RT-THREAD OTA 用户手册

RT-THREAD 文档中心

上海睿赛德电子科技有限公司版权 **@2019**

WWW.RT-THREAD.ORG

Friday 28th September, 2018

版本和修订

Date	Version	Author	Note
2018-06-21	v0.1	MurphyZhao	初始版本

目录

版本和修订 **i**

目录 **ii**

1 rt_ota 介绍 **1**

1.1 文件目录结构 1

1.2 rt_ota 软件框架图 2

1.3 rt_ota 功能特点 3

1.3.1 加密 3

1.3.2 压缩 3

1.3.3 防篡改 4

1.3.4 差分升级 4

1.3.5 断电保护 4

1.3.6 智能还原 5

2 rt_ota 示例应用程序 **6**

2.1 示例介绍 6

3 OTA 工作原理 **7**

4 rt_ota 使用说明 **9**

4.1 使用前的准备 9

4.1.1 依赖软件包下载与移植 9

FAL（必选） 9

Quicklz 或者 Fastlz（可选） 9

TinyCrypt（可选） 10

4.1.2 rt_ota 软件包下载与移植 10
4.1.3 定义配置参数 10
4.2 开发 bootloader 11
4.3 开发 APP 11
4.4 OTA 固件打包 12
4.5 开始升级 14
4.6 参考 14
4.7 注意事项 16

5 rt_ota API **17**

5.1 OTA 初始化 17
5.2 OTA 固件校验 17
5.3 OTA 升级检查 18
5.4 固件擦除 18
5.5 查询固件版本号 19
5.6 查询固件时间戳 19
5.7 查询固件大小 19
5.8 查询原始固件大小 20
5.9 获取目标分区名字 20
5.10 获取固件加密压缩方式 20
5.11 开始 OTA 升级 21
5.12 获取固件加密信息 22
5.13 自定义校验 22

第 1 章

rt_ota 介绍

rt_ota 是 RT-Thread 开发的跨 OS、跨芯片平台的固件空中升级技术（Firmware Over-the-Air Technology），轻松实现对设备端固件的管理、升级与维护。

RT-Thread 提供的 OTA 固件升级技术具有以下优势：

- 固件防篡改：自动检测固件签名，保证固件安全可靠
- 固件加密：支持 AES-256 加密算法，提高固件下载、存储安全性
- 固件压缩：高效压缩算法，降低固件大小，减少 Flash 空间占用，节省传输流量，降低下载时间
- 差分升级：根据版本差异生成差分包，进一步节省 Flash 空间，节省传输流量，加快升级速度
- 断电保护：断电后保护，重启后继续升级
- 智能还原：固件损坏时，自动还原至出厂固件，提升可靠性
- 高度可移植：可跨 OS、跨芯片平台、跨 Flash 型号使用，不依赖具体的 OTA 服务器

1.1 文件目录结构

```
rt_ota
│   README.md                       // 软件包使用说明
│   SConscript                      // RT-Thread 默认的构建脚本
├───docs
│   │   api.md                      // API 使用说明
│   │   introduction.md             // 软件包详细介绍
│   │   port.md                     // 移植说明文档
│   └───user-guide.md               // 用户手册
├───inc                             // 头文件
├───libs                            // 库文件
├───ports                           // 移植文件
```

```
|    └---temp
|           rt_ota_key_port.c        // 移植文件模板
├---samples                          // 示例代码
|    └---ota.c                       // 软件包应用示例代码
└---tools                            // 工具
        fatfs_ota_packaging_tool     // fatfs 文件系统 OTA 打包工具
        firmware_ota_packaging_tool  // OTA 文件打包工具（rbl文件）
```

