REFERENCE ARCHITECTURE FOR ANDROID APPLICATIONS TO SUPPORT THE DETECTION OF MANIPULATED EVIDENCE

H. Pieterse,[*†] **M.S. Olivier**[†] **and R.P. van Heerden**[‡]

[*] *Defence, Peace, Safety and Security, Council for Scientific and Industrial Research, Pretoria, South Africa E-mail: hpieterse@csir.co.za*

[†] *Department of Computer Science, University of Pretoria, Pretoria, South Africa E-mail: molivier@cs.up.ac.za*

[‡] *Meraka Institute, Council for Scientific and Industrial Research, Pretoria, South Africa E-mail: rvheerden@csir.co.za*

Abstract: Traces found on Android smartphones form a significant part of digital investigations. A key component of these traces is the date and time, often formed as timestamps. These timestamps allow the examiner to relate the traces found on Android smartphones to some real event that took place. This paper performs exploratory experiments that involve the manipulation of timestamps found in SQLite databases on Android smartphones. Based on observations, specific heuristics are identified that may allow for the identification of manipulated timestamps. To overcome the limitations of these heuristics, a new reference architecture for Android applications is also introduced. The reference architecture provides examiners with a better understanding of Android applications as well as the associated digital evidence. The results presented in the paper show that the suggested techniques to establish the authenticity and accuracy of digital evidence are feasible.

Key words: Digital forensics, mobile forensics, smartphones, Android, timestamps, reference architecture.

1. INTRODUCTION

The past decade saw the rapid improvement of smartphone technology, allowing these devices to become very popular across the globe. Their current prominence is directly related to the provided capabilities and functionality, which nowadays closely resemble a personal computer. Bundled with a complete operating system, improved connectivity and communication functions, and the option of adding additional third-party applications, smartphones have become powerful devices. The leading smartphone operating system (OS) of 2014 was Android [1], which has been evolving in a remarkable way and continues to gain widespread popularity. The current prevalence of Android led this paper to focus only on this particular smartphone OS.

The extensive and wide use of Android smartphones allows these devices to become a rich source of trace evidence [2]. All events occurring on Android smartphones generate traces that form an important component of digital investigations, especially when the user of the smartphone is involved in criminal activities. The valuable information (such as contacts, text messages, call lists, website visited or instant messages) contained in these traces can provide a well-defined snapshot of a user's actions at a specific time. Besides providing a description of the event, traces found on Android smartphones also often store the time and date component in the form of a timestamp. Timestamps are integral to digital investigations since it provides the examiner the opportunity to relate the traces found on the Android smartphones to some physical event that took place.

To conceal fraudulent activities, smartphone users can use certain techniques to manipulate the timestamps of the traces and change the associated events. These techniques are referred to as Anti-forensics and are primarily used "*to compromise the availability or usefulness of evidence*" [3]. These techniques are applied by smartphone users in an attempt to either hide or change event logs, which results in the alteration of the timestamps associated with those events.

It is thus important for examiners to be able to verify the authenticity and accuracy of timestamps. Without such verification, the collected timestamps might be incorrect or inaccurate due to tampering and will lead the examiner to make unreliable conclusions. Existing research shows few papers that attempt to offer a solution regarding the verification of the authenticity and accuracy of timestamps. Verma et. al. [4] preserve date and time stamps by capturing the system generated modification, access, change and/or creation date and timestamps (MAC DTS) values and storing it in a secure location such as a cloud server outside of the smartphone. The cloud snapshot of the original MAC DTS values can be used to verify the authenticity of MAC DTS values of questionable files on the smartphone [4]. Govindaraj et. al. [5] designed a solution, called iSecureRing, which allows a jailbroken iPhone to be secure and forensic ready by preserving the timestamps. These timestamps are stored outside the device on a secure server or the cloud and can be used during security incidents [5]. Both solutions, however, require the installation of additional functionality on the smartphone prior to seizing the device for investigation. There is, thus, no existing solution (to the best of the

Based on: "Playing Hide-and-Seek: Detecting the Manipulation of Android Timestamps", by H. Pieterse, M.S. Olivier and R. Van Heerden which appeared in the Proceedings of Information Security South African (ISSA) 2015, Johannesburg, 12 & 13 August 2015. © 2015 IEEE

authors' knowledge) that allows for the verification of timestamps collected from seized Android smartphones.

This paper performs exploratory experiments that involve the manipulation of timestamps found in SQLite databases on Android smartphones. While conducting the experiments, the changes occurring on the Android smartphone are observed. Based on those observations, specific heuristics are identified that may indicate the manipulation of timestamps. The identified heuristics can be categorised into two distinct groups. The first group of heuristics are specific changes that occur on the Android file system and the second group refer to inconsistencies in individual SQLite databases. These heuristics are, however, susceptible to external factors that can impact their availability. To further establish the accuracy and authenticity of the timestamps as well as other forms of digital evidence, a new reference architecture for Android applications is also introduced. The purpose of the reference architecture is to provide the examiners with a better understanding of Android applications and how the associated digital evidence originated. The immediate challenges to address are the following: (a) effective manipulation of timestamps found in SQLite databases on Android smartphones, (b) verifying the authenticity of these timestamps by using the identified heuristics and (c) introducing a newly designed reference architecture for Android applications. The current paper provides preliminary evidence that, in terms of the challenges identified, the suggested approach shows potential.

