Analyzing Android's File-Based Encryption

Information Leakage through Unencrypted Metadata

Tobias Groß
tobias.gross@cs.fau.de
Friedrich-Alexander University
Erlangen-Nuremberg, Germany

Matanat Ahmadova
s6ahmata@uni-bonn.de
University of Bonn
Bonn, Germany

Tilo Müller
tilo.mueller@cs.fau.de
Friedrich-Alexander University
Erlangen-Nuremberg, Germany

ABSTRACT

We investigate the amount of information leakage through unencrypted metadata in Android's *file-based encryption* (FBE) which was introduced as an alternative to the previously dominating *full-disk encryption* (FDE) in Android 7.0. We propose a generic method, and provide appropriate tooling, to reconstruct forensic events on Android smartphones encrypted with FBE. Based on a dataset of 3903 applications, we show that metadata of files can be used to reconstruct the name, version and installation date of all installed apps. Furthermore, we show that, depending on a specific app, information leakages through metadata can even be used to reconstruct a user's behavior. For the example of WhatsApp, we show that the point of time a user sent or received her last message can be traced back even though the phone was encrypted. Our approach requires access to the raw data of an encrypted disk only but does not require access to a powered-on device or the bootloader, such as known attacks against FDE including *cold boot* and *evil maid*. We conclude that FBE is significantly more insecure than FDE and was presumably elected for usability reasons like direct boot.

KEYWORDS

Android, File-Based Encryption, Digital Forensics, Metadata

ACM Reference Format:
Tobias Groß, Matanat Ahmadova, and Tilo Müller. 2019. Analyzing Android's File-Based Encryption: Information Leakage through Unencrypted Metadata. In *Proceedings of the 14th International Conference on Availability, Reliability and Security (ARES 2019) (ARES '19), August 26–29, 2019, Canterbury, United Kingdom.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3339252.3340340

1 INTRODUCTION

Android is the most widespread OS for smartphones in the world, with a worldwide market share of 87% at the end of 2018.[1] To protect data on modern smartphones against physical access, Android 7.0 introduced *file-based encryption* (FBE) as an alternative to *full-disk encryption* (FDE), which was available since Android 4.0. FBE enables *direct boot* to allow encrypted devices to boot straight to the lock screen. Previously, with FDE users needed to provide credentials before any data could be accessed, preventing the phone from performing all but the most basic of operations.[2]

Despite the gain in usability, we show that FBE is more insecure than FDE. Our results and methods are of interest from two points of view. On the one hand, state authorities like police have a legitimate interest to subvert data encryption for their investigations and crime prevention. Today, IT systems are not only used in cases of cybercrime but also in traditional crime. The data on smartphones play an increasingly important role to give evidence about the general procedure of crimes. On the other hand, users of smartphones have a legitimate interest of keeping their sensitive data private and secret against unauthorized access, including both criminals and illiberal states.

1.1 Our Contribution

To the best of our knowledge, we are the first investigating the amount of data leakage through unencrypted metadata in Android's file-based encryption. Given full access to the raw disk of an FBE-enabled device, our contributions are:

- Based on the IT forensics toolchain *sleuthkit*, we provide a tool to extract the metadata of files from ext4 partitions. Those data include directory layouts, file sizes, permissions, and creation/modification times.
- Based on the unencrypted metadata we retrieve with our tool, we propose a method to reconstruct a list of

[1]https://www.idc.com/promo/smartphone-market-share/os
[2]https://source.android.com/security/encryption/file-based

installed apps from a device. Our analysis is based on a set of fingerprints from 3903 apps.

- For WhatsApp, we exemplarily show how to even reconstruct a user's behavior from FBE-enabled devices. This step, however, is highly app-specific and naturally limited.

