全盘加密

注意：搭载 Android 7.0 - 9 的设备支持全盘加密。搭载 Android 10 及更高版本的新设备必须使用文件级加密 (/security/encryption/file-based.html)。

全盘加密是使用密钥（密钥本身也经过加密）对 Android 设备上的所有用户数据进行编码的过程。设备经过加密后，所有由用户创建的数据在存入磁盘之前都会自动加密，并且所有读取操作都会在将数据返回给调用进程之前自动解密数据。

全盘加密是在 Android 4.4 版中引入的，不过 Android 5.0 中又引入了以下新功能：

- 新增了快速加密方式，这种加密方式只会对数据分区中已使用的分块进行加密，以免首次启动用时过长。目前只有 EXT4 和 F2FS 文件系统支持快速加密。
- 添加了 `forceencrypt` fstab 标记 (/devices/storage/config.html)，以便在首次启动时进行加密。
- 添加了对解锁图案和无密码加密的支持。
- 添加了由硬件支持的加密密钥存储空间，该空间使用可信执行环境（TEE，例如 TrustZone）的签名功能。如需更多详细信息，请参阅存储已加密的密钥 (#storing_the_encrypted_key)。

注意：对于升级到 Android 5.0 的设备，如果升级之后进行了加密，则可以通过恢复出厂设置还原到未加密状态。在首次启动时加密的新 Android 5.0 设备无法还原到未加密状态。

Android 全盘加密的运作方式

Android 全盘加密基于在块设备层运行的内核功能 `dm-crypt`。因此，这种加密方式适用于以块设备的形式呈现给内核的嵌入式多媒体卡 (eMMC) 和类似闪存设备。YAFFS 会直接与原始 NAND 闪存芯片交互，无法进行全盘加密。

全盘加密采用的是 128 位高级加密标准 (AES) 算法（搭配加密块链 (CBC) 和 ESSIV:SHA256）。对主密钥进行加密时使用的是 128 位 AES 算法（通过对 OpenSSL 库的调用实现）。对于该密钥，您必须使用 128 位或更多位（可以选择 256 位）。

注意：原始设备制造商 (OEM) 可以使用 128 位或更多位对主密钥进行加密。

Android 5.0 版中有以下 4 种加密状态：

- 默认
- PIN 码
- 密码
- 解锁图案

首次启动时，设备会创建一个随机生成的 128 位主密钥，然后会使用默认密码和存储的盐对其进行哈希处理。默认密码是“default_password”。不过，设备还会通过 TEE（例如 TrustZone）为生成的哈希签名。TEE 会使用相应签名的哈希来加密主密钥。

您可以在 Android 开源项目的 cryptfs.cpp (https://android.googlesource.com/platform/system/vold/+/master/cryptfs.cpp) 文件中找到定义的默认密码。

当用户在设备上设置 PIN 码/通行码或密码时，只有 128 位的密钥会被重新加密并存储起来（也就是说，更改用户 PIN 码/通行码/解锁图案不会导致重新加密用户数据）。请注意，受管理的设备 (http://developer.android.com/guide/topics/admin/device-admin.html)可能有 PIN 码、解锁图案或密码限制。

加密操作由 **init** 和 **vold** 管理。 **init** 负责调用 **vold**，然后 vold 会设置相关属性以触发 init 中的事件。系统的其他部分也会查看这些属性以执行各项任务，例如报告状态、提示输入密码，或有严重错误发生时提示恢复出厂设置。为了调用 **vold** 中的加密功能，系统会使用命令行工具 **vdc** 的 **cryptfs** 命令：**checkpw**、**restart**、**enablecrypto**、**changepw**、**cryptocomplete**、**verifypw**、**setfield**、**getfield**、**mountdefaultencrypted**、**getpwtype**、**getpw** 以及 **clearpw**。

要加密、解密或清空 **/data**，**/data** 不得处于装载状态。但要显示任何界面，框架都必须启动，而框架需要 **/data** 才能运行。为了解决这一冲突，**/data** 上会装载一个临时文件系统。通过该文件系统，Android 可以提示输入密码、显示进度或根据需要建议清除数据。不过，该文件系统会带来以下限制：要从临时文件系统切换到实际的 **/data** 文件系统，系统必须停止所有在临时文件系统中打开了文件的进程，并在实际的 **/data** 文件系统中重启这些进程。为此，所有服务都必须位于以下其中一个组内：**core**、**main** 和 **late_start**。

- **core**：启动后一直不会关闭。
- **main**：关闭，然后在用户输入磁盘密码后会重启。
- **late_start**：在 **/data** 未解密并装载之前，一直不会启动。