The remainder of the paper is structured as follows. Section 2 briefly describes the architecture of Android and the internal structure of SQLite databases. Section 3 presents the methodology followed to conduct the exploratory experiments and offers a descriptive summary of the findings. The reference architecture for Android applications is introduced in Section 4. A short discussion and future developments of the research is presented in Section 5. The final conclusions are made in Section 6.

2. BACKGROUND

With the continuous growth in functionality of Android smartphones, increasing number of people make use of these devices during their daily activities. For the traces collected by Android smartphones to be of use during digital investigations, a comprehensive understanding of the architecture of Android is required. An evaluation of SQLite is also required, since most of the traces found on Android smartphones are stored in SQLite databases. This section, therefore, provides a short introduction of the Android architecture and presents the internal structure of SQLite databases.

2.1 Android's Architecture

Android is a popular open source software architecture provided by the Open Handset Alliance [8] that is currently targeting mobile devices, such as smartphones and tablet computers. The Android software architecture (see Figure 1) is divided into five layers: Applications, Application Framework, Libraries, Android Runtime and the Linux Kernel [9]. The uppermost layer, Applications, provides access to a set of core applications. The Application Framework layer implements a software framework that reassembles functions used by existing applications. All available libraries are written in C/C++ and called through a Java interface. The Android runtime consists of a set of core libraries and a Dalvik virtual machine. The bottommost layer is the Linux kernel, which allows for interaction between the upper layers by means of device drivers [9, 10].

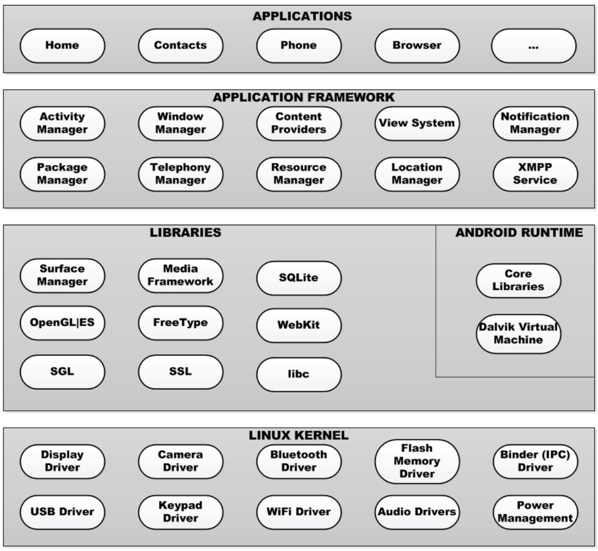


Figure 1: Architecture of the Android Operating Systems [6, 7]

Until Android version 2.2 (*Froyo*), most Android smartphones used Yet Another Flash File System 2 (YAFFS2) [11]. YAFFS2 was developed in 2004 in response for larger sized NAND (Not-AND) flash devices [12]. With the release of Android version 2.3 (*Gingerbread*), the file system for Android devices switched from YAFFS2 to Fourth Extended (EXT4) file system [11]. YAFFS2 was developed with a single-threaded design, which may cause bottlenecks in devices released with a multi-core chipset. The EXT4 file system, which is one of the most used file systems in Linux, does not have this limitation and can run smoothly on multi-core devices. The disk space of the EXT4 file system is divided into logical blocks, which reduce management overhead and improves throughput [13]. The key features of the EXT4 file system promote the development of advance applications and functionalities.

The architecture of Android regularly improves to support more improved applications. It is therefore necessary to continuously evaluate Android's architecture and remain up to date with the current changes.

Table 1: User Data Stored in SQLite Databases

User Data	SQLite Database Location	Table
Call History	/data/data/com.sec.android.provider.logsprovider/databases/log.db	logs
Messages (SMS/MMS)	/data/data/com.android.providers.telephony/databases/mmssms.db	sms
E-mails (Gmail)	/data/data/com.google.android.gm/databases/mailstore.<name@gmail.com>.db	messages
Google Hangouts	/data/data/com.google.android.talk/databases/babel1.db	messages
WhatsApp Messenger	/data/data/com.whatsapp/databases/msgstore.db	messages

2.2 SQLite Databases

SQLite is an open source software library that implements a lightweight Structured Query Language (SQL) database engine for embedded use [14, 15]. The lightweight design of SQLite does not require a separate server and thus allows for the quick processing of stored data by reading and writing directly to a disk file [16]. The main database file, *<database_name>.db* or *<database_name>.db3*, consists out of a complete SQL structure that includes tables, indices, triggers, and views [16]. To support the SQL structure, the main database file is divided into one or more pages and each page share the same size [17]. The first page of the main database file is called the header page and is composed of the database header and the schema table. The database header stores structural information and the schema table contain the table information of the database. The pages following the header page are structured as B-trees and store the actual data [18].

