



CASignTool

User Guide

Issue	06
Date	2016-04-01

Copyright © HiSilicon Technologies Co., Ltd. 2016. All rights reserved.

No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of HiSilicon Technologies Co., Ltd.

Trademarks and Permissions



HISILICON, and other HiSilicon icons are trademarks of HiSilicon Technologies Co., Ltd.

All other trademarks and trade names mentioned in this document are the property of their respective holders.

Notice

The purchased products, services and features are stipulated by the contract made between HiSilicon and the customer. All or part of the products, services and features described in this document may not be within the purchase scope or the usage scope. Unless otherwise specified in the contract, all statements, information, and recommendations in this document are provided "AS IS" without warranties, guarantees or representations of any kind, either express or implied.

The information in this document is subject to change without notice. Every effort has been made in the preparation of this document to ensure accuracy of the contents, but all statements, information, and recommendations in this document do not constitute a warranty of any kind, express or implied.

HiSilicon Technologies Co., Ltd.

Address: Huawei Industrial Base
Bantian, Longgang
Shenzhen 518129
People's Republic of China

Website: <http://www.hisilicon.com>

Email: support@hisilicon.com



About This Document

Purpose

This document describes how to generate dedicated secure BOOT files for advanced conditional access (CA) chips by using the CASignTool.

Related Versions

The following table lists the product versions related to this document.

Product Name	Version
Hi3716C	V100/V200
Hi3716M	V200
Hi3716M	V300/V400/V310/V420/V410/V320/V330
Hi3110E	V300/V500
Hi3719C	V100
Hi3719M	V100
Hi3796C	V100
Hi3798C	V100/V200
Hi3796M	V100
Hi3798M	V100



NOTE

Hi3110E V300, Hi3716C V100, and Hi3716M V200 share an advanced CA solution, and Hi3716M V300/V400/V310/V410/V420/V320/V330, Hi3110E V500, Hi3716C V200, Hi3719C V100, Hi3719M V100, Hi3796C V100, Hi3796M V100, Hi3798M V100, and Hi3798C V100 share an advanced CA solution. Hi3798C V200 uses another advanced CA solution.








Intended Audience

This document is intended for:

- Technical support personnel
- Software development engineers

Symbol Conventions

The symbols that may be found in this document are defined as follows.

Symbol	Description
 DANGER	Alerts you to a high risk hazard that could, if not avoided, result in serious injury or death.
 WARNING	Alerts you to a medium or low risk hazard that could, if not avoided, result in moderate or minor injury.
 CAUTION	Alerts you to a potentially hazardous situation that could, if not avoided, result in equipment damage, data loss, performance deterioration, or unanticipated results.
 TIP	Provides a tip that may help you solve a problem or save time.
 NOTE	Provides additional information to emphasize or supplement important points in the main text.

Change History

Changes between document issues are cumulative. Therefore, the latest document issue contains all changes made in previous issues.

Issue 06 (2016-04-01)

This issue is the sixth official release, which incorporates the following changes:

Hi3716H V100 and Hi3110E V200/V400 are not supported, and Hi3716M V320/V330, Hi3796M V100, Hi3798M V100, and Hi3798C V200 are supported.

Issue 05 (2015-05-30)

This issue is the fifth official release, which incorporates the following changes:

Hi3716M V420, Hi3716M V410, and Hi3110E V500 are supported.

Issue 04 (2015-01-30)

This issue is the fourth official release, which incorporates the following changes:



The document name is changed, and the entire document is updated.

Issue 03 (2014-12-22)

This issue is the third official release, which incorporates the following changes:

Section 2.2.3 is added and section 2.6 is modified.

Issue 02 (2014-10-21)

This issue is the second official release, which incorporates the following changes:

Section 2.1.3.3 is added.

Modifications are made to support the Hi3716M V310 advanced CA chipset.

Issue 01 (2014-06-05)

This issue is the first official release, which incorporates the following changes:

The advanced CA chips Hi3716M V400, Hi3796C V100, and Hi3798C V100 are supported.

Section 3.6.8 and section 4.5 are modified.

Issue 00B09 (2014-04-18)

This issue is the ninth draft release, which incorporates the following change:

The generation mode of the board parameters is changed.

Issue 00B08 (2014-04-15)

This issue is the eighth draft release, which incorporates the following change:

The Hi3716M V400 advanced CA chip is supported.

Issue 00B07 (2014-04-02)

This issue is the seventh draft release, which incorporates the following changes:

Section 2.1.3.1 and section 2.1.3.2 are added, and section 2.5 is modified.

Issue 00B06 (2013-09-16)

This issue is the sixth draft release, which incorporates the following changes:

The function of generating the RSA key is added.

The functions of encrypting/decrypting images and verifying signed BOOT images are added.

The names of sub-menus of tools are changed.

Hi3716C V200/Hi3719C V100/Hi3719M V100/Hi3716M V400 is supported.

Issue 00B05 (2013-08-19)

This issue is the fifth draft release, which incorporates the following change:

The supported chips are updated.



Issue 00B04 (2013-04-15)

This issue is the fourth draft release, which incorporates the following change:

Chapter 2 Operation Guide

Section 2.4 is updated.

Issue 00B03 (2012-12-26)

This issue is the third draft release, which incorporates the following change:

Chapter 2 Operation Guide

Section 2.4 is updated.

Issue 00B02 (2012-05-14)

This issue is the second draft release, which incorporates the following changes:

Chapter 2 Operation Guide

Section 2.1.2 is updated.

Section 2.1.3 is updated.

Section 2.4 is added.

Issue 00B01 (2012-03-22)

This issue is the first draft release.



Contents

About This Document.....	i
1 Introduction to the CASignTool	9
1.1 Overview	9
1.2 Function Description	9
2 Operation Procedures.....	11
2.1 Generating an Unsigned Secure Boot File	11
2.1.1 Environment Preparation	11
2.1.2 Procedures	12
2.1.3 Precautions	17
2.2 Generating a Signed Secure Boot File.....	25
2.2.1 Environment Preparation	25
2.2.2 Procedures	25
2.3 Generating a Self-Signed Secure Boot File.....	30
2.3.1 Environment Preparation	30
2.3.2 Procedures	32
2.3.3 Precautions	34
2.4 Signing Non-Boot Images	34
2.4.1 Special CA Signature for Non-Boot Images	35
2.4.2 Common CA Signature for Non-Boot Images	39
2.4.3 M2-3rd CA Signature for Non-BOOT Images	45
2.5 Generating the RSA Asymmetric Key	49
2.6 Encrypting/Decrypting an Image	51
2.7 Verifying a Signed Boot Image	54
2.8 Obtaining the CASignTool Version Number	54
2.9 Usage of the CASignTool for Linux	55



Figures

Figure 2-1 Boot configuration excel file	11
Figure 2-2 Choosing Sign BootImage Window.....	12
Figure 2-3 Selecting a chipset type	13
Figure 2-4 Selecting cfg.bin	14
Figure 2-5 Selecting the BOOT file	15
Figure 2-6 Displaying a dialog box indicating that files are created in 347431828	16
Figure 2-7 Generating FinalBoot.bin.....	17
Figure 2-8 Configuring the MSID	18
Figure 2-9 MSID of the Hi3110E (print information).....	19
Figure 2-10 MSID of the Hi3110E (in FinalBoot.bin)	20
Figure 2-11 Setting the version ID	20
Figure 2-12 Setting the mask on the Sign BootImage tab page.....	22
Figure 2-13 Setting the auxiliary code on the Sign BootImage tab page	23
Figure 2-14 Setting the key for encrypting the BOOT for Hi3716M V200/Hi3716C V100/Hi3110E V300	24
Figure 2-15 Setting the key for encrypting the BOOT for the BOOT image signature.....	25
Figure 2-16 Selecting files	26
Figure 2-17 Selecting files (for Hi3798C V200 series chips).....	26
Figure 2-18 Merging the files into a final file	27
Figure 2-19 Merging the files into a final file (for the Hi3798C V200 series chips).....	28
Figure 2-20 Specifying the name for the signed BOOT file.....	29
Figure 2-21 Generating the signed BOOT file	30
Figure 2-22 Format of the public key required by the CASignTool.....	31
Figure 2-23 Format of the private key required by the CASignTool.....	31
Figure 2-24 Generating a self-signed BOOT file	32
Figure 2-25 Generating the folder	33
Figure 2-26 Generating FinalBoot.bin.....	34



Figure 2-27 Sign Non-BootImage Window.....	35
Figure 2-28 Special CA signature for non-BOOT images.....	36
Figure 2-29 Information indicating that a new folder has been created	37
Figure 2-30 Generating FinalImage.bin	38
Figure 2-31 Structure of the signed image in special CA signature mode.....	39
Figure 2-32 Common CA signature for non-BOOT images.....	40
Figure 2-33 Generating a new folder.....	42
Figure 2-34 Generating FinalImage.bin	43
Figure 2-35 Structure of the signed image in common CA signature mode (the image is not divided into sections)	43
Figure 2-36 Structure of the signed image in common CA signature mode (the image is divided into sections).....	44
Figure 2-37 M2-3rd CA signature for non-BOOT images	45
Figure 2-38 Generating a new folder.....	46
Figure 2-39 Generating FinalImage.bin	47
Figure 2-40 Structure of the signed image in M2-3rd CA signature mode.....	48
Figure 2-41 Selecting Create RSA Key Window	49
Figure 2-42 Generating the RSA key	50
Figure 2-43 Information indicating that the RSA key has been created successfully.....	50
Figure 2-44 Generated RSA keys	51
Figure 2-45 Selecting Crypto Tool Window.....	52
Figure 2-46 Encrypting/Decrypting images	53
Figure 2-47 Information indicating that the operation is successful.....	54
Figure 2-48 Obtaining the CASignTool version number.....	55
Figure 2-49 Usage of the CASignTool for Linux	55



Tables

Table 1-1 Application scopes of three BOOT files	10
Table 2-1 MSID byte order for advanced CA chips.....	18
Table 2-2 Setting the version ID	21



1 Introduction to the CASignTool

1.1 Overview

Advanced CA chips provide the secure BOOT function. That is, the BOOT signature is automatically verified when a chip starts. Advanced CA chips require dedicated secure BOOT files but not the common BOOT files that are generated in the SDK for common chips. The CASignTool is used to package the common BOOT files generated in the SDK into the secure BOOT files. It can also merge the kernel and file system images into one image for the CA vendor to sign.

The signature verification algorithm RSA2048 bits PKCS #1 v1.5 with SHA256 is used for the secure BOOT of the advanced CA chips.



NOTE

The CASignTool has the Windows and Linux versions.

1.2 Function Description

The secure BOOT function is implemented as follows:

Step 1 Develop and debug the BOOT file.

The signature key of the corresponding CA vendor is written to the advanced CA chips before shipment, but the secure BOOT function is not enabled. The chip does not verify the BOOT signature and can be used to develop and debug the secure BOOT file.

Step 2 Apply for a signature.

After developing a BOOT file, R&D engineers apply to the CA vendor for a signature and enable the secure BOOT function of the chip. During production of a product (for example, an STB), a signed BOOT file must be burnt to the STB, and the secure BOOT function must be enabled.

----End

HiSilicon chips are classified into three types based on the requirements on BOOT files:

- Common chips. These chips support the common BOOT files generated in the SDKs.



- Advanced CA chips with disabled secure BOOT function. These chips use the unsigned secure BOOT files.
- Advanced CA chips with enabled secure BOOT function. These chips use the signed secure BOOT files.

Table 1-1 describes the application scopes of three different BOOT files.

Table 1-1 Application scopes of three BOOT files

Boot Type	Common Chip	Advanced CA Chip with Disabled Secure Boot Function	Advanced CA Chip with Enabled Secure Boot Function
Common BOOT file	√	×	×
Unsigned secure BOOT file	×	√	×
Signed secure BOOT file	×	√	√

The common Boot file and unsigned secure Boot file for the Hi3798C V200 series chips have been combined, that is, they apply to both common chips and advanced CA chips with disabled secure boot function.

The CASignTool supports the following functions:

- Generates unsigned secure BOOT files.
- Generates signed secure BOOT files.
- Generates self-signed secure BOOT files.
- Generates the BOOT image (**FinalBoot.bin**) required for booting advanced CA chips.
- Signs images of applications, kernel, and file systems.
- Generates the [Rivest-Shamir-Adleman](#) (RSA) key required for the STB vendor tests.
- Verifies the signed BOOT file (only for simulated verification).
- Provides the Advanced Encryption Standard (AES) and Triple Data Encryption Standard (3DES) encryption and decryption tools.



2 Operation Procedures

2.1 Generating an Unsigned Secure Boot File

An unsigned secure BOOT file applies to the advanced CA chips with the disabled secure BOOT function. Typically, the unsigned secure BOOT file is used for developing the secure BOOT by R&D engineers.

**NOTE**

The secure BOOT file and common BOOT file for the Hi3798C V200 series chips are the same, and the common BOOT file can be used as the unsigned secure BOOT file by default.

2.1.1 Environment Preparation

Before generating a secure BOOT file, make the preparations as follows:

- Step 1** Set the board parameters by using the BOOT configuration excel file provided in the HiSilicon SDK and generates a *.cfg file. Take Hi3716C V200 as an example. See [Figure 2-1](#).

**NOTE**

For the Hi3798C V200 series chips, this step is skipped.

Figure 2-1 Boot configuration excel file

Boot V1.2		
Hisilicon Technologies Co., Ltd. All rights reserved. ©2012		
Description:		
1. The button [Generate reg bin file(NAND)] will generate config file of spreadsheet which item is noCA and NAND;		
2. The button [Generate reg bin file(eMMC)] will generate config file of spreadsheet which item is noCA and eMMC;		
3. The button [Generate CA config file(NAND)] will generate config file of spreadsheet which item is CA and NAND;		
4. The button [Generate CA config file(eMMC)] will generate config file of spreadsheet which item is CA and eMMC;		
version	vl.1.0	
version	vl.2.0	offset>=0x10000
version	vl.3.0	Add buttons
Import other files		
Generate reg bin file(NAND)		
Generate CA config file(NAND)		
Generate reg bin file(eMMC)		
Generate CA config file(eMMC)		

After configuring the parameters, click **Generate CA config file(NAND)** or **Generate CA config file(eMMC)** based on the type of flash memory on the board to generate the *.cfg file.



For example, if the NAND flash is used and the excel file is **hi3716cdmo2b_hi3716cv200_ddr3_2gbyte_8bitx4_4layers.xlsm**, the generated parameter configuration file is **hi3716cdmo2b_hi3716cv200_ddr3_2gbyte_8bitx4_4layers_nand.cfg**. The generated *.cfg file for Hi3716C V100/Hi3716H V100/Hi3716M V200 cannot be greater than 0x3d0, and that for Hi3716M V300 and later chips cannot be greater than 0x13d0. Otherwise, contact HiSilicon to reduce the size of the *.cfg file. **Generate reg bin file(NAND)** and **Generate reg bin file(eMMC)** are used to generate the configuration parameters of a common BOOT file. Do not press the buttons by mistake.

The file generation time and the comment out information about the original file are added to the end of the *.cfg file for Hi3716C V200 and later versions. You can view information about the *.cfg file by using tools such as the UltraEdit.

Step 2 Generate a common BOOT file by compiling the SDK.

----End



NOTE

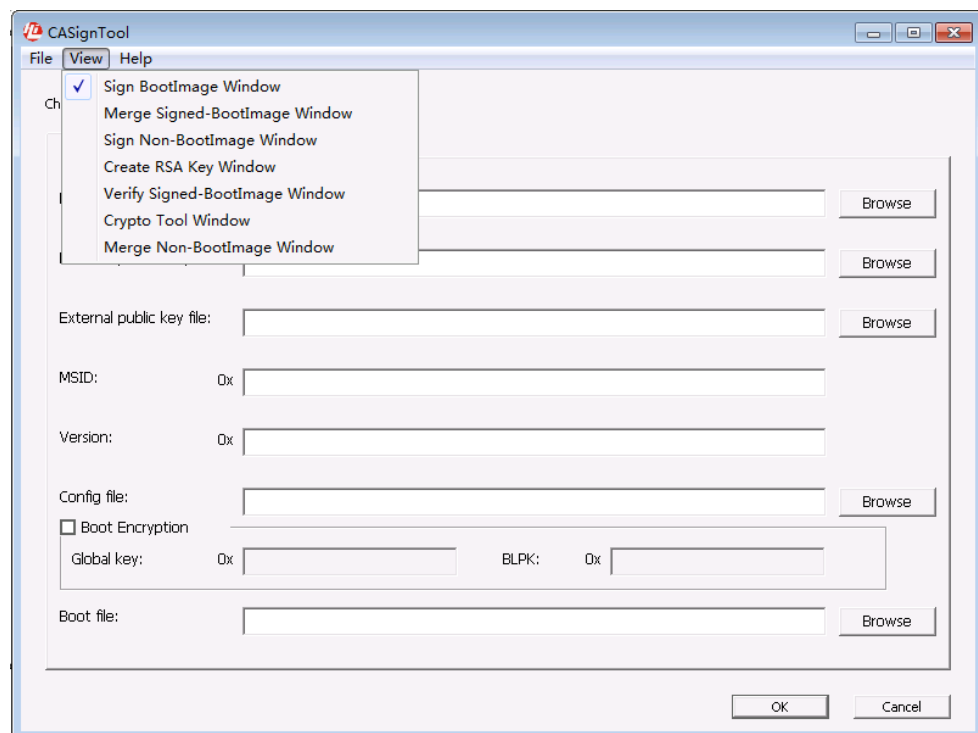
cfg.bin in the following sections refers to the *.cfg file generated by the BOOT configuration file.

2.1.2 Procedures

Take Hi3716M V410 as an example. To generate an unsigned secure BOOT file, perform the following steps:

Step 1 Start the CASignTool, click **View**, and select **Sign BootImage Window**, as shown in [Figure 2-2](#).

