android Bootcamp 2016

New App Lifecycle for Encryption

January 21, 2015

Agenda

- File-based Encryption
- Integration with Android
- User Data Layout Changes
- App Lifecycle Changes

File-based Encryption

android

Why encrypt?

- Protect our users against extraction of data from lost/stolen/appropriated devices
- To avoid more subtle physical attacks, critical data must be encrypted with user’s credentials

Full disk encryption

- Meets requirement that data be encrypted at rest
- Credential requirement is a challenge:
    - Only one user per device can be protected properly
    - That user must log in before anything works
    - Does not lend itself to a great user experience
- No obvious way of solving any of these issues with full disk encryption

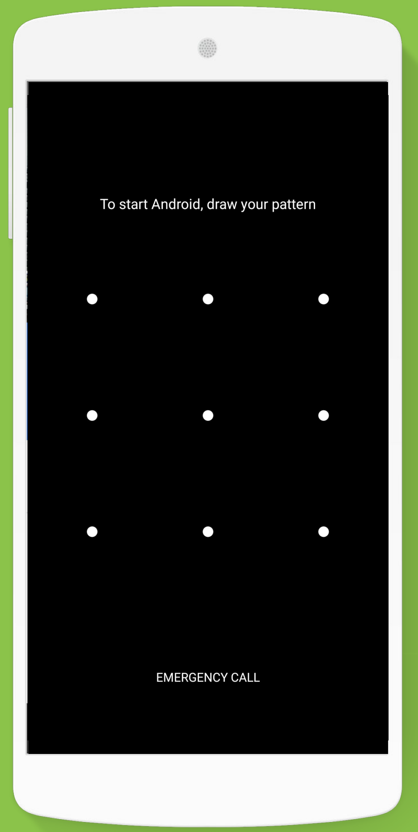
File-based encryption

- Added to `ext4` by Michael Halcrow and Theodore T'so
  - Encrypts file contents with AES-256 in XTS mode
  - File names are also encrypted
  - Policy is applied to a folder and all subfolders
- Performance is slightly better on average than full disk encryption
- Backported to Linux 3.18 common kernel (and Linux 3.10 kernels for testing)
- `f2fs` encryption is also available

Integration with Android

Motivation

- Alarm clock never goes off
- Incoming calls/SMS are forgotten
- Only a handful of built-in “core” apps, no third-party accessibility services
- Wi-Fi configuration is missing
- Bluetooth pairing data is missing



android

Storage areas

Device encrypted (DE)

Files encrypted with a key that is only available after a device has performed a successful verified boot.

There is a single **DE_{sys}** key for the system, and a separate **DE_n** key for each user to support fast wipe.

Credential encrypted (CE)

Files encrypted with a key that is entangled with user credentials, such as PIN, pattern, or password. Only available after a user has presented their credentials.

There is a separate **CE_n** key for each user.

Not encrypted (NE)

Files not encrypted at all, which should be extremely rare. OTA update files are one example.

User Data Layout Changes

android

Layout goals

Apps

- Default app storage must be **CE_n** to keep legacy apps secure
- Apps can access a new **DE_n** storage area

Media

- Internal shared storage must be **CE_n**

System

- Default system storage can be **DE_{sys}** to make migration easier
- System can access new **CE_n** and **DE_n** storage areas to better protect data

Typical layout

DE_{sys}	`/data/system`
DE_0	`/data/system_de/0`
CE_0	`/data/system_ce/0`
CE_0	`/data/user/0`
DE_0	`/data/user_de/0`

CE_0	`/data/media/0`
DE_{sys}	`/data/misc`
DE_0	`/data/misc_de/0`
CE_0	`/data/misc_ce/0`
NE	`/data/misc_ne`

System storage

- System internals persist a lot of data!
  - By default it's probably stored in **DE_{sys}**
  - Look for data that is strongly associated with a user, and migrate to either **DE_n** or **CE_n**, depending on when it's needed

- **You** are responsible for auditing all data stored by your device
  - Consider unexpected data sources, such as temporary camera files

- So how do you decide if data should be **DE** or **CE**?

