# Cyber Security Precautions for Hi3861 V100 / Hi3861L V100 Secondary Development

**Issue** 01

**Date** 2020-04-30

### 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 (Shanghai) Technologies Co., Ltd.

Address: New R&D Center, 49 Wuhe Road,
Bantian, Longgang District,
Shenzhen 518129 P. R. China

Website: https://www.hisilicon.com/en/

Email: support@hisilicon.com

# About This Document

## Purpose

The Hi3861 V100/Hi3861L V100 chip solution delivery package contains the chip documentation, hardware documentation, software development kit (SDK), software reference design, and software documentation. You can customize various products based on this delivery package.

This document analyzes possible cyber security threats related to the SDK software package during the use of the products developed based on this delivery package as well as providing corresponding solutions.

## Related Versions

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

| Product Name | Version |
| --- | --- |
| Hi3861 | V100 |
| Hi3861L | V100 |

## Intended Audience

The document is intended for:

- Technical support engineers
- Software development engineers

## Symbol Conventions

The following table describes the symbols that may be found in this document.

| Symbol | Description |
| --- | --- |
| DANGER | Indicates a hazard with a high level of risk which, if not avoided, will result in death or serious injury. |
| WARNING | Indicates a hazard with a medium level of risk which, if not avoided, could result in death or serious injury. |
| CAUTION | Indicates a hazard with a low level of risk which, if not avoided, could result in minor or moderate injury. |
| NOTICE | Indicates a potentially hazardous situation which, if not avoided, could result in equipment damage, data loss, performance deterioration, or unanticipated results.<br>NOTICE is used to address practices not related to personal injury. |
| NOTE | Supplements the important information in the main text.<br>NOTE is used to address information not related to personal injury, equipment damage, and environment deterioration. |

# Change History

| Issue | Date | Change Description |
| --- | --- | --- |
| 01 | 2020-04-30 | This issue is the first official release.<br>• Added **1.5 Secure Storage of Key Data**.<br>• Added a NOTICE about the number of iterations for the KDF algorithm in **1.6.1 Cipher Driver**. |
| 00B02 | 2020-04-09 | • Added the description of firmware code security in **1.1 Security Architecture**.<br>• Added the purposes of the steps for writing the eFUSE, description of eFUSE items and lock bits, and the method of burning the eFUSE in the production line by referring to the user guide in **1.3.1 Boot Methods**.<br>• Added **1.4 Flash Encryption**.<br>• Added the reference to *Hi3861 V100/ Hi3861L V100 API Development Reference* in **1.6.1 Cipher Driver**.<br>• Added **1.7.3 Maintainability and Testability Precautions**. |
| 00B01 | 2020-01-15 | This issue is the first draft release. |

# Contents

# 1 Product Security Solution



## 1.1 Security Architecture

The product cyber security is systematic engineering that involves all aspects of the product.

Possible threats to the Hi3861/Hi3861L version include:

- Boot security

  This part mainly involves the verification mechanism for all levels of images during the system boot. Hi3861/Hi3861L provides a secure boot solution in which a ROMBoot program is solidified as level-1 boot in ROM. During the boot, ROMBoot verifies the digital signature of FlashBoot and then verifies the digital signature of the kernel in FlashBoot to ensure the security of running the program.
- System upgrade security

  The threats faced by the system upgrade solution are the legitimacy and integrity of upgrade files. Both FlashBoot and firmware can be upgraded. An upgrade file is verified using a digital signature. Both FlashBoot and firmware allow version anti-rollback. The version number is authenticated with the tee_boot and tee_firmware versions in the eFUSE. If the authentication fails, the system cannot be started.
- Firmware code security

The flash encryption function supports the two-level key encryption architecture, providing encryption protection to prevent key code of the flash memory from being read and decompiled.

- JTAG security debugging

  The JTAG debugging function is enabled by default. You are advised to burn the JTM field in the eFUSE to **1** to disable the JTAG debugging function before mass production.

# 1.2 Device Security

To ensure security, you are advised to perform the following operations:

- Enable secure boot.
- Permanently disable JTAG debugging.

