest,48);
}

int main(void)
{
    SYSTEM_Initialize();

    printf("\r\n[Hello~~ Roy is Here!!]\r\n");

    test_dsPIC_SHA384();

    while(1)
    {
    }
}
```



```
[Hello~~ Roy is Here!!]
48215ef9eac2a82839fa625e9f7fc60b31760adee7963452ac29a994da6b3d6a9b91b945ea63191a
2596262c664e8c9a
```



```
48215ef9eac2a82839fa625e9f7fc60b31760adee7963452ac29a994da6b3d6a9b91b945e
a63191a2596262c664e8c9a
```

SHA384 digest

Let’s start the Lab9-1~9-3 & practices

Lab10 – Verify Device Certificate

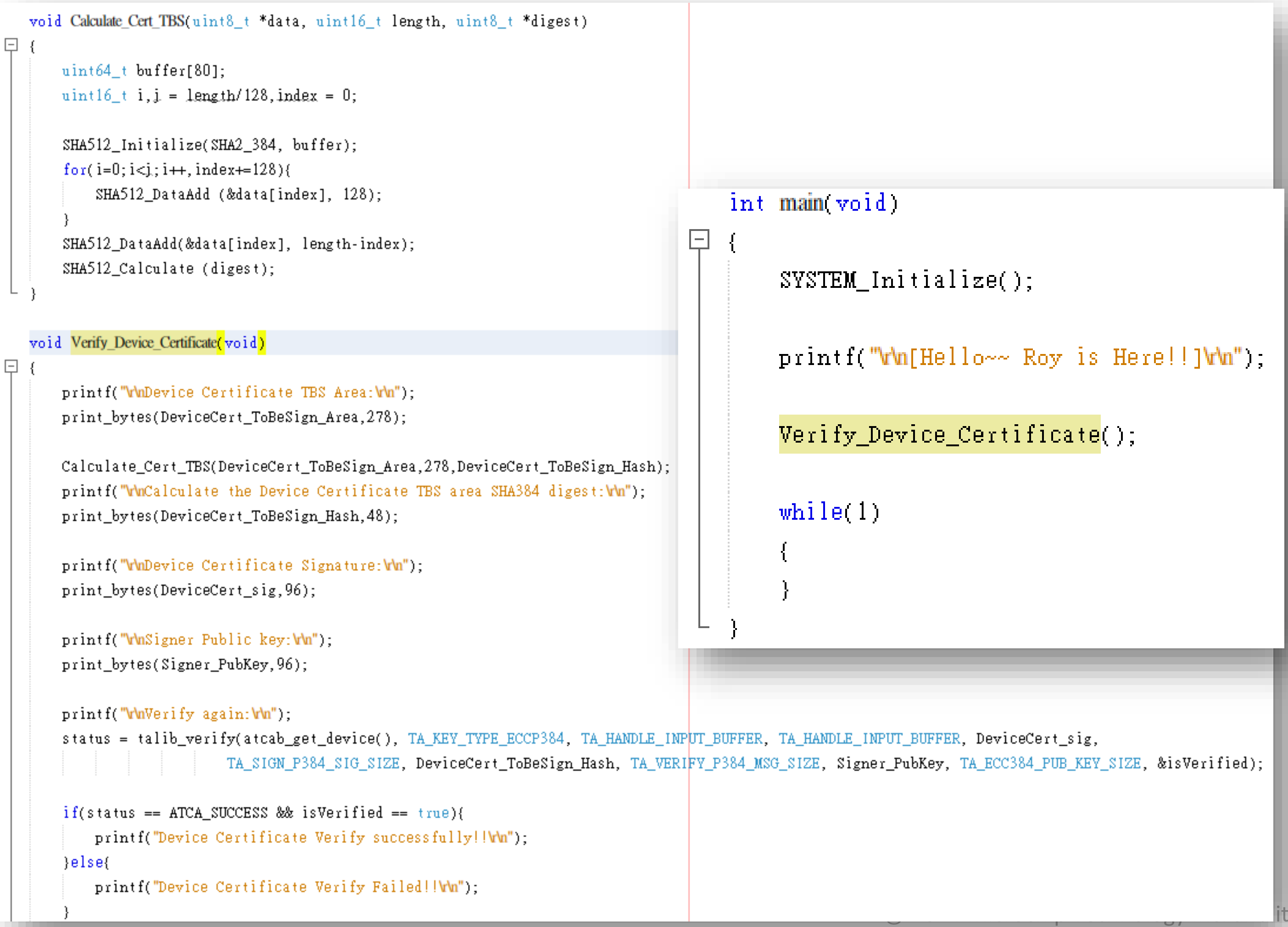
Use: Lab10.txt

sha384.c

sha384_asm.s

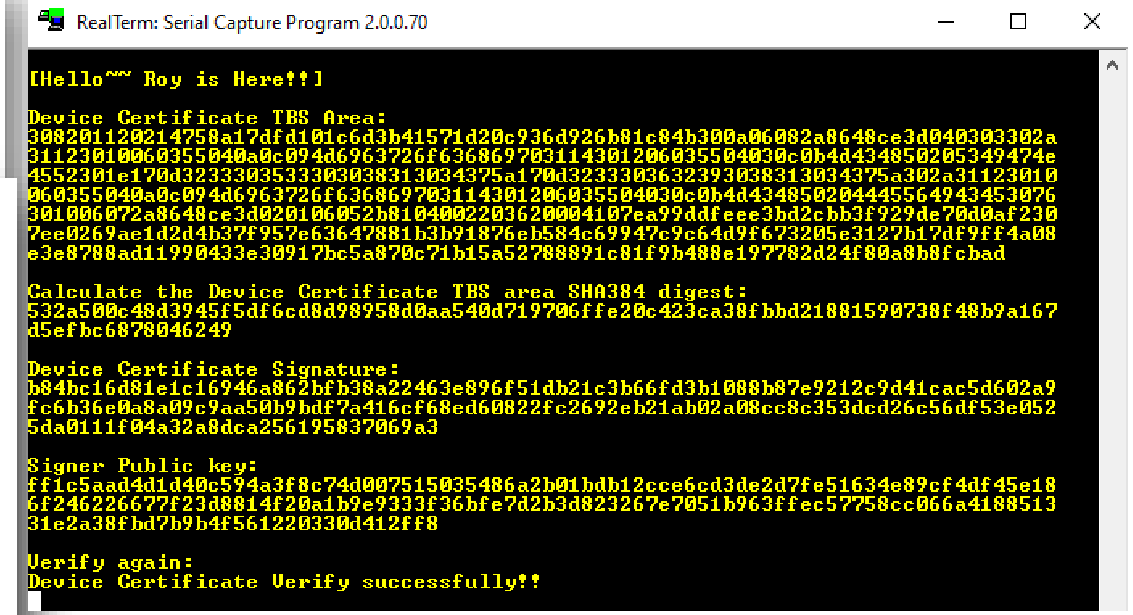
Step 10-1

- Add sha384.c & sha384_asm.s into project (Same as Lab 9)
- Copy functions from Lab10.txt to main.c , Program & Run!



