元数据加密

Android 7.0 及更高版本支持文件级加密 (/security/encryption/file-based) (FBE)。采用 FBE 时，可以使用不同的密钥对不同的文件进行加密，并且可以对加密文件进行单独解密。这些密钥可用于加密文件内容和文件名。采用 FBE 时，其他信息（例如目录布局、文件大小、权限和创建/修改时间）不会被加密。这些统称为“文件系统元数据”。

Android 9 引入了对存在硬件支持的元数据加密的支持。借助元数据加密，启动时出现的单个密钥会加密未通过 FBE 进行加密的任何内容。该密钥受到 Keymaster 的保护，而 Keymaster 受到启动时验证功能的保护。

实现

您可以在搭载 Android 9 的新设备上设置元数据加密，只需设置元数据文件系统，更改 init 序列，然后在设备的 fstab 文件中开启元数据加密即可进行设置。

硬件要求

元数据加密只能在数据分区首次进行格式化时设置。因此，该功能仅适用于新设备；OTA 不应更改此设置。

为了支持当前的元数据加密，您的硬件必须支持使用内嵌加密引擎 (https://blog.google/topics/connected-workspaces/pixel-security-better-faster-stronger/)进行文件级加密。**`fstab.hardware`** 中的用户数据分区的 **`fileencryption=ice`** 指令指明了这一点。

此外，内核中必须存在并启用 **`dm-default-key`** 模块。

设置元数据文件系统

由于在元数据加密密钥出现之前，用户数据分区中的所有内容均无法读取，因此分区表必须留出一个名为“元数据分区”的单独分区，用于存储保护该密钥的 Keymaster Blob。该元数据分区的大小应为 16MB。

`fstab.hardware` 必须为该分区上的元数据文件系统纳入一个条目，并将其装载到 **`/metadata`**（包括 **`formattable`** 标记），以确保在启动时对其进行格式化。f2fs 文件系统不适用于较小的分区；在较小分区中，我们建议您改为使用 ext4。例如：

```
/dev/block/bootdevice/by-name/metadata            /metadata          ext4
```

要确保 **/metadata** 装载点存在，请在 **BoardConfig-common.mk** 中添加下面这行代码：

```
BOARD_USES_METADATA_PARTITION := true
```

更改 init 序列

在使用元数据加密时，必须在装载 **/data** 之前运行 **vold**。为了确保其提前足够长的时间开始运行，请将以下节添加到 **init.hardware.rc** 中：

```
# We need vold early for metadata encryption
on early-fs
    start vold
```

Keymaster 必须在 init 尝试装载 **/data** 之前运行并准备就绪。

init.hardware.rc 应该已经包含一个 **mount_all** 指令，用于将 **/data** 本身装载到 **on late-fs** 节中。请在这行代码前面添加以下指令，以执行 **wait_for_keymaster** 服务：

```
on late-fs
   …
    # Wait for keymaster
    exec_start wait_for_keymaster

    # Mount RW partitions which need run fsck
    mount_all /vendor/etc/fstab.${ro.boot.hardware.platform} --late
```

开启元数据加密

最后，将 **keydirectory=/metadata/vold/metadata_encryption** 添加到用户数据的 **fstab.hardware** 条目中：

```
/dev/block/bootdevice/by-name/userdata             /data             f2fs
```

验证

在实现元数据加密时，请注意以下常见问题并测试您的实现。

常见问题

在调用 **mount_all**（用于装载元数据加密的 **/data** 分区）时，**init** 会执行 vdc 工具。vdc 工具会通过 **vold** 连接到 **binder**，以设置元数据加密的设备并装载分区。在此调用期间，**init** 会被拦截，并且在 **mount_all** 完成之前，尝试读取或设置 **init** 属性的操作也会被拦截。在此阶段，如果 **vold** 的任何一部分工作在读取或设置某个属性时被直接或间接拦截，则会导致死锁。请务必确保 **vold** 能够完成读取密钥、与 Keymaster 交互以及装载数据目录的工作，而无需与 **init** 进一步交互。

如果 Keymaster 在 **mount_all** 运行时没有完全启动，就不会响应 **vold**，直到从 **init** 读取到某些属性为止，从而导致上述死锁。按照相关规定将 **exec_start wait_for_keymaster** 放置在相关的 **mount_all** 调用之前，可确保 Keymaster 提前完全运行，从而避免此类死锁。