# 1.3 Secure Boot

## 1.3.1 Boot Methods

Hi3861/Hi3861L supports secure boot, which can be implemented in the following methods:

- Method 1: FlashBoot uses RSA/ECC for signing. ROMBoot verifies the validity of the signature based on the root key hash in the eFUSE and the signature data of FlashBoot, and then starts FashBoot.
- Method 2: FlashBoot uses RSA/ECC for signing, and the code segments are encrypted in AES-CBC mode. ROMBoot decrypts the code segments, verifies the RSA/ECC signature, and then starts FlashBoot.

For method 1, you need to configure the hash value of the root public key in the eFUSE and enable secure boot. Perform the following steps:

**Step 1** Write the SHA256 value of the RSA root public key to **root_pubkey** in the eFUSE and lock this area.

**Step 2** Write **0xFF** to the secure boot flag of the eFUSE to enable the secure boot function and lock this area.

**----End**

For method 2, you need to configure the hash value of the root public key and the hardware unique key (HUK) of the encryption key in the eFUSE, and enable secure boot and the FlashBoot encryption flag. Perform the following steps:

**Step 1** Write the SHA256 value of the RSA root public key to **root_pubkey** in the eFUSE and lock this area.

**Step 2** Write **0xFF** to the secure boot flag of the eFUSE to enable the secure boot function and lock this area.

**Step 3** Write the 32-byte HUK code to the root_key of the eFUSE and lock this area.

**Step 4** Write **0xFF** to Encrypt Flag of the eFUSE to enable the flash boot encryption flag and lock this area.

**----End**

The secure boot feature requires digital signatures for the FlashBoot, kernel, and upgrade files. You are advised to use method 2, which has higher security performance.

NOTE

- For details about the eFUSE items and lock bits, see the *Hi3861 V100/Hi3861L V100 eFUSE User Guide*.
- For details about how to burn the eFUSE in the production line, see the *Hi3861 V100/Hi3861L V100 Production Line Equipment Test User Guide*.

## 1.3.2 Solution Description

- The RSA/ECC root public key is stored in plaintext in the FlashBoot file header structure.
- The SHA256 value of the RSA/ECC root public key is stored in the eFUSE.
- The RSA/ECC level-2 public key is signed using the root private key.
- FlashBoot is signed by using the RSA/ECC level-2 private key.
- The user root public key is preset in FlashBoot to verify the validity of the lower-level key. When FlashBoot starts the lower-level program, it needs to verify the signature of the program.
- The root key is in RSA4096 or ECDH_BRAIN_POOL_P256R1 format, and the level-2 key is in RSA2048 or ECDH_BRAIN_POOL_P256R1 format. The root key cannot be changed after being confirmed, nor can the key preset in FlashBoot. Other keys can be changed as required.
- The encryption key is derived from the KDF algorithm.
- The IV value of the encryption key is stored in the level-2 key structure of the FlashBoot file.

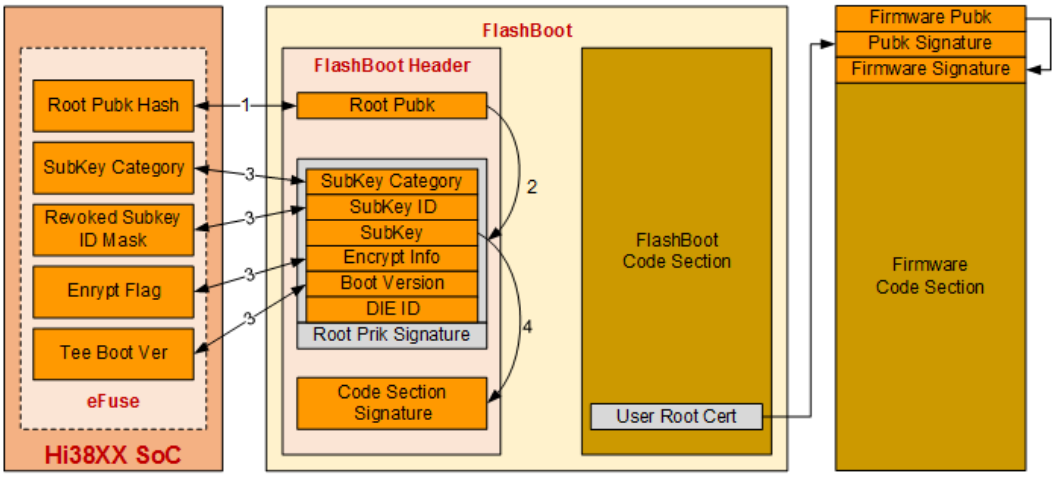
NOTE

The root key and level-2 key must be signed in the same mode. The RSA and ECC modes must not be used together.

