S32K1xx CSEc Tool User Manual

Draft version

Rev. 0.2

30 June2020

CONTENT

1. Introductions

S32K1xx serials have Cryptographic Services Engine (CSEc) module implements a comprehensive set of cryptographic functions as described in the SHE Functional Specification. More information please refer to S32K-RM.

CSEc is independent from other modules and unable to initialize before DFlash partition initialization. So, the problem arises that how to find an easy way to configure CSEc module especially in mass-production. Actually, CSEc configuration is public in application code in normal, it results in plaint key left over in public firmware to make hacker easy attack. S32k1xx CSEc tool provides a way to mask code about CSEc configuration in your release version application and promote security consideration. It is convenience to not configure CSEc with PC software tool but create the firmware include your configuration.

Apart from CESc configuration, CSEc initial code excluded in your public firmware is significant part for security consideration and CSEc initialization must be completed before application firmware download. Two ways are purposed:

1) Configure CSEc by NXP: it has been configured by NXP that was entrusted with user before chip travels to customer. but it's only entrusted for some customers.
2) Run CSEc initialization firmware before application firmware during mass-production: you can have a firmware which just for configure CSEc, it can run in RAM or Flash (RAM better), before you download application firmware, download this firmware and run to finish CSEc initialization.

This tool is useful both on development phase and Mass-production phase, for development phase, tool can be used for verification, for Mass-production phase, regenerate firmware, key configuration, et.

Development: The S3211xx CSEc Tool include AES128 encrypt/decrypt CMAC MPCompress functions, that will help you verify the result which compute by MCU. If you need enable secure boot, it’s a good way to use this tool compute boot mac by load *.bin file which you defined area in your project.

Mass-production: During mass-production, you can use this tool to configure CSEc keys and secure boot and create the appropriate firmware.

2. Install and run

Double click S32K1xx CSEc Tool installer.exe to install the software, when the installer is finished, open the software install path, find S32K1xx CSEc Tool.exe and run it, the installer will create shortcut on desktop, you can also create shortcut by yourself if it didn’t. Maybe it will apply for administrator rights, please approve it.

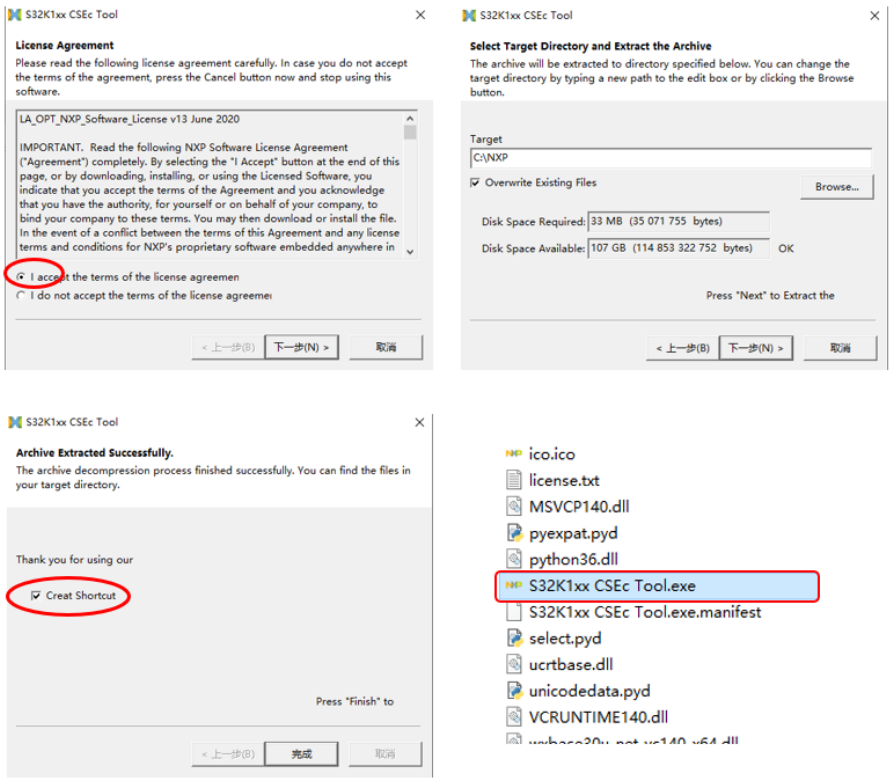

Fig1 S32K1xx CSEc tool -CSEc initialization

3. Create firmware by S32K1xx CSEc tool

S32K1xx CSEc tool(shown on Fig2) include 3 parts include CESc init which is basic information of CPU and application (eg: secure boot), AES tool used for CSEc and Key configuration and Calculate M.

3.1. CSEc initialization

CSEc initialization is about the basic information of CPU and application which are inserted based on customer requirement as the Table1.

