RT-THREAD OTA User Manual

RT-THREAD Document Center

WWW.RT-THREAD.ORG

Friday 28th September, 2018

Versions and Revisions

Date	Version	Author	Note
2018-06-21	v0.1	MurphyZhao	Initial version

Machine Translated by Google

Table of contents

Chapter 1

rt_ota Introduction

rt_ota is a cross-OS and cross-chip platform firmware over-the-air upgrade technology developed by RT-Thread.

Over-the-Air Technology) to easily manage, upgrade and maintain device-side firmware.

The OTA firmware upgrade technology provided by RT-Thread has the following advantages:

- Firmware tamper-proof: Automatically detect firmware signatures to ensure firmware security and reliability
- Firmware encryption: Supports AES-256 encryption algorithm to improve firmware download and storage security
- Firmware compression: efficient compression algorithm, reduce the firmware size, reduce Flash space occupation, save transmission traffic, reduce

  Download time
- Differential upgrade: Generate differential packages based on version differences, further saving Flash space, saving transmission traffic, and speeding up

  Upgrade speed
- Power failure protection: protection after power failure, and continue to upgrade after restart
- Intelligent restore: When the firmware is damaged, it will automatically restore to the factory firmware to improve reliability
- Highly portable: can be used across OS, chip platforms, and Flash models, and does not rely on a specific OTA server

1.1 File Directory Structure

```
rt_ota
ÿ README.md ÿ                                  // Software package instructions
SConscript ÿÿÿÿdocs                            //RT-Thread default build script

ÿ ÿ api.md ÿ ÿ                                 // API usage instructions
introduction.md ÿ ÿ port.md ÿ ÿÿÿÿuser-        // Software package details
guide.md ÿÿÿÿinc                               // Porting documentation
ÿÿÿÿlibs ÿÿÿÿports                             // User Manual
                                               // Header file
                                               // Library file
                                               // Migrate files
```

```
    ÿÿÿÿtemp
            rt_ota_key_port.c            // Migrate file template
ÿ ÿ ÿÿÿÿsamples                          // Example code
ÿ ÿÿÿÿota.c                              // Software package application sample code
ÿÿÿÿtools                                // Tools
        fatfs_ota_packaging_tool         // fatfs file system OTA packaging tool
        firmware_ota_packaging_tool // OTA file packaging tool (rbl file)
```