1.2 rt_ota 软件框架图

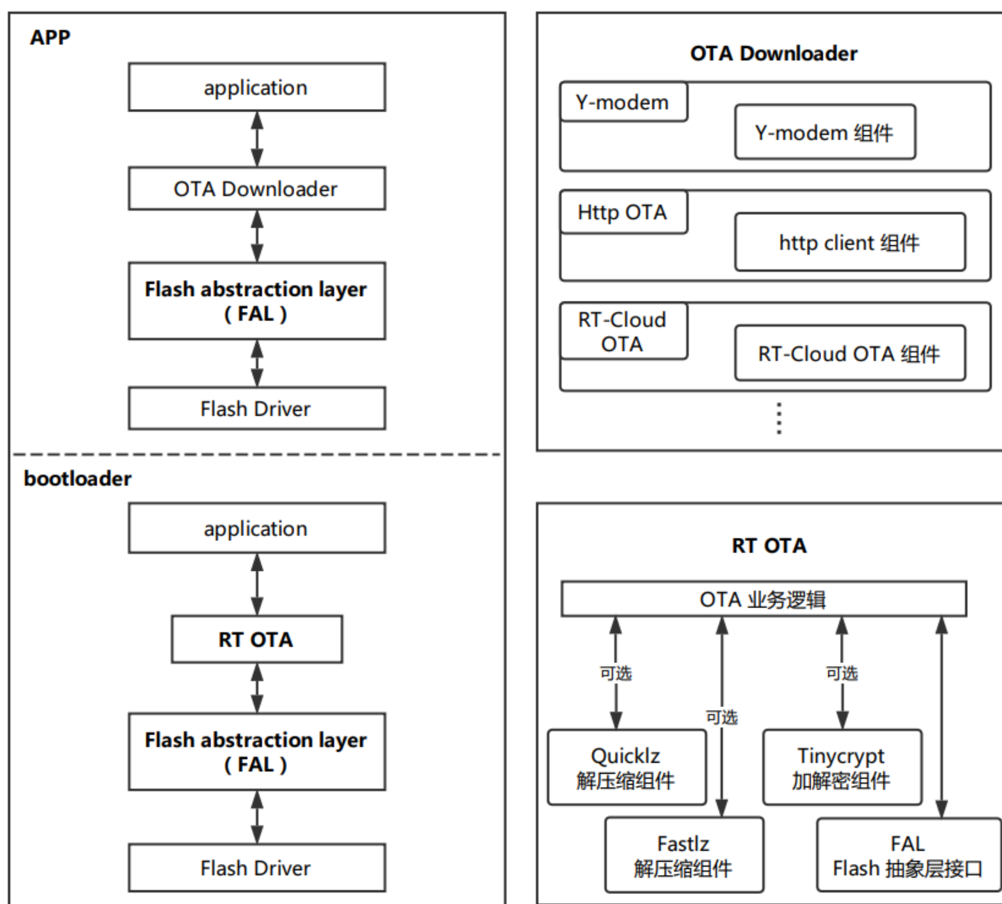

图 1.1: *RT OTA 软件框架图*

如上图所示，该应用框架图展示了 rt_ota 在整个 OTA 应用中所处的位置，以及 rt_ota 应用所涉及到的相关软件组件包。

从 **rt_ota 软件框架图**中可以看到，APP 部分的软件是不需要依赖 rt_ota 软件包的。因为，APP 部分只需要关心如何把升级固件从 OTA 服务器下载到设备，而涉及到系统安全与稳定性的固件校验和固件搬运环节才需要 rt_ota 介入。

OTA Downloader 是与 OTA 服务器对应的客户端程序，用于将 OTA 固件从 OTA 服务器下载到设备。常用的且通用的 **OTA Downloader** 有 `Y-modem`（串口升级）和 `HTTP OTA`（网络升级），开发者使用自己的电脑既可搭建用于 OTA 升级的服务端。私有云或者公有云平台提供的 OTA 服务器，通常需要开发对应的客户端程序，运行在设备端，用于下载 OTA 固件。

1.3 rt_ota 功能特点

1.3.1 加密

为什么要选择加密？

- 未加密的固件可以被任何人以任何方式窃取使用，还可能面临固件被篡改、系统被攻击、产品被山寨等风险
- 客户使用的 OTA 服务多是第三方服务，客户的固件需要上传到第三方服务器，或者发送给第三方机构，固件很容易被泄漏、传播，或恶意使用

为了避免未加密固件存在的各种问题，**rt_ota** 对固件使用了 AES256 加密方式。

> AES（Advanced Encryption Standard）是美国联邦政府采用的一种分组加密标准，也是目前分组密码实际上的工业标准。

rt_ota 采用了 **TinyCrypt** 软件包中实现的 AES256 加密算法，解密速度快，资源占用小。

> 未开优化情况下，**TinyCrypt** 占用 ROM 5244 字节，占用 RAM 8744 字节。

1.3.2 压缩

为什么要支持压缩？

```
嵌入式设备的 Flash 资源往往都比较有限（通常只有 2M 字节），在有限的 Flash
    上通常需要存储 Bootloader、应用程序（app）、OTA 固件、系统和用户参数
   配置等信息，这就使得可用的应用程序代码空间变得很小。
```

为了解决 Flash 资源受限的问题，RT-Thread OTA 引入了高效的压缩算法，来降低固件对 Flash 空间的占用。

目前 RT-Thread 完美支持了 Quicklz、Fastlz 和 MiniLZO 解压缩算法，并在 rt_ota 组件包支持使用 Quicklz 和 Fastlz。

下面表格统计了三种压缩算法在压缩率和资源占用上的对比：（非精准测试，仅供参考）

名称	版权	ROM	解压时 RAM	压缩级别	压缩率
quicklz	GPL	1838	9732	3	67%
fastlz	MIT	3096	9696	2	74%
miniLZO	GPL	2024	9604	LZO1X_1	75%