During transactions, SQLite stores additional information in a second file called either a rollback journal or write-ahead log (WAL) file [17]. The rollback journal is the default method of SQLite to implement an atomic commit and rollback. Beginning with SQLite version 3.7.0, the new WAL approach was introduced and allowed for improved speed and concurrent execution. The WAL approach preserves the original content in the main database file and appends changes to a separate WAL file (*<database_name>.db-wal*), which contains a header and zero or more WAL frames. Transferring the transactions from the WAL file to the main database file is called a "checkpoint". When a checkpoint occurs the updated or new pages in the WAL file are written to the main database file. The checkpoint operation leaves the WAL file untouched, allowing the WAL file to be reused rather than deleted [19]. SQLite does a checkpoint automatically when a file reaches a size of 1000 pages [20].

SQLite databases are a popular choice for data storage in Android applications [14]. An Android application, which uses SQLite, separately includes the SQLite databases and this allows for reduced external dependencies and minimized latency [21]. A lot of events taking place on an Android smartphone generate valuable traces, for example: call history, SMS/MMS messages, e-mails (Gmail) and instant messages generated by Google Hangouts (previously Google Talk) as well as WhatsApp Messenger. A summary of the SQLite databases used to store these traces, as well as the location of these databases on an Android smartphone are provided in Table 1. The examples used throughout the remainder of this paper focus on the SQLite database of the Messaging application.

3. DETECTION OF MANIPULATED TIMESTAMPS

Timestamps of traces found on Android smartphones are integral to digital investigations, especially if the owner of the smartphone participates in criminal activities. Collected timestamps allow the examiner to relate the traces to some physical event and, more importantly, establish a timeline depicting the chronological order of events. Due to the importance of timestamps in digital investigations, smartphone users, or even malicious applications, can alter timestamps to compromise the integrity of traces as evidence.

In order to detect manipulated timestamps, it is necessary to understand the processes involved in the creation of the timestamps. Understanding these processes provide the required insight to manipulate timestamps found in SQLite databases. The exploratory experiments conducted throughout this paper can be categorised into two groups. The first group of experiments observe normal operation of SQLite databases while the second group focuses on detecting changes occurring due to the manipulation of timestamps found in the SQLite databases. All of the experiments and observations were performed on a Samsung Galaxy S2, running Android version 4.1.2 (*Jelly Bean*). The experiments conducted are not limited to the Samsung Galaxy S2 and can be performed on any other Android smartphone.

3.1 Observing SQLite Databases

To understand the underlying structure of SQLite databases and comprehend the operations involved in the creation of timestamps, it is necessary to observe these databases under normal conditions. Applications using SQLite databases to store user-related data are located in the */data/data/<application_package_name>/databases/* directories on an Android smartphone [22]. Observing the SQLite databases located in the */data* directory is not permitted by default and is only accessible by rooting the Android smartphone. The term rooting, which is similar to the Jailbreaking of an iPhone, is often perceived as a negative action [12]. Rooting an Android smartphone merely means to escalate the current rights to root access rights. Root access rights allow any user access to the root directory (/) and provide the necessary permissions to take root actions [12]. The technical process of rooting an Android smartphone is, however, beyond the scope of this paper. The Samsung Galaxy S2, which is used to observe the results of interacting with the SQLite databases, is already rooted and therefore provides access to the required databases.

The purpose of observing SQLite databases is to identify how these databases react and function under normal operations. The observations are made by monitoring the directory containing the SQLite database while simultaneously interacting with the database. Interactions with the SQLite databases occur by sending messages, such as text and instant messages, and making phone calls. After conducting a set of 10 experiments using the rooted Samsung Galaxy S2 smartphone, the final conclusion of the observations led to the identification of several changes occurring as a direct result of the interaction with the SQLite databases. Firstly, from the observations it is possible to infer that all of the data received during the interactions are stored in the *<database_name>.db-wal* file. The data are only transferred from the *<database_name>.db-wal* file to the *<database_name>.db* file when reaching the limit of 1000 pages. Secondly, the timestamps associated with the data are added as new entries, with a unique record identifier, at the end of the database table.

A summary of the findings of observing the SQLite databases is the identification of the files that are altered during normal operations and the location where the timestamps are stored in the tables. The insight gained by observing the SQLite databases will assist with the manipulation of timestamps.