元数据加密测试

我们会将这些测试放到上游，与此同时，请向 **Android.bp** 中添加几行代码，并将 **check_encryption.cpp** 添加到 **platform/system/vold** (https://android.googlesource.com/platform/system/vold/+/master) 以测试您的实现。

对 Android.bp 的更改

下面列出了对 **Android.bp** 的更改。

```
...
}

cc_binary {
    name: "vold",
    defaults: [
        "vold_default_flags",
        "vold_default_libs",
    ],

    srcs: ["main.cpp"],
    static_libs: ["libvold"],
    product_variables: {
        arc: {
            static_libs: [
                "arc_services_aidl",
```

```
                "libarcobbvolume",
            ],
        },
    },
    init_rc: [
        "vold.rc",
        "wait_for_keymaster.rc",
    ],

    required: [
        "check_encryption",
        "mke2fs",
        "vold_prepare_subdirs",
        "wait_for_keymaster",
    ],
}
...

}

cc_binary {
    name: "check_encryption",
    defaults: ["vold_default_flags"],
    srcs: [
        "FileDeviceUtils.cpp",
        "check_encryption.cpp",
    ],
    shared_libs: [
        "libbase",
    ],
}
```

添加 check_encryption.cpp

将 `check_encryption.cpp` 添加到 **platform/system/vold** (https://android.googlesource.com/platform/system/vold/+/master) 中。

```
/*
 * Copyright (C) 2017 The Android Open Source Project
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 *      http://www.apache.org/licenses/LICENSE-2.0
```

```
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */
#include "FileDeviceUtils.h"

#include <cmath>
#include <string>

#include <assert.h>
#include <stdio.h>

#include <android-base/file.h>
#include <android-base/logging.h>
#include <android-base/unique_fd.h>
using android::base::unique_fd;
using android::base::ReadFileToString;
using android::base::WriteStringToFile;

namespace android {
namespace vold {
const size_t sectorsize = 1 << 12;
const int sectorcount = 1024;
static double randomness_score(const std::string& checkme) {
    unsigned int freq[256] = {0};
    unsigned int sum = 256;
    double loglaplace = 0;
    for (auto b : checkme) {
        loglaplace -= 8 + log2(static_cast<double>(++freq[static_cast<uint8_t>
    }
    return loglaplace;
    LOG(INFO) << "Score: " << loglaplace;  // if negative, not random
    return loglaplace < 0;
}
static bool run_test(const std::string& device) {
    unique_fd device_fd(open(device.c_str(), O_RDONLY | O_CLOEXEC));
    if (device_fd.get() == -1) {
        PLOG(ERROR) << "Failed to open " << device;
        return false;
    }
    int randompassed = 0;
    auto buf = std::string(sectorsize, '\0');
    for (int i = 0; i < sectorcount; i++) {
        auto l = read(device_fd.get(), &buf[0], buf.size());
        if (l < 1) {
```

```
            PLOG(ERROR) << "Failed read on sector " << i;
            return false;
        }
        if (((size_t)l) != buf.size()) {
            LOG(ERROR) << "Short read on sector " << i;
            return false;
        }
        auto score = randomness_score(buf);
        if (score >= 0) {
            randompassed++;
            LOG(INFO) << "Passed randomness check on sector " << i << " with s
        } else {
            LOG(ERROR) << "Failed randomness check on sector " << i << " with
        }
    }
    LOG(INFO) << "Passed randomness check on " << randompassed << "/" << secto
    return randompassed == sectorcount;
}
}  // namespace vold
}  // namespace android
int main(int argc, const char* const argv[]) {
    setenv("ANDROID_LOG_TAGS", "*:v", 1);
    android::base::InitLogging(const_cast<char**>(argv), android::base::Stderr
    if (argc != 2) {
        LOG(ERROR) << "Usage: " << argv[0] << " <device>";
        LOG(ERROR) << "example: " << argv[0] << " /dev/block/bootdevice/by-nam
        return -1;
    }
    android::vold::run_test(std::string(argv[1]));
    return 0;
}
```