1.3.3 防篡改

OTA 固件通常是暴露在外网中的，如果固件未经过加密和防篡改处理，就会面临以下问题：

- OTA 固件存放在第三方 OTA 服务器，不受信
- OTA 固件升级下载的过程中，可能被截获，被恶意篡改，不安全
- OTA 固件可能被非法获取，被破解，产品可能被山寨

为了保证客户固件的安全，以及 OTA 升级的可靠，RT-Thread OTA 默认集成了防篡改功能，检查速度快，可靠性强。

1.3.4 差分升级

差分升级是将设备固件与新版固件的差异部分以既定的组织格式打成差分包然后进行升级的一种技术。

在嵌入式设备中常用的差分升级多是 `多bin 升级` 方式，有效地降低了差分升级的复杂度。

多 bin 升级，通常是将一个应用程序划分成不同的部分，生成多个 bin 文件，通过编译器分别链接到 Flash 的不同位置，每次升级只升级其中的一个 bin 文件。

相对于整包升级，差分升级具有以下优势：

- 差分包相对较小，流量成本低
- 下载和升级速度较快，升级时间短
- 网络条件要求低，适用于 LoRa 和 NBiot 应用场景
- 有效减少电能消耗

1.3.5 断电保护

断电保护功能主要应用在 OTA 升级过程中，设备突然断电的场景中。如果没有断电保护功能，设备很有可能因为只升级了一部分固件而变砖返厂。

RT-Thread OTA 安全保护机制的 **断电保护**功能，保证设备升级过程中即使出现了异常中断，下次上电设备仍会继续进行升级，不会导致固件损坏设备变砖。

1.3.6 智能还原

设备可能会因为外部攻击、升级过程中断或者其它某种原因导致设备固件异常，即使发生了这种情况，RT-Thread OTA 安全保护机制的 **智能还原**功能也可以智能地还原设备固件，从而有效保证设备程序正确稳定地运行。

第 2 章

rt_ota 示例应用程序

2.1 示例介绍

示例文件：

```
samples/ota.c
```

该例程是 **rt_ota** 软件包的示例展示，主要是展示用户如何使用 **rt_ota** 软件包快速搭建自己的 OTA 应用程序，以及展示了 OTA 的基本工作流程。

该例程文件可以应用到用户的 Bootloader 工程中使用，也可以基于该例程定制修改适合用户方案的 OTA 流程。

第 3 章

OTA 工作原理

OTA 升级其实就是 IAP 在线编程。在嵌入式设备 OTA 中，通常通过串口或者网络等方式，将升级数据包下载到 Flash，然后将下载得到的数据包搬运到 MCU 的代码执行区域进行覆盖，以完成设备固件升级更新的功能。