3.2 *Manipulation of Individual Timestamps in SQLite Databases*

Manipulation of timestamps found in SQLite databases requires access to the correct files. Observations from the previous exploratory experiments identified the correct files to be the *<database_name>.db* and *<database_name>.db-wal* files. Access to these files requires the enabling of the Universal Serial Bus (USB) debugging functionality [12]. Although the default setting for USB debugging is "disabled", going to Settings, selecting Developer options and touching the checkbox next to USB debugging will turn on this feature. Once USB debugging is enabled, interaction with the root directory can occur using the Android debug bridge (adb). Android Debug Bridge is a versatile command-line tool that communicates with a connected Android smartphone [23]. To allow for complete access to the root directory, adb is restarted with the command, *adb root*, to gain root permission. Full access, with the necessary permissions, has been established and it is now possible to manipulate the timestamps in SQLite databases.

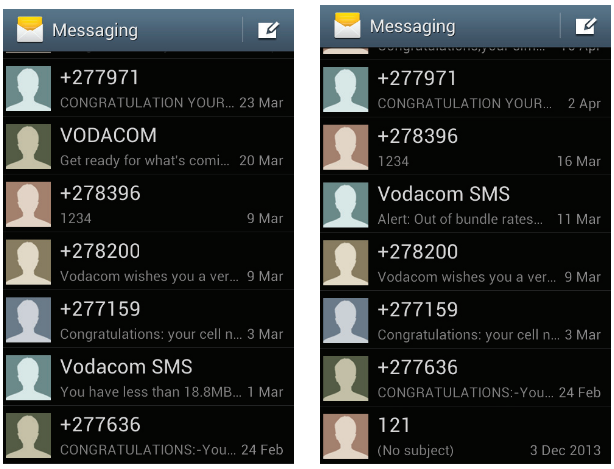
Manipulation of timestamps proceeds through three individual phases: retrieve, manipulate, and return. The first phase retrieves the required SQLite database files from the Android smartphone. Since the sqlite3 command utility, which is required to interact with the SQLite databases, does not come pre-installed on Android smartphones [24], the SQLite databases must be transferred to the local computer. The command, *adb pull <remote><local>*, copies the specified file from the Android smartphone to the local computer [23]. It is necessary to repeat this command for both the main database file, as well as the associated WAL file, which cannot be edited directly. Retrieving both the main database file and the associated WAL file ensures that all the latest records are present. The list of commands to retrieve these files is shown below:

- *adb pull /data/data/<application_package_name> /databases/<database_name>.db C:\<local_folder>*
- *adb pull /data/data/<application_package_name> /databases<database_name>.db-wal C:\<local_folder>*

Manipulating the timestamps found in the copied SQLite database files is performed during the second phase. A script is created to act as a malicious application and randomly manipulate timestamps within the SQLite database. During the execution of the script the main database file is opened, allowing for a checkpoint to occur. Once the execution of the script is completed, only the main database file (*<database_name>.db*) remains and must be returned to the Android smartphone.

The final phase returns the modified SQLite database to the Android smartphone. The command, *adb push <local><remote>*, copies the specified file to the connected Android smartphone [23]. To prevent the changes from being over-written by the existing data in the *<database_name>.db-wal* file, the file must be removed. The first step is to start an interactive shell using *adb shell*, followed by *su*, which provides root permissions within the shell. The next command, *cd /data/data/<application_package_name>/databases/*, change the current working directory to the directory containing the main database and associated WAL file. Using the command *rm <database_name>.db-wal* will delete the WAL file from the directory. For the changes to reflect on the Android smartphone, it is necessary to reboot the device.

Using the Messaging application as an example, Figure 2 provides a snapshot of the messages before and after the manipulation of the timestamps. The comparison shows significant changes to the dates of specific messages and an adjustment of the order of the messages. These changes occur as a direct result of manipulating the timestamps. Close observation during the manipulation allow for the identification of the altered database files (*<database_name>.db* and *<database_name>.db-wal*). Examining the main database file shows that the individual records are ordered incorrectly. These findings are further discussed in the following section.