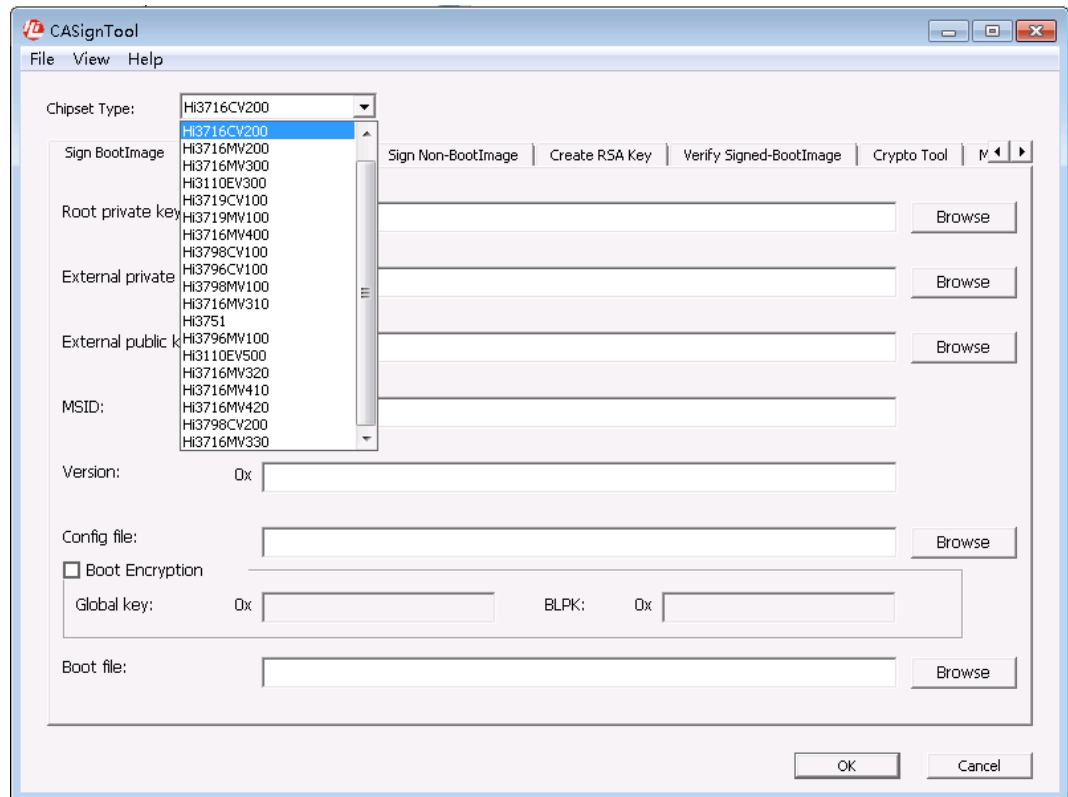
Figure 2-2 Choosing Sign BootImage Window





Step 2 Select a chip from the **Chipset Type** drop-down list.

Figure 2-3 Selecting a chipset type

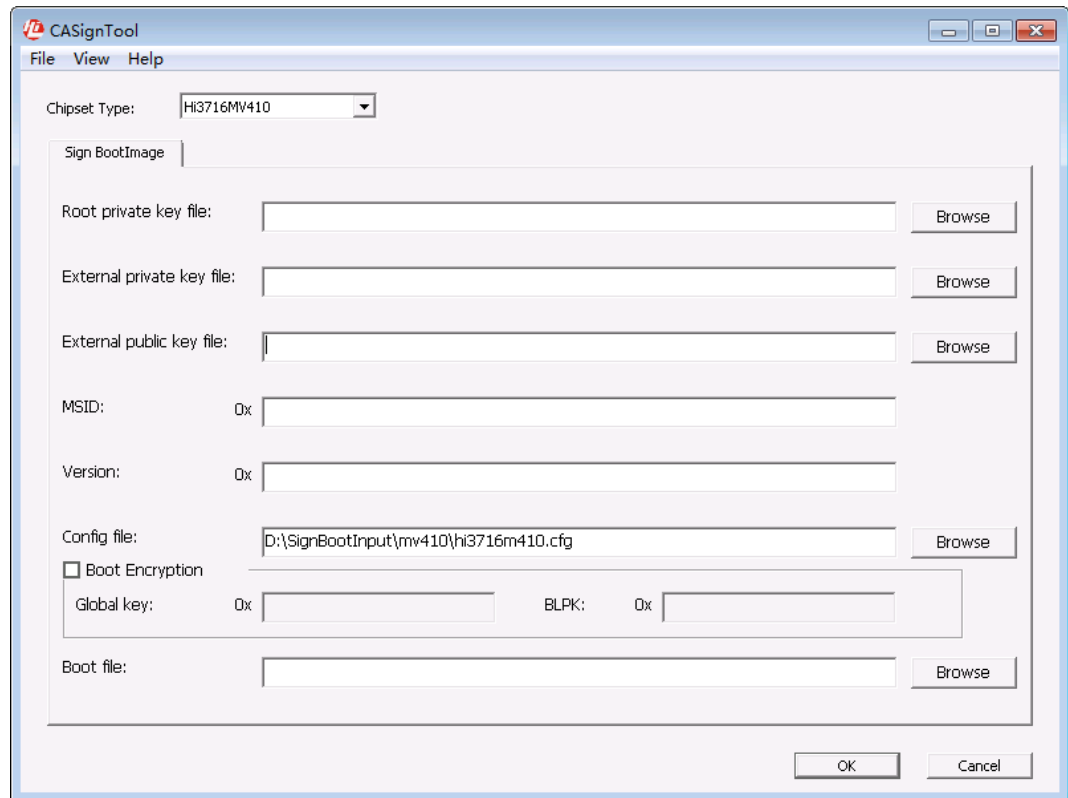


Select the correct chipset type.

Step 3 Click **Browse** after **Config file** and select the **cfg.bin** file generated in [Step 1](#) of section 2.1.1 "Environment Preparation," as shown in [Figure 2-4](#).



Figure 2-4 Selecting cfg.bin



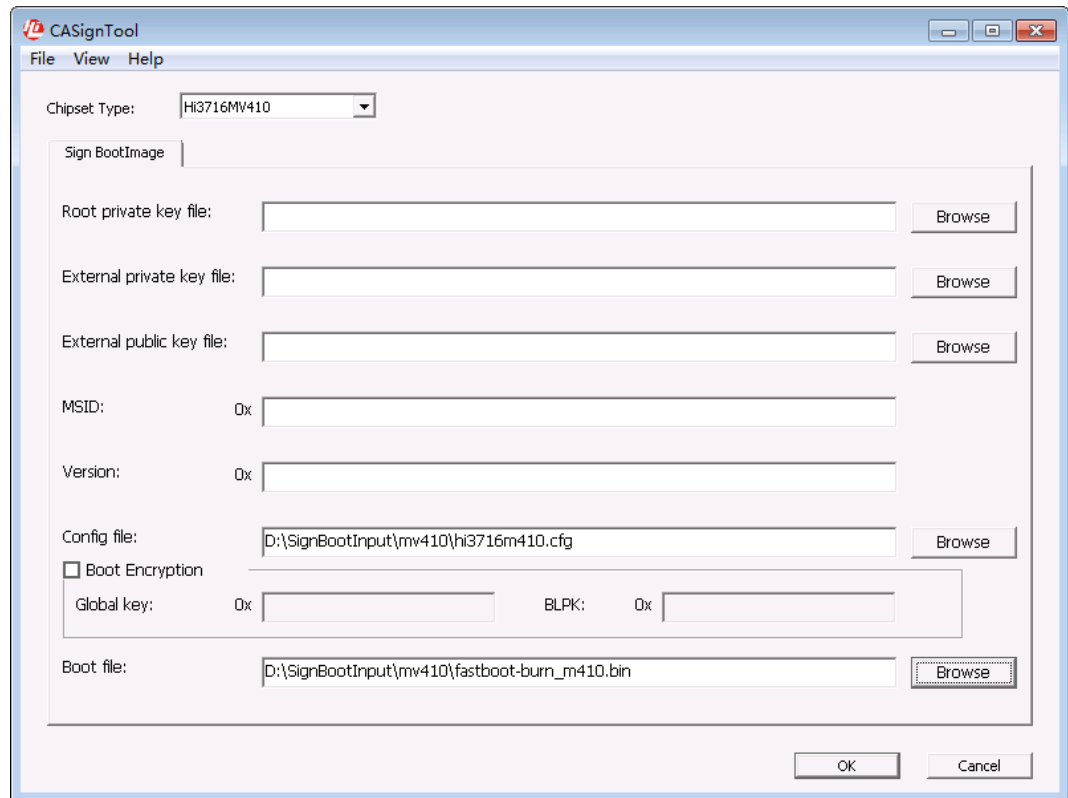
Step 4 Click **Browse** after **Boot file** to select the common BOOT file prepared in [Step 2](#) of section [2.1.1 "Environment Preparation."](#)

This BOOT file is generated after the advanced CA option is enabled in the SDK.

- The BOOT file generated in the HD SDK is **fastBOOT_burn.bin**.
- The BOOT file generated in the SD SDK is **miniBOOT2008.bin**.



Figure 2-5 Selecting the BOOT file

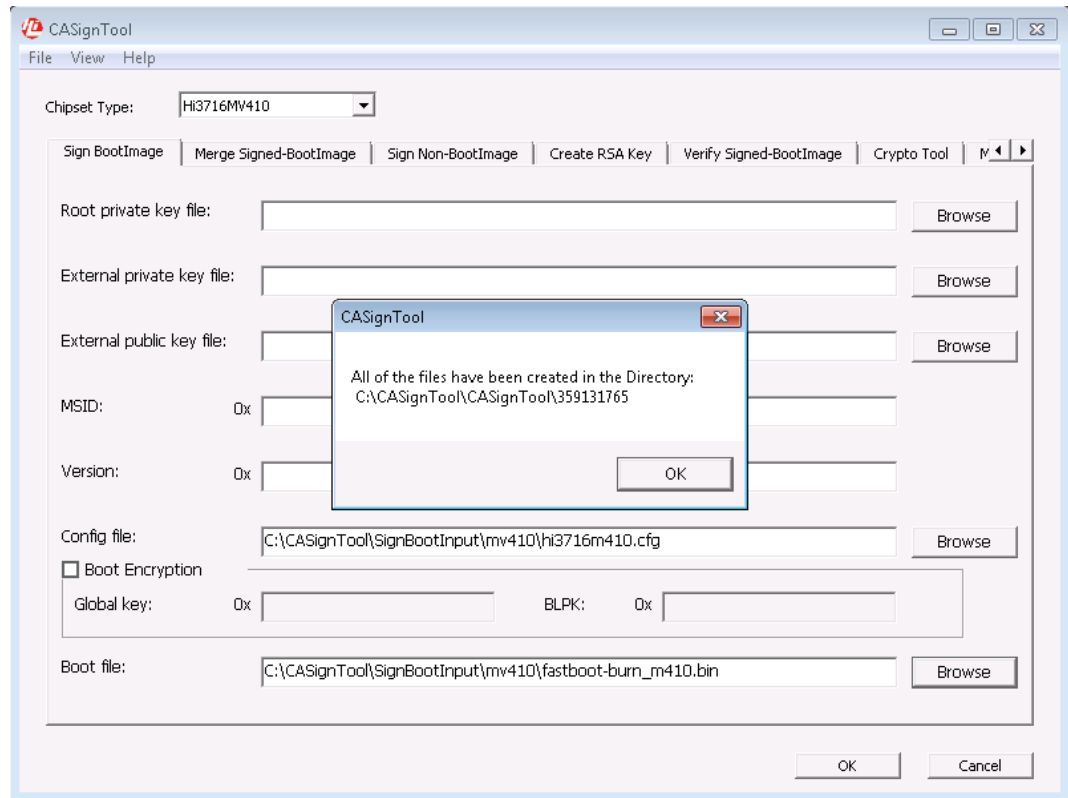


Step 5 Click **OK**.

The CASignTool creates a folder in the folder where the CASignTool locates for storing the generated file. The folder name as shown in [Figure 2-6](#) is random and may be different from the actual file.



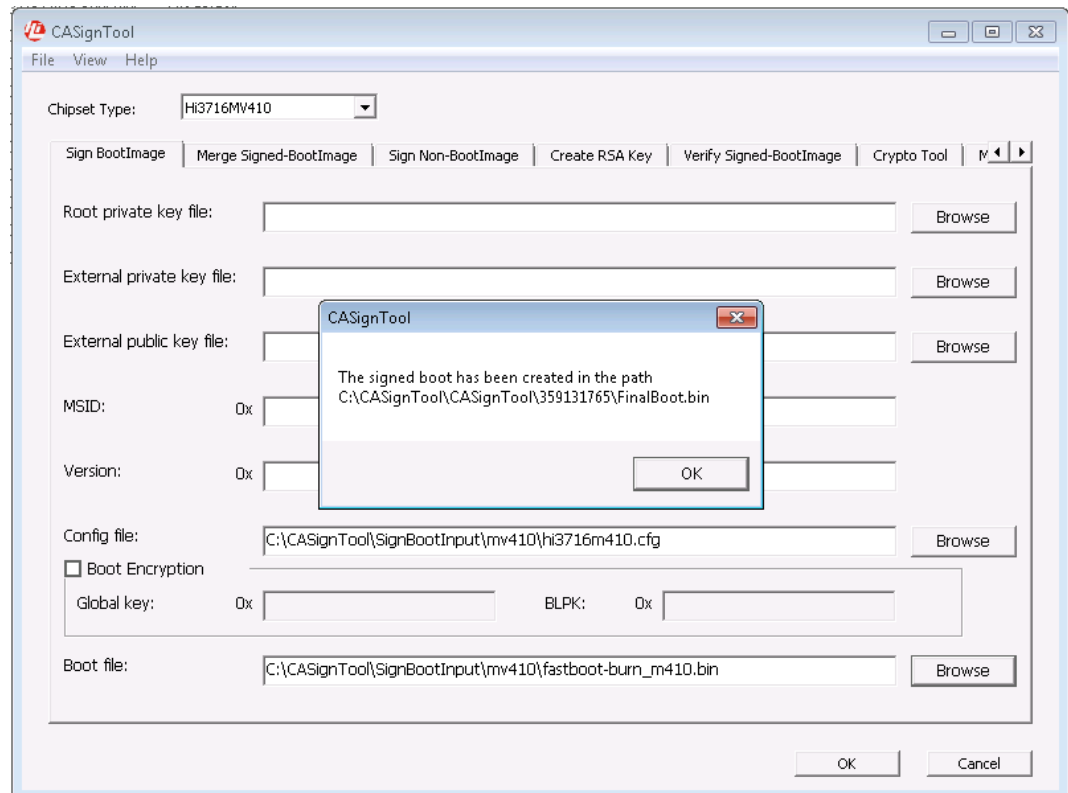
Figure 2-6 Displaying a dialog box indicating that files are created in 347431828



A dialog box is displayed, indicating that the unsigned BOOT file **FinalBoot.bin** has been created.



Figure 2-7 Generating FinalBoot.bin



----End

2.1.3 Precautions

2.1.3.1 Setting the MSID

If the market segment ID (MSID) needs to be configured, set **MSID** on the **Sign BootImage** tab page during the configuration described in section 2.1.2 "Procedures." The other operations are the same as those described in section 2.1.2 "Procedures."



Figure 2-8 Configuring the MSID

The following describes the rules of configuring the MSID by using the 0x67452301 MSID as an example if the CA vendor requires that the MSID be set.

2.1.3.2 MSID Byte Order for Advanced CA Chips

Table 2-1 MSID byte order for advanced CA chips

Item	Hi3716C V200/Hi3719C V100/Hi3719M V100/Hi3716M V400/Hi3798M V100/Hi3716M V420/Hi3716M V410/Hi3110E V500	Hi3716M V300/Hi3716M V310	Hi3716C V100/Hi3716M V200	Hi3110E
HI_UNF_AD VCA_SetMarketId (HI_U8 u8MarketId[4)	u8MarketId[0] = 0x67 u8MarketId[1] = 0x45 u8MarketId[2] = 0x23 u8MarketId[3] = 0x01	u8MarketId[0] = 0x67 u8MarketId[1] = 0x45 u8MarketId[2] = 0x23 u8MarketId[3] = 0x01	u8MarketId[0] = 0x01 u8MarketId[1] = 0x23 u8MarketId[2] = 0x45 u8MarketId[3] = 0x67	u8MarketId[0] = 0x01 u8MarketId[1] = 0x23 u8MarketId[2] = 0x45 u8MarketId[3] = 0x67



Item	Hi3716C V200/Hi3719C V100/Hi3719M V100/Hi3716M V400/Hi3798M V100/Hi3716M V420/Hi3716M V410/Hi3110E V500	Hi3716M V300/Hi3716M V310	Hi3716C V100/Hi3716M V200	Hi3110E
CASignTool	0x67452301	0x67452301	0x67452301	0x67452301
System Register	0xf8ab0120: 0x01 0xf8ab0121: 0x23 0xf8ab0122: 0x45 0xf8ab0123: 0x67	0x10180120: 0x01 0x10180121: 0x23 0x10180122: 0x45 0x10180123: 0x67	0x10000008: 0x01 0x10000009: 0x23 0x1000000a: 0x45 0x1000000b: 0x67	0x101c0008: 0x01 0x101c0009: 0x23 0x101c000a: 0x45 0x101c000b: 0x67
OTP (Internal use)	0xa0: 0x01 0xa1: 0x23 0xa2: 0x45 0xa3: 0x67	0xa0: 0x01 0xa1: 0x23 0xa2: 0x45 0xa3: 0x67	0x2c: 0x01 0x2d: 0x23 0x2e: 0x45 0x2f: 0x67	0x2c: 0x01 0x2d: 0x23 0x2e: 0x45 0x2f: 0x67

Some CA chips have special requirements on the MSID. For details, see the development guides provided by HiSilicon for the related CA vendors.

[Figure 2-9](#) and [Figure 2-10](#) describe the MSID in a more intuitive way by using the Hi3110E as an example.

Figure 2-9 MSID of the Hi3110E (print information)

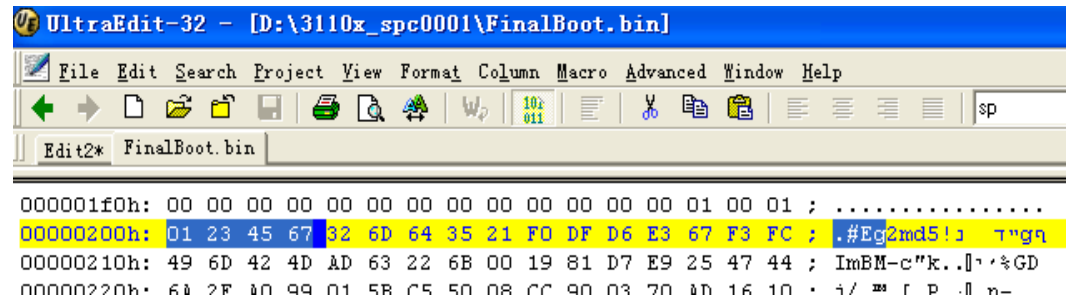
```
MarketID Example: (chipset 3110E X5V400)
CASignTool.exe:
MSID Input: 0x67452301,
UltraEdit-32.exe to open FinalBoot.bin: Offset: 0x200: 01 23 45 67
OTP: 67452301, BYTE:
    0X2C-->0x01, 0X2D-->0x23, 0X2E-->0x45, 0X2F-->0x67

hisilicon # md 60c00200
60c00200: 67452301 35646d32 d6dff021 fcf367e3

# himd.1 0x101c0000
*** Board tools : ver0.0.1_20120501 ***
[debug]: {source/utis/cmdshell.c:166}cmdstr:himd.1
====dump memory 0X101C0000====
0000: 00000432 0000000f 67452301 2a0a0a0a
0010: 67452301 00000000 00000000 00000000
```



Figure 2-10 MSID of the Hi3110E (in FinalBoot.bin)

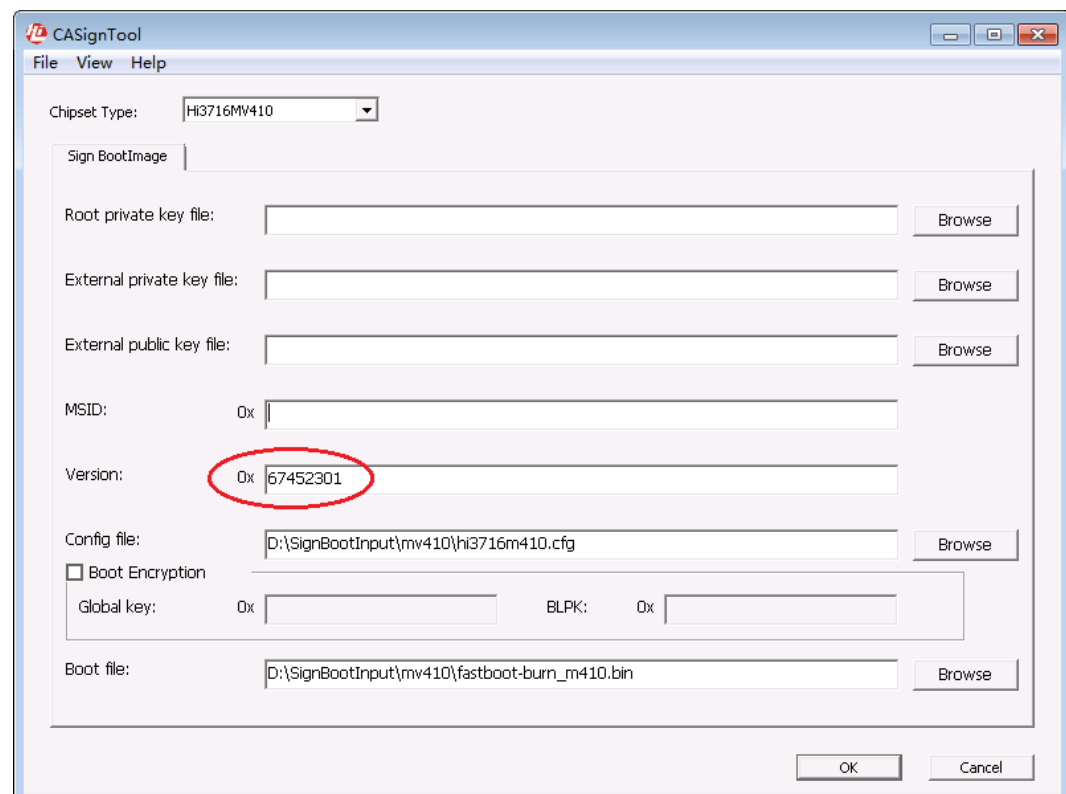


2.1.3.3 Setting the Version ID

If the version ID needs to be configured, set **Version** on the **Sign BootImage** tab page during the configuration described in section 2.1.2 "Procedures." The other operations are the same as those described in section 2.1.2 "Procedures."