嵌入式设备的 OTA 升级一般是不基于文件系统的，而是通过对 Flash 划分为不同的功能区域来完成 OTA 升级功能。

在嵌入式系统方案里，要完成一次 OTA 固件远端升级，通常需要以下三个核心阶段：

1. 上传新固件到 OTA 服务器
2. 设备端下载新的 OTA 固件
3. bootloader 对 OTA 固件进行校验、解密和搬运（搬运到可执行程序区域）

详细的 OTA 升级流程如下图所示：

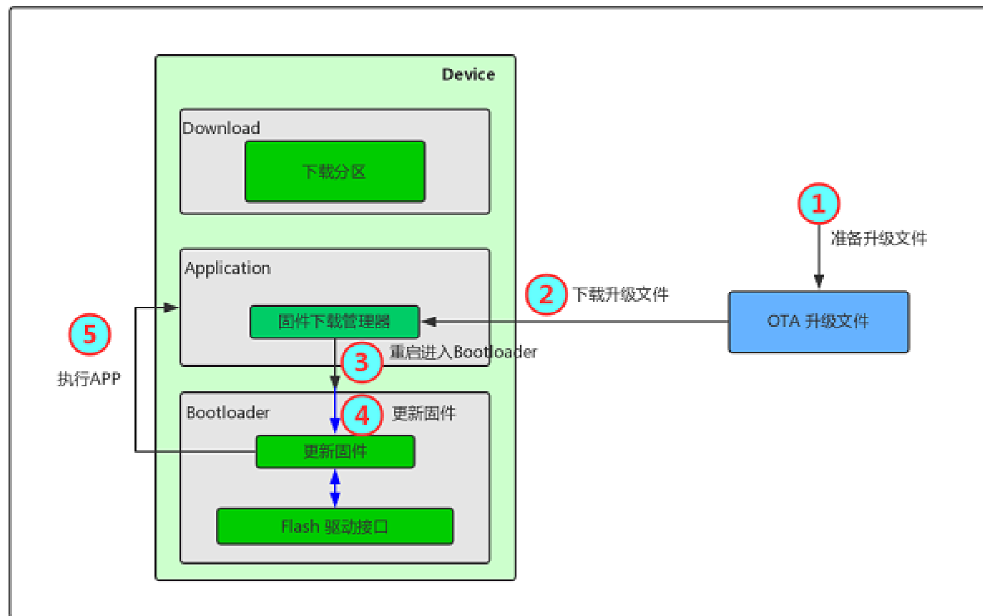

图 3.1: *OTA 升级流程*

第 4 章

rt_ota 使用说明

4.1 使用前的准备

4.1.1 依赖软件包下载与移植

FAL（必选）

FAL 软件包下载：

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

FAL 软件包移植参考 FAL README。

Quicklz 或者 Fastlz（可选）

Quicklz 和 Fastlz 是 rt_ota 支持的解压缩软件包，用户可以选择使用其中的一个。

Quicklz 软件包下载：

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

在 OTA 中启用压缩并且使用 Quicklz 需要在 **rtconfig.h** 文件中定义以下宏定义：

```
#define RT_OTA_USING_CMPRS                    // 启用解压缩功能
#define RT_OTA_CMPRS_ALGO_USING_QUICKLZ       // 使用 Quicklz
#define QLZ_COMPRESSION_LEVEL 3               // 定义使用 Quicklz 3级压缩
```

Fastlz 软件包下载：

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

在 OTA 中启用压缩并且使用 Quicklz 需要在 **rtconfig.h** 文件中定义以下宏定义：

```
#define RT_OTA_USING_CMPRS                  // 启用解压缩功能
#define RT_OTA_CMPRS_ALGO_USING_FASTLZ      // 使用 Fastlz
```

TinyCrypt（可选）

TinyCrypt 是 rt_ota 中使用的用于固件加密的软件包，支持 AES256 加解密。

TinyCrypt 软件包下载：

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

在 OTA 中启用压缩并且使用 TinyCrypt 需要在 **rtconfig.h** 文件中定义以下宏定义：

```
#define RT_OTA_USING_CRYPT                 // 启用 Tinycrypt 组件包
#define TINY_CRYPT_AES                     // 启用 AES 功能
#define RT_OTA_CRYPT_ALGO_USING_AES256     // 启用 AES256 加密功能
```

4.1.2 rt_ota 软件包下载与移植

rt_ota 是闭源包，请联系 **RT-Thread** 获取使用权。

如果您已经拿到 **rt_ota** 的使用权，并下载到了 **rt_ota** 软件包，请详细阅读软件包中的相关说明文档，完成移植工作，参考 **rt_ota** 移植文档。

4.1.3 定义配置参数

依赖软件包下载与移植章节中描述的配置宏定义需要定义到 **rtconfig.h** 文件中，参考文件如下所示：（开发者根据自己的需求配置相关宏定义）

```
#define PKG_USING_RT_OTA                   // 启用 RT_OTA 组件包
#define RT_OTA_USING_CRYPT                 // 启用 Tinycrypt 组件包
#define TINY_CRYPT_AES                     // 启用 AES 功能
#define RT_OTA_CRYPT_ALGO_USING_AES256     // 启用 AES256 加密功能
#define RT_OTA_USING_CMPRS                 // 启用解压缩功能
#define RT_OTA_CMPRS_ALGO_USING_QUICKLZ    // 启用 Quicklz
#define QLZ_COMPRESSION_LEVEL 3            // 定义使用 Quicklz 3级压缩
```

```
#define FAL_PART_HAS_TABLE_CFG          // 启用分区表配置文件（不启用需
    要在 Flash 中查找）
```

4.2 开发 bootloader

rt_ota 软件包完成的是固件校验、认证、搬运的工作，需要配合 BootLoader 中使用，因此用户在拿到 **rt_ota** 软件包后，需要按照自己的需求开发 BootLoader 程序。

1. 开发者首先需要创建目标平台的 BootLoader 工程（可以是裸机工程）
2. 将 **rt_ota** 软件包拷贝到 BootLoader 工程目录
3. 将 **FAL** 软件包拷贝到 BootLoader 工程目录，并完成移植工作，参考 FAL README
4. 将 **Quicklz** 或者 **Fastlz** 软件包拷贝到 BootLoader 工程目录（如果需要解压缩功能）
5. 将 **TinyCrypt** 软件包拷贝到 BootLoader 工程目录（如果需要加密功能）
6. 将 **定义配置参数**章节中的 **rtconfig.h** 文件拷贝到 BootLoader 工程
7. 开发 OTA 具体的业务逻辑，参考 bootloader&OTA 整体流程图（详见参考章节），参考 sample 说明文档