要触发这些操作，请将 **vold.decrypt** 属性设置为各种字符串 (https://android.googlesource.com/platform/system/vold/+/master/cryptfs.c)。要结束和重启服务，请使用以下 **init** 命令：

- **class_reset**：停止相应服务，但允许通过 class_start 重启该服务。
- **class_start**：重启相应服务。
- **class_stop**：停止相应服务并添加 **SVC_DISABLED** 标记。被停止的服务不会对 **class_start** 做出响应。

流程

有 4 种针对已加密设备的流程。每台设备只需加密一次，然后会遵循正常的启动流程。

- 对之前未加密的设备进行加密：
  - 使用 **forceencrypt** 加密新设备：首次启动时强制加密（从 Android L 开始）。
  - 加密现有设备：由用户启动加密（Android K 及更低版本）。
- 启动已加密的设备：
  - 启动无密码的已加密设备：启动未设置密码的已加密设备（适用于运行 Android 5.0 及更高版本的设备）。
  - 启动设有密码的已加密设备：启动设置了密码的已加密设备。

除了上述流程外，设备还可能无法加密 **/data**。下文对上述每种流程进行了详细介绍。

使用 forceencrypt 加密新设备

这是 Android 5.0 设备首次启动时的常规流程。

1. **检测带有 forceencrypt 标记的未加密文件系统**

   /data 未加密，但需要加密，因为 **forceencrypt** 强制要求进行此项加密。卸载 **/data**。

2. **开始加密 /data**

   vold.decrypt = "trigger_encryption" 会触发 **init.rc**，从而使 **vold** 对 **/data** 进行无密码加密。（因为这应该是新设备，还没有设置密码。）

3. **装载 tmpfs**

   vold 会装载一个 tmpfs **/data**（使用 **ro.crypto.tmpfs_options** 中的 tmpfs 选项），并会将 **vold.encrypt_progress** 属性设为 0。 **vold** 会准备 tmpfs **/data** 以便启动已加密的系统，并会将 **vold.decrypt** 属性设为 **trigger_restart_min_framework**

4. **启动框架以显示进度**

   由于设备上几乎没有要加密的数据，加密过程很快就会完成，因此实际上通常并不会显示进度条。如需关于进度界面的更多详细信息，请参阅加密现有设备(#encrypt_an_existing_device)。

5. **`/data` 加密后，关闭框架**

   `vold` 会将 `vold.decrypt` 设为 `trigger_default_encryption`，这会启动 `defaultcrypto` 服务。（这会启动以下流程来装载默认的已加密用户数据。）`trigger_default_encryption` 会检查加密类型，以了解 `/data` 加密是否使用了密码。由于 Android 5.0 设备是在首次启动时加密，应该没有设置任何密码，因此我们要解密并装载 `/data`。

6. **装载 `/data`**

   接下来，`init` 会使用从 `ro.crypto.tmpfs_options`（在 `init.rc` 中设置）中选取的参数在 tmpfs RAMDisk 中装载 `/data`。

7. **启动框架**

   将 `vold` 设为 `trigger_restart_framework`，这会继续常规启动过程。

加密现有设备

当您加密之前搭载 Android K 或更低版本但已迁移至 L 的未加密设备时，则会发生该流程。

该流程由用户启动，在代码中称为“原地加密”。当用户选择对设备进行加密时，界面中将会提示用户确认电池是否已充满电并且交流电源适配器是否已插好，以便有充足的电量来完成加密过程。

警告：如果设备在完成加密之前耗尽电量并关机，文件数据将会处于部分加密状态。如果出现这种情况，必须将设备恢复出厂设置，而这将导致所有数据都会丢失。

为了进行原地加密，`vold` 会启动一个循环来读取实际块设备中每个扇区的数据，然后将其写入到加密块设备。在读取每个扇区的数据以及将其写入到加密块设备之前，`vold` 都会先检查相应扇区是否处于使用状态。对于几乎没有什么数据的新设备来说，这种方式可以大大加快加密速度。

设备状态：设置 `ro.crypto.state = "unencrypted"` 并执行 `on nonencrypted` `init` 触发器以继续启动。

1. **检查密码**

界面会使用 `cryptfs enablecrypto inplace` 命令调用 `vold`，其中 `passwd` 是用户的锁定屏幕密码。

2. **关闭框架**

   `vold` 会检查是否存在错误。如果无法加密，则返回 -1，并在日志中记录原因。如果可以加密，则会将 `vold.decrypt` 属性设为 `trigger_shutdown_framework`。这会使 `init.rc` 停止 `late_start` 类和 `main` 类中的服务。