1.2 rt_ota software framework diagram

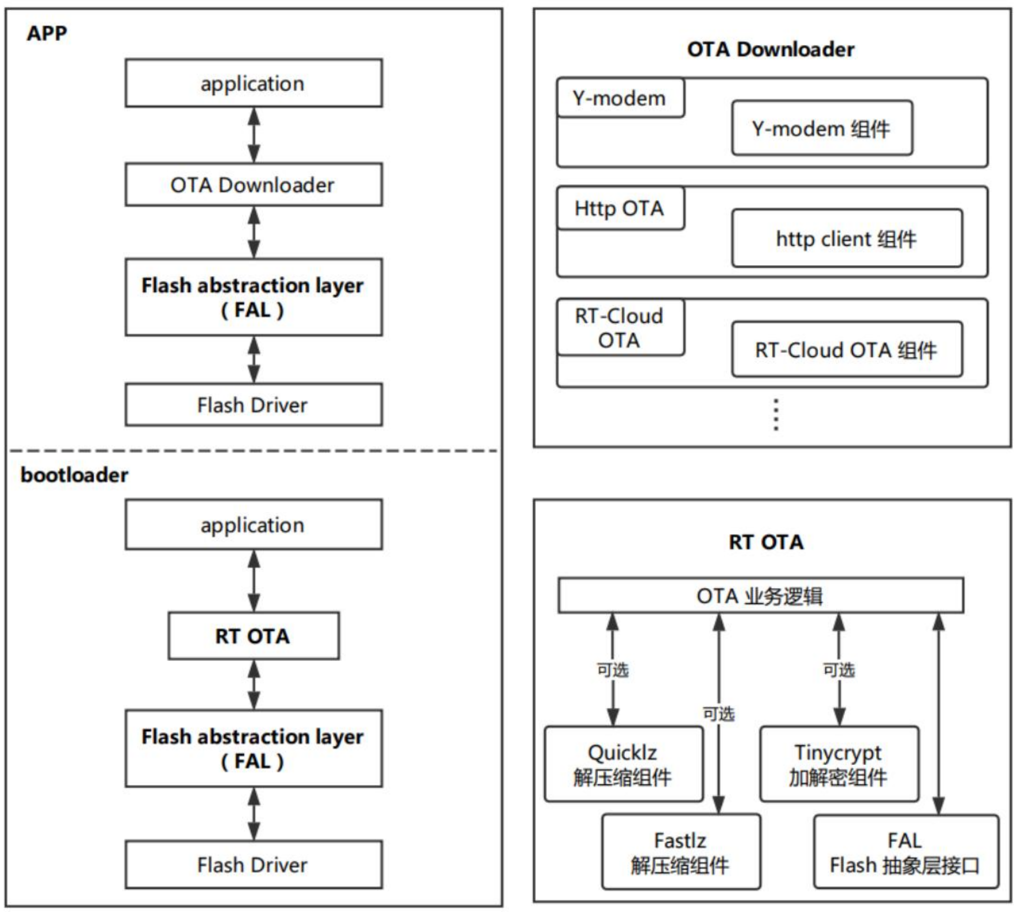

Figure 1.1: *RT OTA* Software framework diagram

As shown in the figure above, the application framework diagram shows the position of rt_ota in the entire OTA application, as well as the Related software component packages involved in the application.

As can be seen from the **rt_ota** software framework diagram, the APP part of the software does not need to rely on the rt_ota software package. Because the APP part only needs to worry about how to download the upgraded firmware from the OTA server to the device, and it involves system security rt_ota is only required for firmware verification and firmware transfer to ensure stability.

The OTA Downloader is a client program that corresponds to an OTA server and is used to download OTA firmware from the OTA server to the device. Common and universal **OTA downloaders** include Y-modem (serial port upgrade) and HTTP OTA (network upgrade). Developers can build an OTA upgrade server using their own computers. OTA servers provided by private or public cloud platforms typically require the development of corresponding client programs that run on the device to download OTA firmware.

1.3 rt_ota Features

1.3.1 Encryption

Why choose encryption?

- Unencrypted firmware can be stolen and used by anyone in any way, and may also be tampered with or attacked. Risks such as attacks and product counterfeiting
- The OTA services used by customers are mostly third-party services, and the customer's firmware needs to be uploaded to the third-party server, or The firmware can be easily leaked, spread, or used maliciously by third parties.

To avoid various problems with unencrypted firmware, rt_ota uses AES256 encryption for the firmware.

> AES (Advanced Encryption Standard) is a block encryption standard adopted by the U.S. federal government and is also the de facto industrial standard for block ciphers.

rt_ota uses **TinyCrypt** The AES256 encryption algorithm implemented in the software package has fast decryption speed and small resource usage.

Without optimization, TinyCrypt It occupies 5244 bytes of ROM and 8744 bytes of RAM.

1.3.2 Compression

Why support compression?

> The Flash resources of embedded devices are often limited (usually only 2M bytes). In the limited Flash, it is usually necessary to store information such as bootloader, application (app), OTA firmware, system and user parameter configuration, which makes the available application code space very small.

In order to solve the problem of limited Flash resources, RT-Thread OTA introduces an efficient compression algorithm to reduce the The Flash space occupied by the software.

Currently RT-Thread perfectly supports Quicklz, Fastlz and MiniLZO decompression algorithms, and is available in the rt_ota group. The package supports the use of Quicklz and Fastlz.

The following table compares the three compression algorithms in terms of compression rate and resource usage: (Not an accurate test, for reference only)

name	copyright	ROM	When decompressing RAM	Compression Level	Compression Ratio
quicklz	GPL	1838	9732	3	67%
fastlz	WITH	3096	9696	2	74%
miniLZO	GPL	2024	9604	LZO1X_1	75%

1.3.3 Anti-tampering

OTA firmware is usually exposed to the Internet. If the firmware is not encrypted and tamper-proof, it will face the following risks: question:

- OTA firmware is stored on a third-party OTA server and is not trusted.
- The OTA firmware upgrade download process may be intercepted and maliciously tampered with, which is unsafe.
- OTA firmware may be illegally obtained, cracked, and the product may be counterfeited

To ensure the security of customer firmware and the reliability of OTA upgrades, RT-Thread OTA integrates tamper protection by default. Improved functions, fast inspection speed and strong reliability.