The version ID cannot be configured for Hi3716C V100, Hi3716M V200, Hi3110E V300, and Hi3110E V400, but can be configured for Hi3716M V300, Hi3716C V200, Hi3719C V100, Hi3719M V100, Hi3716M V400, Hi3798M V100, Hi3716M V310, Hi3716M V420, Hi3716M V410, Hi3110E V500, and later chips.

Figure 2-11 Setting the version ID



The following describes the rules of configuring the version ID by using the 0x67452301 version ID as an example if the CA vendor requires that the version ID be set.



Table 2-2 Setting the version ID

Item	Hi3716C V200/Hi3719C V100/Hi3719M V100/Hi3716M V400/Hi3798M V100/Hi3716M V420/Hi3716M V410/Hi3110E V500	Hi3716M V300/Hi3716M V310
HI_UNF_ADVCA_SetVersionId(HI_U8 u8VersionId[4])	u8VersionId [0] = 0x67; u8VersionId [1] = 0x45; u8VersionId [2] = 0x23; u8VersionId [3] = 0x01;	u8VersionId [0] = 0x67; u8VersionId [1] = 0x45; u8VersionId [2] = 0x23; u8VersionId [3] = 0x01;
CASignTool	0x67452301	0x67452301
System Register	0xf8ab0124: 0x01; 0xf8ab0125: 0x23; 0xf8ab0126: 0x45; 0xf8ab0127: 0x67	0x10180124: 0x01; 0x10180125: 0x23; 0x10180126: 0x45; 0x10180127: 0x67
OTP (Internal use)	0xa4: 0x01; 0xa5: 0x23; 0xa6: 0x45; 0xa7: 0x67	0xa4: 0x01; 0xa5: 0x23; 0xa6: 0x45; 0xa7: 0x67

2.1.3.4 Setting the Mask

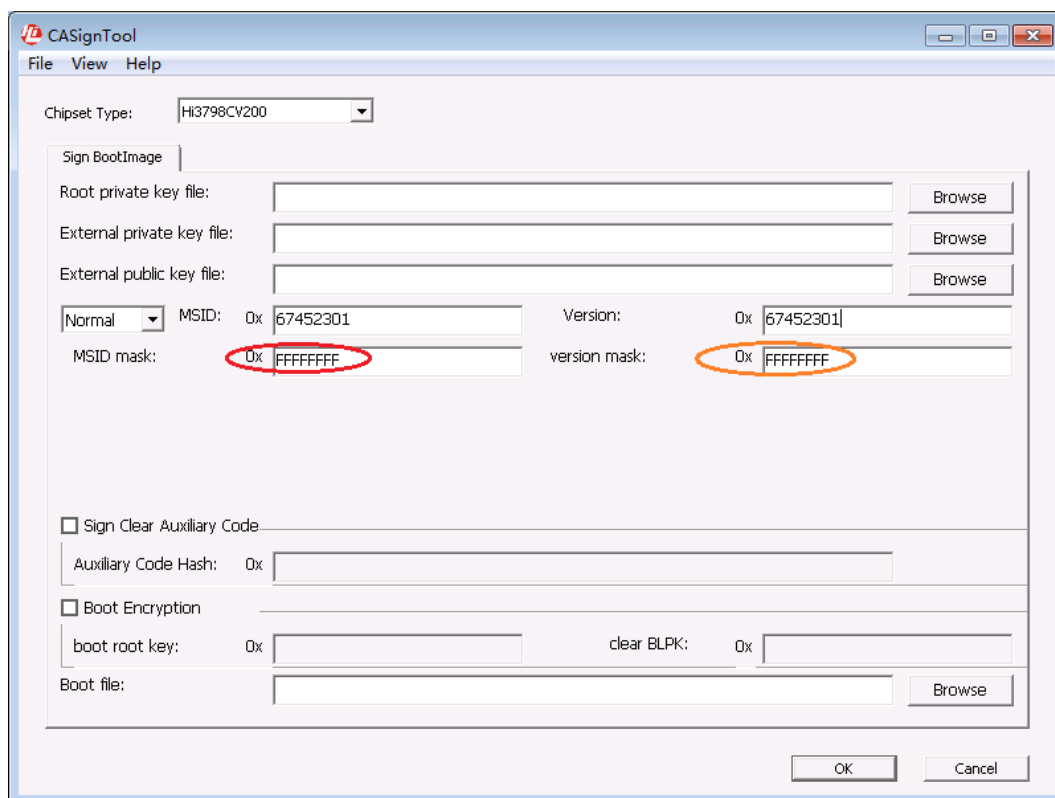
For the Hi3798C V200 series chips, the **MSID mask** and **version mask** may need to be set as required by CA vendors during signing. Mask is the mask value, indicating the bits in MSID and version ID that are checked. The default value is 0x00000000. For example, when the MSID is 0x12345678 and the MSID mask is 0xffff0000, the result is 0x12340000.

The MSID and version ID have two modes: normal mode and debug mode.

The debug mode is required by specified CA vendors. For details, see the development guides for the related CA vendors. The normal mode is used by default.



Figure 2-12 Setting the mask on the Sign BootImage tab page

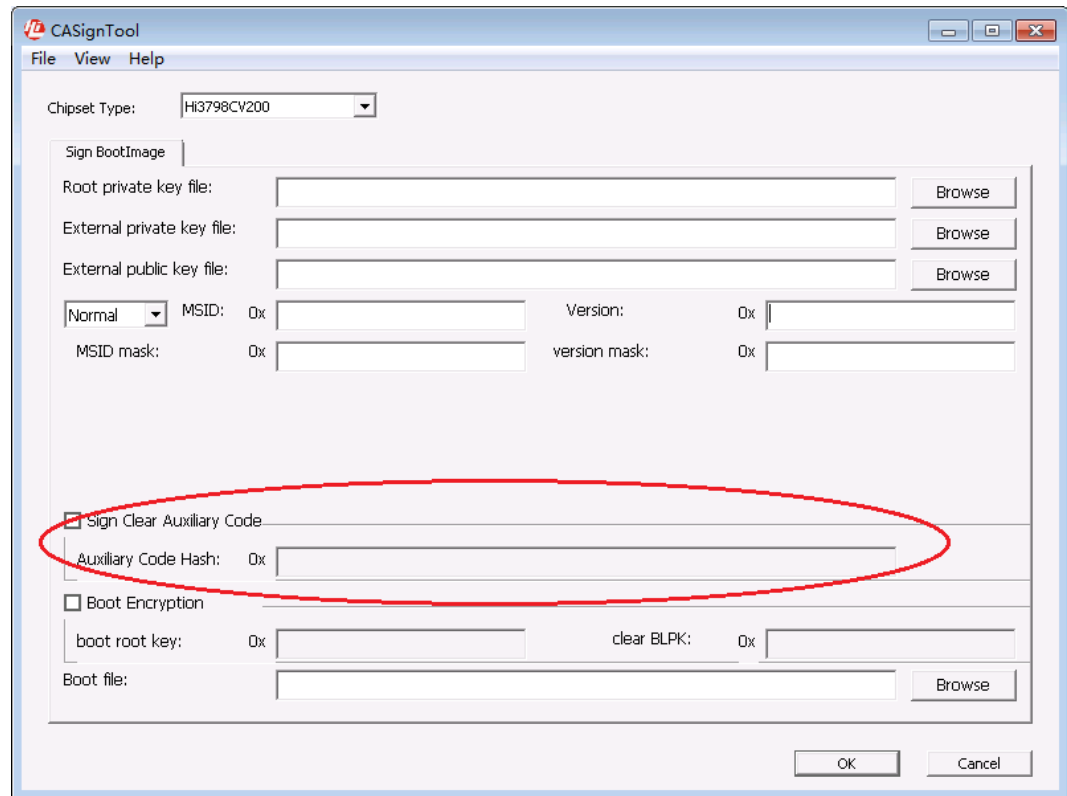


2.1.3.5 Setting the Auxiliary Code

For the Hi3798C V200 series chips, the BOOT contains the auxiliary code. Set **Auxiliary Code Hash** to control whether the auxiliary code signs the plain text during signing. That is, perform signing before encryption. This setting is required by special CA vendors and is not performed by default.



Figure 2-13 Setting the auxiliary code on the Sign BootImage tab page

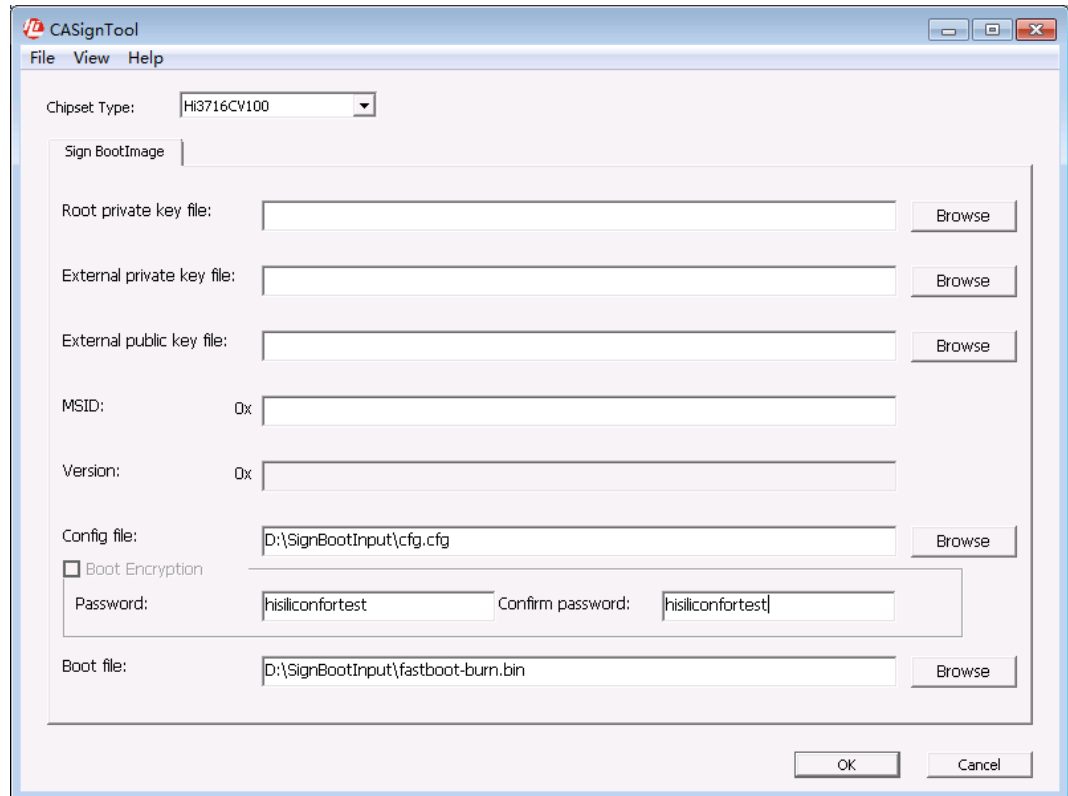


2.1.3.6 Setting the Key for Encrypting the Boot

The BOOT for the Hi3716M V200/Hi3716C V100/Hi3110E V300 advanced CA chip is encrypted and then signed. For these chips, the CASignTool can be used to encrypt the BOOT. If the BOOT file needs to be encrypted, set **Password** on the **Sign BootImage** tab page to a string of 16 ASCII codes during configuration described in section 2.1.2 "Procedures." The other operations are the same as those described in section 2.1.2 "Procedures."



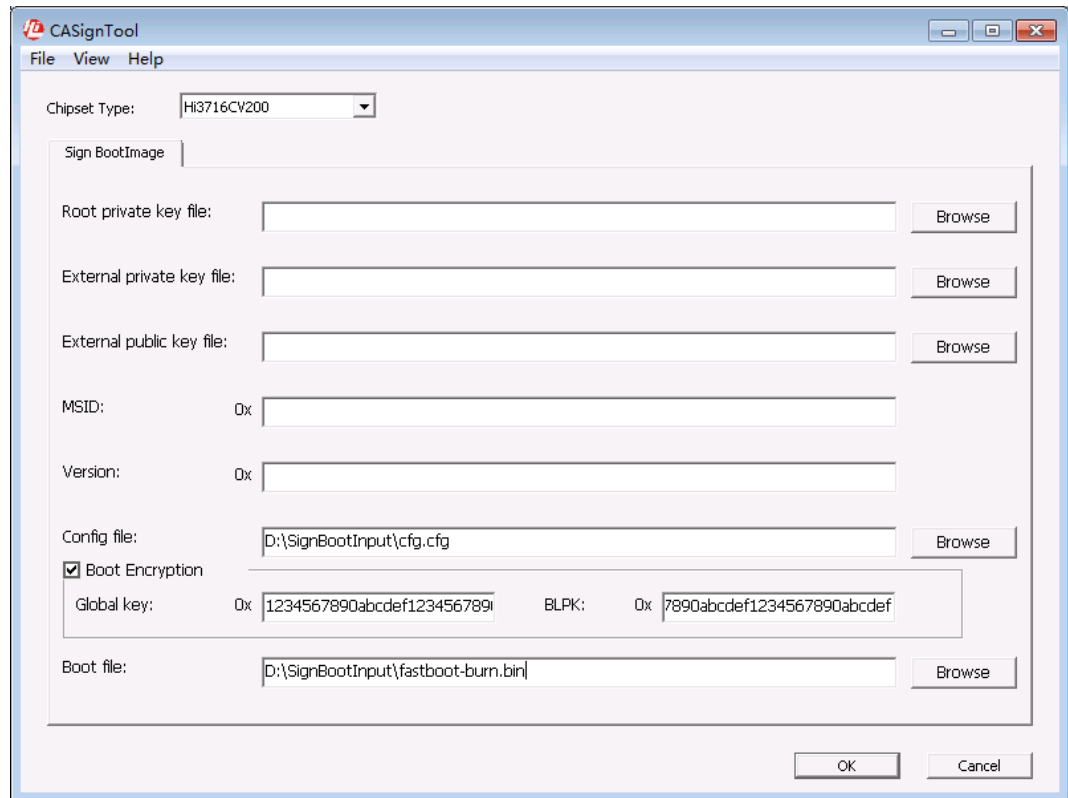
Figure 2-14 Setting the key for encrypting the BOOT for Hi3716M V200/Hi3716C V100/Hi3110E V300



The BOOT for the Hi3716M V300/Hi3716C V200/Hi3719C V100/Hi3719M V100/Hi3716M V400/Hi3716M V310/Hi3716M V420/Hi3716M V410/Hi3110E V500 advanced CA chip is signed and then encrypted. For most CA vendors, the BOOT for these chips cannot be encrypted by using the CASignTool. You need to burn **FinalBoot.bin** into the flash memory of the STB. The application then encrypts the BOOT after the STB starts properly. In this case, do not select **Boot Encryption** on the **Sign BootImage** tab page. However, for some special CA vendors, the BOOT file needs to be encrypted by using the STB root key, which is the same for all chips. In this case, select **Boot Encryption**, and set **Global key** to the STB root key and **BLPK** to the protection key for encrypting the BOOT file. The other operations are the same as those described in section 2.1.2 "Procedures."



Figure 2-15 Setting the key for encrypting the BOOT for the BOOT image signature



NOTE

The values of both **Global key** and **BLPK** are 16-byte hexadecimal digits. You can separate every two bytes by using a space or not.

2.2 Generating a Signed Secure Boot File

A signed secure BOOT file applies to the advanced CA chips with the enabled secure BOOT function.

2.2.1 Environment Preparation

For details, see section [2.1.1 "Environment Preparation."](#)

2.2.2 Procedures

To generate a signed secure BOOT file, perform the following steps:

Step 1 Perform operations according to steps 1 to 5 described in section [2.1.2 "Procedures."](#)

During STB development, some STB vendors may have got the Ext_RSA_Pub_Key from the CA vendor. In this case, you need to set **External public key file** to **external_rsa_pub.txt** to generate **KeyArea.bin**.

Step 2 Select **KeyArea.bin**, **ParamArea.bin**, and **BootArea.bin** in the generated folder and send them to the CA vendor for signature.



For the Hi3798C V200 series chips, **KeyArea.bin**, **ParamArea.bin**, **BootArea.bin**, and **AuxArea.bin** need to be selected and sent to the CA vendor as required for signature application.