Note: *During our work, which was based on a Nexus 5X with Android 8.0, the latest version of Android 9.0 was released. According to Google, Android 9.0 introduces support for metadata encryption where hardware support is present. As a consequence, the flaws we identified are not applicable on up-to-date Android 9.0 smartphones, proving that Google was aware about privacy concerns of FBE independent of our results.*

1.2 Related Work

Loftus et al. [6] give an overview of Android's file-based encryption and investigate whether known attacks against full-disk encryption are still applicable. As a result, they state that some attacks appear still feasible on Android 7.0 and later. In contrast to the work by Loftus et al. [6], we investigate a new vulnerability of FBE, not present on FDE-enabled devices, namely, how to extract unencrypted metadata and exploit it to, e.g., reconstruct usage behavior.

Other works [9–11] evaluate the security of Android's FDE and proposed novel methods for hardening FDE. Müller et al. [8] implemented a tool called FROST to extract the disk encryption key from a cold booted Android device. In contrast, Götzfried and Müller [3] implemented a method to harden Android's FDE encryption against cold boot attacks. They also showed the vulnerability of Android to evil maid attacks. All this work shows the importance of the security research on disk encryption methods, but until today was limited to FDE.

Garfinkel [2] developed a tool called fiwalk for extracting filesystem structures and metadata into XML files. We adapted this concept and implemented our own version of fiwalk which in addition extracts ext4 specific and FBE related metadata.

Research about reconstructing usage behavior from a filesystem's metadata was performed by the following papers: James et al. [4] generated signatures for identifying user actions in Microsoft Internet Explorer, Mozilla Firefox and Microsoft MSN Messenger. The generated signatures are based on Windows registry modifications and changes to the filesystem's metadata. Kälber et al. [5] have used a fingerprinting approach for reconstructing user actions in different Windows apps. As fingerprints they used NTFS timestamp changes and evaluated their approach on Mozilla Thunderbird and ICQ. In contrast to the previous work, we extracted fingerprints from metadata for Android apps and despite FBE being in place.

2 BACKGROUND

In this section we give necessary background information about file-based encryption (2.1), app fingerprints (2.2), and the file and folder structure of Android devices (2.3).

2.1 File Based Encryption

Starting with Android 7.0, users can encrypt the userdata partition with FBE instead of FDE. Technically Android's FBE is implemented as a feature of the ext4 FS. The basic element in ext4 is an *inode*, which represents and stores metadata of a file or folder. Metadata include modify, access, create or change (MAC) timestamps, ownership information, size and a generation ID. File and folder names are not stored in an inode structure and are hence not present unencrypted.

Inodes hold pointers to the content data and to more metadata called extended attributes. The extended attributes are used to store data of filesystem extensions. In the case of FBE they also manage important data for the encryption. Other important metadata for our work is the generation ID. Generation IDs are used on NFS to distinguish new files reusing an inode from an older, already deleted file. The combination of an inode and generation ID is unique for the filesystem's lifetime.

File and folder names are only part of a folder's content data. Figure 1 shows a simplified example of a folder containing a file. The content data of a folder is a list of names and inode reference. These entries define the name of child files/folders and link to their inode. This allow to nest folders and to build up the filesystem structure. With FBE the content of each file or folder gets encrypted with an individual per-file-key. Since file names are the content data of the parent folder, all names of files in the same folder get encrypted with the same key, namely the parent folders per-file-key.

The per-file-key of a file is inherited from a master key. For this process the extended attributes hold a random nonce and a key descriptor defining which master key to use. The per-file-key is the result of the encryption using AES128-ECB of the nonce with the master key. In our experiments AES256-XTS is used to encrypt the content data of a file with the per-file-key. For the folder entries AES256-CBC-CTS is used with the per-file-key of the folder to encrypt the file and folder names.

There are two master keys in a single user Android system. One is used for files which are needed after booting the device without unlocking to support basic functionality like emergency calls. These files are called *device encrypted* (DE). The DE master key is tied to the physical device. The second master key is for files which are only usable after unlocking the device with the user's passcode. These files are called *credential encrypted* (CE) as their master key is inherited of the passcode.