1.3.4 Differential Upgrade

Differential upgrade is to package the differences between the device firmware and the new version of the firmware into differential packages in a predetermined format and then upgrade A level of technology.

The commonly used differential upgrade method in embedded devices is multi- bin upgrade , which effectively reduces the complexity of differential upgrade. Degrees.

Multi- **bin** upgrade usually divides an application into different parts and generates multiple bin files. The compilers are linked to different locations of the Flash respectively, and each upgrade only upgrades one of the bin files.

Compared with the full package upgrade, the differential upgrade has the following advantages:

- Differential packets are relatively small, and traffic costs are low
- Fast download and upgrade speed, short upgrade time
- Low network requirements, suitable for LoRa and NBiot application scenarios
- Effectively reduce power consumption

1.3.5 Power failure protection

The power-off protection function is mainly used in the scenario where the device suddenly loses power during the OTA upgrade process. Without the protection function, the device may be bricked and returned to the factory because only part of the firmware has been upgraded.

The power-off protection function of the RT-Thread OTA security protection mechanism ensures that even if an abnormality occurs during the device upgrade process, If the upgrade is interrupted, the device will continue to upgrade next time it is powered on, and the firmware will not be damaged or the device will become bricked.

1.3.6 Intelligent Restore

The device may become abnormal due to external attacks, interruption of the upgrade process or other reasons. In this case, the intelligent restore function of RT-Thread OTA security protection mechanism can also intelligently restore the device firmware. software, thereby effectively ensuring the correct and stable operation of the device program.

Chapter **2**

rt_ota Sample Application

2.1 Example Introduction

Example file:

samples/ota.c

This example is an example of the **rt_ota** software package, mainly showing how users can **quickly** build
Build your own OTA application and demonstrate the basic OTA workflow.

This routine file can be applied to the user's Bootloader project, and the OTA process can also be customized based on this routine to suit the user's solution.

Chapter **3**

How **OTA** works

OTA upgrades are essentially IAP (In-App Programming). In embedded device OTA, the upgrade data package is typically downloaded to Flash memory via a serial port or network. The downloaded data package is then moved to the MCU's code execution area for overwriting, completing the device firmware upgrade.

OTA upgrades for embedded devices are generally not based on the file system, but rather on dividing the Flash into different functional areas. The OTA upgrade function can be completed in the region.

In embedded systems, completing an OTA firmware remote upgrade typically involves the following three core stages:

1. Upload the new firmware to the OTA server 2. The device downloads the new OTA firmware 3. The bootloader verifies, decrypts, and moves the OTA firmware (moving it to the executable program area)

The detailed OTA upgrade process is shown in the figure below:

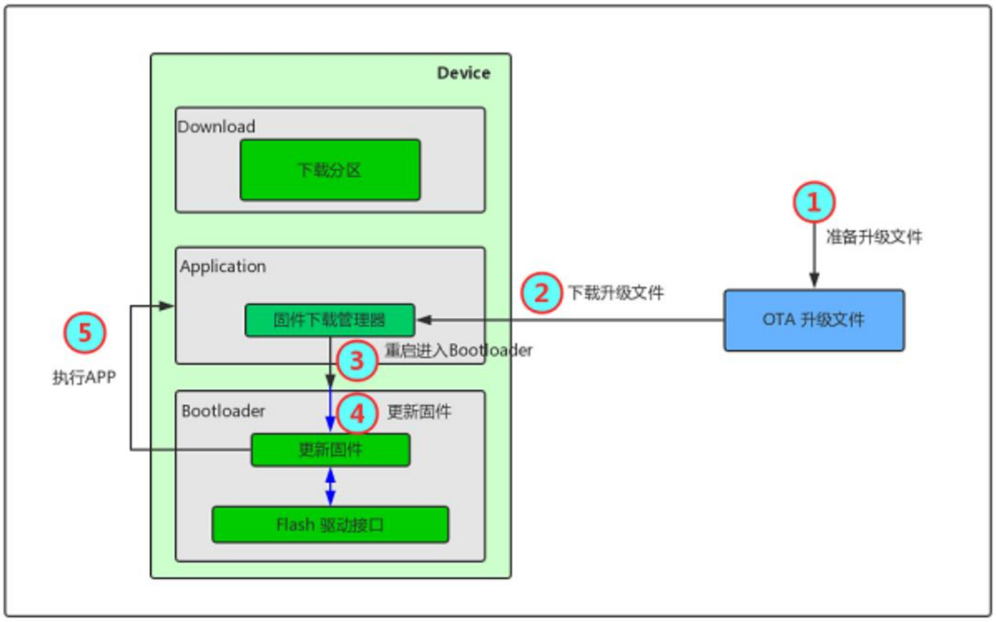

Figure 3.1: *OTA* Upgrade Process

Chapter 4

rt_ota Usage Instructions

4.1 Preparation before use

4.1.1 Downloading and porting dependent software packages

FAL (required)