3. **创建加密页脚**

4. **创建路径文件**

5. **重新启动**

6. **检测路径文件**

7. **开始加密 `/data`**

   接下来，`vold` 会设置加密映射，这将创建一个映射到实际块设备的虚拟加密块设备，但会在写入每个扇区的数据时对相应扇区进行加密，并在读取每个扇区的数据时对相应扇区进行解密。然后，`vold` 会创建并写出加密元数据。

8. **在加密时装载 tmpfs**

   `vold` 会装载一个 tmpfs `/data`（使用 `ro.crypto.tmpfs_options` 中的 tmpfs 选项），并会将 `vold.encrypt_progress` 属性设为 0。`vold` 会准备 tmpfs `/data` 以便启动已加密的系统，并会将 `vold.decrypt` 属性设为 `trigger_restart_min_framework`

9. **启动框架以显示进度**

   `trigger_restart_min_framework` 会使 `init.rc` 启动 `main` 类中的服务。当框架看到 `vold.encrypt_progress` 已设为 0 时，它会打开进度条界面，该界面每 5 秒查询一次该属性并更新进度条。加密循环会在已加密的分区比例每增加百分之一时更新一次 `vold.encrypt_progress`。

10. **`/data` 加密后，更新加密页脚**

    `/data` 成功加密后，`vold` 会清除元数据中的 `ENCRYPTION_IN_PROGRESS` 标记。

    当设备成功解锁后，接下来便会使用密码加密主密钥，并且会更新加密页脚。

    如果重新启动因某个原因失败了，`vold` 会将 `vold.encrypt_progress` 属性设为 `error_reboot_failed`，并且界面中应显示一条消息，提示用户按某个按钮以重新启动。这种情况不应发生。

启动采用默认加密的已加密设备

当您启动无密码的已加密设备时，则会发生该流程。由于 Android 5.0 设备是在首次启动时加密，应该没有设置任何密码，因此属于“默认加密”状态。

1. **检测无密码的已加密 `/data`**

   会发现 Android 设备已加密，因为 `/data` 无法装载，并且设置了 `encryptable` 或 `forceencrypt` 标记。

   `vold` 会将 `vold.decrypt` 设为 `trigger_default_encryption`，这会启动 `defaultcrypto` 服务。`trigger_default_encryption` 会检查加密类型，以了解 `/data` 加密是否使用了密码。

2. **解密 /data**

   基于块设备创建 `dm-crypt` 设备，接下来设备就可以使用了。

3. **装载 /data**

   然后，`vold` 会装载已解密的实际 `/data` 分区，并准备新的分区。它会将 `vold.post_fs_data_done` 属性设为 0，接着将 `vold.decrypt` 设为 `trigger_post_fs_data`。这会使 `init.rc` 运行其 `post-fs-data` 命令。这些命令会创建所有必要的目录或链接，然后将 `vold.post_fs_data_done` 设为 1。

   当 `vold` 看到该属性中的 1 时，会将 `vold.decrypt` 属性设为 `trigger_restart_framework`. 这会使 `init.rc` 再次启动 `main` 类中的服务，并启动 `late_start` 类中的服务（这是设备启动后首次启动这些服务）。

4. **启动框架**

   现在，框架会使用已解密的 `/data` 启动其所有服务，接下来系统就可以使用了。

启动未采用默认加密的已加密设备

当您启动设有密码的已加密设备时，则会发生该流程。设备的密码可以是 PIN 码、解锁图案或密码。

1. **检测设有密码的已加密设备**

   会发现 Android 设备已加密，因为设置了 `ro.crypto.state = "encrypted"` 标记

   由于 `/data` 是使用密码加密的，因此 `vold` 会将 `vold.decrypt` 设为 `trigger_restart_min_framework`。

2. **装载 tmpfs**

   `init` 会设置 5 个属性，以保存为 `/data`（包含从 `init.rc` 传入的参数）提供的初始装载选项。 `vold` 会使用这些属性来设置加密映射：

a. `ro.crypto.fs_type`

b. `ro.crypto.fs_real_blkdev`

c. `ro.crypto.fs_mnt_point`

d. `ro.crypto.fs_options`

e. `ro.crypto.fs_flags` （ASCII 码 8 位十六进制数字，以 0x 开头）

3. **启动框架以提示输入密码**

   框架会启动并看到 `vold.decrypt` 已设为 `trigger_restart_min_framework`。这让框架知道自己是在 tmpfs `/data` 磁盘中启动的，并且需要获取用户密码。