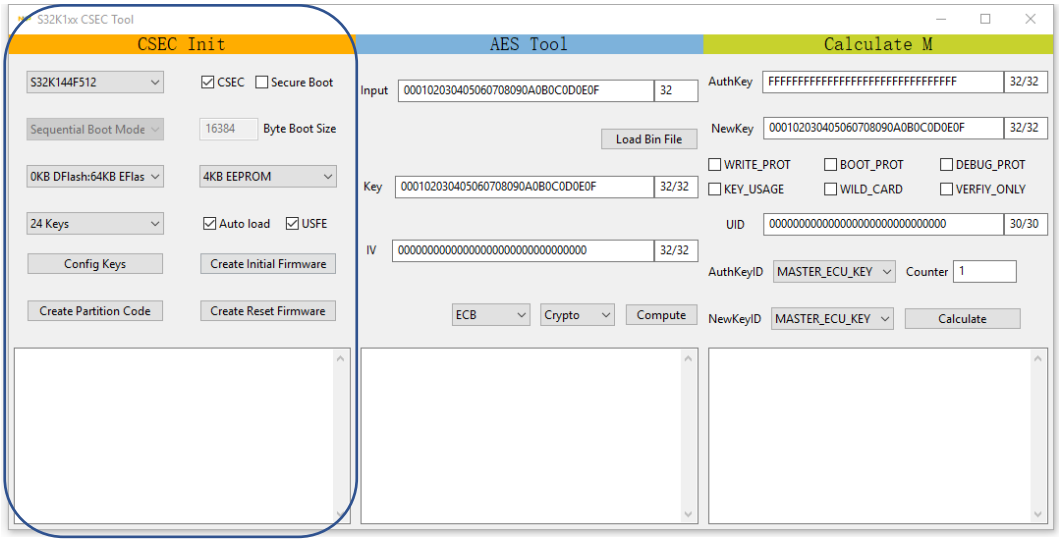

Fig2 S32K1xx CSEc tool -CSEc initialization

Table1 Selection on CSEc initialization

MCU part number	Choose MCU part number.
CSEC	Enable/disable CSEc, if disable, partition code creation is used only.
Secure Boot	Enable/disable secure boot, if enable it, BOOT_MAC and BOOT_MAC_KEY must be imported in key configure panel. BOOT_MAC can be also computed by this tool.
Secure mode	Chose secure mode if secure boot is enabled. **Note: Please set strict sequential boot mode carefully!**
Secure boot size	Define secure boot size.
DE-Flash partition	Set DE-Flash partition.
EEPROM size	Set EEPROM size.
CSEC key numbers	Set CSEC key numbers
Auto load	Set FlexRAM loaded with valid EEPROM data during reset sequence.
USFE	uSFE Security Flag Extension, it should be false if the function is not used for configuring CSEc part.
Config keys	Show key config panel.
Create initial firmware	Click to create initial firmware.
Create partition code	Click to create partition firmware.
Create reset firmware	Click to create reset firmware.

3.2. Demo setup

There is S32K144F512 for example to show how to configure and use tool.
Based information as below should be inserted on CSEc Init
MCU: S32K144F512
Secure boot: Enable. Sequential boot mode
Secure boot size: 8K bytes (4 bytes alignment, max 512K bytes)

Others CSEC configurations as flows show:

Step 1:

Compute boot mac (if secure boot enabled) as Fig3. Load Bin file and secure boot size could be calculated. Size can be compared with the value setting on CSEc initialization, choose boot mode to compute the result.

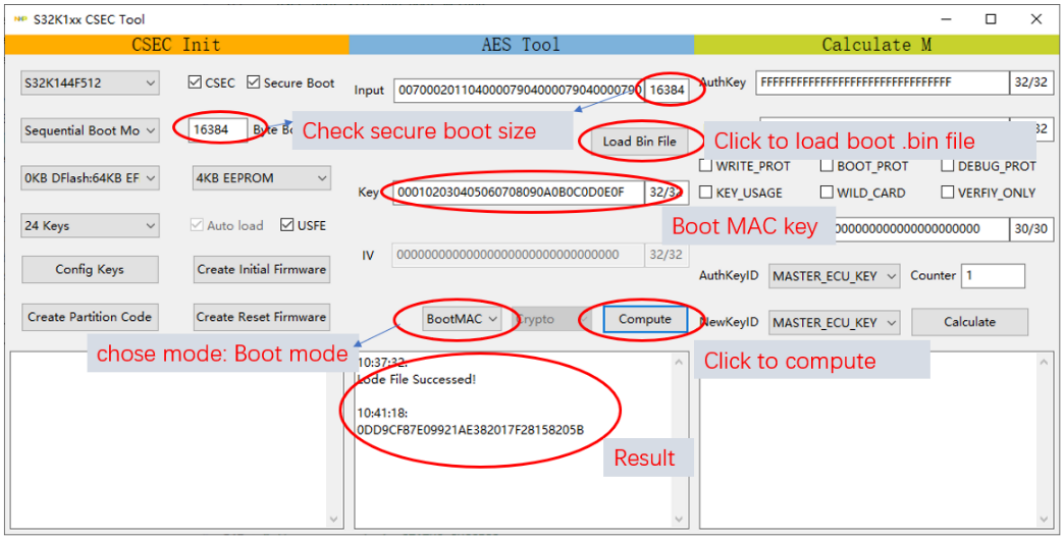

Fig3 boot mac computation

Step 2:

Configure CSEC and keys, click Config keys and open the configuration panel, click import keys and key can be loaded in the panel automatically.

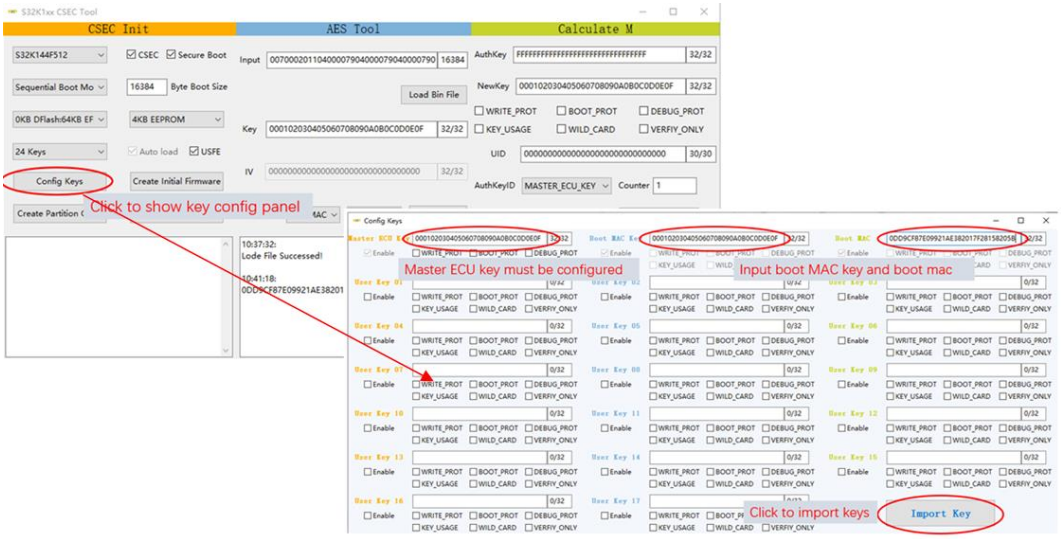

Fig4 Configure CSEC and keys

Note

Boot MAC Key and Boot MAC must be configured before secure boot enable.

Step 3:

Create firmware by click Create Initial Firmware button to generate new firmware and

more details information can be shown as Fig5.

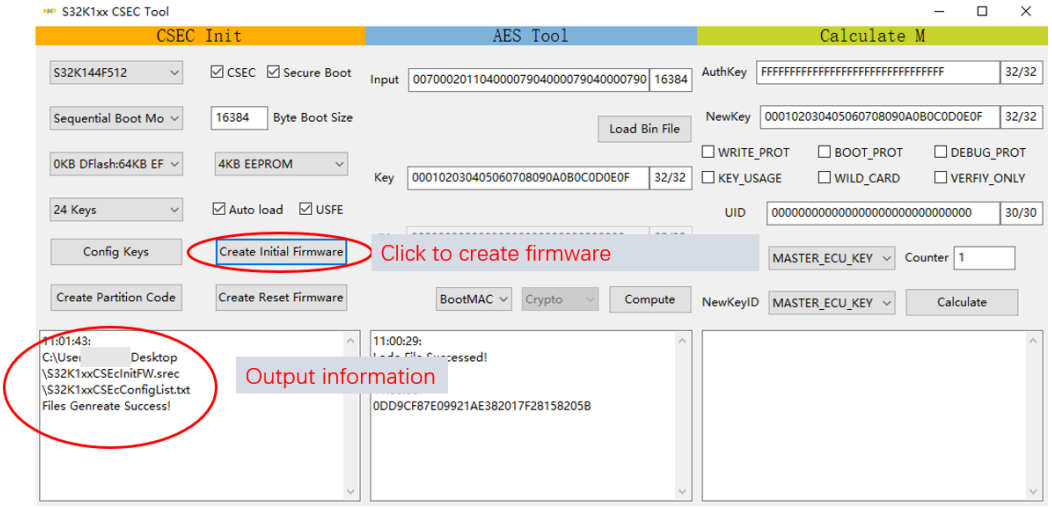

Fig5 Firmware creation

Now you can download the firmware to target MCU and run it as usual. The CSEc will be initiated by your configuration now.