Figure 2-16 Selecting files

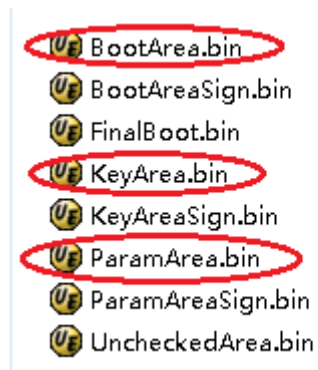
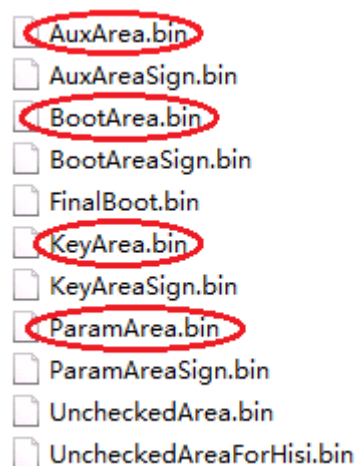


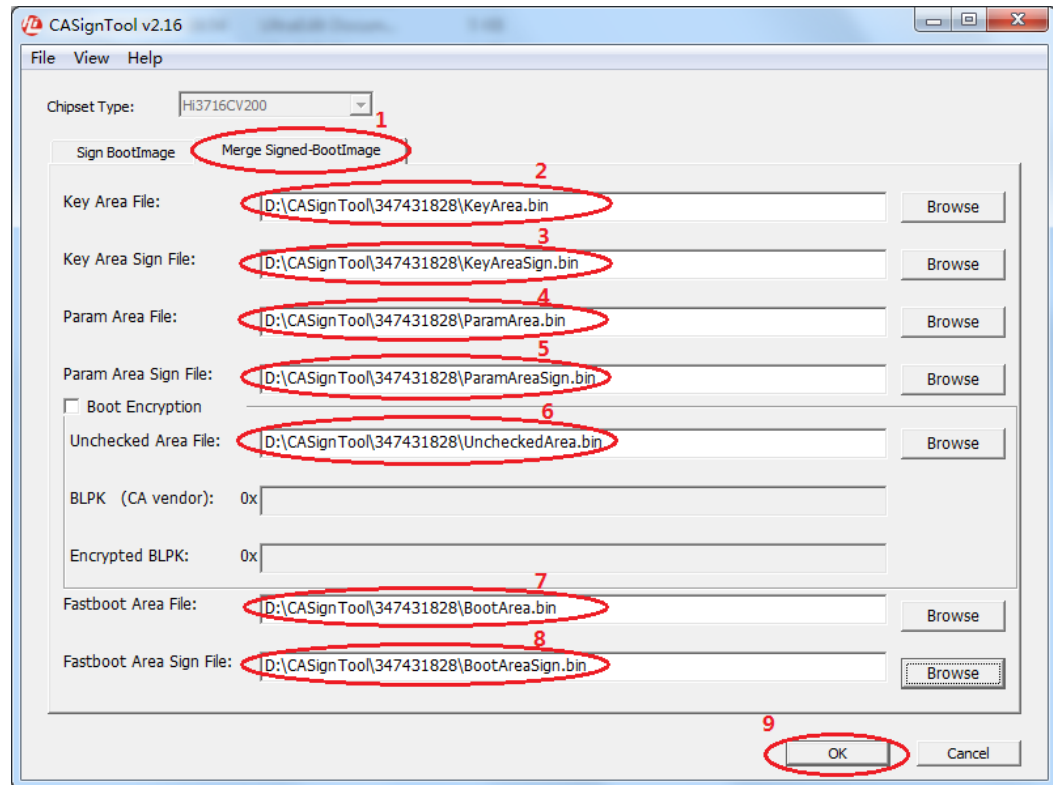
Figure 2-17 Selecting files (for Hi3798C V200 series chips)



Step 3 After obtaining the signature from the CA vendor, click **View**, select **Merge Signed-BootImage Window**, and then click the **Merge Signed-BootImage** tab.



Figure 2-18 Merging the files into a final file

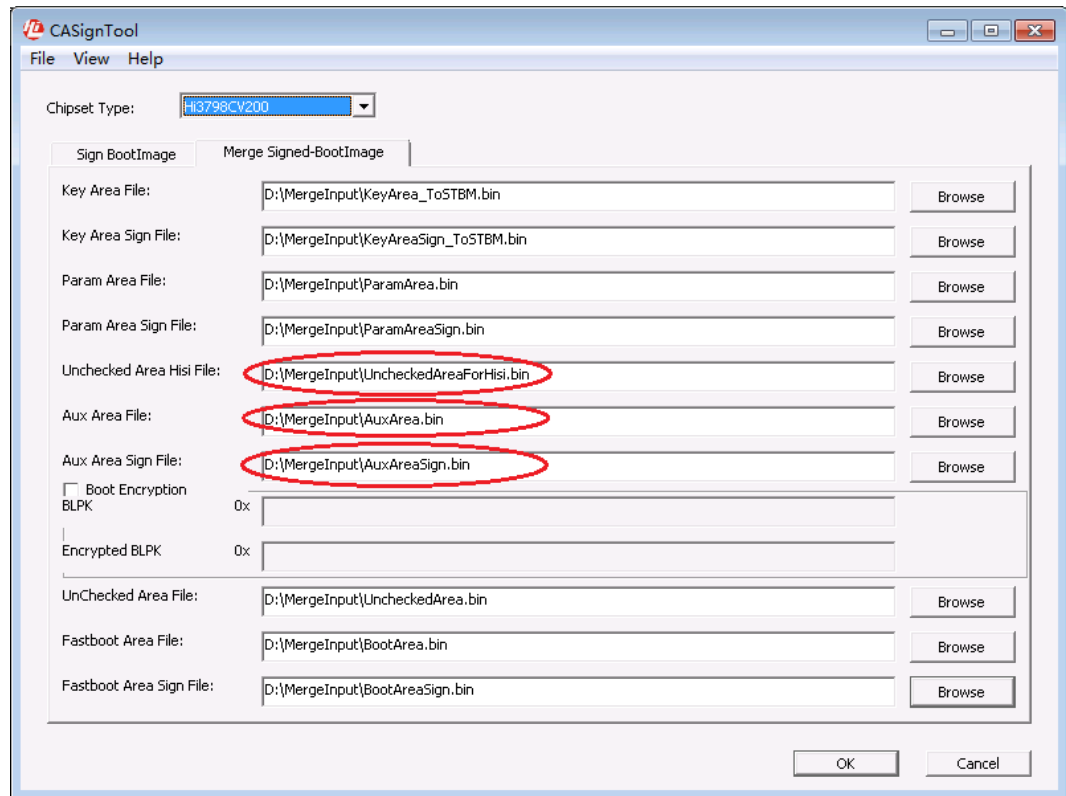


- Step 4** Click **Browse** after **Key Area File** to select **KeyArea.bin** generated in step 1, and click **Browse** after **Key Area Sign File** to select the **KeyArea.bin** signature obtained from the CA vendor.
- Step 5** Click **Browse** after **Param Area File** to select **ParamArea.bin** generated in step 1, and click **Browse** after **Param Area Sign File** to select the **ParamArea.bin** signature obtained from the CA vendor.
- Step 6** Click **Browse** after **Unchecked Area File** to select **UncheckedArea.bin** generated in step 1.
- Step 7** Click **Browse** after **FastBOOT Area File** to select **BootArea.bin** generated in step 1, and click **Browse** after **FastBOOT Area Sign File** to select the **BootArea.bin** signature obtained from the CA vendor.

For the Hi3798C V200 series chips, the signature of **AuxArea.bin** obtained from the CA vendor, and **UncheckedAreaForHisi.bin** and **AuxArea.bin** generated in step 1 also need to be selected.



Figure 2-19 Merging the files into a final file (for the Hi3798C V200 series chips)

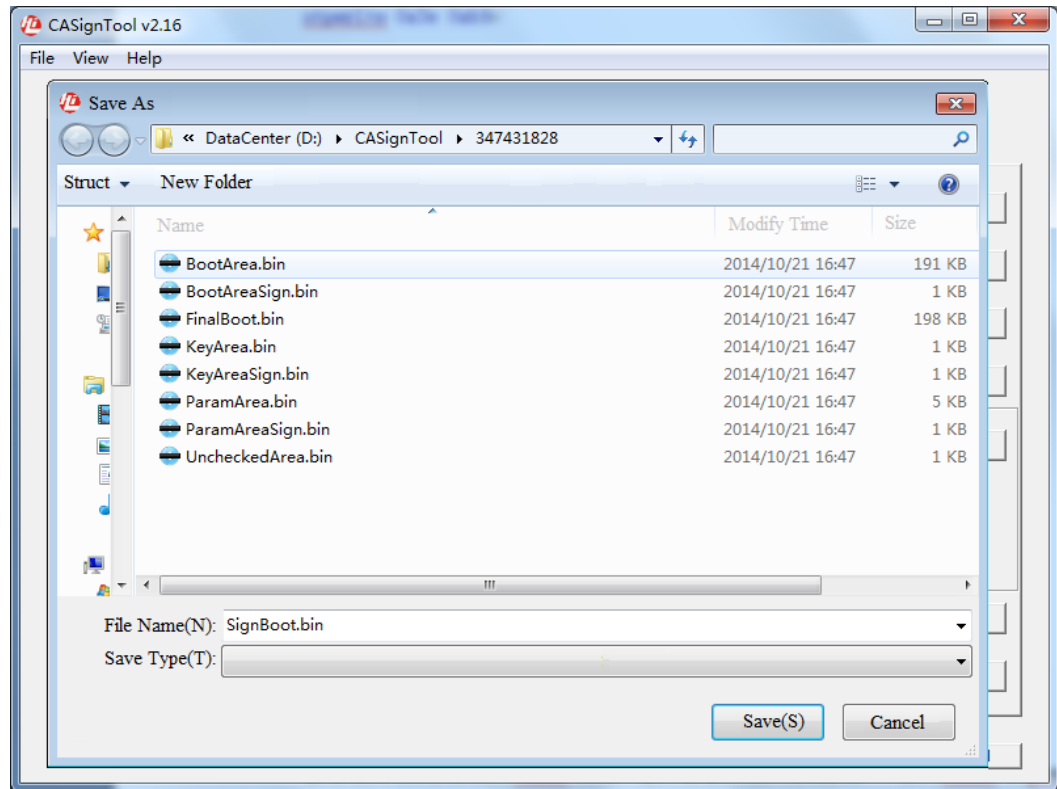


Step 8 Click **OK**.

Step 9 Specify the name for the signed BOOT file.



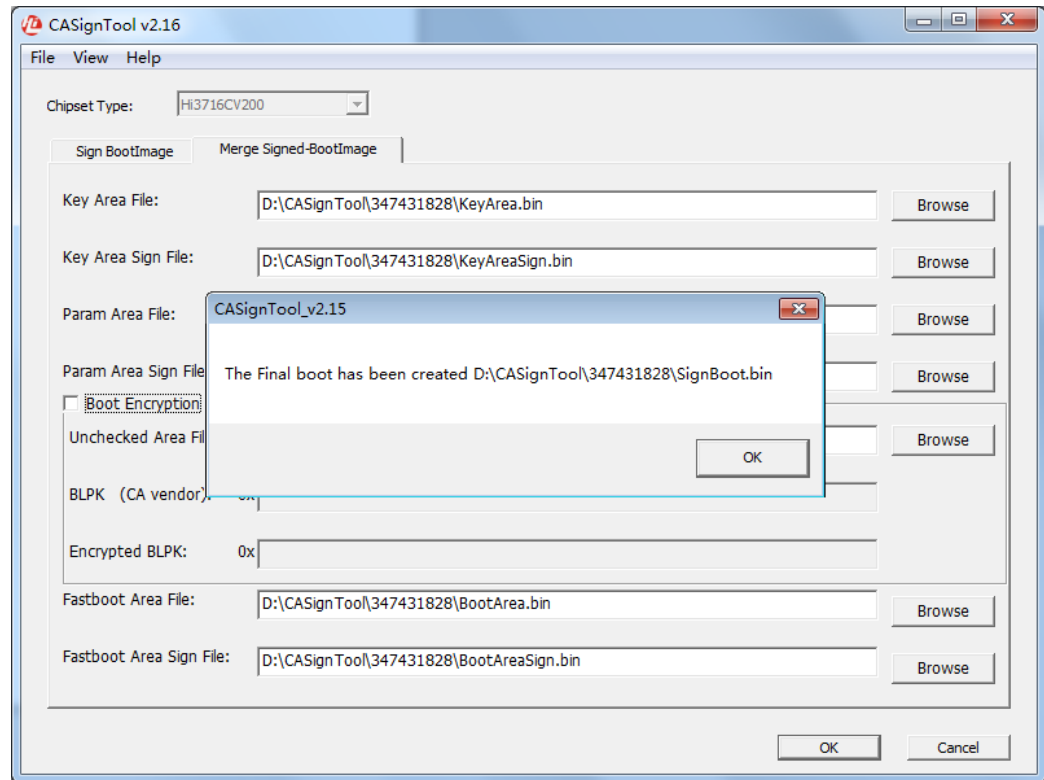
Figure 2-20 Specifying the name for the signed BOOT file



A dialog box is displayed, indicating that the signed BOOT file **FinalBoot.bin** has been created.



Figure 2-21 Generating the signed BOOT file



----End

2.3 Generating a Self-Signed Secure Boot File

When the CA vendor does not keep the signature key, you can still sign the secure BOOT file by using the CASignTool.

2.3.1 Environment Preparation

Perform the following steps:

Step 1 Configure two pairs of keys.

The RSA2048 passwords are used for signing the BOOT files for HiSilicon advanced CA chips. Two pairs of RSA2048 keys are required:

- A pair of root RSA keys
- A pair of external RSA keys

Each pair of keys includes a public key and private key. Therefore, there are four keys:

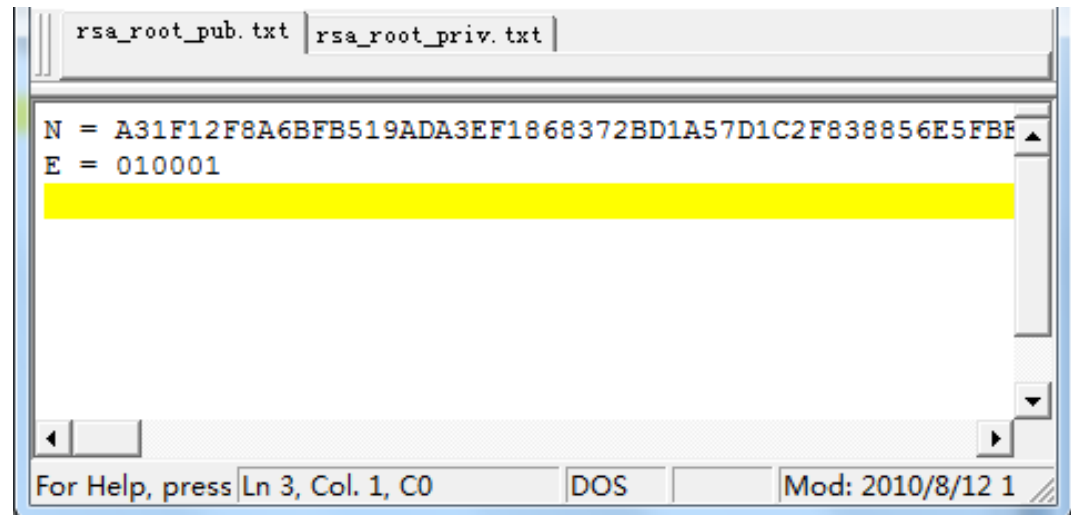
- Public root key
- Private root key
- Public external key
- Private external key



The public root key needs to be burnt to the one-time programmable (OTP) area. For details, see **ca_writeRSAkey.c** in the SDK.

The private root key, public external key, and private external key are used by the CASignTool. [Figure 2-22](#) shows the format of the public key required by the CASignTool.

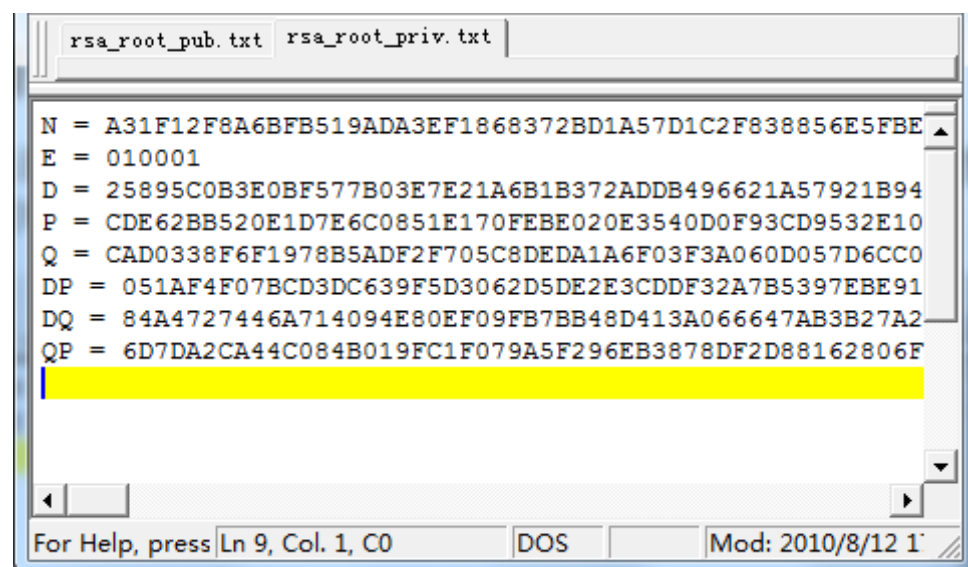
Figure 2-22 Format of the public key required by the CASignTool



The public key file is a **.txt** file. *N* is a 2048-bit hexadecimal value. That is, there are 512 hexadecimal values after "N =".

[Figure 2-23](#) shows the format of the private key required by the CASignTool.

Figure 2-23 Format of the private key required by the CASignTool



The private key file is also a **.txt** file.

Step 2 Set the board parameters. For details, see **cfg.bin** in section [2.1.1 "Environment Preparation."](#)



Step 3 Prepare a common BOOT file.

----End



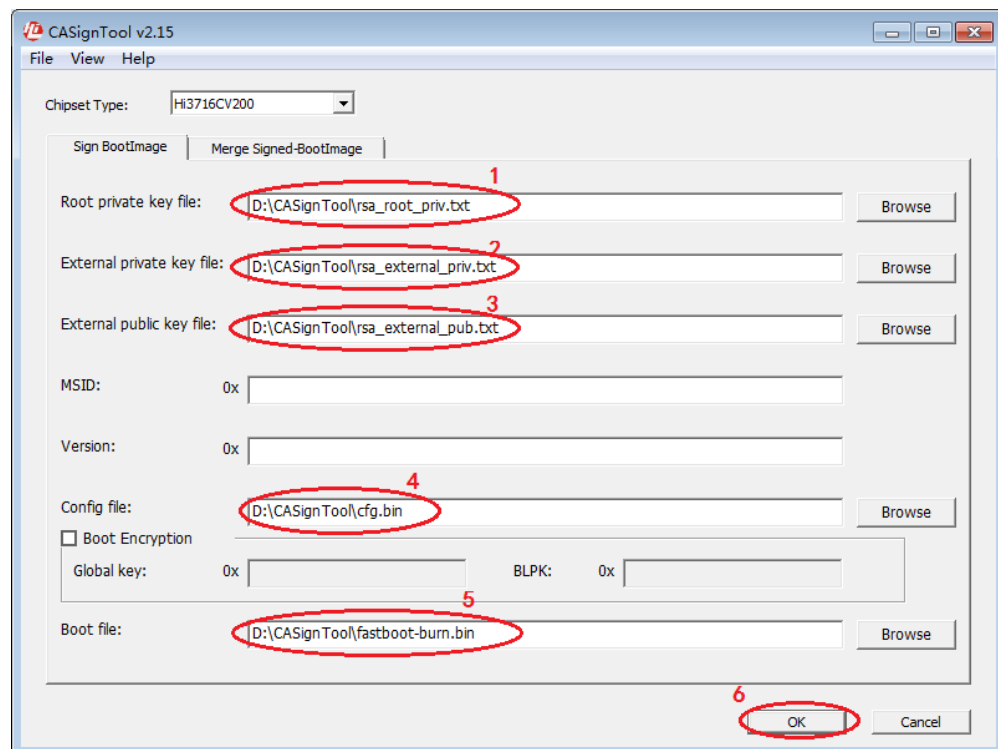
CAUTION

- If you want to use the HiSilicon CASignTool, your RSA private key must contain five groups of data (N, E, D, P, Q).
- You can download **mbed TLS** from <https://tls.mbed.org/>, modify **rsa_genkey.c** in **mbedtls/programs/pkey**, compile and run **rsa_genkey** to generate the RSA key data required by HiSilicon.
- The STB vendor can generate the RSA key for development on the **Create RSA Key** tab page. During production, the CA vendor maintains the private RSA key. The STB vendor can request the public RSA key from the CA vendor and submit images to the CA vendor for signature.