2.2 App Fingerprints

To reconstruct user activities on an Android smartphone even without decrypting the userdata partition, we take the concept of app fingerprints from Kälber et al. [5] and James et al. [4]. The idea is that every user interaction with an app leaves patterns in the filesystem metadata. This is because apps often need to access data from files like program code and resources or modify or create new information stored in files. All file accesses and modifications are reflected in a file's MAC timestamp which is part of the metadata. Every user interaction of an app might touch those timestamps. For the case of instant messengers like WhatsApp, sending and receiving text messages or photos likely modifies a local database. Sending a photo must access the sent media file and receiving a photo must create a new media file.

Furthermore, we differentiate between *ordinary fingerprints* and *characteristic fingerprints*. Characteristic fingerprints are changes in the metadata which are only triggered by one specific action. Ordinary fingerprints are those which can also emerge from multiple user actions. Characteristic fingerprints are more valuable than ordinary fingerprints because they cannot be overlaid when other actions are executed after the action with characteristic fingerprints. This allows for reconstructing the event with characteristic fingerprints regardless other events happened afterwards.

2.3 Files and Folder Structure

Android apps, in contrast to desktop apps on Windows or Linux, are strictly sandboxed, meaning that apps are restricted to their own folders when modifying, accessing or creating files. The userdata partition is typically mounted at `/data`, holding all third-party apps. We distinguish between static and dynamic app data. Static app data consists of program code, libraries and resources like graphics and sounds. The static app data is stored in a folder located under `/data/app` at install or update time and never gets changed afterwards. Dynamic app data can change after install time. There are several places where dynamic app data is stored, including `/data/misc/profiles/{cur, ref}/appName`, `/data/ user_de` and `/data/data/appName`. For our work only `/data/app` is used for installation event reconstruction and `/data/data` for user action reconstruction.

3 ATTACKER MODEL

We assume an "attacker" (who can impersonate both sides, criminals and legitimate authorities) having physical access to an Android 7.0 or 8.0 device with FBE-enabled. This attacker is able to access the raw data of the encrypted disk, either through chip-off data recovery or logical access due to exploits or unlocked bootloaders. Modern smartphones like the Nexus 5X uses eMMC flash memory as persistent disk memory. Etemadieh et al. [1] showed that it is possible to dump the data from an unsoldered eMMC chip with relatively little effort. Even for NAND chips, which can also be used as persistent memory, there exist methods for dumping data from the chip [7].

4 IMPLEMENTATION

For our implementation we used a Nexus 5X device with Android 8.0.0 installed. We implemented our own version of fiwalk [2], which outputs all files and folders to a file in the DFXML[3] format. Our version of fiwalk also includes the ext4 specific metadata generation ID and FBE related metadata like nonce, master key descriptor and cipher.

4.1 Installation Event Reconstruction

To reconstruct installation events, we built a database of app fingerprints. We identified the location of an app's static files which do not change in size or content after installation and used these files to fingerprint apps. The static data of an app is structured as pictured in figure 2. On the root level of an app's folder is the `base.apk`. This file is an archive for the whole app data. Subfolders of the `lib` folder store native libraries used by the app. These libraries get extracted from the `base.apk` at install time. So as time of installation we just take the creation timestamp of the `base.apk`. After installing this file, all native libraries get extracted and last, the app code gets compiled.

The *Android runtime* (ART) compiles program code of an app to machine code, typically known as *ahead-of-time* (AOT) compilation and stores it in files located inside child folders of `oat`. File sizes and content may differ little from one installation to another. Therefore, we cannot use these files for fingerprints. Instead, we used the file sizes of `base.apk` and native libraries, as these files are static in size even on different devices. To identify apps, our fiwalk module iterates over all apps in `/data/app` and tries to match the file sizes with a fingerprint inside the database. To exclude AOT files from matching without file and folder names, we exclude them based on the file count of a folder and the cr-times of files. In the `oat` folder are always two or three files, see figure 2, while in the `lib` folder an arbitrary number of files is stored.