Messaging

+277971
CONGRATULATION YOUR... 23 Mar

VODACOM
Get ready for what's comi... 20 Mar

+278396
1234 9 Mar

+278200
Vodacom wishes you a ver... 9 Mar

+277159
Congratulations: your cell n... 3 Mar

Vodacom SMS
You have less than 18.8MB... 1 Mar

+277636
CONGRATULATIONS:-You... 24 Feb

(a) Original timestamps

Messaging

+277971
CONGRATULATION YOUR... 2 Apr

+278396
1234 16 Mar

Vodacom SMS
Alert: Out of bundle rates... 11 Mar

+278200
Vodacom wishes you a ver... 9 Mar

+277159
Congratulations: your cell n... 3 Mar

+277636
CONGRATULATIONS:-You... 24 Feb

121
(No subject) 3 Dec 2013

(b) Manipulated timestamps

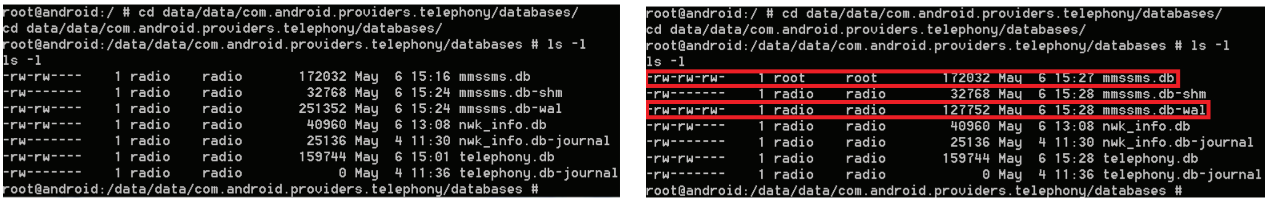
Figure 2: Messaging application showing messages with the original (a) and manipulated (b) timestamps

3.3 Discussion of Findings and Identification of Heuristics

The findings and observations of the exploratory experiments, conducted to identify changes occurring due to the manipulation of timestamps, can be categorised into two groups. The first group contains a collection of heuristics that identifies the presence of certain changes in the Android file system, which are indicators of the manipulation of the SQLite database. The second group subsequently focuses on the individual SQLite databases and the identification of inconsistencies in these databases. The presence of specific file system changes as well as inconsistencies in the associated SQLite database indicates that the authenticity of the timestamps might be compromised.

Android File System Changes: Android File System Flags (AFS-Flags) are indicators of the potential tampering of SQLite databases on Android smartphones. Each AFS-Flag represents a change that occurs in the Android file system due to the modification or removal of a SQLite database or any other associated database files. The presence of any of these AFS-Flags is not an indication of the manipulation of timestamps but merely that the SQLite databases have been tampered with. The following four individual AFS-Flags offer guidance regarding the tampering of SQLite databases:

- **File permissions:** associated with the SQLite database files in the directory of a specific application are set to give only read/write access to the file owner and the group members. For each application, the current file owner and group members are only the individual application. Any modification or removal of a file within this directory will change the existing file permissions of the modified file from *-rw-rw—-* to *-rw-rw-rw-*. The following changes to the file permissions are therefore an indication of the possible manipulation of timestamps in the SQLite databases:
  - File permission of *<database_name>.db* file changed from *-rw-rw—-* to *-rw-rw-rw-*.
  - File permission of *<database_name>.db-wal* file changed from *-rw-rw—-* to *-rw-rw-rw-*.
- **Ownership:** of the SQLite databases is given to the specific application using the database. The file owner and group members are thus set to the user ID (UID) of the application, which is unique and specific to the application. The UID remains constant for the duration of the application on the particular Android smartphone. Modifications to any SQLite database files will result in a change of ownership and subsequently change the UID of both the file owner and group members. The following change to the ownership of the main database file is a possible indication of the manipulation the databases:
  - The current ownership for the file owner and group members of the *<database_name>.db* file changed from the current *UID* to *root*.
- **File Size:** of the main database file is expected to be smaller than the size of the associated WAL file, since all new transactions are appended to the WAL file. The size of the main database file is only expected to grow after a checkpoint, when all the transactions from the WAL file is transferred to the main database file. A checkpoint, however, occurs only after the WAL file accumulated 1000 entries (leading to a file size of approximately 4MB), and thus the size of the main database remains relatively small. An automatic checkpoint occurs when the main database file is opened to allow for the manipulation of the timestamps. The WAL file must be deleted to prevent the changes made in the main database file from being overwritten by the existing content located in the WAL file. Once the Android smartphone is rebooted to reflect the changes, a new WAL file is automatically



```
root@android:/ # cd data/data/com.android.providers.telephony/databases/
cd data/data/com.android.providers.telephony/databases/
root@android:/data/data/com.android.providers.telephony/databases # ls -l
ls -l
-rw-rw----    1 radio    radio       172032 May  6 15:16 mmssms.db
-rw-------    1 radio    radio        32768 May  6 15:24 mmssms.db-shm
-rw-rw----    1 radio    radio       251352 May  6 15:24 mmssms.db-wal
-rw-rw----    1 radio    radio        40960 May  6 13:08 nwk_info.db
-rw-------    1 radio    radio        25136 May  4 11:30 nwk_info.db-journal
-rw-rw----    1 radio    radio       159744 May  6 15:01 telephony.db
-rw-------    1 radio    radio            0 May  4 11:36 telephony.db-journal
root@android:/data/data/com.android.providers.telephony/databases #
```