```
void Calculate_Cert_TBS(uint8_t *data, uint16_t length, uint8_t *digest)
{
    uint64_t buffer[80];
    uint16_t i,j = length/128,index = 0;

    SHA512_Initialize(SHA2_384, buffer);
    for(i=0;i<j;i++,index+=128){
        SHA512_DataAdd (&data[index], 128);
    }
    SHA512_DataAdd(&data[index], length-index);
    SHA512_Calculate (digest);
}

void Verify_Device_Certificate(void)
{
    printf("\r\nDevice Certificate TBS Area:\r\n");
    print_bytes(DeviceCert_ToBeSign_Area,278);

    Calculate_Cert_TBS(DeviceCert_ToBeSign_Area,278,DeviceCert_ToBeSign_Hash);
    printf("\r\nCalculate the Device Certificate TBS area SHA384 digest:\r\n");
    print_bytes(DeviceCert_ToBeSign_Hash,48);

    printf("\r\nDevice Certificate Signature:\r\n");
    print_bytes(DeviceCert_sig,96);

    printf("\r\nSigner Public key:\r\n");
    print_bytes(Signer_PubKey,96);

    printf("\r\nVerify again:\r\n");
    status = talib_verify(atcab_get_device(), TA_KEY_TYPE_ECCP384, TA_HANDLE_INPUT_BUFFER, TA_HANDLE_INPUT_BUFFER, DeviceCert_sig,
                    TA_SIGN_P384_SIG_SIZE, DeviceCert_ToBeSign_Hash, TA_VERIFY_P384_MSG_SIZE, Signer_PubKey, TA_ECC384_PUB_KEY_SIZE, &isVerified);

    if(status == ATCA_SUCCESS && isVerified == true){
        printf("Device Certificate Verify successfully!!\r\n");
    }else{
        printf("Device Certificate Verify Failed!!\r\n");
    }
```

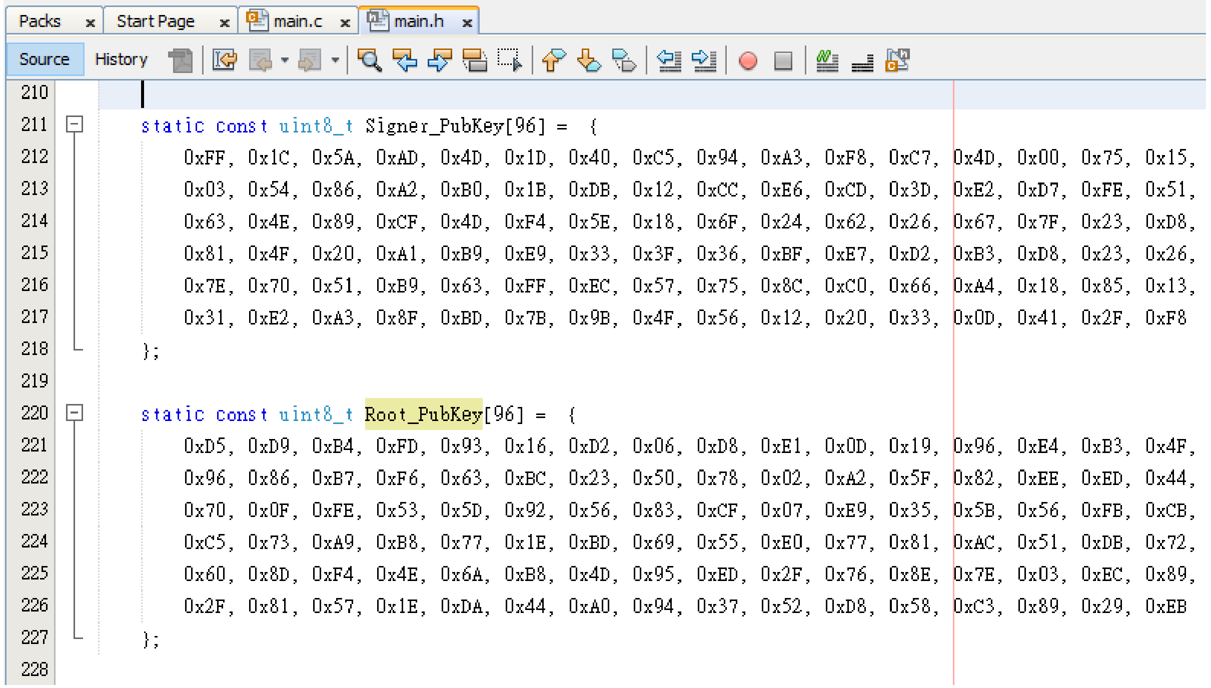
```
int main(void)
{
    SYSTEM_Initialize();

    printf("\r\n[Hello~~ Roy is Here!!]\r\n");