4. Merge CSEc initial and application firmware

CSEc initialization is independent from application firmware. It is hard implement in Mass-production. But CSEc initialization could be merged with application in cyclone official tool by generating SAP file. More details as below.

1. Run CreateImage.exe

CreateImage.exe located in cyclone installation files and open it. Shown as Fig6

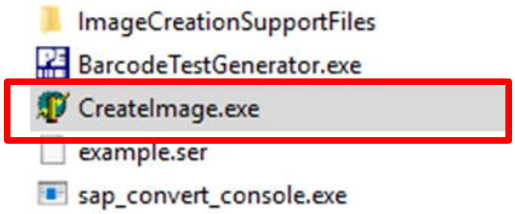

Fig6 Open CreateImage.exe

2. Device chosen

Choose the ARM devices as specify Target Architecture and select the target device as below Fig7(S32K144F512M15 for example).

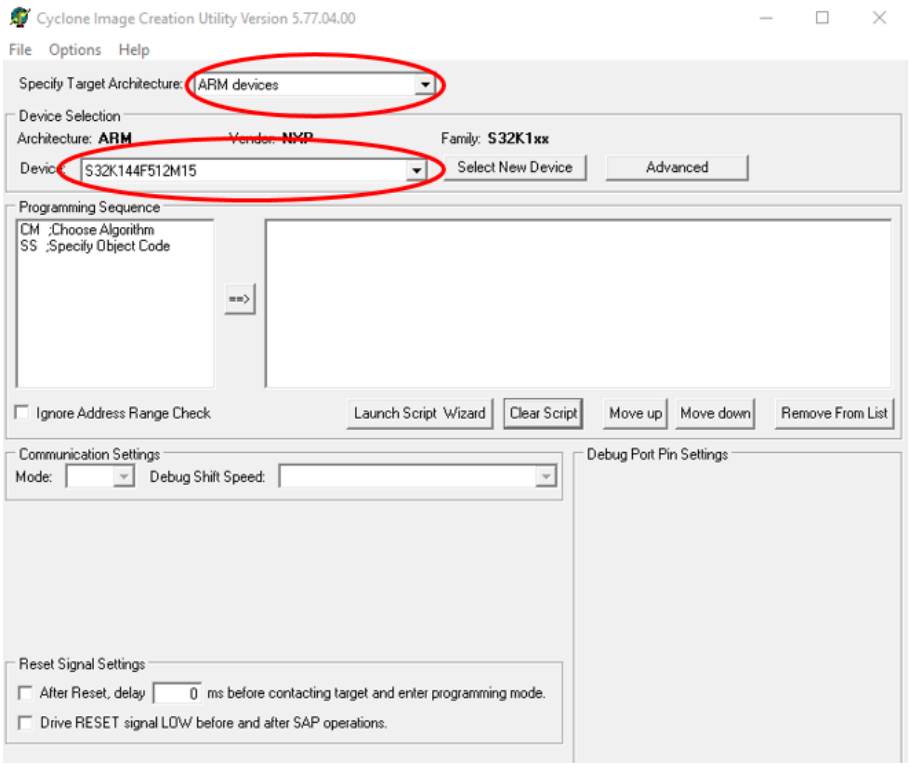

Fig7 Device chosen (S32K144F512M15 for example)

3. Launch script

Click launch script wizard, launch cyclone *.arp script and CSEc initial firmware.

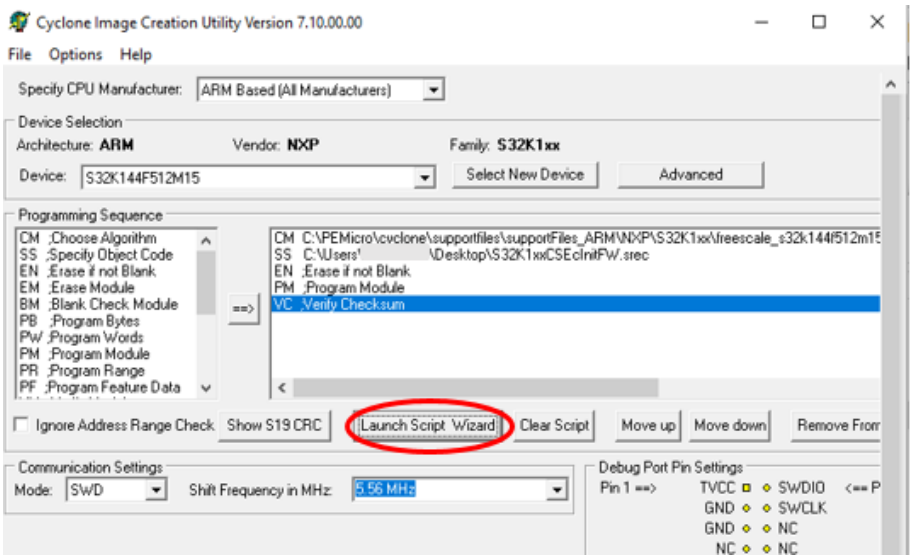

Fig8 Launch script setting

4. Add run command and App firmware.

More commands as Table2 located in programming sequence can be added by the requirement of customer.