4.3 开发 APP

APP 中所要完成的工作主要是讲 OTA 升级文件下载到设备的 Flash 中。

1. 建立 RT-Thread 应用程序工程
2. 使用 RT-Thread 的包管理器打开 FAL 组件包，并完成移植工作，参考 FAL README（移植代码可以与 Bootloader 中的通用）
3. 选择一个 OTA Downloader（RT-Thread 包管理工具提供 Y-modem 和 HTTP OTA）
    - Ymodem
    - HTTP OTA
    - 其它（需要自行开发 OTA 固件下载客户端程序）
4. 开发应用程序业务逻辑
5. 修改链接脚本配置

通常，我们的程序都是从 Flash 代码区的起始地址开始运行的。但是，Flash 代码区的起始地址开始的空间被 bootloader 程序占用了，因此我们需要修改链接脚本，让 application 程序从 Flash 的 application 区域起始地址开始排放。

一般，我们只需要修改链接脚本里中 Flash 和 SECTION 段的起始地址为 application 分区的起始地址即可。application 分区信息必须跟对应 MCU 平台的 Flash 分区表完全一致。

以 GCC 链接脚本为例，修改示例如下图所示：

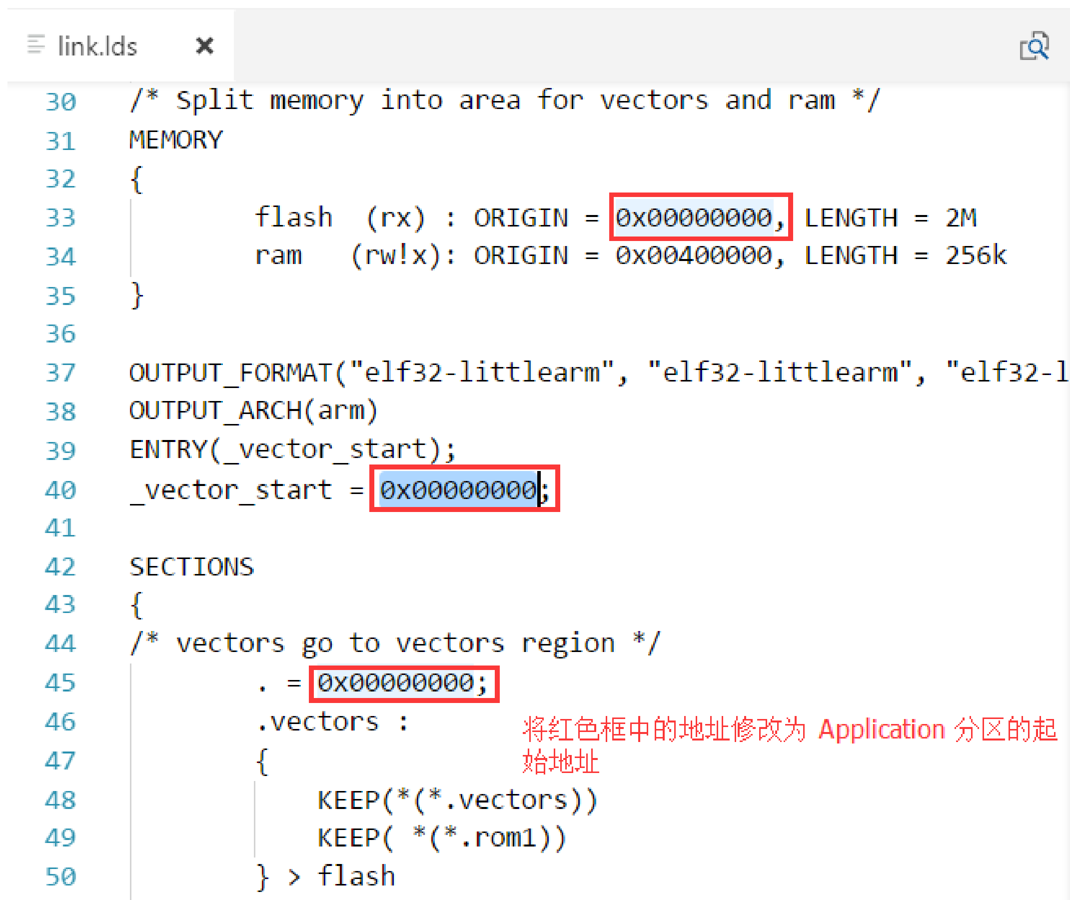

图 4.1: *链接脚本示例*

6. 修改链接脚本后，重新编译生成固件 **rtthread.bin**。

4.4 OTA 固件打包

编译器编译出来的应用程序 `rtthread.bin` 属于原始固件，并不能用于 RT-Thread OTA 的升级固件，需要用户使用 RT-Thread OTA 固件打包器 打包生成 `.rbl` 后缀名的固件，然后才能进行 OTA 升级。