2.3.2 Procedures

Perform the following steps:

Figure 2-24 Generating a self-signed BOOT file



Step 1 Start the CASignTool, click **View**, and select **Sign BootImage Window**.

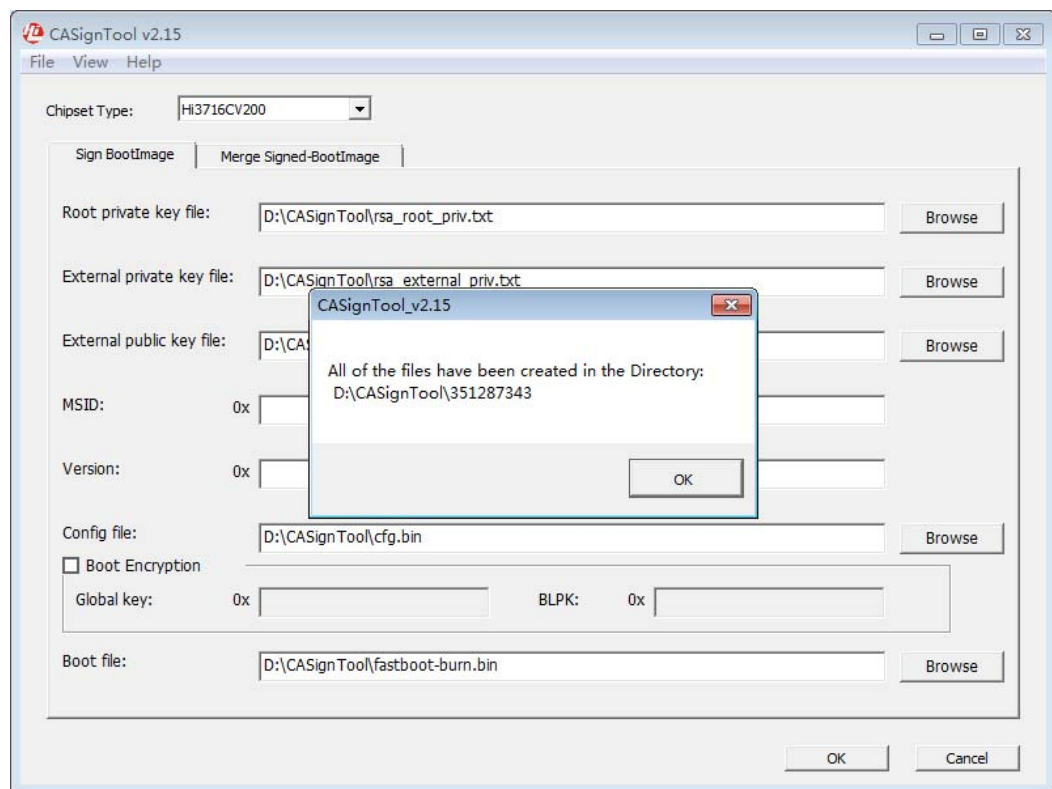
Step 2 Select a chip from the **Chipset Type** drop-down list.



- Step 3** Click **Browse** after **Root private key file** to select the private root key prepared in section 2.3.1 "Environment Preparation."
- Step 4** Click **Browse** after **External private key file** to select the private external key prepared in section 2.3.1 "Environment Preparation."
- Step 5** Click **Browse** after **External public key file** to select the public external key prepared in section 2.3.1 "Environment Preparation."
- Step 6** Click **Browse** after **Config file** to select **cfg.bin** prepared in section 2.3.1 "Environment Preparation."
- Step 7** Click **Browse** after **Boot file** to select the BOOT file prepared in section 2.3.1 "Environment Preparation."
- Step 8** Click **OK**.

The CASignTool creates a folder in the folder where the CASignTool locates for storing the generated file. The folder name **351287343** shown in Figure 2-25 is random and may be different from the actual file.

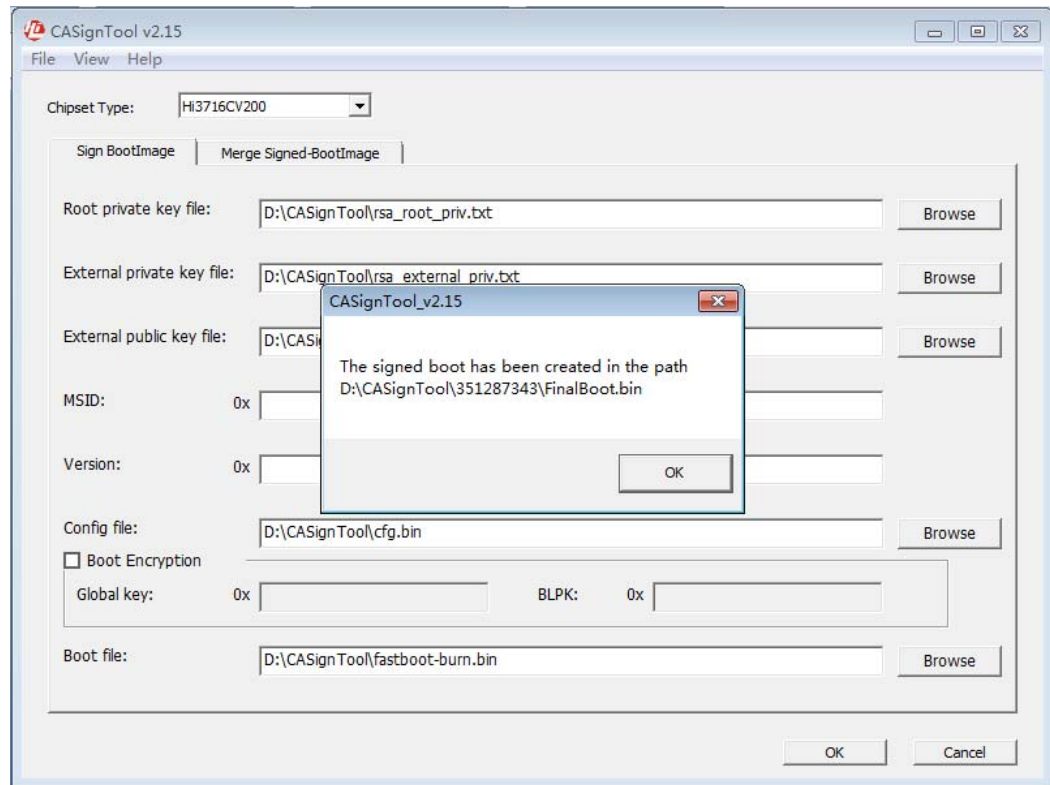
Figure 2-25 Generating the folder



A dialog box is displayed, indicating that the signed BOOT file **FinalBoot.bin** has been created.



Figure 2-26 Generating FinalBoot.bin



----End

2.3.3 Precautions

If you want to configure the MSID and version ID or encrypt the BOOT, see section [2.1.3 "Precautions."](#)

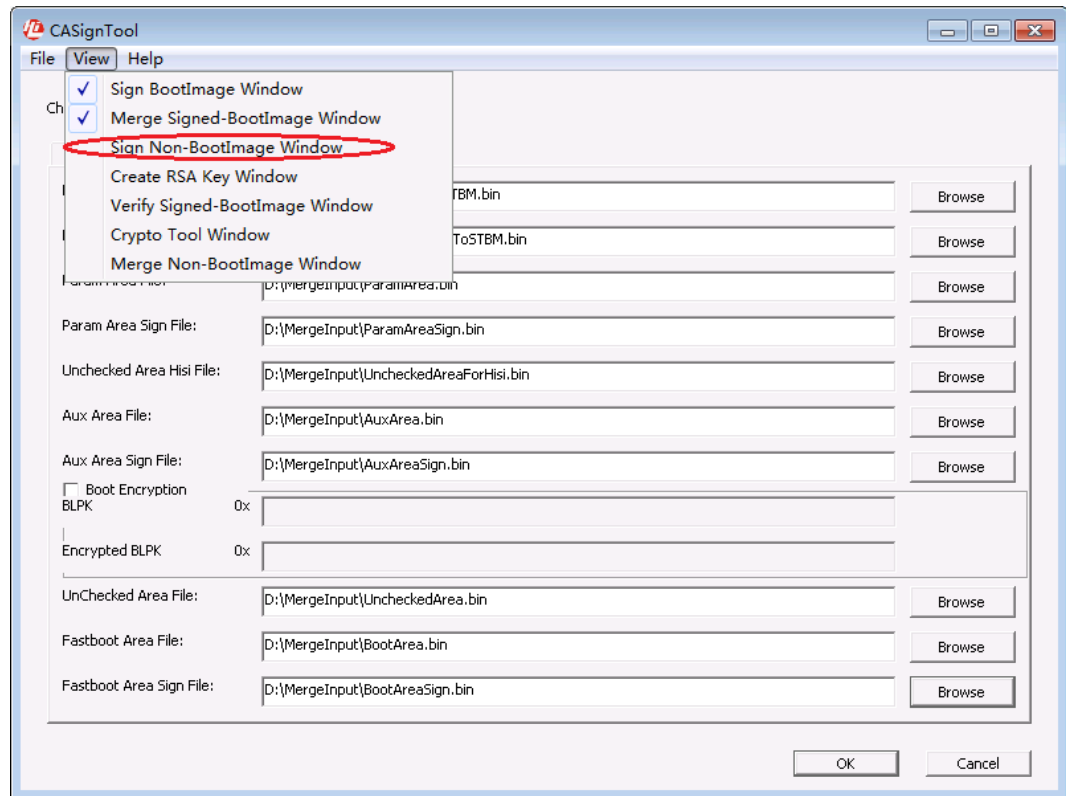
2.4 Signing Non-Boot Images

The CASignTool can merge images of the kernel, file system, and applications into an image in the required format, and then sign the image.

Before using this function, click **View**, and select **Sign Non-BootImage Window** to display the **Sign Non-BootImage** tab, as shown in [Figure 2-27](#).



Figure 2-27 Sign Non-BootImage Window



HiSilicon provides methods for signing non-BOOT images: special CA signature, common CA signature, and M2-3rd signature for non-BOOT file. For details about the application scenarios, see the *Advanced Secure CA Development Guide*.

2.4.1 Special CA Signature for Non-Boot Images

This signature mode is used when the image must be encrypted and the memory file system is used. The generated signature data consists of two parts: the index information is stored in the fore part of the image, and other data is stored in the tail.

Storing index information in the fore part of the image allows you to quickly find the signature and image parameters.

The format of the signature data is as follows:

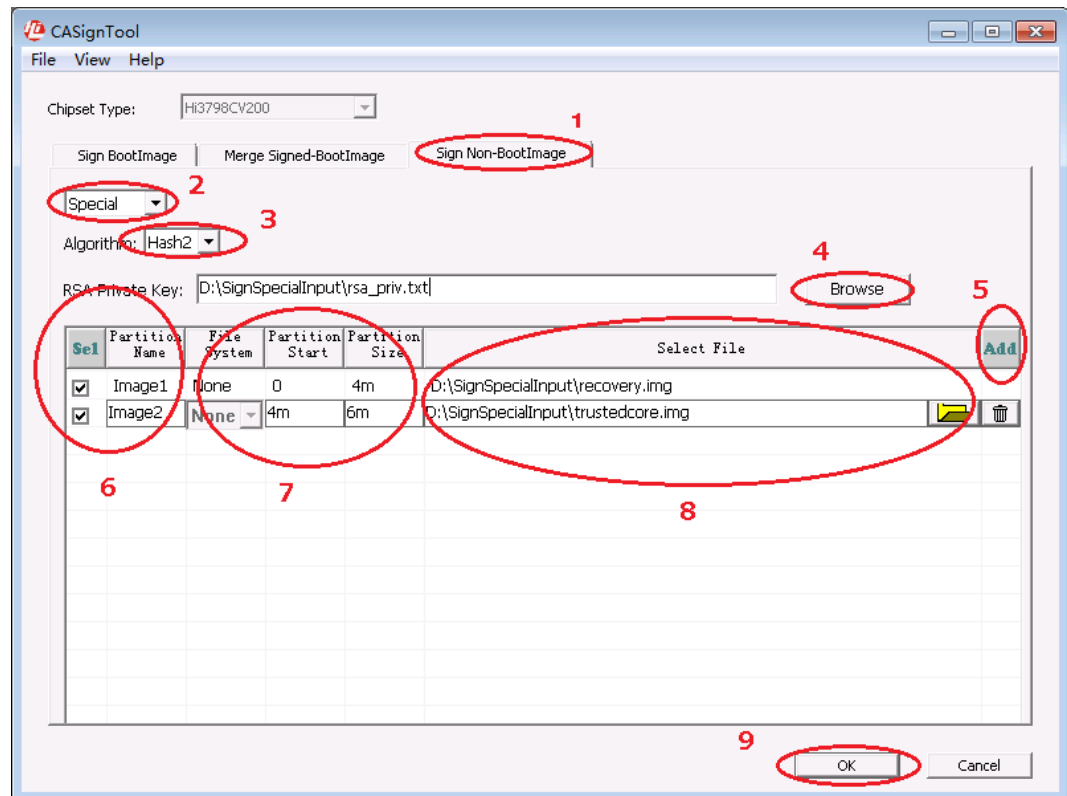
```
typedef struct hi_CAImgHead_S
{
    HI_U8  u8MagicNumber[32];           //Magic Number:
    "Hisilicon_ADVCA_ImgHead_MagicNum"
    HI_U8  u8Version[8];                //version: "V000 0003"
    HI_U32 u32TotalLen;                 //Total length
    HI_U32 u32CodeOffset;               //Image offset
    HI_U32 u32SignedImageLen;           //Signed Image file size
    HI_U32 u32SignatureOffset;          //Signed Image offset
    HI_U32 u32SignatureLen;              //Signature length
}
```



```
HI_U32 u32BlockNum; //Image block number
HI_U32 u32BlockOffset[IMG_MAX_BLOCK_NUM]; //Each Block offset
HI_U32 u32BlockLength[IMG_MAX_BLOCK_NUM]; //Each Block length
HI_U32 Reserved[32];
HI_U32 u32CRC32; //CRC32 value
} HI_CAIImgHead_S;
```

Perform the following steps:

Figure 2-28 Special CA signature for non-BOOT images



- Step 1** Click the **Sign Non-BootImage** tab.
- Step 2** Select **Special CA Signature**.
- Step 3** Select a signature algorithm. Hash 1 and Hash 2 are supported, and Hash 2 is selected by default.
- Step 4** Click **Browse** after **RSA Private Key** to select the RSA key file used for the signature.
- Step 5** Click **Add** to add a row.
- Step 6** Select the images to be signed.
- Step 7** Specify the relative start address and length for each image in the flash partition.



NOTE

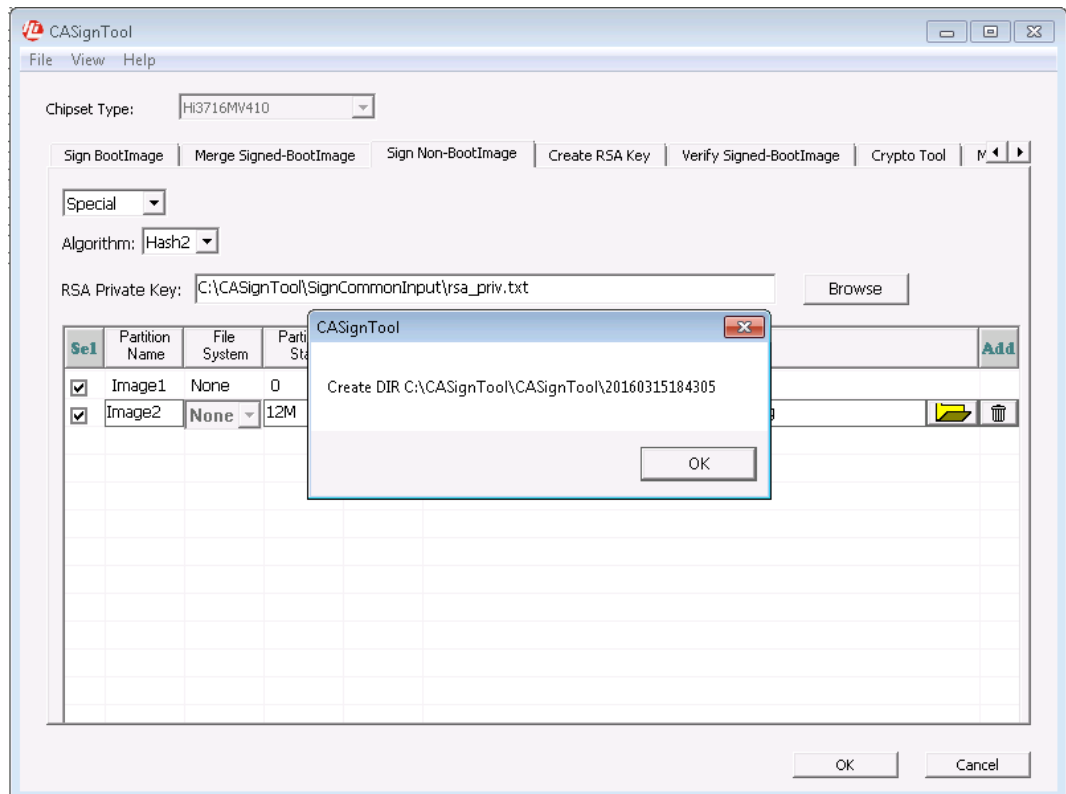
- The relative start address for the first file must be **0**.
- The unit is KB or MB.
- The value of **Partition Start** must be greater than or equal to (**Partition Start** + **Partition Size**) of the previous partition.
- The size of the signed image depends on the actual size of the image in the last partition but not the size of the last partition.

Step 8 Specify the images to be signed.

- If you select only one file, a final signed image for the file is generated.
- If you select multiple files, these files are merged into one image and the image is signed.

Step 9 Click **OK**. A new folder is generated in the **CASignTool** folder.

