# 文件级加密

Android 7.0 及更高版本支持文件级加密 (FBE)。采用文件级加密时，可以使用不同的密钥对不同的文件进行加密，也可以对加密文件单独解密。

本文介绍了如何在新设备上启用文件级加密，以及系统应用如何利用 Direct Boot API 尽可能为用户提供最佳、最安全的体验。

**注意**：

- 搭载 Android 10 及更高版本的新设备需要使用文件级加密 (/security/encryption/file-based.html)。
- 搭载 Android 9 及更高版本的设备可以使用可合并的存储设备和文件级加密。
- 在搭载 Android 7.0 - 8.1 的设备上，无法同时使用文件级加密和可合并的存储设备 (/devices/storage/adoptable)。
- 对于使用文件级加密的设备，必须将新添加的存储媒介（如 SD 卡）用作传统存储设备 (/devices/storage/traditional)。

## 直接启动

借助文件级加密，Android 7.0 中引入了一项称为直接启动 (https://developer.android.com/training/articles/direct-boot)的新功能。该功能处于启用状态时，已加密设备在启动后将直接进入锁定屏幕。之前，在使用全盘加密 (/security/encryption/full-disk) (FDE) 的已加密设备上，用户在访问任何数据之前都需要先提供凭据，从而导致手机只能执行最基本的操作。例如，手机甚至无法接听电话，只能执行基本的紧急拨号操作，而且闹钟无法运行，无障碍服务也不可用。

引入文件级加密 (FBE) 和可以将应用设为加密感知型应用的新 API 后，应用将能够在受限环境中运行。这意味着，应用可以在用户提供凭据之前运行，同时系统仍能保护私密用户信息。

在启用了 FBE 的设备上，每位用户均有两个可供应用使用的存储位置：

- 凭据加密 (CE) 存储空间：这是默认存储位置，只有在用户解锁设备后才可用。
- 设备加密 (DE) 存储空间：在直接启动模式期间以及用户解锁设备后均可用。

这种区分能够使工作资料更加安全，因为这样一来，加密不再只基于启动密码，从而能够同时保护多位用户。

Direct Boot API 允许加密感知型应用访问上述任何一个存储空间。应用生命周期会发生一些变化，以便系统在以下情况下通知应用：用户的 CE 存储空间因用户在锁定屏幕上首次输入凭据而解锁时，或者在工作资料提供工作资料质询代码 (https://developer.android.com/about/versions/nougat/android-7.0.html#android_for_work)时。无论是否实现了 FBE，运行 Android 7.0 的设备都必须要支持这些新的 API 和生命周期。不过，如果没有启用 FBE，DE 和 CE 存储空间将始终处于解锁状态。

Android 开源项目 (AOSP) 中提供了 Ext4 和 F2FS 文件系统中的文件级加密的完整实现。在满足相关要求的设备上，只需启用该实现即可使用该功能。选择使用 FBE 的制造商可能想要了解基于所用系统芯片 (SoC) 优化该功能的方法。

AOSP 中的所有必要程序包均已更新为直接启动感知型程序包。不过，如果设备制造商使用的是这些应用的定制版本，则需要确保至少存在能够提供以下服务的直接启动感知型程序包：

- 电话服务和拨号器
- 用于在锁定屏幕中输入密码的输入法

## 示例和源代码

Android 提供了文件级加密的参考实现，其中 vold (system/vold (https://android.googlesource.com/platform/system/vold/)) 负责提供用于管理 Android 上的存储设备和存储卷的功能。添加 FBE 会为 vold 提供一些新命令，以便支持对多位用户的 CE 密钥和 DE 密钥进行密钥管理。除了为使用内核中的文件级加密 (#kernel-support)功能而进行的核心更改外，许多系统程序包（包括锁定屏幕和 SystemUI）也经过了修改，以支持 FBE 和“直接启动”功能。这些选项包括：