FAL package download:

```
git clone https://github.com/RT-Thread-packages/fal.git
```

For FAL package porting, refer to FAL README.

Quicklz or **Fastlz (optional)**

Quicklz and Fastlz are decompression packages supported by rt_ota, and users can choose to use either one of them.

Quicklz package download:

```
git clone https://github.com/RT-Thread-packages/quicklz.git
```

To enable compression in OTA and use Quicklz, define the following macros in the **rtconfig.h** file:

```
#define RT_OTA_USING_CMPRS                           // Enable decompression
#define RT_OTA_CMPRS_ALGO_USING_QUICKLZ // Use Quicklz // Define
#define QLZ_COMPRESSION_LEVEL 3                      using Quicklz level 3 compression
```

Fastlz package download:

git clone https://github.com/RT-Thread-packages/fastlz.git

To enable compression in OTA and use Quicklz, define the following macros in the **rtconfig.h** file:

```
#define RT_OTA_USING_CMPRS #define              // Enable decompression
RT_OTA_CMPRS_ALGO_USING_FASTLZ                  // Using Fastlz
```

TinyCrypt (optional)

TinyCrypt is a software package used in rt_ota for firmware encryption, supporting AES256 encryption and decryption.

TinyCrypt package download:

git clone https://github.com/RT-Thread-packages/tinycrypt.git

To enable compression in OTA and use TinyCrypt, define the following macros in the **rtconfig.h** file:

```
#define RT_OTA_USING_CRYPT #define              // Enable the Tinycrypt component package
TINY_CRYPT_AES #define                          // Enable AES functionality
RT_OTA_CRYPT_ALGO_USING_AES256 // Enable AES256 encryption
```

4.1.2 Downloading and Porting the rt_ota Software Package

rt_ota is a closed source package, please contact **RT-Thread** Obtain usage rights.

If you have obtained the right to use **rt_ota** and downloaded the **rt_ota** software package, please read the To complete the porting work, refer to the **rt_ota** porting documentation.

4.1.3 Defining Configuration Parameters

The configuration macros described in the Dependency Package Download and Porting section need to be defined in the **rtconfig.h** file. The file is as follows: (Developers configure relevant macro definitions according to their own needs)

```
#define PKG_USING_RT_OTA #define                // Enable RT_OTA component package
RT_OTA_USING_CRYPT #define                      // Enable the Tinycrypt component package
TINY_CRYPT_AES #define                          // Enable AES functionality
RT_OTA_CRYPT_SOMETHING_USING_AES256 // Enter AES256 as well
#define RT_OTA_USING_CMPRS #define              // Enable decompression
RT_OTA_CMPRS_ALGO_USING_QUICKLZ // ÿ ÿ Quicklz
#define QLZ_COMPRESSION_LEVEL 3                 // Define the use of Quicklz level 3 compression
```

```
#define FAL_PART_HAS_TABLE_CFG                    // Enable the partition table configuration file (do not enable
        To find in Flash)
```

4.2 Developing the bootloader

The **rt_ota** software package completes the work of firmware verification, authentication, and transfer, and needs to be used in conjunction with BootLoader. Therefore, after obtaining the **rt_ota** software package, users need to develop the BootLoader program according to their own needs.

1. Developers first need to create a BootLoader project for the target platform (can be a bare metal project). 2. Copy the **rt_ota** software package to the BootLoader project directory. 3. Copy the **FAL** software package to the BootLoader project directory and complete the porting work. Refer to the FAL README. 4. Copy the **Quicklz** or **Fastlz** software package to the BootLoader project directory (if the decompression function is required). 5. Copy the **TinyCrypt** software package to the BootLoader project directory (if the encryption function is required). 6. Copy the **rtconfig.h** file in the Defining Configuration Parameters section to the BootLoader project. 7. Develop the specific business logic of OTA. Refer to the bootloader & OTA overall flow chart (see the reference section for details) and the sample documentation.