RT-Thread OTA 固件打包器 如下图所示：

图 4.2: *OTA 打包工具*

用户可以根据需要，选择是否对固件进行加密和压缩，提供多种压缩算法和加密算法支持，基本操作步骤如下：

- 选择待打包的固件
- 选择生成固件的位置
- 选择压缩算法
- 选择加密算法
- 配置加密密钥（不加密则留空）
- 配置加密 IV （不加密则留空）
- 填写固件名称（对应分区名称）
- 填写固件版本
- 开始打包
- OTA 升级

Note：

- **加密密钥**和 **加密 IV** 必须与 BootLoader 程序中的一致，否则无法正确加密固件
- 固件打包过程中有 `固件名称` 的填写，这里注意需要填入 Flash 分区表中对应分区的名称，不能有误（通常应用程序区名称为 `app`）

4.5 开始升级

如果开发者使用的 OTA 下载器部署在公网服务器，则需要将 OTA 升级固件上传到对应的服务器中。

如果开发者使用的是 Y-modem 方式，需要在 RT-Thread MSH 命令行中输入 `update` 命令来进行升级。

不同 OTA 升级方式的操作方法，请参考对应升级方式的用户手册。

4.6 参考

- bootloader&OTA 整体流程图

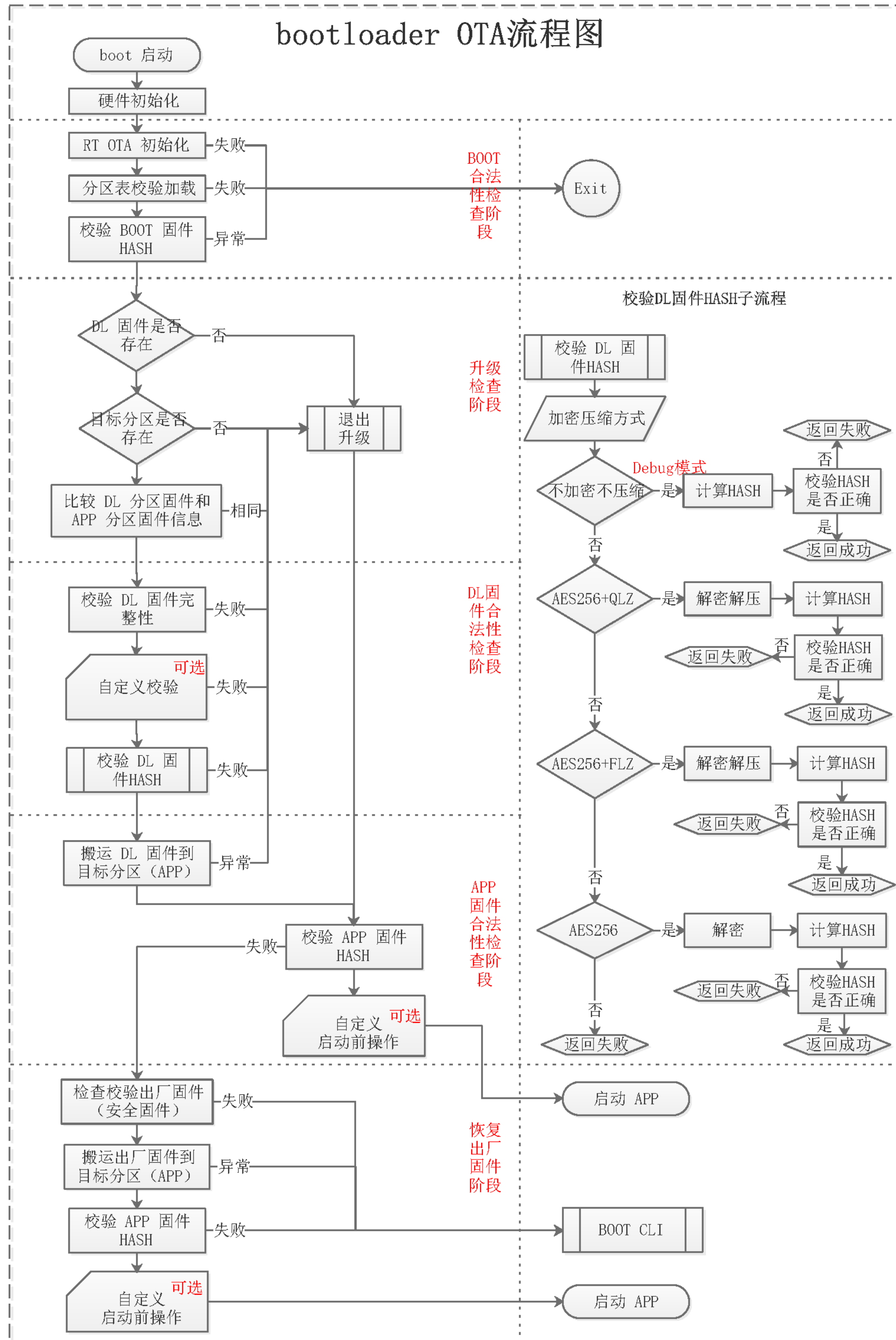

图 4.3: *Bootloader OTA 流程图*

- RT_OTA 软件框架图

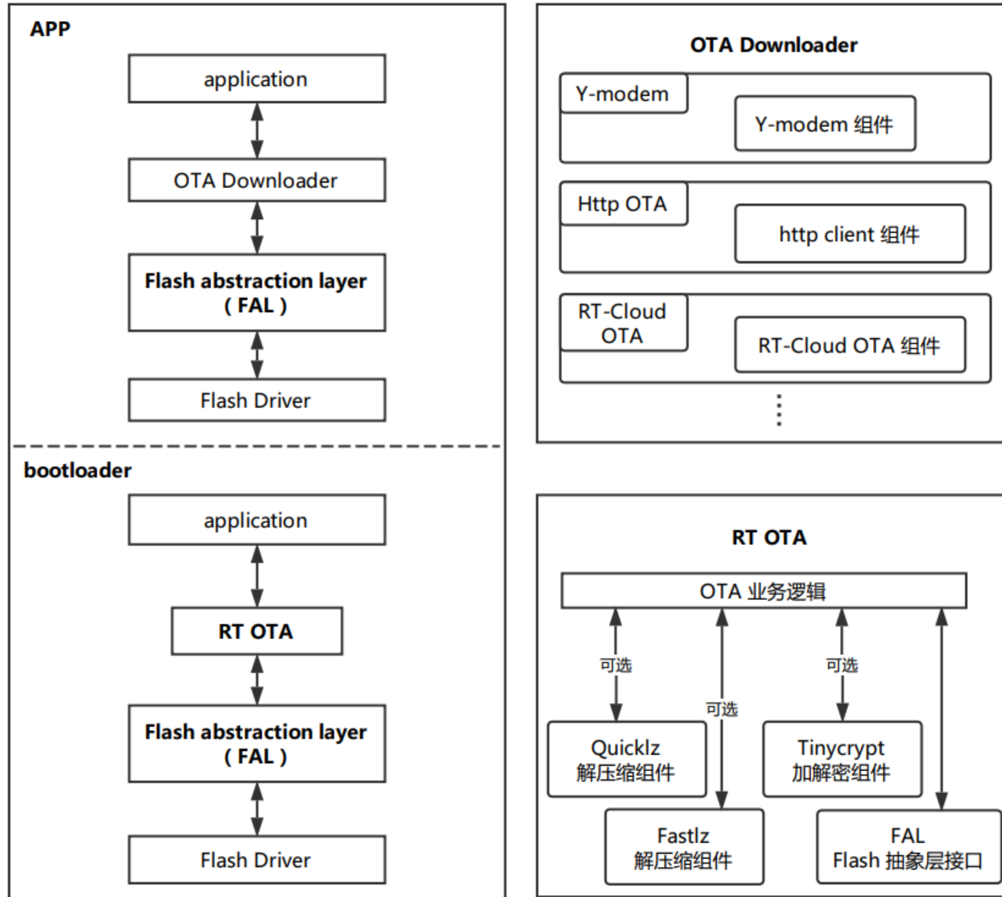