Table2 Commands of cyclone

CMD	Description	Function
CM	Choose algorithm	Use for download CSEc initial firmware to target MCU
SS	Add object file	

EN	Erase if target not blank	
PM	Program module	
VC	Verify checksum	
RE	Reset	Run the firmware to initialize target MCU
GO	Run code	
DE	delay	
NOTE		
After this, the CSEc initial firmware has been downloaded to target MCU and run. The CSEc module also has been initialed.		
CM	Choose algorithm	Use for download application firmware to target MCU
SS	Add object file	
EN	Erase if target not blank	
PM	Program module	
VC	Verify checksum	

5. Save the cyclone SAP file.

*.sap file which is generated in consideration with customer's requirement can be saved to run in the next step. This *.sap file could guide cyclone to download CSEc initialization first and then delay for a while to download application code subsequently. The setting of sap file can make sure the one-time download on mass-production and the right download sequence.

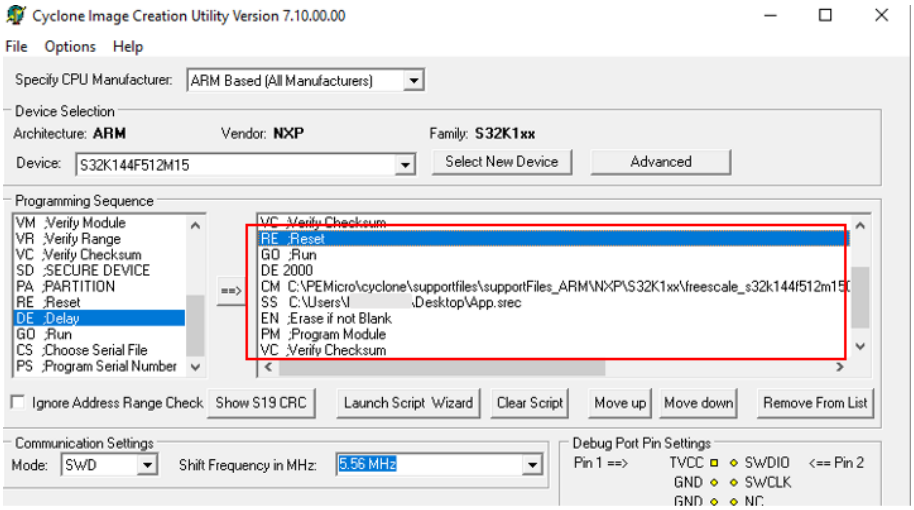

Fig9 Save sap file

6. Run SAP file by cyclone PE.

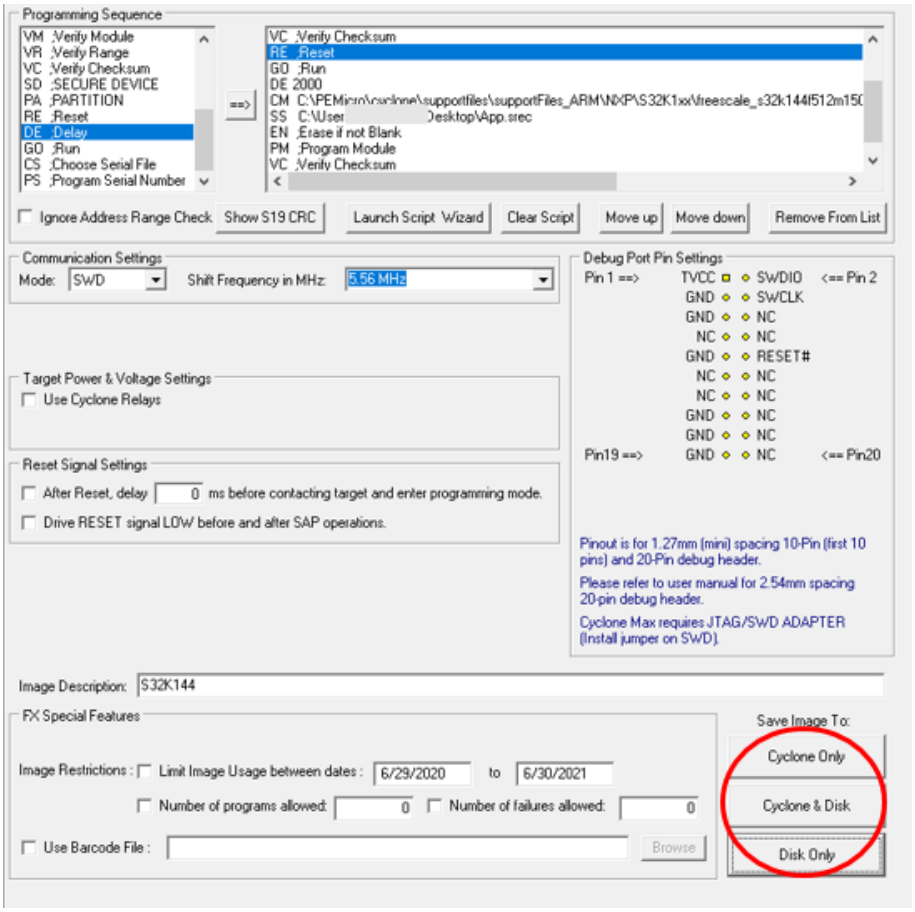

Fig10 Run SAP file by cyclone PE.

5. Source code introductions

For customize the tools by customer. This chapter is to introduce how to redevelop and modify the tool in according with specific project or mass-production.

5.1. Implement model of CESc tool