4.3 Developing the App

The main task to be completed in the APP is to download the OTA upgrade file to the device's Flash.

1. Create an RT-Thread application project. 2. Use the RT-Thread package manager to open the FAL component package and complete the porting. Refer to the FAL README. (The ported code can be the same as the one in the Bootloader)
3. Select an OTA Downloader (RT-Thread package management tool provides Y-modem and HTTP OTA)
   - Ymodem
   - HTTP OTA
   - Others (need to develop OTA firmware download client program by yourself)
4. Develop application business logic 5. Modify link script configuration

Normally, our programs start running from the start address of the Flash code area. However, the space starting from the start address of the Flash code area is occupied by the bootloader program, so we need to modify the link script to allow the application program to start from the start address of the Flash application area.

Generally, we only need to modify the starting address of the Flash and SECTION segments in the link script to the starting address of the application partition. The application partition information must be completely consistent with the Flash partition table of the corresponding MCU platform.

Taking the GCC link script as an example, the modification example is shown in the figure below:

link.lds

```
/* Split memory into area for vectors and ram */
MEMORY
{
        flash  (rx) : ORIGIN = 0x00000000, LENGTH = 2M
        ram   (rw!x): ORIGIN = 0x00400000, LENGTH = 256k
}

OUTPUT_FORMAT("elf32-littlearm", "elf32-littlearm", "elf32-l
OUTPUT_ARCH(arm)
ENTRY(_vector_start);
_vector_start = 0x00000000;

SECTIONS
{
/* vectors go to vectors region */
        . = 0x00000000;
        .vectors :
        {
            KEEP(*(*.vectors))
            KEEP( *(*.rom1))
        } > flash
```

将红色框中的地址修改为 Application 分区的起始地址

Figure 4.1: Linker Script Example

6. After modifying the link script, recompile and generate the firmware **rtthread.bin.**

4.4 OTA Firmware Packaging

The application rtthread.bin compiled by the compiler is the original firmware and cannot be used for RT-Thread OTA To upgrade the firmware, users need to use the RT-Thread OTA firmware packager to generate the firmware with the .rbl suffix, and then OTA upgrade is available.

The RT-Thread OTA firmware packager is shown in the following figure:

Figure 4.2: *OTA* Packaging Tools

Users can choose whether to encrypt and compress the firmware according to their needs. A variety of compression and encryption algorithms are supported.

The basic operation steps are as follows:

• Select the firmware to be packaged
• Select the location to generate the
firmware • Select the
compression algorithm •
Select the encryption algorithm • Configure the encryption key
(leave it blank if not encrypted) • Configure the encryption IV
(leave it blank if not encrypted) • Fill in the firmware name
(corresponding to the partition
name) • Fill in the
firmware version • Start packaging • OTA upgrade

Noteÿ

• The encryption key and encryption **IV** must be consistent with those in the BootLoader program, otherwise the firmware cannot be encrypted correctly. • During the firmware packaging process, there is a firmware name to be filled in. Please note that you need to fill in the name of the corresponding partition in the Flash partition table. The name cannot be wrong (usually the application area is called app)

4.5 Start Upgrading

If the OTA downloader used by the developer is deployed on a public network server, the OTA upgrade firmware needs to be uploaded to the to the corresponding server.

If the developer is using the Y-modem method, you need to enter the update command in the RT-Thread MSH command line to upgrade.

For the operation methods of different OTA upgrade methods, please refer to the user manual of the corresponding upgrade method.