### 1.3.2.1 Boot Process

**Figure 1-1** shows the secure boot process.

**Figure 1-1** Secure boot process




The secure boot process is as follows:

**Step 1** Check whether the secure boot mode is used (secure boot is recommended). In non-secure boot mode (not recommended), calculate the hash value of FlashBoot from the start to the end of the code segment and compare it with the hash value at the end of FlashBoot. If they are the same, start FlashBoot directly. If the secure boot mode is used, go to **Step 2**.

**Step 2** Verify the root public key of the FlashBoot header using the HASH value of the root public key stored in the eFUSE.

**Step 3** Verify the level-2 public key (subkey) of the FlashBoot header by using the root public key of the FlashBoot header.

**Step 4** Verify that the category of the level-2 public key matches the category stored in the eFUSE.

**Step 5** Check whether the ID of the level-2 public key is within the range of [0, 23].

**Step 6** Check whether the ID of the level-2 public key is revoked by comparing it with the RSIM item in the eFUSE. Revoked Subkey ID Mask (RSIM) is a 24-bit bitmap that indicates the status of each level-2 public key. If **Subkey ID** of the RSIM is **1** (((1<<Subkey_ID)&RSIM)==1), the public key is revoked and the identity authentication fails.

**Step 7** If neither **Encrypt Flag** in the eFUSE nor that in the level-2 key is **0x42**, the code segment is decrypted. The decryption key is derived from the HUK and salt in the eFUSE through the KDF. The salt is 32 bytes. The first 16 bytes are in the level-2 key structure, and the last 16 bytes are hard-coded in ROMBoot.

**Step 8** Verify the FlashBoot version to prevent version rollback. Check whether the version number is within the range of [0, 16], and then verify the identity based on the tee_boot version in the eFUSE. The algorithms are as follows:

- If boot_ver==0 and tee_boot_ver==0, the verification is successful.
- If boot_ver > 0 and (tee_boot_ver>> boot_ver–1)) ==1, the verification is successful.

**Step 9** Verify the digital signature of the firmware public key using the user public key preset in FlashBoot.

**Step 10** Verify the digital signature of the firmware using the firmware public key.

**Step 11** Start the firmware.

**----End**

The maintenance and debugging process of the board is as follows:

**Step 1** Provide the die ID information for the customer, who then transfers the die ID in the FlashBoot signing.

**Step 2** During the boot, ROMBoot verifies the die ID after determining that the FlashBoot version is a maintenance version. If the verification passes, FlashBoot is started.

**----End**

### 1.3.2.2 Level-2 Key ID

The ID of a level-2 key ranges from 0 to 23.

## 1.4 Flash Encryption

The flash encryption function uses the two-level key encryption architecture to provide encryption protection for key code in the flash memory. The working principle is as follows:

- eFUSE item **FLASH_CRYPT_CFG**: 1 bit. This bit indicates whether to enable flash encryption (**0**: no; **1**: yes).
- eFUSE item **FLASH_CRYPT_CNT**: 12 bits. When the even bits {0, 2, 4, 6, 8, 10, 12} are set to **1**, the flash memory is not encrypted. When the odd bits {1, 3, 5, 7, 9, 11} are set to **1**, the flash memory is encrypted. The default value is **0**.
- HiBurn checks whether **FLASH_CRYPT_CFG** is enabled after burning the flash memory. If this function is enabled, the least significant even bit that is not **0** in **FLASH_CRYPT_CNT** is written as **1**, indicating that the flash memory is not encrypted.
- After the system starts, the system checks whether the most significant bit whose value is **1** in **FLASH_CRYPT_CNT** is an even bit. If the number of bits is an even number, the system boots after the flash memory is encrypted. If the number of bits is an odd number, the system boots after the flash memory is decrypted.