(a) Before manipulation of the mmssms.db

```
root@android:/ # cd data/data/com.android.providers.telephony/databases/
cd data/data/com.android.providers.telephony/databases/
root@android:/data/data/com.android.providers.telephony/databases # ls -l
ls -l
-rw-rw-rw-    1 root     root        172032 May  6 15:27 mmssms.db
-rw-------    1 radio    radio        32768 May  6 15:28 mmssms.db-shm
-rw-rw-rw-    1 radio    radio       127752 May  6 15:28 mmssms.db-wal
-rw-rw----    1 radio    radio        40960 May  6 13:08 nwk_info.db
-rw-------    1 radio    radio        25136 May  4 11:30 nwk_info.db-journal
-rw-rw----    1 radio    radio       159744 May  6 15:28 telephony.db
-rw-------    1 radio    radio            0 May  4 11:36 telephony.db-journal
root@android:/data/data/com.android.providers.telephony/databases #
```

(b) After manipulation of the mmssms.db

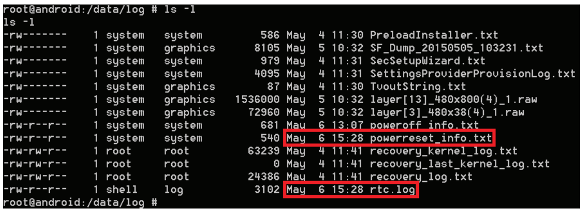
Figure 3: Comparison of changes in the directory containing the SQLite database for the Messaging application

generated. This new WAL file contains limited data and thus has a file size that is smaller than the size of the main database file. A WAL file with a file size smaller than the size main database file is therefore an indication of the possible manipulation of that database.

- **System Reboot:** is required for the changes made to the SQLite databases to reflect on the Android smartphone. The system reboot must occur after making the changes to the SQLite database. Therefore the timestamps of the files associated with a system reboot will follow after the timestamp that shows when the main database file was last modified. Multiple experiments revealed that the following files are indicators of a system reboot:

  - *rtc.log* file located in the */data/log/* directory.
  - *powerreset_info.txt* file located in the */data/log/* directory.
  - *SYSTEM_BOOT@[timestamp].txt* file generated in the */data/system/dropbox/* directory.
  - *event_log@[timestamp].txt* file generated in the */data/system/dropbox/* directory.

  Android log data are written to certain files in the */data/log/* directory [25]. Two files, *rtc.log* and *powerreset_info.txt*, are existing files in this directory. These files are updated with a new entry after every system reboot and every entry shows the boot time of the Android smartphone. The files, *SYSTEM_BOOT@[timestamp].txt* and *event_log@[timestamp].txt*, are located in the directory */data/system/dropbox/* [26, 27]. This folder is used by a service known as DropBoxManager (unrelated to the DropBox cloud storage service) and persistently stores chunks of data from various sources such as application crashes and kernel log records [26]. The *SYSTEM_BOOT@[timestamp].txt* file is generated consistently at boot time, with the timestamp forming part of the file name showing when the Android smartphone was booted. The other file, *event_log@[timestamp].txt*, is generated at 30 minute intervals and also indicates the time when the Android smartphone was rebooted. A system reboot occurring closely after the modification date of a SQLite database provides a possible indication of the modification of timestamps. A system reboot can, however, occur at any time after pushing the modified main database file onto the Android smartphone and is thus necessary to establish a time frame in which this particular AFS-Flag will be deemed reliable.



```
root@android:/data/log # ls -l
ls -l
-rw-------    1 system   system        586 May  4 11:30 PreloadInstaller.txt
-rw-------    1 system   graphics     8105 May  5 10:32 SF_Dump_20150505_103231.txt
-rw-------    1 system   system        979 May  4 11:31 SecSetupWizard.txt
-rw-------    1 system   system       4095 May  4 11:31 SettingsProviderProvisionLog.txt
-rw-------    1 system   graphics       87 May  4 11:30 TvoutString.txt
-rw-------    1 system   graphics  1536000 May  5 10:32 layer[13]_480x800(4)_1.raw
-rw-------    1 system   graphics    72960 May  5 10:32 layer[3]_480x38(4)_1.raw
-rw-r--r--    1 system   system        681 May  6 13:07 poweroff_info.txt
-rw-r--r--    1 system   system        540 May  6 15:28 powerreset_info.txt
-rw-rw-r--    1 root     root        63239 May  4 11:41 recovery_kernel_log.txt
-rw-rw-r--    1 root     root            0 May  4 11:41 recovery_last_kernel_log.txt
-rw-rw-r--    1 root     root        24386 May  4 11:41 recovery_log.txt
-rw-r--r--    1 shell    log          3102 May  6 15:28 rtc.log
root@android:/data/log #
```

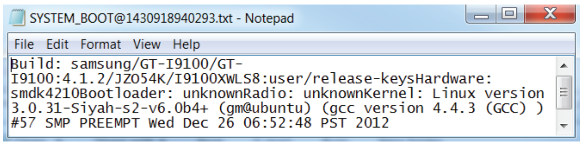
Figure 4: The */data/log/* directory containing the *rtc.log* and *powerreset_info.txt*

Figure 3 provides a comparison of the changes that occurred in the directory containing the SQLite database that stores the SMS/MMS messages. Figure 3 (b) indicates the existence of three AFS-Flags. The first AFS-Flag is the file permissions of both the *mmssms.db* and the *mmssms.db-wal*, which changed from *-rw-rw—-* to *-rw-rw-rw-*. The second AFS-Flag is the ownership for both the file owner and group members of the main database file that changed from *radio* to *root*. The final AFS-Flag is the file size of the *mmssms.db-wal*, which is smaller than the size of the main database file, indicating that the *mmssms.db-wal* file was possibly deleted.