4.6 Reference

• Bootloader & OTA overall flow chart

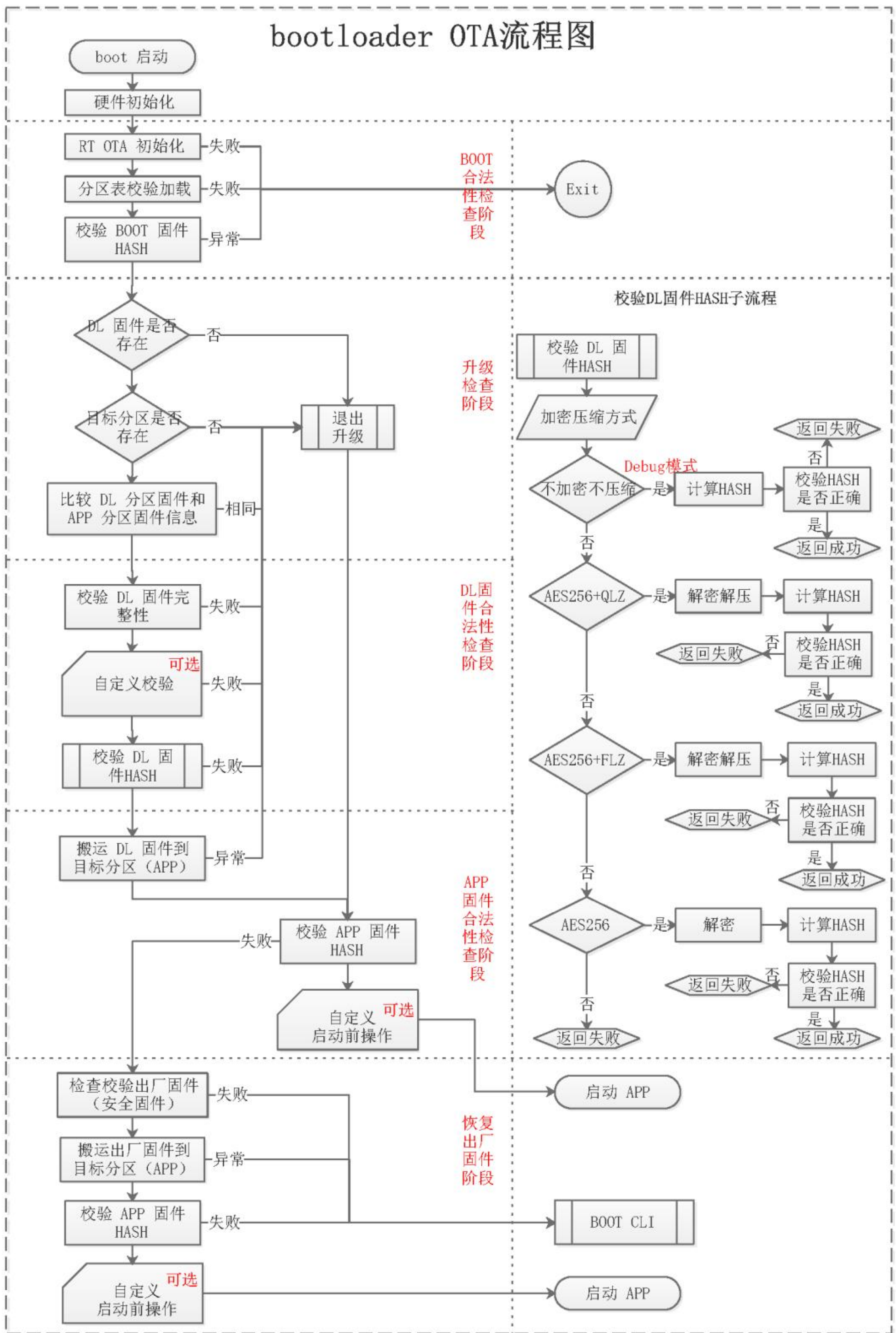

Figure **4.3:** *Bootloader OTA* flow chart

• RT_OTA software framework diagram

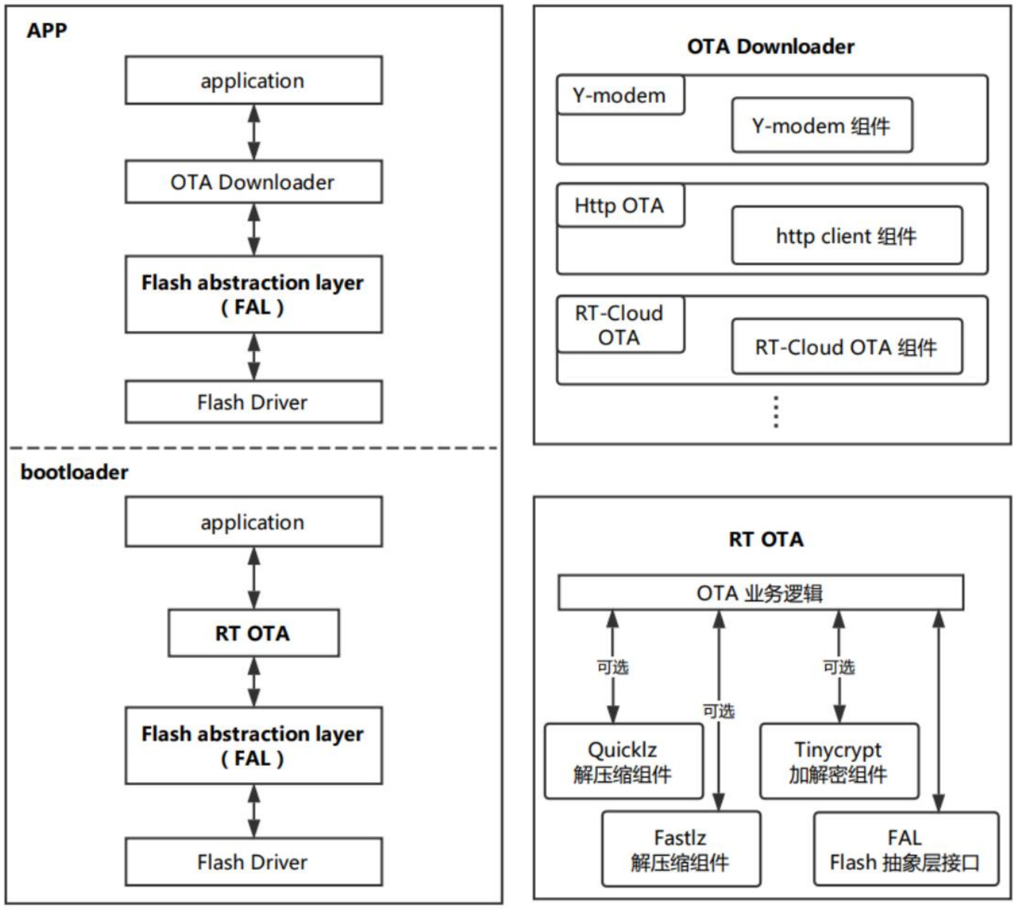

Figure 4.4: *RT OTA* Software framework diagram

4.7 Notes

- The encryption key and encryption **IV** used in the firmware packaging tool must be consistent with those in the BootLoader program, otherwise

  Unable to properly encrypt the firmware

Chapter 5

rt_ota API

5.1 OTA Initialization

```
int rt_ota_init(void);
```

OTA global initialization function, which belongs to the application layer function and needs to **be called before using the OTA function. rt_ota_init**

The function interface integrates the initialization of **FAL** (FAL: Flash abstraction layer) functions.

parameter	describe
none	none
return	describe
>= 0	success
-1	Partition table not found
-2	Download partition not found

5.2 OTA Firmware Verification

```
int rt_ota_part_fw_verify(const struct fal_partition *part);
```

Verify the integrity and validity of the firmware in the specified partition.

parameter	describe
part	Pointer to the partition to be verified
return	describe
>= 0	success

parameter	describe
-1	Verification failed

5.3 OTA Upgrade Check

```
int rt_ota_check_upgrade(void);
```

Check if the device needs to be upgraded. This function interface will first verify the firmware of the download partition. Check whether the download partition has firmware by using the header information. If the download partition has firmware, compare the download partition with The firmware header information of the firmware in the target partition (app partition). If the firmware header information is inconsistent, an upgrade is required.

parameter	describe
none	none
return	describe
1	Need to upgrade
0	No upgrade required

5.4 Firmware Erase

```
int rt_ota_erase_fw(const struct fal_partition *part, size_t new_fw_size);
```

Erase the target partition firmware information. This interface will erase the firmware in the target partition. Please confirm the target partition before using it. correctness.

parameter	describe
part	Pointer to the partition to be erased
new_fw_size	Specify the erase area as the size of the new firmware
return	describe
>= 0	Actual erased size
< 0	mistake

5.5 Query the firmware version number

const char *rt_ota_get_fw_version(const struct fal_partition* part);

Get the version of the firmware in the specified partition.

parameter	describe
part	Pointer to the Flash partition
return	describe
!= NULL	Successfully obtain the version number and return a pointer to the version number
NULL	fail

5.6 Query the firmware timestamp

uint32_t rt_ota_get_fw_timestamp(const struct fal_partition *part);

Get the timestamp information of the firmware in the specified partition.