CESc key configuration area is one part of public firmware which generated by S32DS project to achieves configuration by itself but should be privacy. CESc tool is used to modify the configuration area by user set and generate a new firmware shown as Fig11:

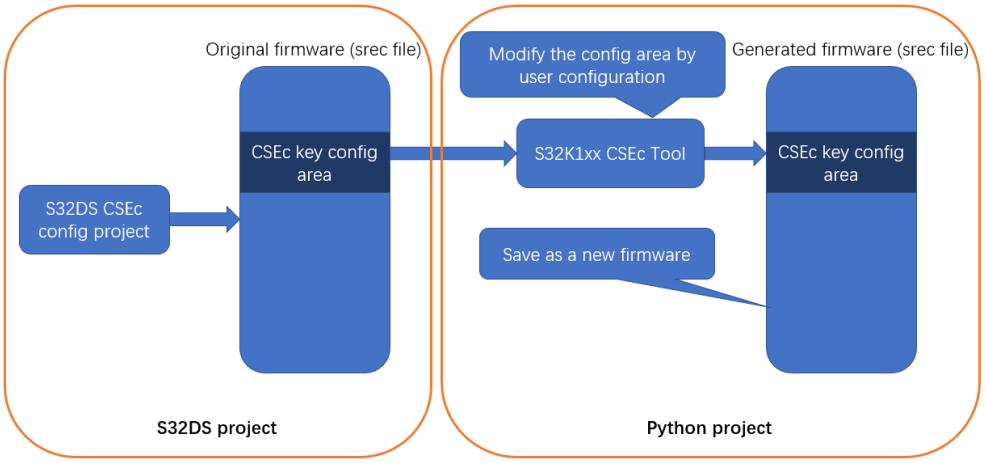

Fig11 The implantation model of CESc tool

Following the step, you can achieve a new security firmware:

1) Create MCU project base on S32DS, used for configuring CSEc.
2) Save all CSEc configurable parameters in Pflash designated area.
3) Build project and output .srec(.s19) file.
4) Modify *.srec file specified area (store CSEc configuration) by PC tool (python project) that can input user configuration in the firmware.
5) Save as new .srec file, this firmware that include you own configuration.

5.2. Development environment requirement

Table3 Development environment requirement

Run time environment	windows 10
python version	3.6
wxpython version	4.1.0
UI design tool	wxFormBuilder version 3.9
S32DS project	S32DS ARM 2018.R1 SDK：RTM3.0.0
Dependency library	wxPython, cryptodemo(AES).

5.3. Files construction

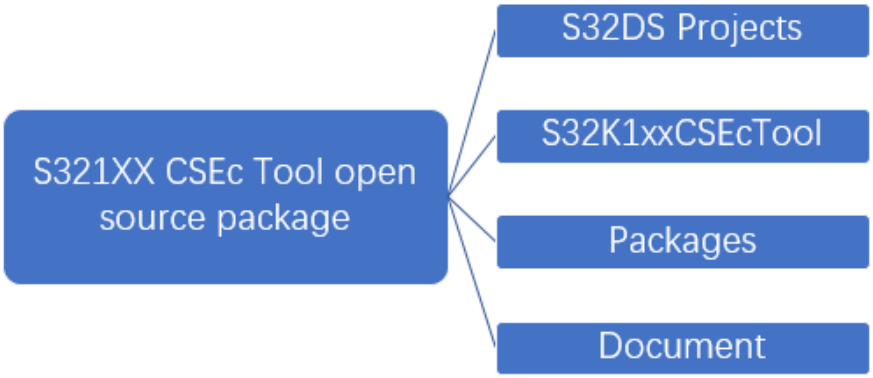

Fig12 Files construction of CESc tool source code

S32DS Projects: this tool can configure CSEc on several platform. Now it can be achieved on S32K116, S32K118, S32K142, S32K144, S32K146 and S32K148.