Figure 4 shows that the *rtc.log* and *powerreset_info.txt* files were last modified at 15:28 on May 6 and Figure 5 presents the contents of a *SYSTEM_BOOT@1430918940293.txt*, which indicates that a reboot occurred at 15:29 on 6 May. All three files illustrate that a reboot occurred approximately at 15:28 on May 6, which follows after the last modified date of the main database file (15:27 on May 6). The existence of these files verifies that a system reboot occurred after pushing the modified main database file onto the Android smartphone.

The presence of all four AFS-Flags indicates the possible manipulation of timestamps within the SQLite database storing the SMS/MMS messages. Identification of these manipulated timestamps requires the analysis of the SQLite database for potential inconsistencies.

SQLite Database Inconsistencies: SQLite databases are the prominent choice for data storage in the Android OS. The association of one or more AFS-Flags


SYSTEM_BOOT@1430918940293.txt - Notepad
File Edit Format View Help

```
Build: samsung/GT-I9100/GT-
I9100:4.1.2/JZO54K/I9100XWLS8:user/release-keysHardware:
smdk4210Bootloader: unknownRadio: unknownKernel: Linux version
3.0.31-Siyah-s2-v6.0b4+ (gm@ubuntu) (gcc version 4.4.3 (GCC) )
#57 SMP PREEMPT Wed Dec 26 06:52:48 PST 2012
```

Figure 5: The *SYSTEM_BOOT@[timestamp].txt* file is generated after a reboot

with a specific SQLite database indicates the potential manipulation of the stored timestamps. Detection of the manipulated timestamps requires the further analysis of the SQLite database for any potential inconsistencies. An inconsistency in a SQLite database is described as a record that is listed incorrectly when ordered according to the following fields: primary key and a field containing dates or timestamps. The identification of inconsistencies in the tables of SQLite databases requires the evaluation of the above mentioned fields.

The tool selected to perform the evaluation is SQL. SQL is a powerful query language, allowing for the formulation of queries that can be of forensics use [28]. The evaluation of the tables available in the SQLite databases proceeds through three steps and use the following SQL statements: *CREATE TABLE*, *INSERT INTO*, and *SELECT*. To preserve the integrity of the data stored in the original table, a new temporary table is created using the *CREATE TABLE* statement. The purpose of the *CREATE TABLE* statement is to define the physical structure of the new temporary table [29]. The temporary table contains a primary key, which is an integer value that auto-increments, and all the fields that are necessary to identify the inconsistencies. The query to create the temporary table is as follows:

CREATE TABLE temp (new_id INTEGER PRIMARY KEY AUTOINCREMENT, original_id INTEGER, timestamps INTEGER);

Following the creation of the new temporary table is the population of this table with all the records currently located in the original table, which is being investigated. To perform this action, a combination of the *INSERT INTO* and *SELECT* statements is used. The *SELECT* statement selects all the records from the table currently under investigation while the *INSERT INTO* statement inserts these selected records into the temporary table. Continuing with the SMS/MMS SQLite database of the Messaging application as an example, the SQL query required to copy the records from the *sms* table into the temporary table is as follows:

INSERT INTO temp (original_id, timstamps) SELECT _id, date FROM sms;

To locate any inconsistencies in the records collected in the temporary table, it is necessary to compare the values in the timestamps field of subsequent records. Since all the values in the timestamps field are expected to follow one another (each new record is appended at the end of the table), the difference between two subsequent values in the timestamps field must be smaller than or equal to zero. A positive difference is an indication of a timestamp that is out of order and the cause of this inconsistency is the manipulation of the timestamp. The SQL query used to detect the records that are inconsistent is as follows:

SELECT T1.original_id, T1.timestamps, (T1.timestamps - T2.timestamps) AS difference FROM temp T1, temp T2 WHERE T2.new_id = T1.new_id + 1 AND difference>0;

Applying this SQL query to the records in the temporary table leads to the identification of multiple inconsistencies in the SMS/MMS SQLite database. The existence of these inconsistent records in the SMS/MMS SQLite database invalidates the authenticity of the database. The examiner must thus decide whether to exclude the manipulated records from the investigation or only focus the investigation around the manipulated records.