- AOSP 拨号器 (packages/apps/Dialer)
- 桌面时钟 (packages/apps/DeskClock)
- LatinIME (packages/inputmethods/LatinIME)*
- “设置”应用 (packages/apps/Settings)*
- SystemUI (frameworks/base/packages/SystemUI)*

**使用 `defaultToDeviceProtectedStorage` (#supporting-direct-boot-in-system-applications) 清单属性的系统应用*

通过在 AOSP 源代码树的框架或程序包目录中运行 `mangrep directBootAware` 命令，可以找到更多加密感知型应用和服务的示例。

## 依赖关系

要安全地使用 AOSP 提供的 FBE 实现，设备需要满足以下依赖关系：

- 对 Ext4 加密或 F2FS 加密的**内核支持**（内核配置选项：`CONFIG_EXT4_ENCRYPTION` 或 `CONFIG_F2FS_FS_ENCRYPTION`）
- 基于 1.0 或 2.0 版 HAL 的 **Keymaster 支持** (/security/keystore/index)。不支持 Keymaster 0.3，因为它既不提供必要的功能，也不能保证为加密密钥提供充分保护。
- 必须在可信执行环境 (/security/trusty/index) (TEE) 中实现 **Keymaster/Keystore** (/security/keystore/index) **和 Gatekeeper**，以便为 DE 密钥提供保护，从而使未经授权的操作系统（刷写到设备上的定制操作系统）无法直接请求 DE 密钥。
- 内核中的**加密性能**必须要在使用 AES XTS 时至少达到 50MB/s，以确保良好的用户体验。
- **硬件信任根**和**验证启动**需要绑定到 Keymaster 初始化进程，以确保未经授权的操作系统无法获取设备加密凭据。

**注意**：存储政策会应用到文件夹及其所有子文件夹。对于以未加密形式存入 OTA 文件夹以及存入系统解密密钥存放文件夹的内容，制造商应加以限制。大多数内容都应存放在凭据加密存储空间（而非设备加密存储空间）内。

## 实现

最重要的一点是，应根据直接启动 (https://developer.android.com/training/articles/direct-boot.html)开发者文档将诸如闹钟、电话、无障碍功能等应用设为 android:directBootAware。

### 内核支持

Android 通用内核 3.18 及更高版本中提供了对 Ext4 和 F2FS 加密的内核支持。

除了对 Ext4 或 F2FS 加密提供功能支持外，设备制造商还可以考虑实现加密加速功能，以便加快文件级加密的速度并改善用户体验。

### 启用文件级加密

通过将 **fileencryption=contents_encryption_mode[:filenames_encryption_mode]** 标记添加到 **userdata** 分区最后一列的 **fstab** 行中，可以启用 FBE。
**contents_encryption_mode** 参数定义用于文件内容加密的算法，
**filenames_encryption_mode** 参数定义用于文件名加密的算法。
**contents_encryption_mode** 只能是 **aes-256-xts**。 **filenames_encryption_mode** 有两个可能的值：**aes-256-cts** 和 **aes-256-heh**。如果未指定 **filenames_encryption_mode**，则使用 **aes-256-cts** 值。

## 与 Keymaster 集成

**vold** 负责处理密钥生成和内核密钥环管理。AOSP 的 FBE 实现要求设备支持 1.0 或更高版本的 Keymaster HAL。更低版本的 Keymaster HAL 不受支持。

首次启动时，在启动过程的早期阶段会生成并安装用户 0 的密钥。到 **init** 的 **on-post-fs** 阶段完成时，Keymaster 必须已做好处理请求的准备。在 Nexus 设备上，这是通过设置一个脚本块处理的：

- 确保 Keymaster 在 **/data** 装载之前启动
- 指定文件加密算法套件：AOSP 的文件级加密实现会用到采用 XTS 模式的 AES-256 算法

★ **注意**：所有加密都基于采用 XTS 模式的 AES-256 算法。XTS 的定义方式决定了它需要两个 256 位密钥；因此实际上 CE 密钥和 DE 密钥都是 512 位密钥。