Figure 2-29 Information indicating that a new folder has been created



Step 10 Click **OK**. Then the final signed image **FinalImage.bin** is generated, as shown in [Figure 2-30](#).



Figure 2-30 Generating FinallImage.bin

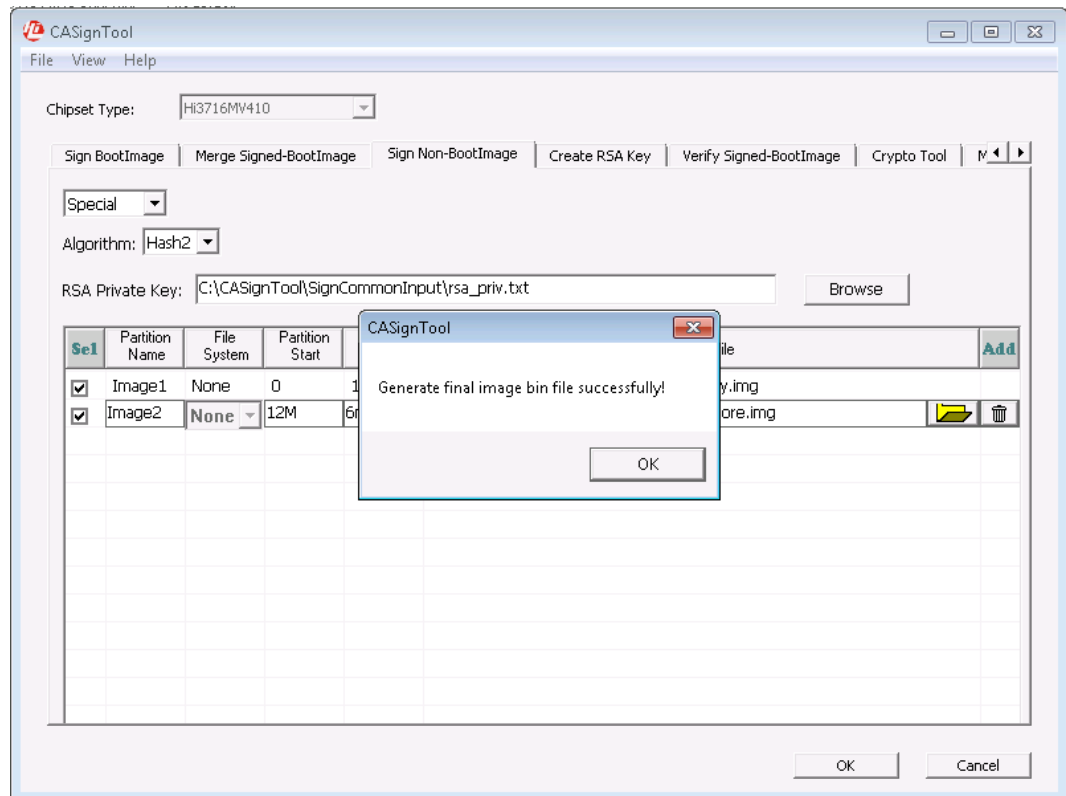


Figure 2-31 shows the structure of the signed image.

Figure 2-31 Structure of the signed image in special CA signature mode



----End

2.4.2 Common CA Signature for Non-Boot Images

This signature mode is used when the image file does not need to be encrypted, that is, when the non-memory file system is used. The generated signature data is stored in the tail of the image.

Because the signature data is stored in the tail of the image, the specific position for storing the signature data needs to be specified by using other methods.

The format of the signature data is as follows:

```
typedef struct
{
    unsigned char u8MagicNumber[32];           //Magic Number:
    "Hisilicon_ADVCA_ImgHead_MagicNum"
    unsigned char u8Version[8];                //version: "V000 0003"
    unsigned int u32CreateDay;                  //yyyymmdd
    unsigned int u32CreateTime;                 //hhmmss
    unsigned int u32HeadLength;                 //The following data size
    unsigned int u32SignedDataLength;           //signed data length
    unsigned int u32IsYaffsImage;              //Yaffsr image need to
    special read-out, No use
}
```



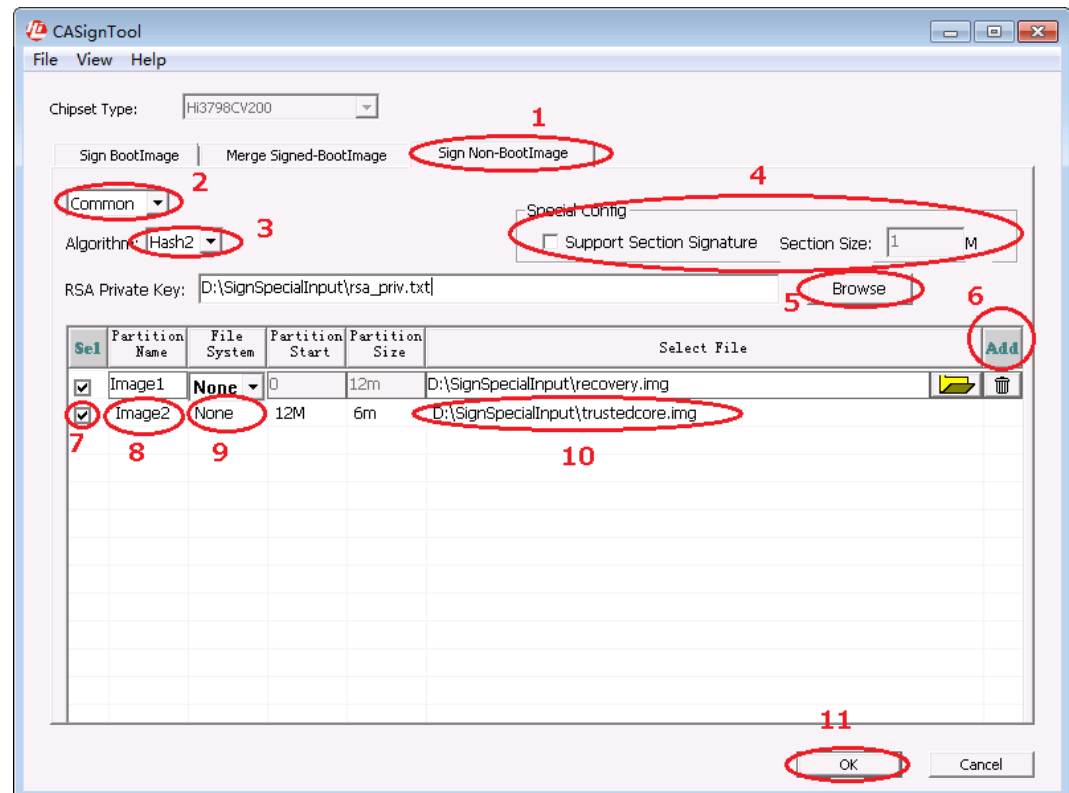
```

        unsigned int  u32IsConfigNandBlock;           //Yaffsr image need to
special read-out, No use
        unsigned int  u32NandBlockType;              //Yaffsr image need to
special read-out, No use
        unsigned int  u32IsNeedEncrypt;              //if "1", code need to be
encrypted.
        unsigned int  u32IsEncrypted;                //if "1", code has encrypted.
        unsigned int  u32HashType;    //if "0", u8Sha save sha1 of code, if "1",
u8Sha save sha256 of code
        unsigned char u8Sha[32];                    //SHA value
        unsigned int  u32SignatureLen;               //Actual Signature length
        unsigned char u8Signature[256];             //Max length:0x100
        unsigned char OriginalImagePath[256];       //Max length:
        unsigned char RSAPrivateKeyPath[256];       //Max length:0x100
        unsigned int  u32CurrentsectionID;           //begin with 0
        unsigned int  u32SectionSize;                //section size
        unsigned int  u32TotalsectionID;             //Max section ID > 1
        unsigned int  CRC32;                         //CRC32 value
    } HI_CASignImageTail_S;

```

Perform the following steps:

Figure 2-32 Common CA signature for non-BOOT images





Step 1 Click the **Sign Non-BootImage** tab.

Step 2 Select **Common CA Signature**.

Step 3 Select a signature algorithm. Hash 1 and Hash 2 are supported, and Hash 2 is selected by default.

Step 4 (Optional) Select **Support Section Signature** as required. (Divide the image file into multiple sections, each of which requires a signature.)

- If **Support Section Signature** is selected, specify **Section Size**. The default value is 1 MB.
- The size of a section is at most 100 MB and cannot be greater than 1/2 of the image size.
- This function can be used if the image to be signed is large (for example, the Android file system is over 200 MB). Signature verification takes a long time if the image is large. In this case, if the large image is divided into multiple sections, you can randomly verify the signatures of only several sections.

Step 5 Click **Browse** after **RSA Private Key** to select the RSA key file used for the signature.

Step 6 Click **Add** to add a row.

- If you select only one file, a final signed image for the file is generated.
- If multiple files are selected, the following files are generated for each file:
 - A file that contains the image and the signature
 - An independent signature data file

Step 7 Select the images to be signed.

Step 8 Set the name of each partition that stores the images in the flash memory.

Step 9 Specify the partition type (**None** or **Yaffs2**, **None** by default).



NOTE

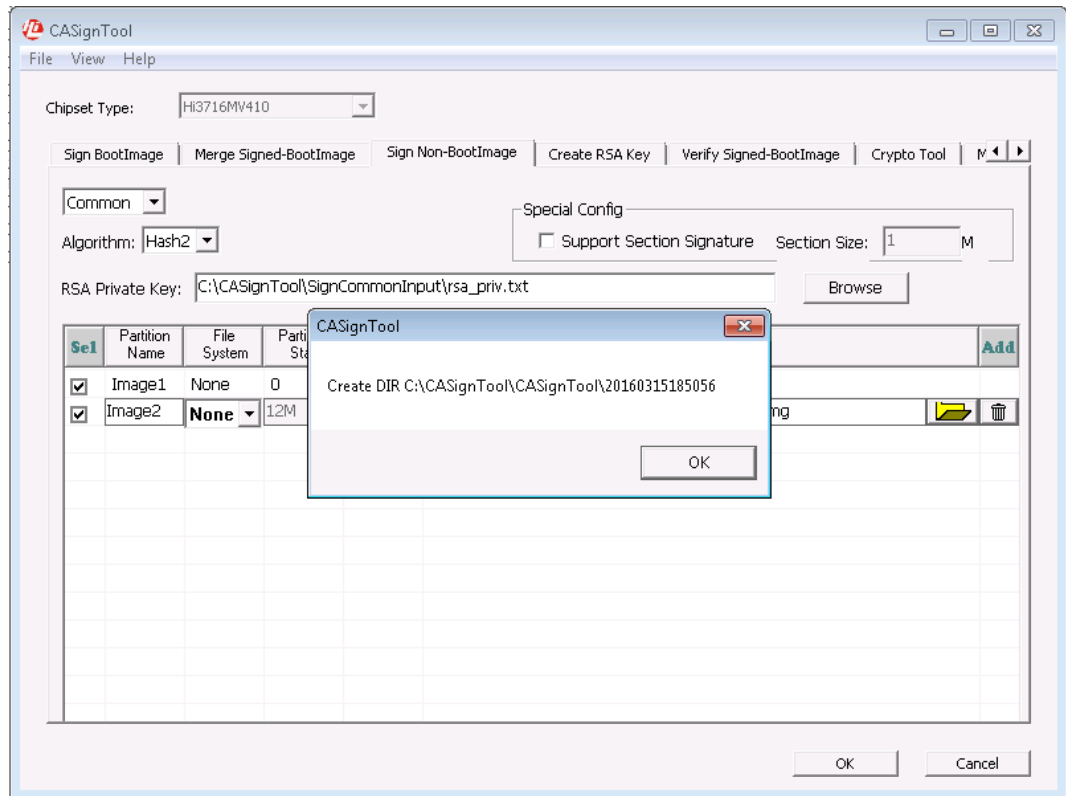
If the partition type is set to **Yaffs2**, data to be signed in the Yaffs file system includes the OOB data. The CASignTool generates only an independent signature data file for this image. The Yaffs2 file system is not recommended.

Step 10 Specify the images to be signed.

Step 11 Click **OK**. A new folder is generated in the **CASignTool** folder.



Figure 2-33 Generating a new folder



Step 12 Click **OK**. Then the final signed image is generated, as shown in [Figure 2-34](#).



Figure 2-34 Generating FinallImage.bin

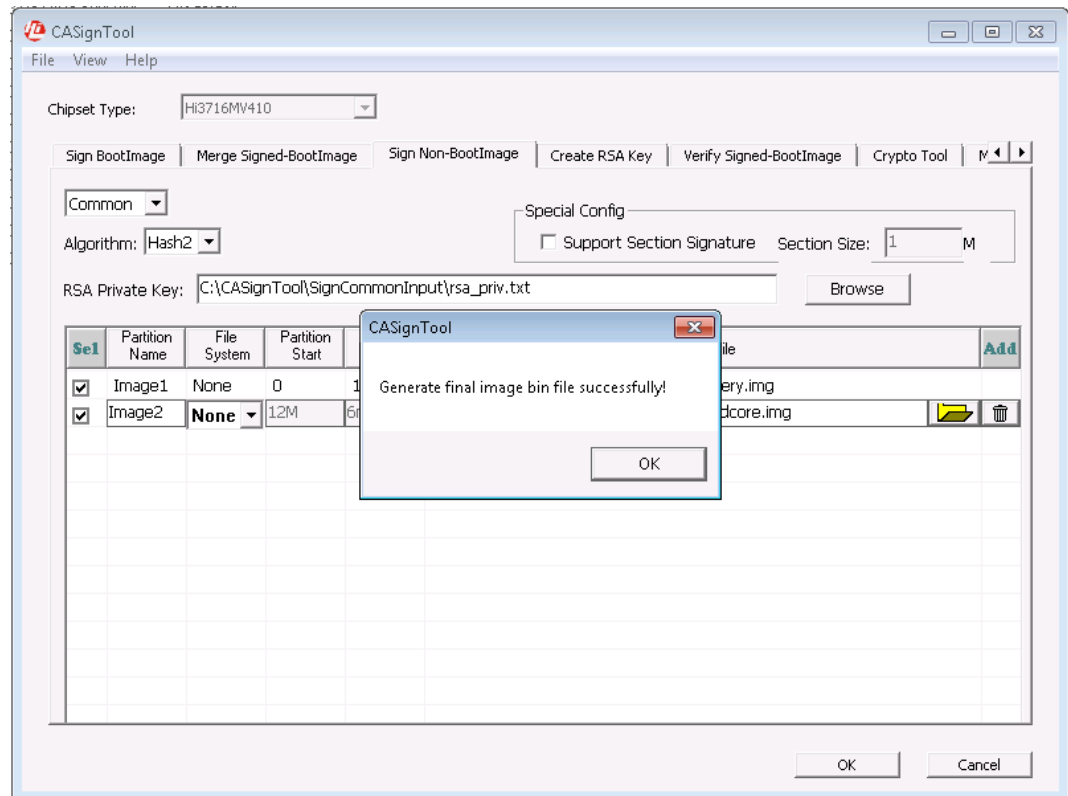


Figure 2-35 and Figure 2-36 show the structure of the signed image.

Figure 2-35 Structure of the signed image in common CA signature mode (the image is not divided into sections)

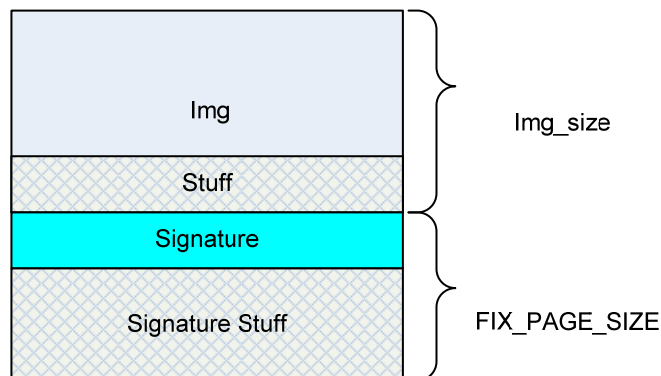
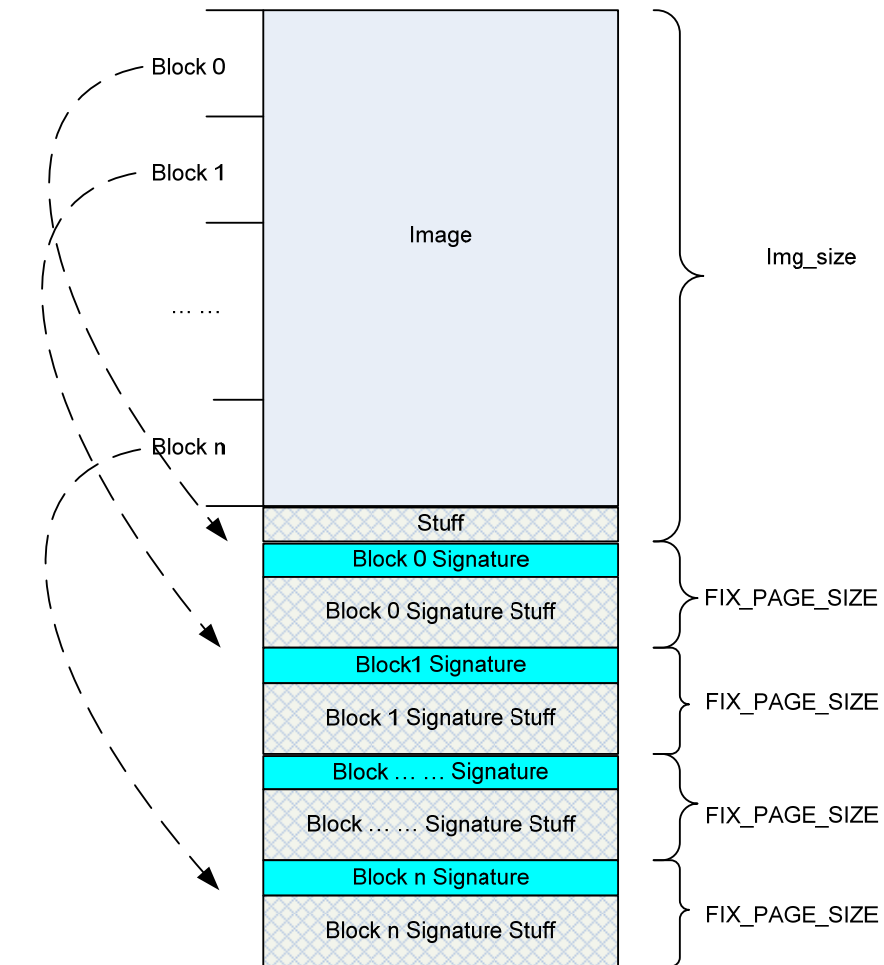




Figure 2-36 Structure of the signed image in common CA signature mode (the image is divided into sections)