S32K1xxCSEcTool:

Tool development is depending on python package as below,

python project folder, files tree shows as below:

main.py: the software entrance

app.py: Application functions

csec.py: include AES128 CBC ECB encrypt/decrypt, CMAC and MPCompress.

utility.py: include utility functions such as compute checksum.

uiCbk.py: UI control callback functions included.

ui.py: include all UI component element. it's created by wxFormBuilder.

Firmware folder: original firmware

ico.ico: app ico.

ui.fbp: wxFormBuilder project.

Packages: development packages include individual GUI tool.

Document: development guide to make sure customer quick start on usage and redevelopment of CESc tool.

5.4. S32DS project

1. Dividing the area for CSEc config by modify link file (S32K144F512 project):

```
/* Specify the memory areas */
MEMORY
{
  /* Flash */
  m_interrupts          (RX)  : ORIGIN = 0x00000000, LENGTH = 0x00000400
  m_flash_config        (RX)  : ORIGIN = 0x00000400, LENGTH = 0x00000010
  m_csec_key            (RX)  : ORIGIN = 0x00000410, LENGTH = 0x00000290
  m_text                (RX)  : ORIGIN = 0x000006A0, LENGTH = 0x0007F960
  /* SRAM_L */
  m_data                (RW)  : ORIGIN = 0x1FFF8000, LENGTH = 0x00008000

  /* SRAM_U */
  m_data_2              (RW)  : ORIGIN = 0x20000000, LENGTH = 0x00007000
}

  .csec_keys :
  {
    . = ALIGN(4);
    KEEP(*(.CSEcKeys))    /* CSEc Keys*/
    . = ALIGN(4);
  } > m_csec_key
```

Fig13 Link file setting

2. Store the CSEc config in this area (in main.c):

```
#define CSEC_KEYS_MEM __attribute__((section (".CSEcKeys")))

uint8_t    CSEC_KEYS_MEM PartitionCode[16]={/*default setting*/
                                    144,  //0:MCU partnumber,S32K144
                                    0x02, //1:EEP size,          0x02:4k eeprom
                                    0x04, //2:DEflash partition, 0x04:64K EFlash:0K DFlash
                                    0x03, //3:total key number,  0x03:24Key
                                    0x1,  //4:USFE,              0x1: Enable
                                    0x01, //5:Flex ram auto load eep data 0x1:Enable must be 0x1 when use csec
                                    0x03, //6:secure boot mode   0x03: CSEC_BOOT_NOT_DEFINED
                                    0x00,0x00,0x00,0x00, //7~10:boot size Low byte frist
                                    0x1,  //11:0-Init 1-reset,   0x1: reset CSEc
                                    0x00,0x00,0x00,0x00 //unused
                                    };

CSEcKeys_t CSEC_KEYS_MEM CSEc_Keys_Buffer[20]=
{
    //      ID          Enable   Flags   reserved  Counter  reserved buffer       Key
    {CSEC_MASTER_ECU,   1,       0,      0,        0x1,     RESERVED_8,  BLANK_KEY},
    {CSEC_BOOT_MAC_KEY, 0,       0,      0,        0x1,     RESERVED_8,  BLANK_KEY},
    {CSEC_BOOT_MAC,     0,       0,      0,        0x1,     RESERVED_8,  BLANK_KEY},
    {CSEC_KEY_1,        0,       0,      0,        0x1,     RESERVED_8,  BLANK_KEY},
    {CSEC_KEY_2,        0,       0,      0,        0x1,     RESERVED_8,  BLANK_KEY},
    {CSEC_KEY_3,        0,       0,      0,        0x1,     RESERVED_8,  BLANK_KEY},
    {CSEC_KEY_4,        0,       0,      0,        0x1,     RESERVED_8,  BLANK_KEY},
    {CSEC_KEY_5,        0,       0,      0,        0x1,     RESERVED_8,  BLANK_KEY},
    {CSEC_KEY_6,        0,       0,      0,        0x1,     RESERVED_8,  BLANK_KEY},
    {CSEC_KEY_7,        0,       0,      0,        0x1,     RESERVED_8,  BLANK_KEY},
    {CSEC_KEY_8,        0,       0,      0,        0x1,     RESERVED_8,  BLANK_KEY},
    {CSEC_KEY_9,        0,       0,      0,        0x1,     RESERVED_8,  BLANK_KEY},
    {CSEC_KEY_10,       0,       0,      0,        0x1,     RESERVED_8,  BLANK_KEY},
    {CSEC_KEY_11,       0,       0,      0,        0x1,     RESERVED_8,  BLANK_KEY},
    {CSEC_KEY_12,       0,       0,      0,        0x1,     RESERVED_8,  BLANK_KEY},
    {CSEC_KEY_13,       0,       0,      0,        0x1,     RESERVED_8,  BLANK_KEY},
    {CSEC_KEY_14,       0,       0,      0,        0x1,     RESERVED_8,  BLANK_KEY},
    {CSEC_KEY_15,       0,       0,      0,        0x1,     RESERVED_8,  BLANK_KEY},
    {CSEC_KEY_16,       0,       0,      0,        0x1,     RESERVED_8,  BLANK_KEY},
    {CSEC_KEY_17,       0,       0,      0,        0x1,     RESERVED_8,  BLANK_KEY},
};
```