   不过，它首先需要确认磁盘是否已经过适当加密。它会向 `vold` 发送 `cryptfs cryptocomplete` 命令。 如果加密已成功完成，`vold` 会返回 0；如果发生内部错误，则会返回 -1；如果加密未成功完成，则会返回 -2。`vold` 通过查看 `CRYPTO_ENCRYPTION_IN_PROGRESS` 标记的加密元数据来确定应返回的值。如果设置了此标记，则表示加密过程中断了，并且设备上没有可用的数据。如果 `vold` 返回错误，界面中应显示一条消息，提示用户重新启动设备并将其恢复出厂设置，并且界面中应为用户提供一个用于执行该操作的按钮。

4. **通过密码解密数据**

   `cryptfs cryptocomplete` 成功后，框架会显示一个界面，提示用户输入磁盘密码。界面会向 `vold` 发送 `cryptfs checkpw` 命令来检查用户输入的密码。如果密码正确（通过以下方式判定：在临时位置成功装载已解密的 `/data`，然后将其卸载），`vold` 会将已解密块设备的名称保存在 `ro.crypto.fs_crypto_blkdev` 属性中，并向界面返回状态 0。如果密码不正确，则向界面返回 -1。

5. **停止框架**

   界面会显示加密启动图形，然后使用 `cryptfs restart` 命令调用 `vold`。`vold` 会将 `vold.decrypt` 属性设为 `trigger_reset_main`，这会使 `init.rc` 执行 `class_reset main` 命令。此命令会停止 main 类中的所有服务，以便卸载 tmpfs `/data`。

6. **装载 `/data`**

   然后，`vold` 会装载已解密的实际 `/data` 分区，并准备新的分区（如果加密时采用了首次发布不支持的数据清除选项，则可能永远无法准备就绪）。它会将 `vold.post_fs_data_done` 属性设为 0，接着将 `vold.decrypt` 设为 `trigger_post_fs_data`。这会使 `init.rc` 运行其 `post-fs-data` 命令。这些命令会创建所有必要的目录或链接，然后将 `vold.post_fs_data_done` 设为 1。当 `vold` 看到该属性中的 1 时，会将 `vold.decrypt` 属性设为 `trigger_restart_framework`。这会使 `init.rc` 再次启动 `main` 类中的服务，并启动 `late_start` 类中的服务（这是设备启动后首次启动这些服务）。

7. **启动整个框架**

   现在，框架会使用已解密的 `/data` 文件系统启动其所有服务，接下来系统就可以使用了。

失败

有一些原因可能会导致设备无法解密。设备会先按照一系列常规步骤启动：

1. 检测设有密码的已加密设备
2. 装载 tmpfs
3. 启动框架以提示输入密码

但在框架打开后，设备可能会遇到一些错误：

- 密码匹配但无法解密数据
- 用户输错密码的次数达到了 30 次

如果这些错误未解决，则会**提示用户清除数据并恢复出厂设置**：

如果 `vold` 在加密过程中检测到错误，并且任何数据都尚未被销毁，而框架处于打开状态，`vold` 会将 `vold.encrypt_progress` 属性设为 `error_not_encrypted`。界面中会提示用户重新启动系统，并提醒他们加密过程并未开始。如果错误发生在框架关闭之后、进度条界面显示之前，`vold` 会重新启动系统。如果重新启动失败，则会将 `vold.encrypt_progress` 设为 `error_shutting_down` 并返回 -1；但却无法捕获相应错误。这种情况不应发生。

如果 `vold` 在加密过程中检测到错误，则会将 `vold.encrypt_progress` 设为 `error_partially_encrypted` 并返回 -1。随后，界面中应显示一条消息，告诉用户加密失败，并且界面中应为用户提供一个用于将设备恢复出厂设置的按钮。

存储已加密的密钥

已加密的密钥存储在加密元数据中。硬件支持是通过使用可信执行环境 (TEE) 的签名功能实现的。以前在加密主密钥时，需要使用通过对用户的密码和存储的盐应用 scrypt 生成的密钥。为了使该密钥能够抵御盒外攻击，我们通过使用存储的 TEE 密钥为生成的密钥签名，扩展了这种算法。然后，通过再次应用 scrypt，生成的签名会转变成具有适当长度的密钥。该密钥随后会用于加密和解密主密钥。存储该密钥的步骤如下：

1. 生成 16 个字节的随机磁盘加密密钥 (DEK) 和 16 个字节的盐。
2. 对用户密码和盐应用 scrypt，以生成 32 个字节的中间密钥 1 (IK1)。