    Verify_Device_Certificate();

    while(1)
    {
    }
}
```



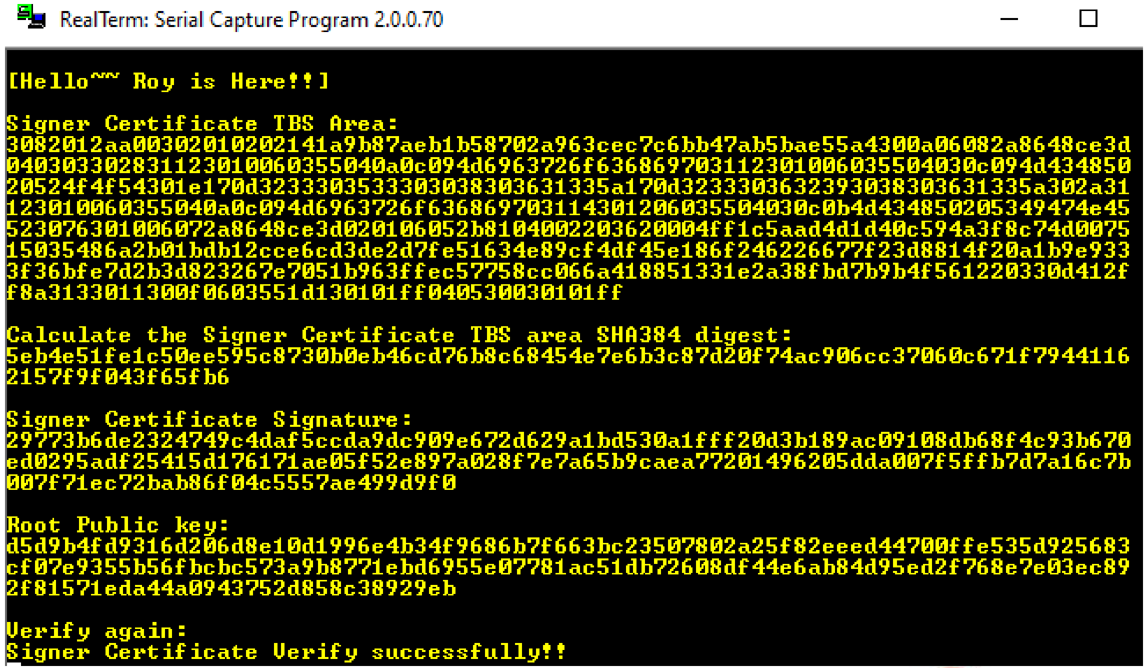
```
[Hello~~ Roy is Here!!]

Device Certificate TBS Area:
308201120214758a17dfd101c6d3b41571d20c936d926b81c84b300a06082a8648ce3d040303302a
3112301006035504 0a0c094d6963726f636869703114301206035504030c0b4d43485020534 9474e
4552301e170d3233303533303038313034375a170d3233303632393038313034375a302a31123010
060355040a0c094d6963726f63686970311430120603550403 0c0b4d434850204445564943453076
301006072a8648ce3d020106052b8104002203620004107ea99ddfeee3bd2cbb3f929de70d0af230
7ee0269ae1d2d4b37f957e63647881b3b91876eb584c69947c9c64d9f673205e3127b17df9ff4a08
e3e8788ad11990433e30917bc5a870c71b15a52788891c81f9b488e197782d24f80a8b8fcbad

Calculate the Device Certificate TBS area SHA384 digest:
532a500c48d3945f5df6cd8d98958d0aa540d719706ffe20c423ca38fbbd21881590738f48b9a167
d5efbc6878046249

Device Certificate Signature:
b84bc16d81e1c16946a862bfb38a22463e896f51db21c3b66fd3b1088b87e9212c9d41cac5d602a9
fc6b36e0a8a09c9aa50b9bdf7a416cf68ed60822fc2692eb21ab02a08cc8c353dcd26c56df53e052
5da0111f04a32a8dca256195837069a3