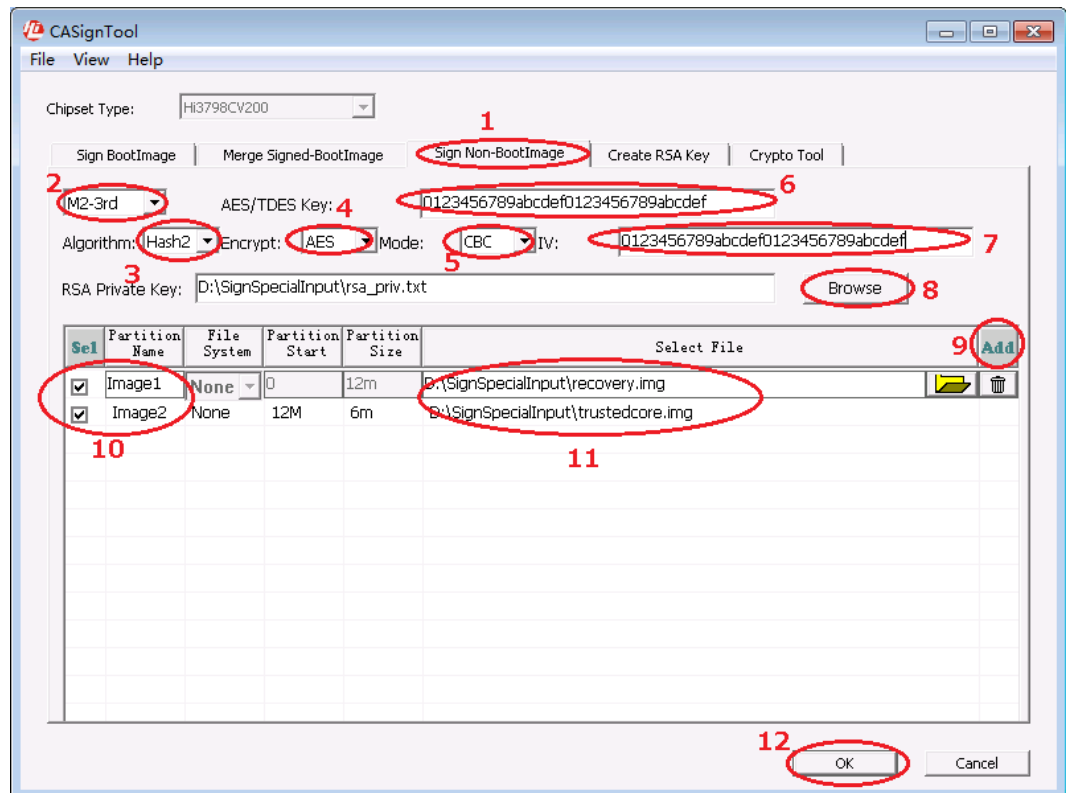


----End



2.4.3 M2-3rd CA Signature for Non-BOOT Images

Figure 2-37 M2-3rd CA signature for non-BOOT images

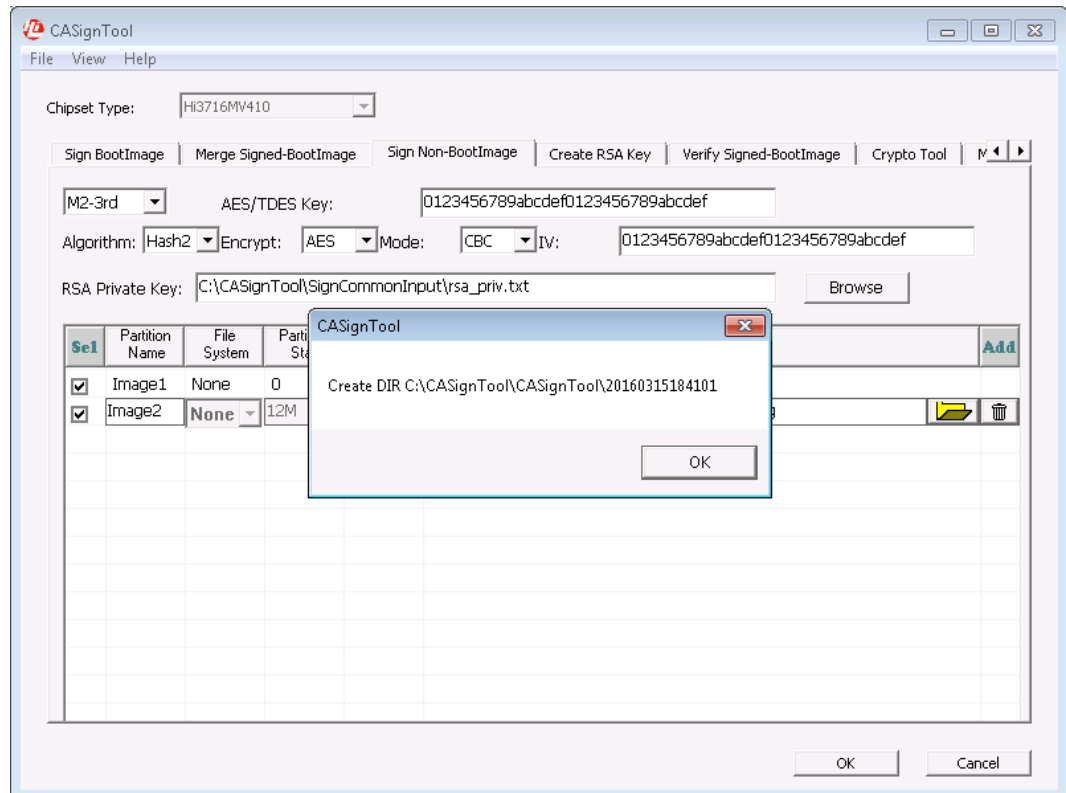


Perform the following steps:

- Step 1** Click the **Sign Non-BootImage** tab.
 - Step 2** Select **M2-3rd CA Signature**.
 - Step 3** Select a signature algorithm. Hash 1 and Hash 2 are supported, and Hash 2 is selected by default.
 - Step 4** Select an encryption algorithm. AES and TDES are supported.
 - Step 5** Select an encryption mode. CBC and ECB are supported.
 - Step 6** Input the encryption key.
 - Step 7** Input the IV vector value (only in CBC mode).
 - Step 8** Select the private key for signing.
 - Step 9** Click **Add** to add a row.
- If various files are selected, each file may generate a file containing images and the signature.
- Step 10** Select the images to be signed.
 - Step 11** Specify the images to be signed.
 - Step 12** Click **OK**. A new folder is generated in the **CASignTool** folder.



Figure 2-38 Generating a new folder



Click **OK**. Then the final signed image is generated, as shown in [Step 12](#) in section 2.4.2 "Common CA Signature for Non-Boot Images."



Figure 2-39 Generating FinallImage.bin

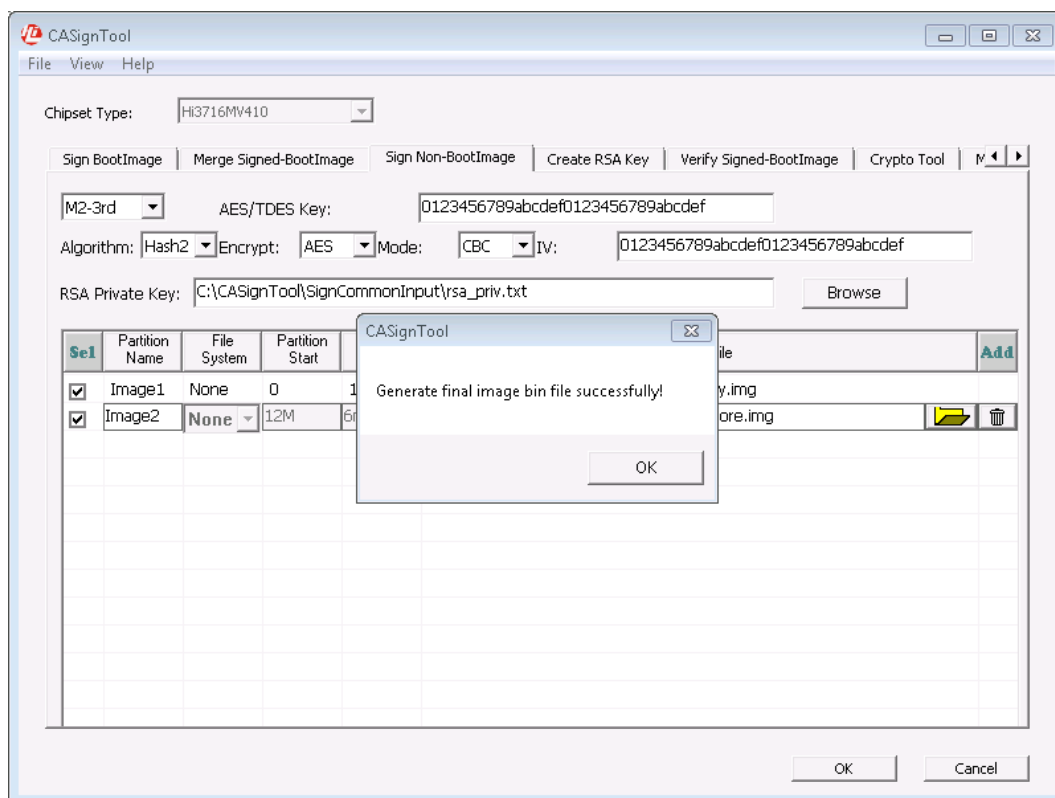
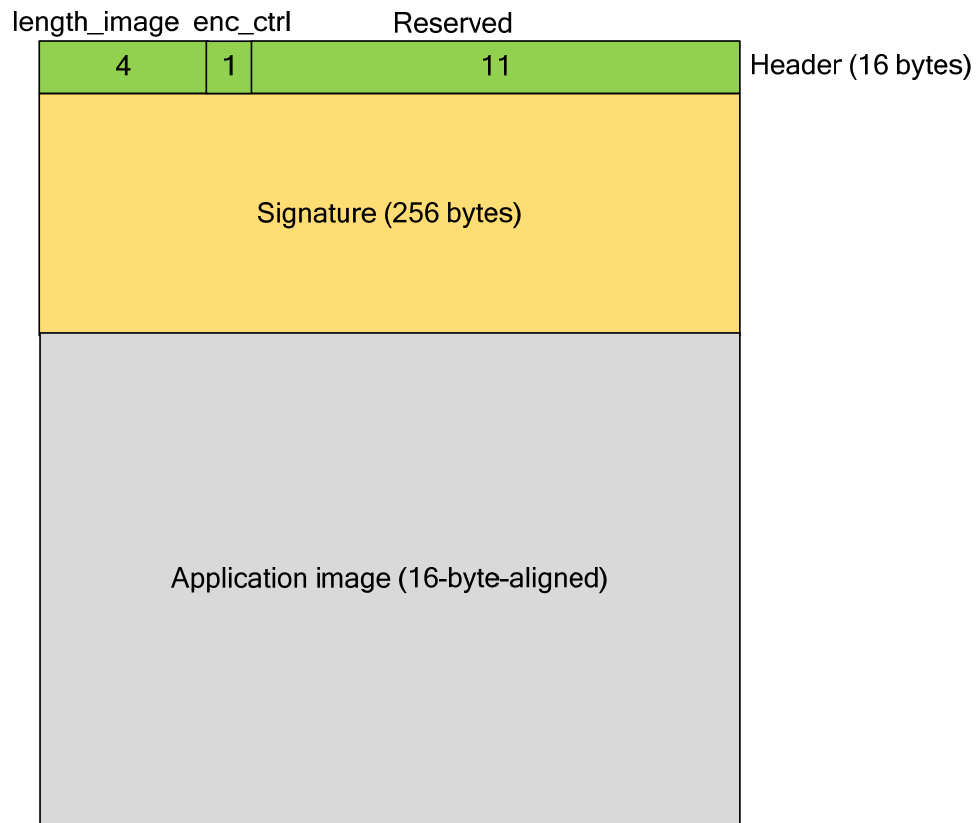


Figure 2-40 shows the structure of the signed image.

Figure 2-40 Structure of the signed image in M2-3rd CA signature mode



The header length is 16 bytes. The data structure is defined as follows:

```
typedef struct
{
    HI_U32 u32ImageLen;
    struct
    {
        HI_U8 enc_flag      :1;
        HI_U8 reserved      :7;
    }enc_ctrl;
    HI_U8 au8Reserved[11];
}VMX_APPLICATION_HEADER_S
```

Members of the structure are as follows:

- **u32ImageLen**: Length of the image. It occupies 4 bytes. The signature value and the length of the header are not included. The image must be 16-byte-aligned.
- **enc_flag**: image encryption flag bit indicating whether the image is encrypted
 - 0: The image is not encrypted.
 - 1: The image is encrypted.
- **au8Reserved[11]**: reserved space

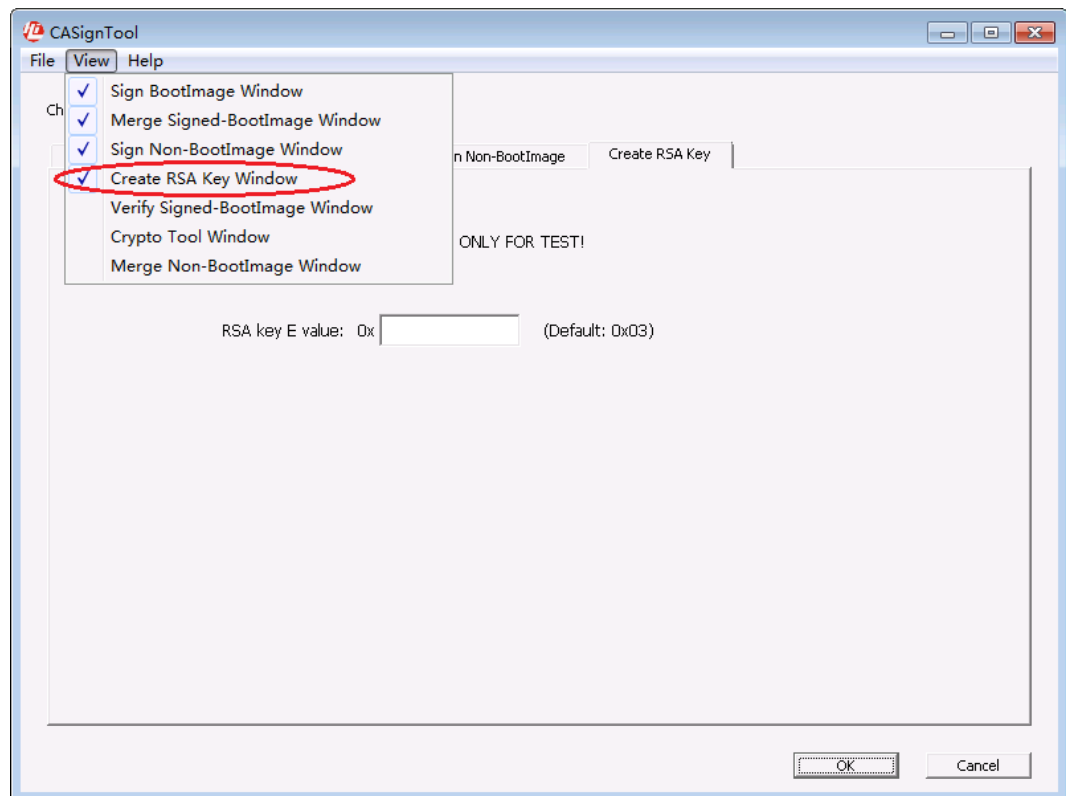


----End

2.5 Generating the RSA Asymmetric Key

The RSA asymmetric key for tests can be generated on the **Create RSA Key** tab page. Before using this function, click **View**, and select **Create RSA Key Window** to display the **Create RSA Key** tab, as shown in [Figure 2-41](#).

Figure 2-41 Selecting Create RSA Key Window



Perform the following steps:

- Step 1** Click the **Create RSA Key** tab.
- Step 2** Set **RSA Key E value** to **0x03** or **0x10001**. 0x03 is recommended.
- Step 3** Click **OK**. The RSA key is generated.



Figure 2-42 Generating the RSA key

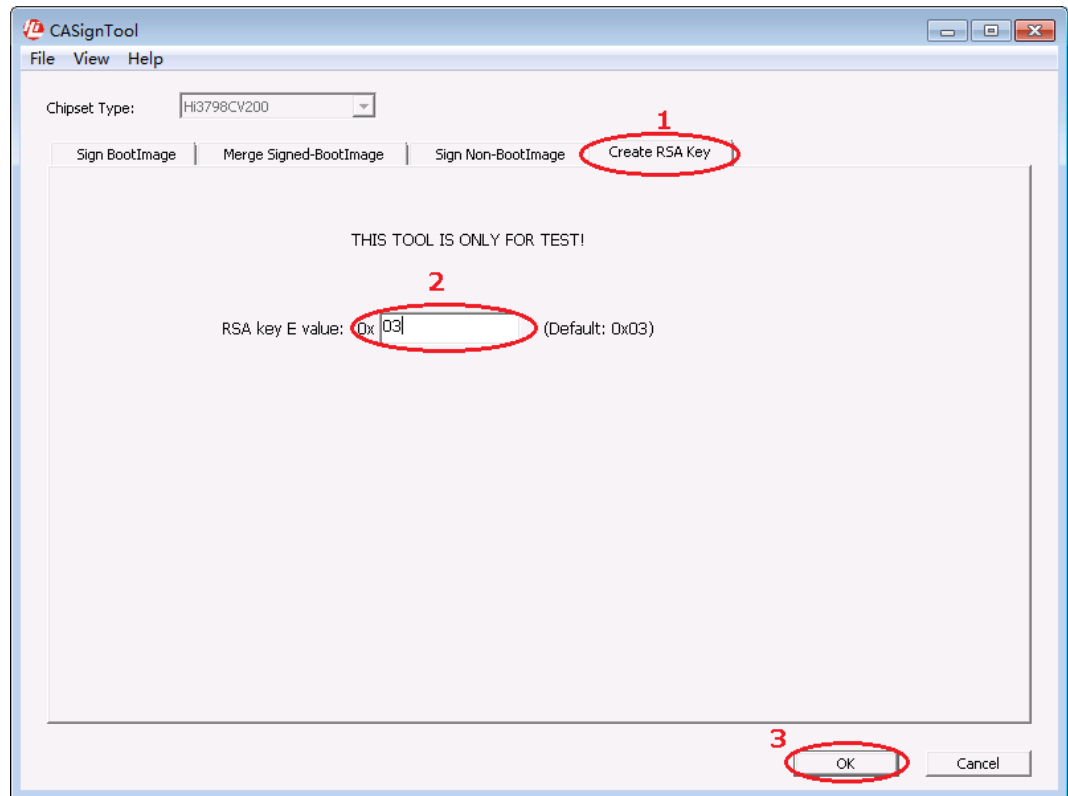


Figure 2-43 Information indicating that the RSA key has been created successfully

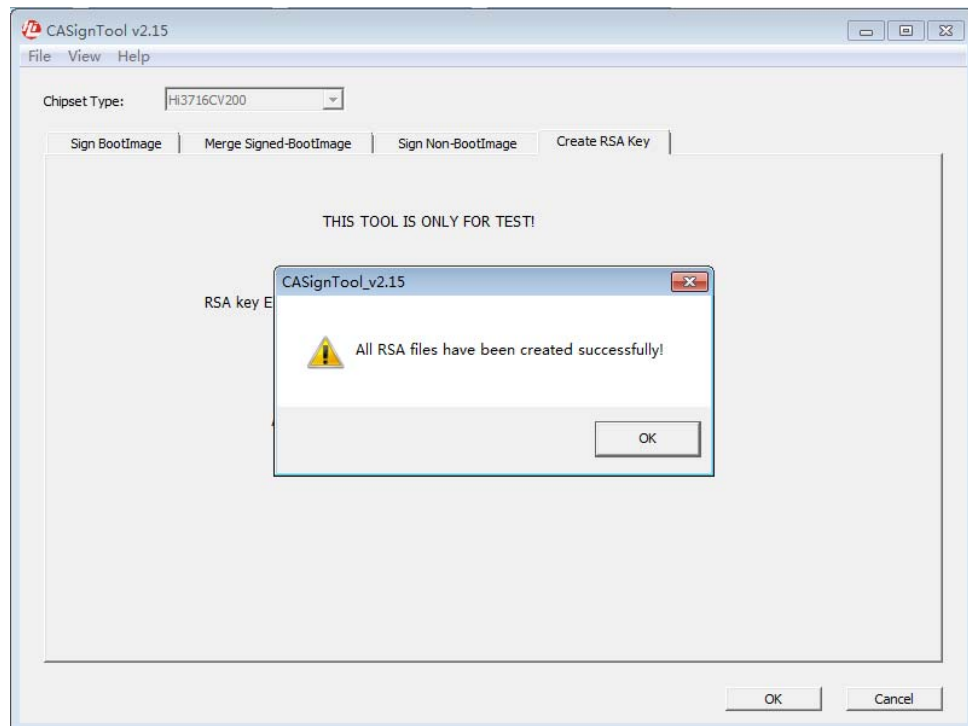









Figure 2-44 shows the generated RSA keys.