## 加密政策

文件级加密在目录级应用加密政策。首次创建设备的 **userdata** 分区时，会由 **init** 脚本应用基本结构和政策。这些脚本将触发创建首位用户（用户 0）的 CE 密钥和 DE 密钥，并定义要使用这些密钥加密哪些目录。创建其他用户和资料时，会生成必要的其他密钥并将其存储在密钥代码库中；接下来会为密钥创建凭据和设备存储位置，并且加密政策会将这些密钥关联到相应目录。

在 AOSP 当前提供的文件级加密实现中，加密政策被硬编码到了以下位置：

```
/system/extras/libfscrypt/fscrypt_init_extensions.cpp
```

在 Android P 及更早版本中，该位置为：

```
/system/extras/ext4_utils/ext4_crypt_init_extensions.cpp
```

可以在该文件中添加例外情况，以彻底防止特定目录被加密，具体方法是将相应目录添加到 **directories_to_exclude** 列表中。如果进行了此类修改，设备制造商应添加 SELinux 政策（/security/selinux/device-policy），以便仅向需要使用未加密目录的应用授予访问权限（应排除所有不可信的应用）。

目前唯一可接受的使用这种方法的情况是在支持旧版 OTA 功能方面。

## 在系统应用中支持直接启动

### 将应用设为直接启动感知型应用

为了实现系统应用的快速迁移，新增了两个可在应用级别设置的属性。**defaultToDeviceProtectedStorage** 属性仅适用于系统应用，**directBootAware** 属性则适用于所有应用。

```
<application
    android:directBootAware="true"
    android:defaultToDeviceProtectedStorage="true">
```

应用级别的 **directBootAware** 属性的含义是将相应应用中的所有组件均标记为加密感知型组件。

**defaultToDeviceProtectedStorage** 属性用于将默认的应用存储位置重定向到 DE 存储空间（而非 CE 存储空间）。使用此标记的系统应用必须要仔细审核存储在默认位置的所有数据，并将敏感数据的路径更改为使用 CE 存储空间。使用此选项的设备制造商应仔细检查要存储的数据，以确保其中不含任何个人信息。

在这种模式下运行时，以下系统 API 可在需要时用于明确管理由 CE 存储空间支持的 Context（这些 API 与设备保护存储空间适用的同类 API 相对应）。

- `Context.createCredentialProtectedStorageContext()`
- `Context.isCredentialProtectedStorage()`

### 支持多位用户

多用户环境中的每位用户均会获得单独的加密密钥。每位用户均会获得两个密钥：一个 DE 密钥和一个 CE 密钥。用户 0 由于是特殊用户，因此必须要先登录设备。这部分适用于使用

设备管理功能 (/devices/tech/admin/index.html)的情况。

加密感知型应用按照以下方式与各用户进行互动：**INTERACT_ACROSS_USERS** 和 **INTERACT_ACROSS_USERS_FULL** 允许应用与设备上的所有用户互动。不过，这些应用只能访问已解锁用户的 CE 加密目录。

应用或许能够与各个 DE 区域自由互动，但一位用户已解锁并不意味着设备上的所有用户均已解锁。应用在尝试访问这些区域之前，应先检查解锁状态。

每个工作资料用户 ID 也会获得两个密钥：一个 DE 密钥和一个 CE 密钥。当满足工作挑战时，资料用户会被解锁，并且 Keymaster（在 TEE 中）可以提供资料的 TEE 密钥。

## 处理更新

恢复分区无法访问 userdata 分区中采用 DE 保护的存储空间。强烈建议实现 FBE 的设备支持 A/B 系统更新 (/devices/tech/ota/ab_implement) OTA 机制。由于可以在正常操作期间安装 OTA 更新，因此恢复分区无需访问已加密存储卷中的数据。