[3]https://github.com/dfxml-working-group/dfxml_schema

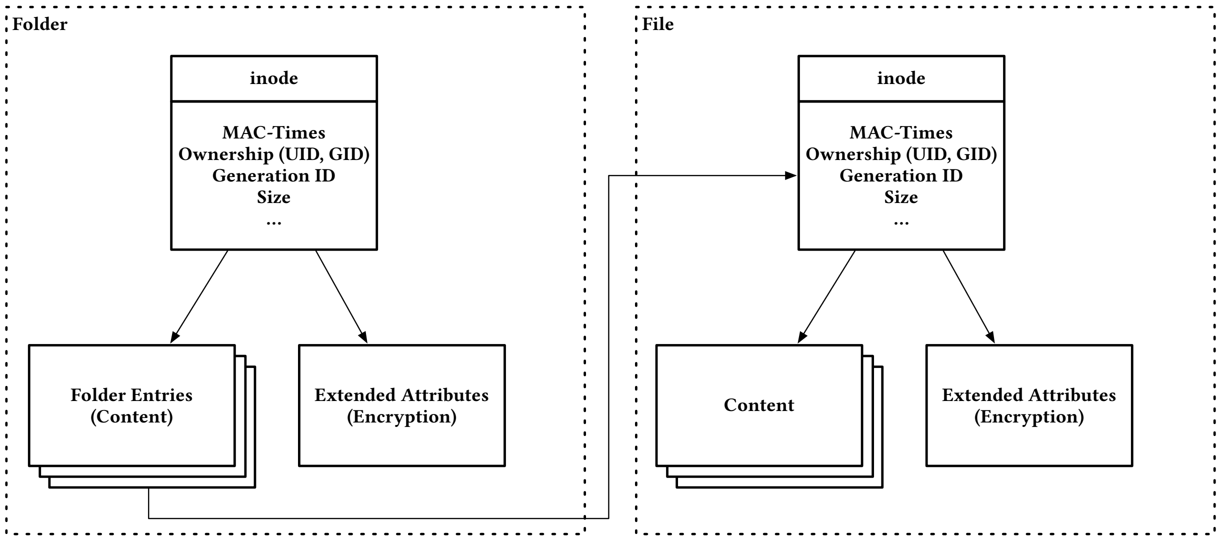

Figure 1: Simplified file and folder organization in ext4 filesystem

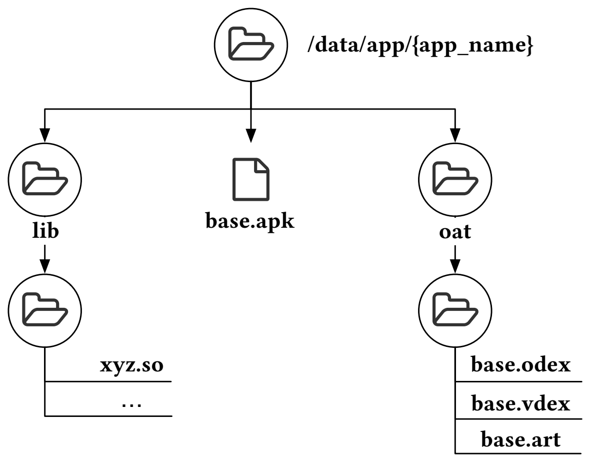

Figure 2: Layout of static app data

4.2 Dynamic Data Folder Identification

Our linking tool identifies all app-related folders despite the app folder at `/data/app` which is identified by the first module and only stores static data. To reconstruct user actions in apps we need dynamic data which gets identified by this tool. The linking tool takes the userdata partition file/folder structure, metadata and the inode of the app folder in `/data/app` and outputs the inodes of other app-related folders created during app installation. We investigated the installation process of an app and identified the following folders, in which app related folders get created during installation: `/data/data`, `/data/user_de`, `/data/misc/profiles/cur/0` and `/data/misc/profiles/ref`. But in the end we used the behavior of generation IDs to identify all these app-related folders, as described in the following.