3. 为 IK1 填充若干个零字节，以便达到绑定到硬件的私钥 (HBK) 的大小。具体来说就是按照以下方式进行填充：00 || IK1 || 00..00；1 个零字节，32 个 IK1 字节，223 个零字节。
4. 使用 HBK 为已填充的 IK1 签名，以生成 256 个字节的 IK2。
5. 对 IK2 和盐（与第 2 步中使用的盐相同）应用 scrypt，以生成 32 个字节的 IK3。
6. 将 IK3 的前 16 个字节用作 KEK，后 16 个字节用作 IV。
7. 使用 AES_CBC、密钥 KEK 和初始化矢量 IV 加密 DEK。

更改密码

当用户选择在设置中更改或移除密码时，界面会向 `vold` 发送 `cryptfs changepw` 命令，然后 `vold` 会使用新密码重新加密磁盘主密钥。

加密属性

`vold` 和 `init` 之间通过设置属性进行通信。下面列出了可用的加密属性。

vold 属性

属性	说明
`vold.decrypt trigger_encryption`	以无密码方式加密存储卷。
`vold.decrypt trigger_default_encryption`	检查存储卷是否采用了无密码加密。如果是，则解密并装载存储卷；如果不是，则将 `vold.decrypt` 设为 trigger_restart_min_framework。
`vold.decrypt trigger_reset_main`	由 vold 设置，用于关闭提示输入磁盘密码的界面。
`vold.decrypt trigger_post_fs_data`	由 vold 设置，用于准备具有必要目录等内容的 `/data`。
`vold.decrypt trigger_restart_framework`	由 vold 设置，用于启动实际框架和所有服务。
`vold.decrypt trigger_shutdown_framework`	由 vold 设置，用于关闭整个框架以开始加密。

`vold.decrypt trigger_restart_min_framework`	由 vold 设置，用于启动加密进度条界面或提示输入密码，具体取决于 `ro.crypto.state` 的值。
`vold.encrypt_progress`	框架启动时，如果设置了此属性，则会进入进度条界面模式。
`vold.encrypt_progress 0 to 100`	进度条界面中应按照设置显示百分比值。
`vold.encrypt_progress error_partially_encrypted`	进度条界面中应显示一条消息，告诉用户加密失败，并且界面中应为用户提供一个用于将设备恢复出厂设置的按钮。
`vold.encrypt_progress error_reboot_failed`	进度条界面中应显示一条消息，告诉用户加密已完成，并且界面中应为用户提供一个用于重新启动设备的按钮。此错误不应发生。
`vold.encrypt_progress error_not_encrypted`	进度条界面中应显示一条消息，告诉用户发生错误，没有已加密的数据或数据已丢失，并且界面中应为用户提供一个用于重新启动系统的按钮。
`vold.encrypt_progress error_shutting_down`	进度条界面未运行，因此不清楚谁将响应此错误。在任何情况下，都不应发生此错误。
`vold.post_fs_data_done 0`	由 `vold` 在将 `vold.decrypt` 设为 `trigger_post_fs_data` 的前一刻设置。
`vold.post_fs_data_done 1`	由 `init.rc` 或 `init.rc` 在完成 `post-fs-data` 任务之后立即设置。

init 属性

属性	说明
`ro.crypto. fs_crypto_blkdev`	由 `vold` 命令 `checkpw` 设置，供 `vold` 命令 `restart` 以后使用。
`ro.crypto.state unencrypted`	由 `init` 设置，用于说明相应系统正在未加密的 `/data ro.crypto. state encrypted` 中运行。由 `init` 设置，用于说明相应系统正在已加密的 `/data` 中运行。
`ro.crypto.fs_type ro.crypto.fs_real_blkdev ro.crypto.fs_mnt_point ro.crypto.fs_options ro.crypto.fs_flags`	这 5 个属性由 `init` 在尝试装载 `/data`（包含从 `init.rc` 传入的参数）时设置。`vold` 会使用这些属性来设置加密映射。
`ro.crypto.tmpfs_options`	由 `init.rc` 设置，包含 init 在装载 tmpfs `/data` 文件系统时应使用的选项。

init 操作

```
on post-fs-data
on nonencrypted
on property:vold.decrypt=trigger_reset_main
on property:vold.decrypt=trigger_post_fs_data
on property:vold.decrypt=trigger_restart_min_framework
on property:vold.decrypt=trigger_restart_framework
on property:vold.decrypt=trigger_shutdown_framework
on property:vold.decrypt=trigger_encryption
on property:vold.decrypt=trigger_default_encryption
```