如果使用旧版 OTA 解决方案（该解决方案要求恢复分区访问 **userdata** 分区中的 OTA 文件），则需要执行以下操作：

1. 在 **userdata** 分区中创建一个顶级目录（例如“**misc_ne**”）。
2. 将该顶级目录添加到加密政策例外情况中（请参阅上文中的加密政策 (#encryption-policy)）。
3. 在顶级目录中创建一个用于存放 OTA 更新包的目录。
4. 添加 SELinux 规则和文件环境，以便控制对该文件夹及其内容的访问。应当只有接收 OTA 更新的进程或应用能够对该文件夹进行读取和写入操作。任何其他应用或进程都不应具有访问该文件夹的权限。

## 验证

为了确保实现的 FBE 功能版本能够按预期工作，需要部署多种 CTS 加密测试 (https://android.googlesource.com/platform/cts/+/master/hostsidetests/appsecurity/src/android/appsecurity/cts/DirectBootHostTest.java)

。

可以顺利为您的主板编译内核后，您还应该为 x86 编译内核并在 QEMU 下运行该内核，以便通过 kvm-xfstests 进行测试。对于 Ext4，请使用：

```
$ kvm-xfstests -c ext4/encrypt -g auto
```

对于 F2FS，请使用：

```
$ kvm-xfstests -c f2fs/encrypt -g auto
```

此外，设备制造商可以在启用了 FBE 的设备上进行以下手动测试：

- 检查 **ro.crypto.state** 是否存在
    - 确认 **ro.crypto.state** 是否已加密
- 检查 **ro.crypto.type** 是否存在
    - 确认 **ro.crypto.type** 是否已设为 **file**

此外，测试人员可以在主用户已设置锁定屏幕的情况下启动一个 **userdebug** 实例。然后通过 **adb** shell 命令进入设备，并使用 **su** 获得 root 权限。确认 **/data/data** 中是否包含加密的文件名；如果没有，则表示存在问题。

## AOSP 实现详情

本部分详细介绍了 AOSP 的文件级加密实现，并讲解了文件级加密的运作方式。设备制造商应该无需执行任何更改，即可在其设备上使用 FBE 和“直接启动”功能。

### fscrypt 加密

AOSP 实现会用到内核中的“fscrypt”加密（受 ext4 和 f2fs 支持），并配置为：

- 借助采用 XTS 模式的 AES-256 算法加密文件内容
- 借助采用 CBC-CTS 模式的 AES-256 算法加密文件名

### 密钥派生

磁盘加密密钥（512 位 AES-XTS 密钥）以加密形式存储：通过另一个存放在 TEE 中的密钥（256 位 AES-GCM 密钥）进行加密。要使用该 TEE 密钥，需要具备以下三项：

- 身份验证令牌

- 扩展凭据
- secdiscardable hash

身份验证令牌是一个经过加密身份验证的令牌，由 Gatekeeper (/security/authentication/gatekeeper) 在用户成功登录时生成。除非用户提供的身份验证令牌正确无误，否则 TEE 将拒绝用户使用该密钥。如果用户没有任何凭据，则不使用也不需要使用身份验证令牌。

扩展凭据是使用 `scrypt` 算法进行加盐和扩展处理后的用户凭据。实际上，凭据在被传递到 `vold`（以便传递到 `scrypt`）之前，会在锁定设置服务中接受一次哈希处理。扩展凭据会以加密形式绑定到 TEE 中的相应密钥，并享有适用于 `KM_TAG_APPLICATION_ID` 的所有保证。如果用户没有凭据，则不使用也不需要使用扩展凭据。

`secdiscardable hash` 是 16 KB 随机文件的 512 位哈希，和用于重建相应密钥的其他信息（例如种子）存储在一起。该文件会在相应密钥被删除时一并被安全删除，或者会以新的方式被加密；采用这种附加的保护措施后，攻击者要恢复相应密钥，必须要先恢复这个被安全删除的文件中的每一个位。secdiscardable hash 同样会以加密形式绑定到 TEE 中的相应密钥，并享有适用于 `KM_TAG_APPLICATION_ID` (/security/keystore/tags#application_id) 的所有保证。