The first generation ID ready for allocation is chosen randomly at each device boot. When creating a new file, this file gets the initial generation ID and then the generation ID is incremented. The following applies for two installed apps a and b, where b was installed after a:

$$\forall a \in A, b \in B : G(a) < G(b)$$

where $G(x)$ is the generation ID of a folder x and the set A consists of app-related child folders directly under `/data/app`, `/data/data`, `/data/user_de`, `/data/misc/profiles/cur/0` and `/data/misc/profiles/ref`. Analogous, the set B consists of folders for an app b.

So we can identify the set A of all app-related folders of app a by knowing the folder in `/data/app` checking the following condition:

$$A = \{x \in F \mid G(x) > G(z) \wedge G(x) < G(z) + \epsilon\}$$

where F is the set of all subfolders of `/data/data`, `/data/user_de`, `/data/misc/profiles/cur/0` and `/data/misc/profiles/ref` and z the app related subfolder in `/data/app`.

4.3 User Action Reconstruction

To reconstruct user actions in apps we have to create a database with characteristic fingerprints for each of the targeted actions. The database creation can be done offline and once-only. For simplicity, the generation should be performed on a device with folder and filenames in cleartext and access to the whole filesystem. This is the case at devices with no

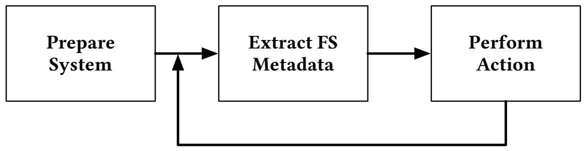

Figure 3: Fingerprint generation process

disk encryption enabled and raw storage device access, e.g., through an unlocked bootloader or rooted device.

The process of fingerprint generation is shown in figure 3. In the prepare system phase, the apps we want to generate fingerprints for get installed and set ready to use. To extract the metadata, we reboot the device into the bootloader and boot a TWRP recovery image to dump the complete metadata and file structure in the extract filesystem metadata phase. At the end of each cycle, we boot into the Android system and perform the action we want to identify characteristic fingerprints.

To find only changes in the metadata which are related to the chosen action, we have to perform each cycle multiple times to segregate noise from background processes and system tasks. The changes in the filesystem occurred during an app's installation are extracted by comparing the dumps of cycle n with $n+1$ with the *idifference*[4] tool which returns a list of new, changed, modified and deleted files. Fingerprints related to the app action are those which occurred in nearly all cycles.

After identifying fingerprints for different actions, we can calculate the characteristic fingerprints of all actions. Characteristic are all those metadata changes which only occurred for one action. These characteristic fingerprints have more significance because they are not triggered by other actions.

The general process of matching fingerprints operates as follows. We extract the metadata of the target device and then divide the files into timeframes by their MAC-times. This provides us with a list of modified, accessed and created files for each timeframe. Last, we can match each timeframe with the fingerprint database. We get a hit if all characteristic traces of an action are contained in a timeframe and we can reconstruct the action to be happened at the timeframe.

The part of matching an exhibit's userdata partition to reconstruct user actions is highly dependent on the app and the identified fingerprints. The fingerprints use file and folder names to clearly identify which file's and folder's metadata change by executing an action.

[4]https://www.forensicswiki.org/wiki/Fiwalk

5 EVALUATION

In this section we evaluate our method for installation event reconstruction and adapt our concepts of user action reconstruction to WhatsApp.

5.1 Installation Event Reconstruction

For evaluating our installation event reconstruction approach, we used a Google Play Store crawler for half a year. Regularly, we downloaded the top 20 of free apps for every store category. For all apps once downloaded we checked for new versions and also downloaded these. At the end we had 1326 different popular apps and 850 of them in at least two different versions. In total we used 3903 different APKs for our installation event reconstruction evaluation.