parameter	describe
part	Pointer to the Flash partition
return	describe
!= 0	Success, return timestamp
0	fail

5.7 Query the firmware size

uint32_t rt_ota_get_fw_size(const struct fal_partition *part);

Get the size of the firmware in the specified partition.

parameter	describe
part	Pointer to the Flash partition
return	describe
!= 0	Success, returns the firmware size
0	fail

parameter	describe

5.8 Query the original firmware size

```
uint32_t rt_ota_get_raw_fw_size(const struct fal_partition *part);
```

Get the original size information of the firmware in the specified partition. For example, the firmware stored in the download partition (download partition) It may be a compressed and encrypted firmware. Use this interface to obtain the original firmware size before compression and encryption.

parameter	describe
part	Pointer to the Flash partition
return	describe
!= 0	Success, returns the firmware size
0	fail

5.9 Get the target partition name

```
const char rt_ota_get_fw_dest_part_name(const struct fal_partition part);
```

Get the name of the target partition within the specified partition. For example, the target partition in the download partition may be app or other partitions (such as parameter area, file system area).

parameter	describe
part	Pointer to the Flash partition
return	describe
!= 0	Success, returns the firmware size
0	fail

5.10 Obtaining the firmware encryption and compression method

```
rt_ota_algo_t rt_ota_get_fw_algo(const struct fal_partition *part);
```

Get the encryption compression method of the firmware in the specified partition.

parameter	describe
part	Pointer to the Flash partition
return	describe
RETURN_VALUE	Returns the firmware encryption compression type

Get encryption type: RETURN_VALUE & RT_OTA_CRYPT_STAT_MASK

Get compression type: RETURN_VALUE & RT_OTA_CMPRS_STAT_MASK

Encryption compression type	describe
RT_OTA_CRYPT_ALGO_NONE	No encryption or compression
RT_OTA_CRYPT_ALGO_XOR	XOR encryption
RT_OTA_CRYPT_ALGO_AES256	AES256 encryption
RT_OTA_CMPRS_ALGO_GZIP	GZIP compression
RT_OTA_CMPRS_ALGO_QUICKLZ	Quicklz compression method
RT_OTA_CMPRS_ALGO_FASTLZ	FastLz compression method

OTA upgrade starts on **May 11th**

```
int rt_ota_upgrade(void);
```

Start the firmware upgrade and move the OTA firmware from the download partition to the target partition (app partition).

parameter	describe
none	none
return	describe
rt_ota_err_t type error	For detailed error types, see the definition of rt_ota_err_t.

Error Type	value
RT_OTA_NO_ERR	0
RT_OTA_GENERAL_ERR	-1
RT_OTA_CHECK_FAILED	-2
RT_OTA_ALGO_NOT_SUPPORTED	-3

Error Type	value
RT_OTA_COPY_FAILED	-4
RT_OTA_FW_VERIFY_FAILED	-5
RT_OTA_NO_MEM_ERR	-6
RT_OTA_PART_READ_ERR	-7
RT_OTA_PART_WRITE_ERR	-8
RT_OTA_PART_ERASE_ERR	-9

5.12 Obtaining firmware encryption information

void rt_ota_get_iv_key(uint8_t * * key_buf); iv_buf, uint8_t

The porting interface needs to be implemented by the user and obtains the iv and key used for firmware encryption from the user-specified location.

parameter	describe
iv_buf	Pointer to the storage of firmware encryption iv, cannot be empty
key_buf	Pointer to the firmware encryption key, cannot be empty
return	describe
none	none

5.13 Custom Verification

int rt_ota_custom_verify(const struct fal_partition *cur_part, long offset, const*

uint8_t buf, size_t len);

User-defined verification interface, which is used to extend the user-defined firmware verification method and needs to be re-implemented by the user.

This interface gets the OTA firmware content of the **len** parameter size through **the buf** parameter. The offset address of the firmware is **offset.** If the user needs to perform customized operations on this part of the firmware, he can implement this interface to handle it.

Note that users cannot modify the contents of the buffer pointed to by **buf** within this interface .

parameter	describe
cur_part	OTA firmware download partition

parameter	describe
offset	OTA firmware offset address
buf	Points to a temporary buffer where OTA firmware is stored and cannot be modified. change
only	Firmware size in OTA firmware buffer
return	describe
>= 0	success
< 0	fail