Signer Public key:
ff1c5aad4d1d40c594a3f8c74d007515035486a2b01bdb12cce6cd3de2d7fe51634e89cf4df45e18
6f246226677f23d8814f20a1b9e9333f36bfe7d2b3d823267e7051b963ffec57758cc066a4188513
31e2a38fbd7b9b4f561220330d412ff8

Verify again:
Device Certificate Verify successfully!!
```

Lab 10 – Practice – Verify Signer Certificate

- Get the Signer Certificate TBS area. (refer to Lab 8 practice)
- Get the Signer Certificate Signature. (refer to Lab 8 practice)
- Root Public key is placed in main.h
- Use Root Pub key to verify the Signer Certificate. (refer to Lab10)



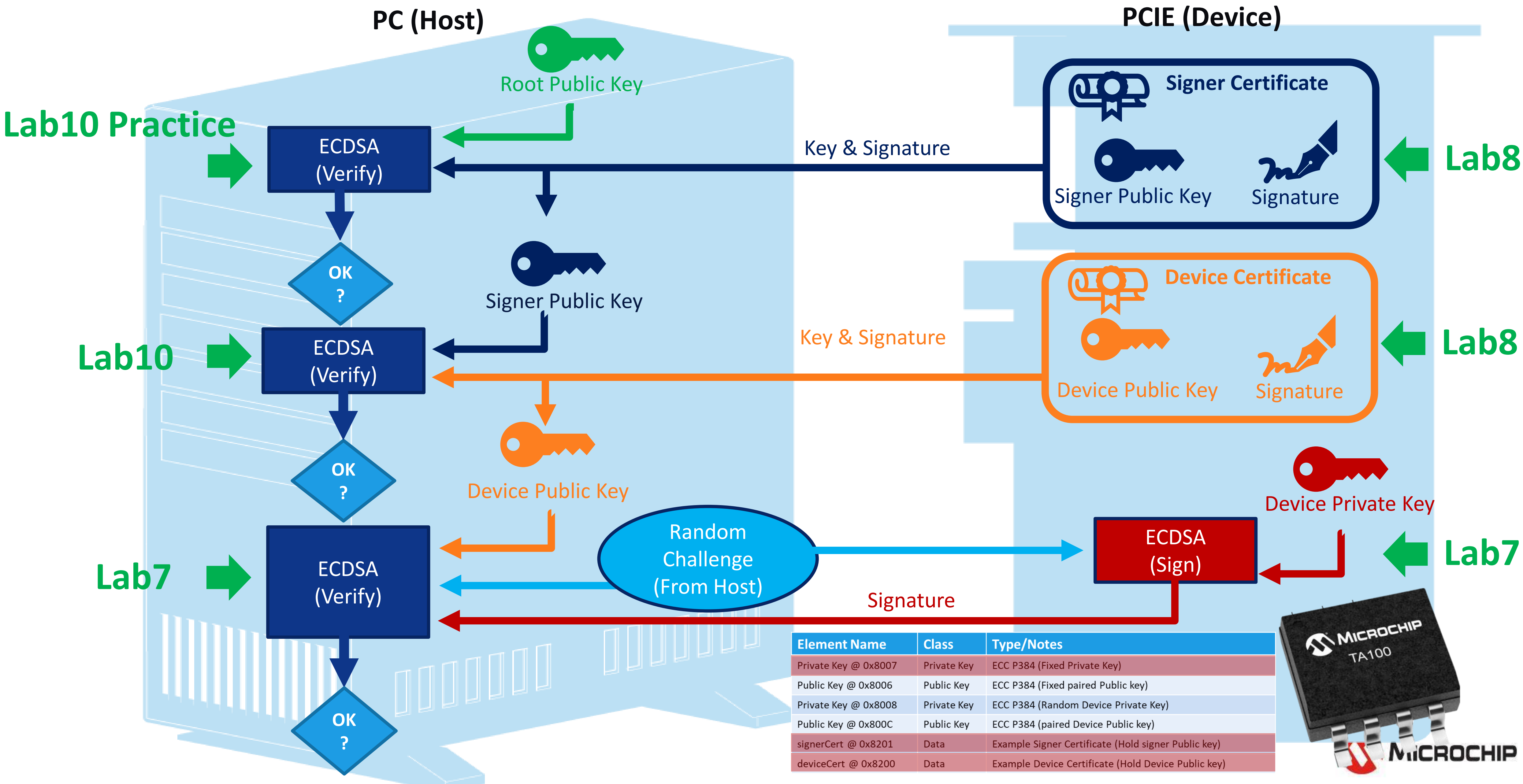
```c