Fig14. Store the CSEc config

3. The output .srec file as Fig15 which is the origin file for tool input.

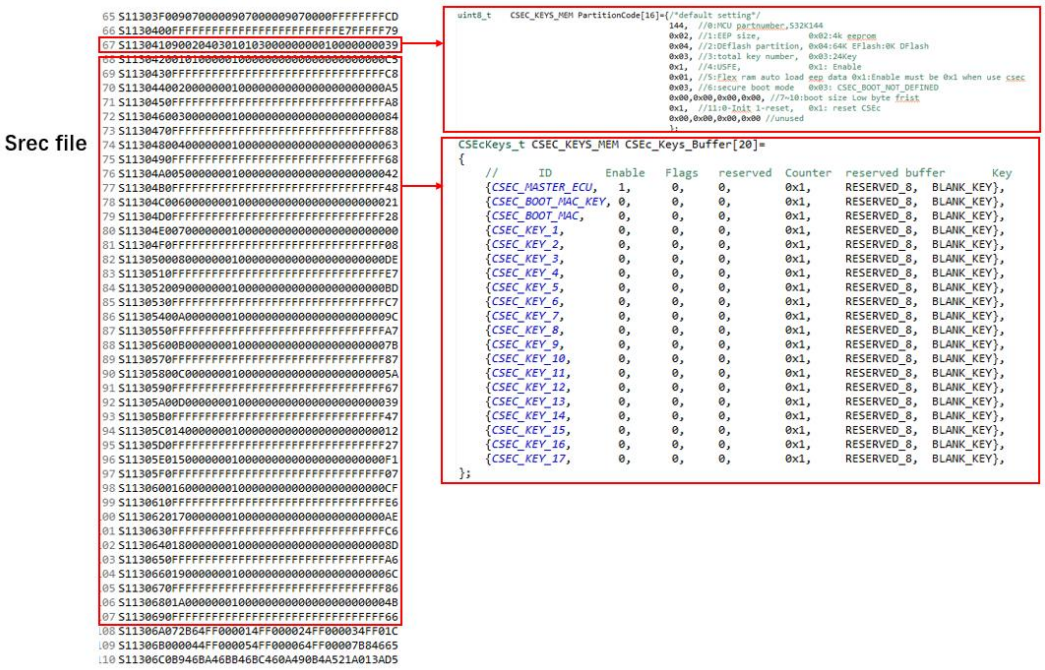

Fig15 .srec file

5.5. Python project module relationships

For understand of tool development. The relationship between python package shown as Fig16

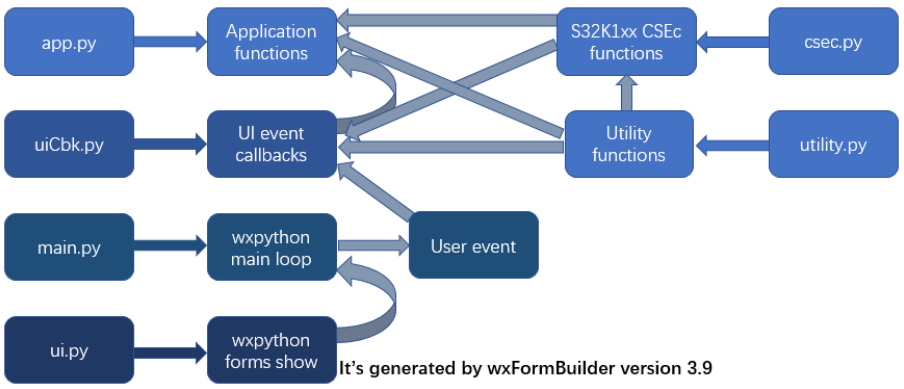

Fig16 The relationship of python package

When the software starts running, instantiating a form app depend on ui.py which generated by wxFormBuilder. Then wxpython main loop will running and detecting user events.

uiCbk.py include two class UI_MainFrame and UI_CfgKeyFrame. Those two class are child class of wx.Frame. when user occurred, event call back will be called to handle user events. These callback functions use functional functions to other modules as needed just as above shows.

Please refer resource code to get more information