In the first part of our evaluation we built a database of app fingerprints with these 3903 different APKs. We utilized a rooted Nexus 5X device and installed the APKs with only one version of an app at a time. Afterwards we extracted the file sizes of `base.apk` and all libraries to store them in our database.

We checked the uniqueness of the fingerprint for each specific app version. In our test set we encountered no two different apps with the same fingerprint. We encountered only 26 apps having the same fingerprint for different app versions. In conclusion this means for the reconstruction process that it is extremely unlikely that an installation event for a wrong app gets reconstructed. Only the reconstruction of the exact app version remains uncertain.

In the second part we run tests on our reconstruction tool. In total we randomly selected 132 apps from our pool and installed them on a test device. Among the selected apps are also some of them which have no unique fingerprint for different versions. For all installed apps we identified the correct app name and installation time in the region of seconds. Also, the correct app version gets reconstructed, for those with no unique fingerprint all matching versions get suggested.

5.2 WhatsApp User Action Reconstruction

Last we conducted a case study to show that we can reconstruct user actions in WhatsApp using unencrypted metadata only. This case study consists of multiple steps. First, we investigated which fingerprints are left behind on the metadata for different user actions. Afterwards, we selected fingerprints which originate in files which we can identify on an FBE-enabled device using structural information. Last, we conducted some tests to show that we can match those fingerprints on our FBE test device and reconstruct user actions in WhatsApp.

In the first step we run the fingerprint generation process, presented in section 4.3 for the following user actions:

- send message to an existing conversation (CSM)
- send photo to existing conversation (CSP)
- send message (new conversation) (NM)
- send photo (new conversation) (NP)
- receive message and open conversation (RM)
- receive photo and open conversation (RP)
- open WA and close in task manager (OC)
- receive photo without opening WA (PNP)
- receive message without opening WA (PNM)

We performed the generation process ten times for every user action to track files which change timestamps during user action or get created. Afterwards we seek for characteristic fingerprints of investigated actions. We could not identify characteristic fingerprints in our set of user actions for WhatsApp. In the next step we selected fingerprints with files we can locate on an FBE-enabled device with no file and folder names. In our fingerprints we discovered that for every action, WhatsApp's databases get updated timestamps.

After the installation, the WhatsApp folder in `/data/data/` is empty except the two symbolic links `cache` and `code_cache`. On the first start of WhatsApp six folders are created in the specific order: `app_minidumps`, `app_traces`, `databases`, `files`, `no_backup` and `shared_prefs`. All WhatsApp databases get created in the `databases` folder. On the first start, eight database files get created, each time in the exact same order. After the registration of WhatsApp, 19 database files are present, all in the same order each time WhatsApp is used. More databases are created on demand when specific WhatsApp features are used by the user, like opening the emoji list for the first time or receiving media files for the first time. Interesting for our case, when a text message is received the first time *web_session.db* and *web_session.db-journal* are created. This creation can happen at the same time as media databases are created. Then web session files get created first.

We cluster our unknown database files by creation time and generation ID and by knowing the creation patterns we can identify each database file. The optional databases can be identified by the different number per cluster. There we distinguish the case when web session databases get created together with media databases and the case when these files get created separately. In the first case they build a cluster with five files and in the second case one cluster with three and one cluster with two files. We evaluated our WhatsApp specific approach for user event reconstruction together with the linking tool mentioned in section 4.2 and our app fingerprint database. For the following event reconstruction, we limit to two combined user actions, because we cannot differentiate more user actions only using database files. We match for send/receive text event which changes only mtime of *axolotl.db* and send/receive photo event which changes mtime in *axolotl.db* and *media.db*.

We tested the reconstruction with ten runs per action combination for these ten different action combinations: CSM - CSP, CSP - CSM, RM - CSM, RM - CSP, RP - CSM, RP - CSP, FP - CSM, PNP, PNM and NM. We were able to identify the static WhatsApp data with our app fingerprint database for all 100 runs. Furthermore, the linking tool identified WA's */data/data* folder correctly and our WhatsApp specific database identification approach was able to find the required databases in all runs. For all runs we reconstructed send/receive photo event if the last action was CSP and PNP as expected. If these actions were performed before the last action, we could only match the fingerprints for send/receive photo partly. If those partly matched fingerprints are characteristic for send/receive photo, then we are also able to reconstruct the event with certainty.