    static const uint8_t Signer_PubKey[96] =  {
        0xFF, 0x1C, 0x5A, 0xAD, 0x4D, 0x1D, 0x40, 0xC5, 0x94, 0xA3, 0xF8, 0xC7, 0x4D, 0x00, 0x75, 0x15,
        0x03, 0x54, 0x86, 0xA2, 0xB0, 0x1B, 0xDB, 0x12, 0xCC, 0xE6, 0xCD, 0x3D, 0xE2, 0xD7, 0xFE, 0x51,
        0x63, 0x4E, 0x89, 0xCF, 0x4D, 0xF4, 0x5E, 0x18, 0x6F, 0x24, 0x62, 0x26, 0x67, 0x7F, 0x23, 0xD8,
        0x81, 0x4F, 0x20, 0xA1, 0xB9, 0xE9, 0x33, 0x3F, 0x36, 0xBF, 0xE7, 0xD2, 0xB3, 0xD8, 0x23, 0x26,
        0x7E, 0x70, 0x51, 0xB9, 0x63, 0xFF, 0xEC, 0x57, 0x75, 0x8C, 0xC0, 0x66, 0xA4, 0x18, 0x85, 0x13,
        0x31, 0xE2, 0xA3, 0x8F, 0xBD, 0x7B, 0x9B, 0x4F, 0x56, 0x12, 0x20, 0x33, 0x0D, 0x41, 0x2F, 0xF8
    };

    static const uint8_t Root_PubKey[96] =  {
        0xD5, 0xD9, 0xB4, 0xFD, 0x93, 0x16, 0xD2, 0x06, 0xD8, 0xE1, 0x0D, 0x19, 0x96, 0xE4, 0xB3, 0x4F,
        0x96, 0x86, 0xB7, 0xF6, 0x63, 0xBC, 0x23, 0x50, 0x78, 0x02, 0xA2, 0x5F, 0x82, 0xEE, 0xED, 0x44,
        0x70, 0x0F, 0xFE, 0x53, 0x5D, 0x92, 0x56, 0x83, 0xCF, 0x07, 0xE9, 0x35, 0x5B, 0x56, 0xFB, 0xCB,
        0xC5, 0x73, 0xA9, 0xB8, 0x77, 0x1E, 0xBD, 0x69, 0x55, 0xE0, 0x77, 0x81, 0xAC, 0x51, 0xDB, 0x72,
        0x60, 0x8D, 0xF4, 0x4E, 0x6A, 0xB8, 0x4D, 0x95, 0xED, 0x2F, 0x76, 0x8E, 0x7E, 0x03, 0xEC, 0x89,
        0x2F, 0x81, 0x57, 0x1E, 0xDA, 0x44, 0xA0, 0x94, 0x37, 0x52, 0xD8, 0x58, 0xC3, 0x89, 0x29, 0xEB
    };

```



```
[Hello~~ Roy is Here!!]