Figure 2-44 Generated RSA keys

 rsa_priv.txt	2014/12/10 16:12	Text Document	3 KB
 rsa_pub.bin	2014/12/10 16:12	UltraEdit Docum...	1 KB
 rsa_pub.h	2014/12/10 16:12	C/C++ Header	4 KB
 rsa_pub.txt	2014/12/10 16:12	Text Document	1 KB
 rsa_pub_crc.bin	2014/12/10 16:12	UltraEdit Docum...	1 KB

rsa_priv.txt is the private RSA key, **rsa_pub.txt** and **rsa_pub.bin** are the public RSA key, **rsa_pub.h** is the public RSA key in the C-language array format, and **rsa_pub_crc.bin** is the public RSA key with the CRC32 code.



CAUTION

The RSA keys generated by the CASignTool may be the same sometimes. STB vendors can use the RSA keys generated by the CASignTool in the STB development tests. However, the RAS keys are not recommended for mass production.

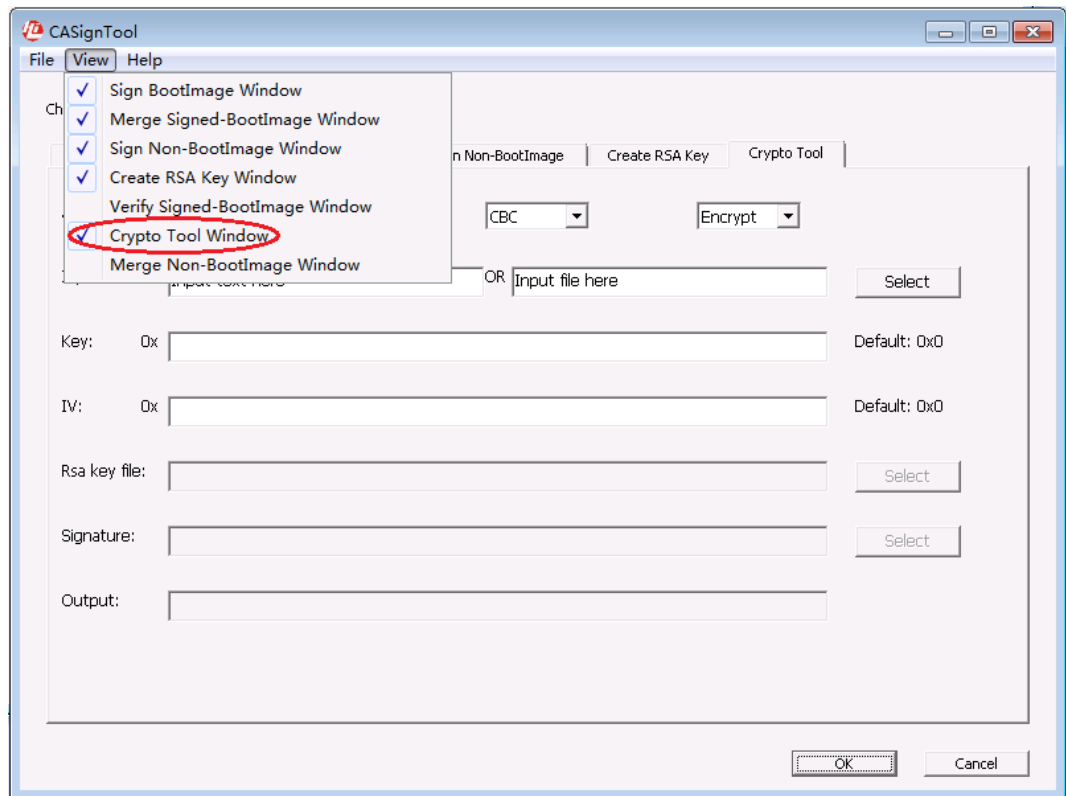
----End

2.6 Encrypting/Decrypting an Image

Images can be encrypted and decrypted on the **Crypto Tool** tab page. Click **View**, and select **Crypto Tool Window** to display the **Crypto Tool** tab, as shown in Figure 2-45.



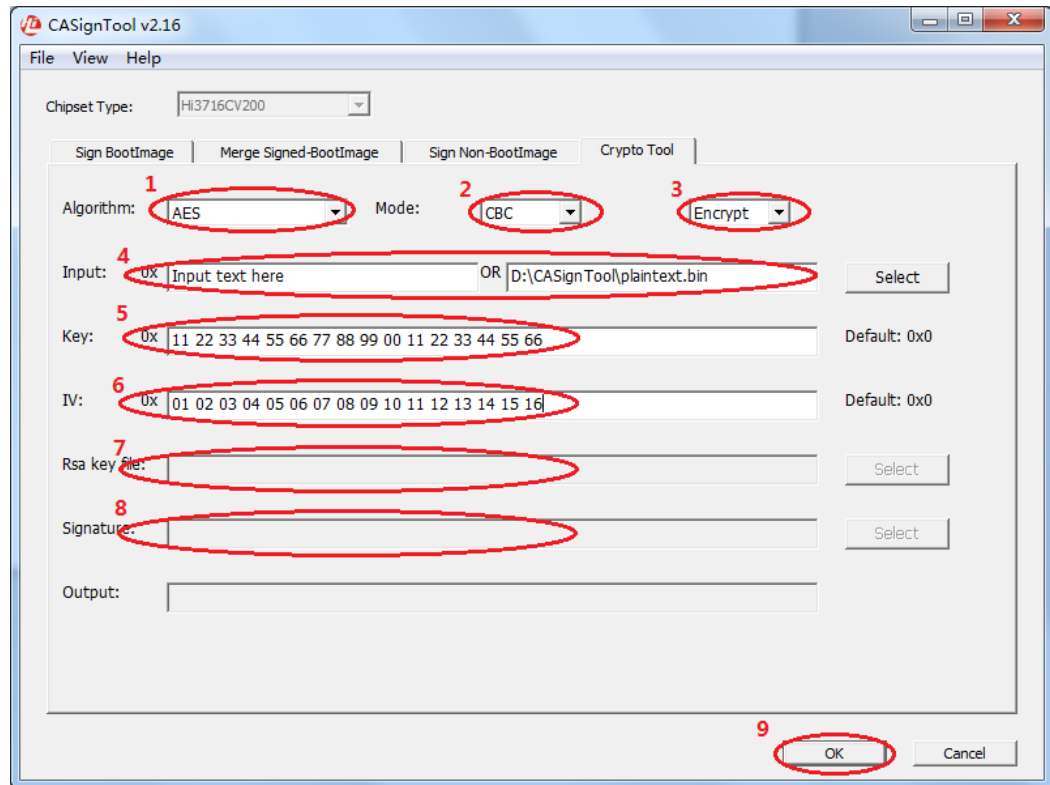
Figure 2-45 Selecting Crypto Tool Window



To encrypt/decrypt an image, perform the following steps:



Figure 2-46 Encrypting/Decrypting images

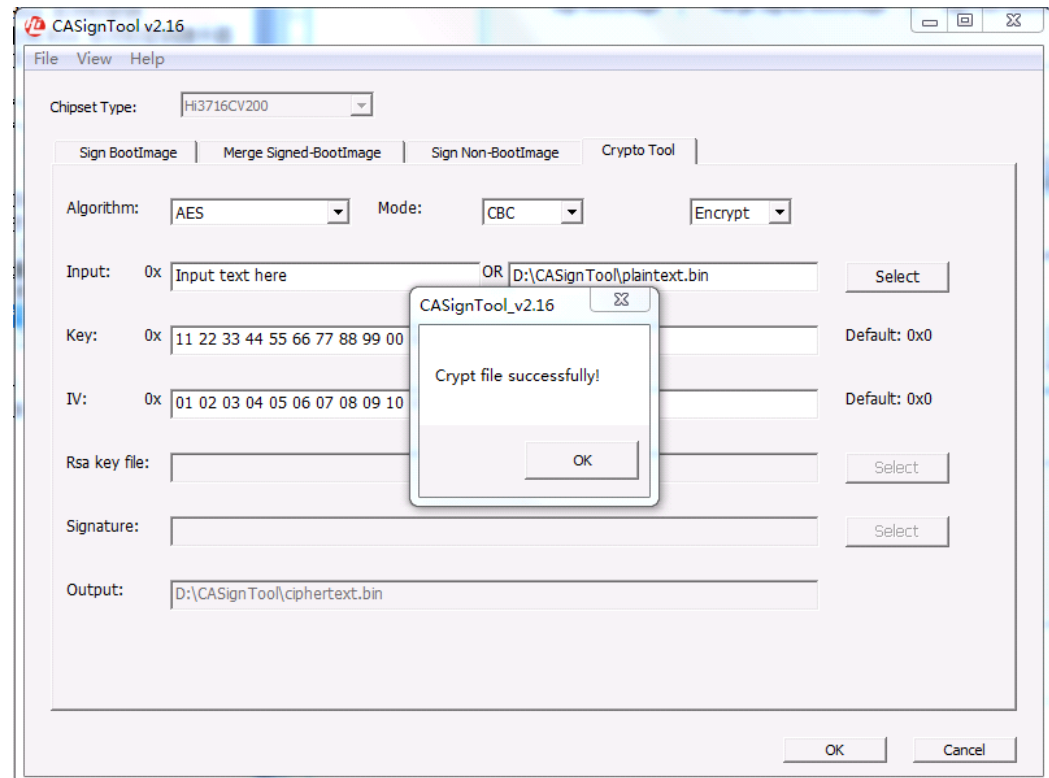


- Step 1** Click the **Crypto Tool** tab.
- Step 2** Select an algorithm. The following algorithms are available: AES, TDES, SHA1, SHA256, HiCRC16, RSA_SIGN, and RSA_VERIFY.
- Step 3** Select a mode, which can be ECB, CBC, or CTR. This option is valid only when the algorithm is set to **AES** or **TDES**. The CTR mode is valid only for the AES algorithm.
- Step 4** Select **Encrypt** or **Decrypt**. This option is valid only when the algorithm is set to **AES** or **TDES**.
- Step 5** Specify the original data to be calculated. You can enter a string of characters or specify a binary file as required.
- Step 6** Enter the key. The input value should be 16-byte hexadecimal numbers. You can separate every two bytes by using a space or not. This option is valid only when the algorithm is set to **AES** or **TDES**.
- Step 7** Enter the IV vector. The input value should be 16-byte hexadecimal numbers. You can separate every two bytes by using a space or not. This option is valid only when the algorithm is set to **AES** or **TDES** and the mode is **CBC** or **CTR**.
- Step 8** Select an RSA key file. This option is valid only when the algorithm is set to **RSA_SIGN** or **RSA_VERIFY**. Select the private RSA key if the algorithm is **RSA_SIGN** and the public RSA key if the algorithm is **RSA_VERIFY**.
- Step 9** Select a signature file. This option is valid only when the algorithm is **RSA_VERIFY**.
- Step 10** Click **OK**.



For the HiCRC16 algorithm, the **Output** text box directly displays the calculation result; for other algorithms, it displays the path of the selected output file.

Figure 2-47 Information indicating that the operation is successful



----End

2.7 Verifying a Signed Boot Image

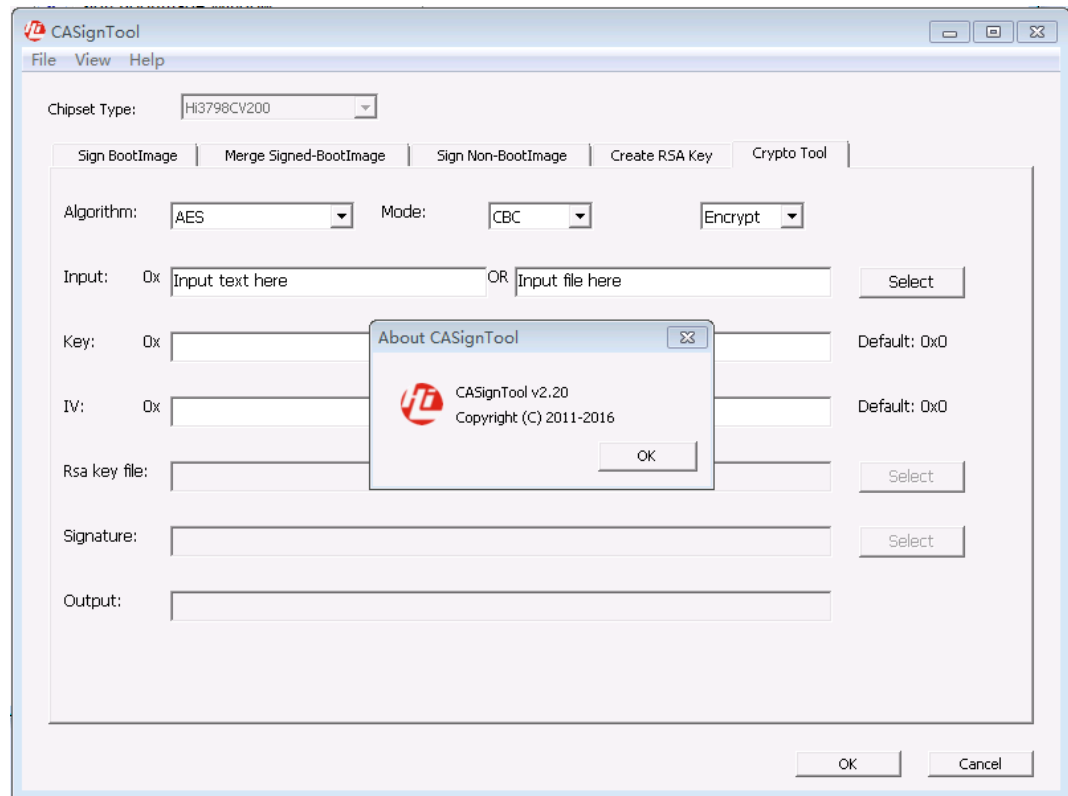
The **Verify Signed-BootImage** tab page is used for internal debugging of HiSilicon and STB vendors do not require this function currently.

2.8 Obtaining the CASignTool Version Number

To obtain the CASignTool version number, choose **Help > About**.



Figure 2-48 Obtaining the CASignTool version number



2.9 Usage of the CASignTool for Linux

Run `./CASignTool` in the **bin** directory of the CASignTool. Then the usage of the CASignTool is displayed, as shown in [Figure 2-49](#).

Figure 2-49 Usage of the CASignTool for Linux

```
Usage: ./CASignTool [flag] [cfgFile] [Optional]
Usage: ./CASignTool -v
Usage: ./CASignTool 0 Signboot_config.cfg
Optional: -k keyDirPath -b bootCFGDirPath -r bootDirPath -o finalbootDirPath
Usage: ./CASignTool 1 merge_config.cfg
Optional: -r bootDirPath -o finalBootDirPath
Usage: ./CASignTool 2 common_config.cfg
Optional: -k keyDirPath -r srcDirPath -o destDirPath
Usage: ./CASignTool 3 special_config.cfg
Optional: -k keyDirPath -r srcDirPath -o destDirPath
Usage: ./CASignTool 4 crypto_config.cfg
Usage: ./CASignTool 5 VMXSignCrypto_config.cfg -K keyFilePath -R bootFilePath -O destFilePath
Usage: ./CASignTool 6 special_crypto_config.cfg
Optional: -k keyDirPath -r srcDirPath -o destDirPath
Usage: ./CASignTool 7 verifyboot.cfg
Usage: ./CASignTool 8 eValue destKeyDirPath or ./CASignTool 8 destKeyDirPath(Default eValue: 3
)
Usage: ./CASignTool 9 RSAPubKeyFilePath RSAPriKeyFilePath AESKeyFilePath DrrInitFilePath UBootFilePa
th OutputFilePath
```



The CASignTool for Linux contains options corresponding to the following functions of the CASignTool for Windows:

- **flag 0**: Generates an unsigned secure BOOT.
- **flag 0**: Generates a signed secure BOOT.
- **flag 0**: Generates a self-signed secure BOOT.
- **flag 1**: Merges images to generate the boot image (**FinalBoot.bin**) required for booting advanced CA chips.
- **flag 2, flag 3, and flag 5**: Signs image files of applications, including the kernel and file systems.
- **flag 8**: Generates the RSA key required for the STB vendor tests.
- **flag 7**: Verifies the signed BOOT (only for simulated verification).
- **flag 4**: Provides the Advanced Encryption Standard (AES) and Triple Data Encryption Standard (3DES) encryption and decryption tools.

The command is in the following format: `./CASignTool [flag] [cfgFile] [Optional]`. The usage of **flag**, **cfgFile**, and **Optional** parameters is as follows:

- **flag**: Fill this parameter according to the preceding function.
- **cfgFile**: Configure this parameter by filling the value required by the .cfg file and then run the command. The CASignTool provides the template of the .cfg file. Fill **cfgFile** as required based on the descriptions of parameters in the template.
- **Optional**: Configure this parameter by filling the optional parameter that can be selected during the execution of commands. The optional parameters vary according to usage methods of the CASignTool. For details, see [Figure 2-49](#). Seven optional parameters are as follows:
 - **-k** specifies the path where the key file is stored. When the path of the input file is also configured in the .cfg file, the path configured by the optional parameter is used.
 - **-r** specifies the path where the image file is stored. When the path of the input file is also configured in the .cfg file, the path configured by the optional parameter is used.
 - **-o** specifies the path where the output file is stored. When the path of the output file is also configured in the .cfg file, the path configured by the optional parameter is used.
 - **-b** specifies the path where the **boot.cfg** excel file is stored. When the path of the input file is also configured in the .cfg file, the path configured by the optional parameter is used.
 - **-K** specifies the path where the key file is stored.
 - **-R** specifies the path where the image file is stored.
 - **-O** specifies the path where the output file is stored.