NOTE

- The flash encryption function can effectively prevent key code from being read and stolen after decompilation. You are advised to enable this function in mass production products.
- For details about the code segment encryption of the flash memory and how to use the function, see the *Hi3861 V100/Hi3861L V100 Cipher Module User Guide*.

**NOTICE**

eFUSE burning is irreversible due to the one-time programmable (OTP) feature of the eFUSE. However, **FLASH_ENCPT_CNT** of the eFUSE in Hi3861 has only 12 bits. Therefore, after this function is enabled, the flash memory can be burnt only six times.

# 1.5 Secure Storage of Key Data

When confidential information such as the account and password is recorded, the data storage security must be ensured. Therefore, the data must be encrypted before being stored. For details about the secure storage solution for key user data, see the *Hi3861 V100/Hi3861L V100 Cipher Module User Guide*.

# 1.6 Driver Security Precautions

## 1.6.1 Cipher Driver

A cipher driver realizes standard symmetric encryption AES, asymmetric encryption RSA, ECDH, digest algorithms SHA256 and HMAC, and key derivation algorithm KDF without using any private algorithms. Note that the longer the cipher key is, the higher the security level is. Therefore, you are advised to use the AES 128-bit or longer key and RSA 2048-bit or longer key. For details, see the *Hi3861 V100/Hi3861L V100 API Development Reference*.

**NOTICE**

If the number of iterations is less than 1000, the KDF algorithm may be cracked. Therefore, you are advised to set the number of iterations to a value greater than or equal to 1000 when using the KDF algorithm to derive keys.

## 1.6.2 Serial Port

A serial port is a serial communication interface. There are debugging serial ports and service serial ports.

- Debugging serial ports are based on RS232, mainly used for near-end bottom-layer equipment debugging.
- Service serial ports are based on RS232, mainly used for transmitting and receiving service packets.

  You are advised to add security authentication and packet encryption mechanisms if conditions permit.

If no serial port is used, the eFUSE can be configured at factory (UTMx set to **1**) to permanently disable the serial port function.

NOTICE

The serial port function cannot be enabled again once it is permanently disabled.

# 1.7 Other Precautions

## 1.7.1 JTAG Interface

Through the JTAG interface, malicious attackers can tamper with any configurations of the system and maliciously destroy the system. Therefore, you are advised to configure eFuse (JTM set to **1**) and permanently disable JTAG at factory.

## 1.7.2 Code Security

Cyber security threats triggered by code errors are usually derived from basic code specification issues, such as pointer access violation, array access violation, and unchecked input arguments. The following measures are recommended:

- Perform full scan with a code health scanning tool commonly used in the industry.
- Perform full fuzz testing of all APIs (including device driver interfaces) with a fuzz testing tool.
- Scan the open-source software in use with a vulnerability scanning tool commonly used in the industry.

## 1.7.3 Maintainability and Testability Precautions

- The maintainability and testability solution can be enabled only during debugging. You are advised to disable it in the release version.
- Currently, the HSO debugging tool is used only to print logs during SDK fault locating.
- For details about how to use the maintainability and testability APIs, see the *Hi3861 V100/Hi3861L V100 SDK Development Guide*.

# 2 Conclusions

You need to take necessary measures for Hi3861/Hi3861L based on the security threat analysis. The following security principles are for reference:

- Appropriate security

  Security design is based on the analysis of specific security risk scenarios. Considering the impact of performance, costs, and services, security design should adopt the most appropriate security measures.

- Minimum authority

  The minimum permissions and resources are assigned to users, maintenance personnel, network units, programs, and processes based on responsibilities. This reduces potential security risks.

- Active collaborative defense

  Malicious attack sources should be identified in a timely manner and the connection between malicious users and the network must be automatically torn down before the attacks cause significant harms. Alternatively, the bandwidth and service quality of the connection can be reduced to minimize the negative impact.

- Deep defense

  Defense-in-depth involves the use of multiple layers of defense against threats. For example, when a defense layer is not enough to defend threats, another defense layer will be enabled to prevent worse damage.