图 4.4: *RT OTA 软件框架图*

4.7 注意事项

- 固件打包工具中使用的 **加密密钥**和 **加密 IV** 必须与 BootLoader 程序中的一致，否则无法正确加密固件

第 5 章

rt_ota API

5.1 OTA 初始化

int rt_ota_init(void);

OTA 全局初始化函数，属于应用层函数，需要在使用 OTA 功能前调用。**rt_ota_init** 函数接口内部集成了 **FAL** （FAL: Flash abstraction layer，Flash 抽象层）功能的初始化。

参数	描述
无	无
返回	**描述**
`>= 0`	成功
-1	分区表未找到
-2	下载分区未找到

5.2 OTA 固件校验

int rt_ota_part_fw_verify(const struct fal_partition *part);

对指定分区内的固件进行完整性和合法性校验。

参数	描述
part	指向待校验的分区的指针
返回	**描述**
`>= 0`	成功

参数	描述
-1	校验失败

5.3 OTA 升级检查

int rt_ota_check_upgrade(void);

检查设备是否需要升级。该函数接口首先会通过校验下载分区（Download 分区）固件头信息的方式检查下载分区是否存在固件，如果下载分区存在固件，则对比检查下载分区和目标分区（app 分区）中固件的固件头信息，如果固件头信息不一致，则需要进行升级。