Triage examples

DE

- Wi-Fi credentials, Bluetooth pairing data
- Alarm clock details
- Wallpaper, active ringtones

CE

- Account auth tokens/passwords
- Contacts, calendar, SMS history, call log
- Location/browsing history
- Recent tasks, screenshots

System APIs

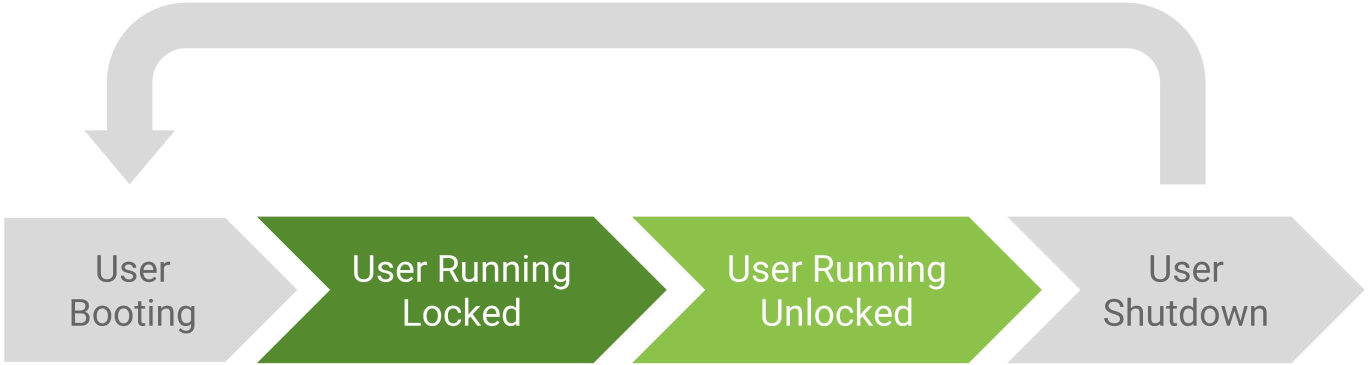
- Inside the system server, paths can be constructed using:
  - `Environment.getUserSystemDirectory(int userId)` **DE_{sys}**
  - `Environment.getUserSystemDeviceEncryptedDirectory(int userId)` **DE_n**
  - `Environment.getUserSystemCredentialEncryptedDirectory(int userId)` **CE_n**
  - `Environment.getMiscNotEncryptedDirectory()` **NE**

App APIs

- By default, Context points at **CE_n** storage, but you can obtain a Context that redirects all file operations to **DE_n** storage:
  - `Context.createDeviceEncryptedStorageContext()`

- System apps can change their default Context to point at **DE_n** storage, and then use a similar method to get back to **CE_n** storage:
  - `<application android:forceDeviceEncrypted=”true”>`
  - `Context.createCredentialEncryptedStorageContext()`
  - **Extreme care** must be used with this flag; all data stored by the app must be audited, and this is only designed to make migration easier

App Lifecycle Changes



- When running locked, only **DE_{sys}** and **DE_n** storage are available
    - Only apps that are *encryption aware* can be run safely
- When running unlocked, **DE_{sys}**, **DE_n**, and **CE_n** storage are available
    - All apps can be run safely
    - User must go through full shutdown to eject **CE_n** keys

Encryption aware apps

- Apps can explicitly mark components as being *encryption aware*, which signals that they can safely run while **CE_n** is unavailable.
  - Includes activities, services, receivers, providers
  - `<receiver ... android:encryptionAware=”true”>`

- All `PackageManager` queries are filtered based on the user state:
  - When locked, **only** encryption aware components are matched
  - When unlocked, **both** encryption aware and unaware components are matched
  - Flags can be used to override this matching behavior when needed

App APIs

- New `UserManager.isUserUnlocked()` API to detect current state
- When user enters locked state:
  - New `LOCKED_BOOT_COMPLETED` broadcast is sent
- When user enters unlocked state:
  - Components started while locked are left running without being killed
  - Existing `BOOT_COMPLETED` broadcast is sent
  - New `USER_UNLOCKED` foreground broadcast is sent
  - Encryption unaware providers are started in already running apps

Common Questions

What if my device doesn’t support file-based encryption?

Devices without file-based encryption support will still need to provide the various filesystem paths described earlier, even if they’re all backed by a single encryption key. These devices can immediately transition from the “running locked” to “running unlocked” state when starting users. Together these ensure we give developers a consistent experience through broadcasts and storage APIs.

THANK YOU