Signer Certificate TBS Area:
3082012aa00302010202141a9b87aeb1b58702a963cec7c6bb47ab5bae55a4300a06082a8648ce3d
04030330283112301006035504a0c094d6963726f636869703112301006035504030c094d434850
20524f4f54301e170d3233303533303038303631335a170d3233303632393038303631335a302a31
12301006035504a0c094d6963726f63686970311430120603550403 0c0b4d43485020534947 4e45
523076301006072a8648ce3d020106052b8104002203620004ff1c5aad4d1d40c594a3f8c74d0075
15035486a2b01bdb12cce6cd3de2d7fe51634e89cf4df45e186f246226677f23d8814f20a1b9e933
3f36bfe7d2b3d823267e7051b963ffec57758cc066a418851331e2a38fbd7b9b4f561220330d412f
f8a3133011300f0603551d130101ff040530030101ff

Calculate the Signer Certificate TBS area SHA384 digest:
5eb4e51fe1c50ee595c8730b0eb46cd76b8c68454e7e6b3c87d20f74ac906cc37060c671f7944116
2157f9f043f65fb6

Signer Certificate Signature:
29773b6de2324749c4daf5ccda9dc909e672d629a1bd530a1fff20d3b189ac09108db68f4c93b670
ed0295adf25415d176171ae05f52e897a028f7e7a65b9caea77201496205dda007f5ffb7d7a16c7b
007f71ec72bab86f04c5557ae499d9f0

Root Public key:
d5d9b4fd9316d206d8e10d1996e4b34f9686b7f663bc23507802a25f82eeed44700ffe535d925683
cf07e9355b56fbcbc573a9b8771ebd6955e07781ac51db72608df44e6ab84d95ed2f768e7e03ec89
2f81571eda44a0943752d858c38929eb

Verify again:
Signer Certificate Verify successfully!!
```

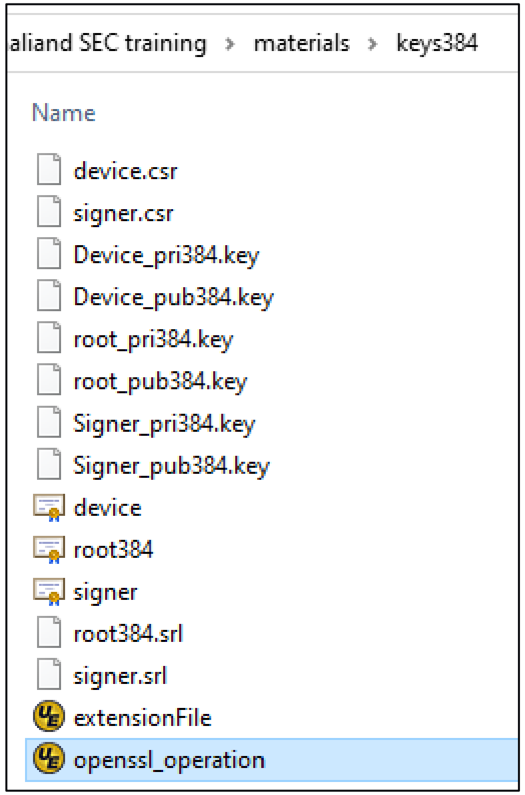
Microchip Proprietary and STRICTLY Confidential

Chain of trust – using Certificates verification



Element Name	Class	Type/Notes
Private Key @ 0x8007	Private Key	ECC P384 (Fixed Private Key)
Public Key @ 0x8006	Public Key	ECC P384 (Fixed paired Public key)
Private Key @ 0x8008	Private Key	ECC P384 (Random Device Private Key)
Public Key @ 0x800C	Public Key	ECC P384 (paired Device Public key)
signerCert @ 0x8201	Data	Example Signer Certificate (Hold signer Public key)
deviceCert @ 0x8200	Data	Example Device Certificate (Hold Device Public key)

Let’s start the Lab10 & practice



aliand SEC training › materials › keys384

Name

- device.csr
- signer.csr
- Device_pri384.key
- Device_pub384.key
- root_pri384.key
- root_pub384.key
- Signer_pri384.key
- Signer_pub384.key
- device
- root384
- signer
- root384.srl
- signer.srl
- extensionFile
- openssl_operation

The End

Questions?

If you wanted to generate your own Root/Signer/Device Certificates & Keys, Please refer to “**openssl_operation.txt**”