参数	描述
无	无
返回	**描述**
1	需要升级
0	不需要升级

5.4 固件擦除

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

擦除目标分区固件信息。该接口会将目标分区内的固件擦除，使用前，请确认目标分区的正确性。

参数	描述
part	指向待擦除分区的指针
new_fw_size	指定擦除区域为新固件的大小
返回	**描述**
`>= 0`	实际擦除的大小
< 0	错误

5.5 查询固件版本号

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

获取指定分区内的固件的版本。

参数	描述
part	指向 Flash 分区的指针
返回	**描述**
!= NULL	成功获取版本号，返回指向版本号的指针
NULL	失败

5.6 查询固件时间戳

uint32_t rt_ota_get_fw_timestamp(const struct fal_partition *part);

获取指定分区内的固件的时间戳信息。

参数	描述
part	指向 Flash 分区的指针
返回	**描述**
!= 0	成功，返回时间戳
0	失败

5.7 查询固件大小

uint32_t rt_ota_get_fw_size(const struct fal_partition *part);

获取指定分区内的固件的大小信息。

参数	描述
part	指向 Flash 分区的指针
返回	**描述**
!= 0	成功，返回固件大小
0	失败

参数	描述

5.8 查询原始固件大小

uint32_t rt_ota_get_raw_fw_size(const struct fal_partition *part);

获取指定分区内的固件的原始大小信息。如下载分区（download 分区）里存储的固件可能是经过压缩加密后的固件，通过该接口获取压缩加密前的原始固件大小。

参数	描述
part	指向 Flash 分区的指针
返回	**描述**
!= 0	成功，返回固件大小
0	失败

5.9 获取目标分区名字

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

获取指定分区内的目标分区的名字。如下载分区（download 分区）中目标分区可能是 `app` 或者是其他分区（如参数区、文件系统区）。

参数	描述
part	指向 Flash 分区的指针
返回	**描述**
!= 0	成功，返回固件大小
0	失败

5.10 获取固件加密压缩方式

rt_ota_algo_t rt_ota_get_fw_algo(const struct fal_partition *part);

获取指定分区内固件的加密压缩方式。

参数	描述
part	指向 Flash 分区的指针
返回	**描述**
RETURN_VALUE	返回固件加密压缩类型

获取加密类型：RETURN_VALUE & RT_OTA_CRYPT_STAT_MASK

获取压缩类型：RETURN_VALUE & RT_OTA_CMPRS_STAT_MASK

加密压缩类型	描述
RT_OTA_CRYPT_ALGO_NONE	不加密不压缩
RT_OTA_CRYPT_ALGO_XOR	XOR 加密方式
RT_OTA_CRYPT_ALGO_AES256	AES256 加密方式
RT_OTA_CMPRS_ALGO_GZIP	GZIP 压缩方式
RT_OTA_CMPRS_ALGO_QUICKLZ	Quicklz 压缩方式
RT_OTA_CMPRS_ALGO_FASTLZ	FastLz 压缩方式

5.11 开始 OTA 升级

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

启动固件升级，将 OTA 固件从下载分区搬运到目标分区（app 分区）。

参数	描述
无	无
返回	**描述**
rt_ota_err_t 类型错误	详细的错误类型查看 rt_ota_err_t 定义

错误类型	值
RT_OTA_NO_ERR	0
RT_OTA_GENERAL_ERR	-1
RT_OTA_CHECK_FAILED	-2
RT_OTA_ALGO_NOT_SUPPORTED	-3

错误类型	值
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 获取固件加密信息

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

移植接口，需要用户自行实现，从用户指定的地方获取固件加密使用的 iv 和 key。

参数	描述
iv_buf	指向存放固件加密 iv 的指针，不能为空
key_buf	指向存放固件加密 key 的指针，不能为空
返回	**描述**
无	无

5.13 自定义校验

int rt_ota_custom_verify(const struct fal_partition *cur_part, long offset, const uint8_t* buf, size_t len);

用户自定义校验接口，该接口用于扩展用户自定义的固件校验方法，需用用户重新实现。

该接口通过 **buf** 参数拿到 **len** 参数大小的 OTA 固件内容，固件的偏移地址为 **offset**，用户如果需要对这部分固件做自定义的操作，可以实现该接口来进行处理。

注意，用户不能在该接口内修改 **buf** 指向的缓冲区里的内容。

参数	描述
cur_part	OTA 固件下载分区

参数	描述
offset	OTA 固件的偏移地址
buf	指向存放 OTA 固件的临时缓冲区，不能修改
len	OTA 固件缓冲区中的固件大小
返回	**描述**
`>= 0`	成功
< 0	失败