6 SUMMARY

Summarizing, we investigated the amount of information leakage on Android FBE-enabled devices through metadata left unencrypted. In digital forensics, our insights could be used by state authorities to reconstruct a list of installed apps and user behavior in apps, not only limited to WhatsApp. Apparently, Google was aware of these security shortcomings and introduced metadata encryption in Android 9.0.

REFERENCES

[1] Amir Etemadieh, CJ Heres, and Khoa Hoang. 2017. Hacking Hardware With $10 SD Card Reader. https://bh2017.exploitee.rs/Hacking_Hardware_With_A_10_Reader-wp.pdf Blackhat 2017.

[2] Simson L. Garfinkel. 2009. Automating Disk Forensic Processing with SleuthKit, XML and Python. In *Fourth International IEEE Workshop on Systematic Approaches to Digital Forensic Engineering, SADFE 2009, Berkeley, California, USA, May 21, 2009*. 73–84. https://doi.org/10.1109/SADFE.2009.12

[3] Johannes Götzfried and Tilo Müller. 2014. Analysing Android's Full Disk Encryption Feature. *Journal of Wireless Mobile Networks, Ubiquitous Computing and Dependable Applications* 5 (2014), 84–100.

[4] Joshua James, Pavel Gladyshev, and Yuandong Zhu. 2013. Signature Based Detection of User Events for Post-Mortem Forensic Analysis. *CoRR* abs/1302.2395 (2013). http://arxiv.org/abs/1302.2395

[5] Sven Kälber, Andreas Dewald, and Felix C. Freiling. 2013. Forensic Application-Fingerprinting Based on File System Metadata. In *Seventh International Conference on IT Security Incident Management and IT Forensics, IMF 2013, Nuremberg, Germany, March 12-14, 2013*. 98–112. https://doi.org/10.1109/IMF.2013.20

[6] Ronan Loftus, Marwin Baumann, Rick van Galen, and Rachelle de Vries. 2017. Android 7 File Based Encryption and the Attacks Against It. https://www.semanticscholar.org/paper/Android-7-File-Based-Encryption-and-the-Attacks-It-Loftus-Baumann/e1cf9ad5614f3a24b24088e4b22e9218f0abc3a0.

[7] Igor Mikhaylov. 2016. Chip-Off Technique in Mobile Forensics. https://www.digitalforensics.com/blog/chip-off-technique-in-mobile-forensics/ Accessed: 18.03.2019.

[8] Tilo Müller, Michael Spreitzenbarth, and Felix C. Freiling. 2014. Frost: Forensic Recovery of Scrambled Telephones. In *International Conference on Applied Cryptography and Network Security*.

[9] Adam Skillen, David Barrera, and Paul C. van Oorschot. 2013. Deadbolt: Locking Down Android Disk Encryption. In *Proceedings of the Third ACM Workshop on Security and Privacy in Smartphones & Mobile Devices (SPSM '13)*. ACM, New York, NY, USA, 3–14. https://doi.org/10.1145/2516760.2516771

[10] Peter Teufl, Andreas Fitzek, Daniel Hein, Alexander Marsalek, Alexander Oprisnik, and Thomas Zefferer. 2014. Android encryption systems. In *2014 International Conference on Privacy and Security in Mobile Systems (PRISMS)*. 1–8. https://doi.org/10.1109/PRISMS.2014.6970599

[11] Zhaohui Wang, Rahul Murmuria, and Angelos Stavrou. 2012. Implementing and Optimizing an Encryption Filesystem on Android. In *2012 IEEE 13th International Conference on Mobile Data Management*. 52–62. https://doi.org/10.1109/MDM.2012.31