4. REFERENCE ARCHITECTURE FOR ANDROID APPLICATIONS

The exploratory experiments conducted during this research identified two categories of heuristics that can be used to establish the authenticity and accuracy of timestamps in SQLite databases. The successful application of these heuristics depends, however, on the following two external factors. Firstly, the skills of the smartphone user or the sophistication of the malicious application performing the manipulation can influence the availability of the heuristics. Smartphone users or the malicious application may be aware of the changes that occur due to the manipulation of timestamps in SQLite databases. To prevent detection, these changes can be removed or altered in an attempt to thwart the examiner performing the investigation. Secondly, the timeframe between the manipulation of the timestamps and the seizing of the smartphone can also influence the availability of certain AFS-Flags. An extended timeframe can cause specific AFS-Flags (such as File size and System reboot) to be deleted or be overwritten by the Android OS. The aim of these heuristics is to assist with the identification of manipulated timestamps and not other forms of digital evidence.

Due to the limitations of the identified heuristics, it becomes necessary to explore other techniques that can also be used to establish the authenticity and accuracy of digital evidence. Identifying such techniques requires a deeper understanding of Android applications, which is responsible for creating the evidence. It is possible to obtain such an understanding by designing reference architecture for Android applications. A reference architecture captures the common architectural elements as well as the relationships between these elements for a specific domain [30]. Using a reference architecture to model Android applications allow the examiner to comprehend how the digital evidence originated and whether the evidence is authentic and accurate. While reference architectures exist for many domains such as

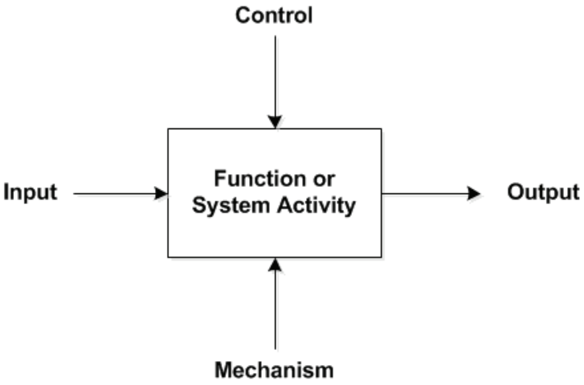


Figure 6: Structured Analysis and Design Technique notation

web browsers [31] and web servers [32], this is the first reference architecture, to the best of the authors' knowledge, which allows for the modelling of Android applications.

The design of the reference architecture is accomplished by using the Structured Analysis and Design Technique (SADT). SADT is a graphical language for describing systems by using a set of diagrams consisting primarily of boxes, which are interconnected by arrows. The boxes represent a specific function or system activity and the interconnected arrows provide external interfaces to the defined function or system activity. The external interfaces are the following [33, 34]:

- Input represents data or other consumables required for functioning.
- Output is the results produced by the function or system activity.
- Control influences or regulates the execution but are not consumed.
- Mechanism is a component used to accomplish the function or system activity.

The notation of SADT, as defined above is used in this paper to illustrate the reference architecture for Android applications (see Figure 6).

Designing a reference architecture for Android applications requires the examination of a wide variety of existing Android applications. This paper closely focused on the Messaging application, which allowed for the identification of certain architectural elements as well as the relationship between these elements. In order to confirm whether the identified architectural elements and the relationship between these elements are prevalent among other applications, additional Android applications were also thoroughly examined. The examination of these Android applications led to the discovery two common architectural elements: application activity and SQLite databases.

The application activity is responsible for launching the graphical user interface and initialising the logic of the Android application. The graphical user interface, structured according to a specified layout and styled following a certain theme, accepts as input an action, along with optional data, from the end user. The graphical user interface is therefore the space where interactions between end users and the Android application occur and accepts specific sets of input, which leads to expected results. The application logic contains the work-flow logic of the Android application and executes the received input to produce results accordingly.

The data involved in the requested action is transferred and retained in a SQLite database. This includes the original data supplied by the end user during the admission of the action and a timestamp, indicating when the action was performed. The retention of the data proceeds according to the policies or set guidelines of the SQLite library [35], which describes what data will be stored, how such data is stored and for how long it will be kept. The SQLite library receives the incoming data and transforms the data according to the rules and requirements of both the SQLite library and SQLite database. Once transformed, the data is retained in the *<database_name>.db-wal* file until the WAL file reaches the limit of 1000 pages. Once the limit is reached, the SQLite library transfers the data to the *<database_name>.db* file.

The reference architecture for Android applications consists of two core components: application activity and SQLite database. The final reference architecture for Android application is shown in Figure 7. The purpose and ultimate goal of the reference architecture is to organise conventional Android applications according to a standardised model. From this standardisation, an examiner conducting a digital forensics investigation can establish certain particulars regarding the Android applications currently under investigation. Using the newly designed reference architecture, a collection of general characteristics regarding Android applications can be identified:

- The Android application must first receive an action from the end-user
- Only after receiving the action can changes be made to data retained in the SQLite database.
- Each individual application accepts a limited selection of inputs.
- Each input contains an action and possibly optional data.
- Every input, along with the action and optional data, leads to expected results.
- The action to be executed by the Android application is provided by the end user, who is either a human operator or another application.

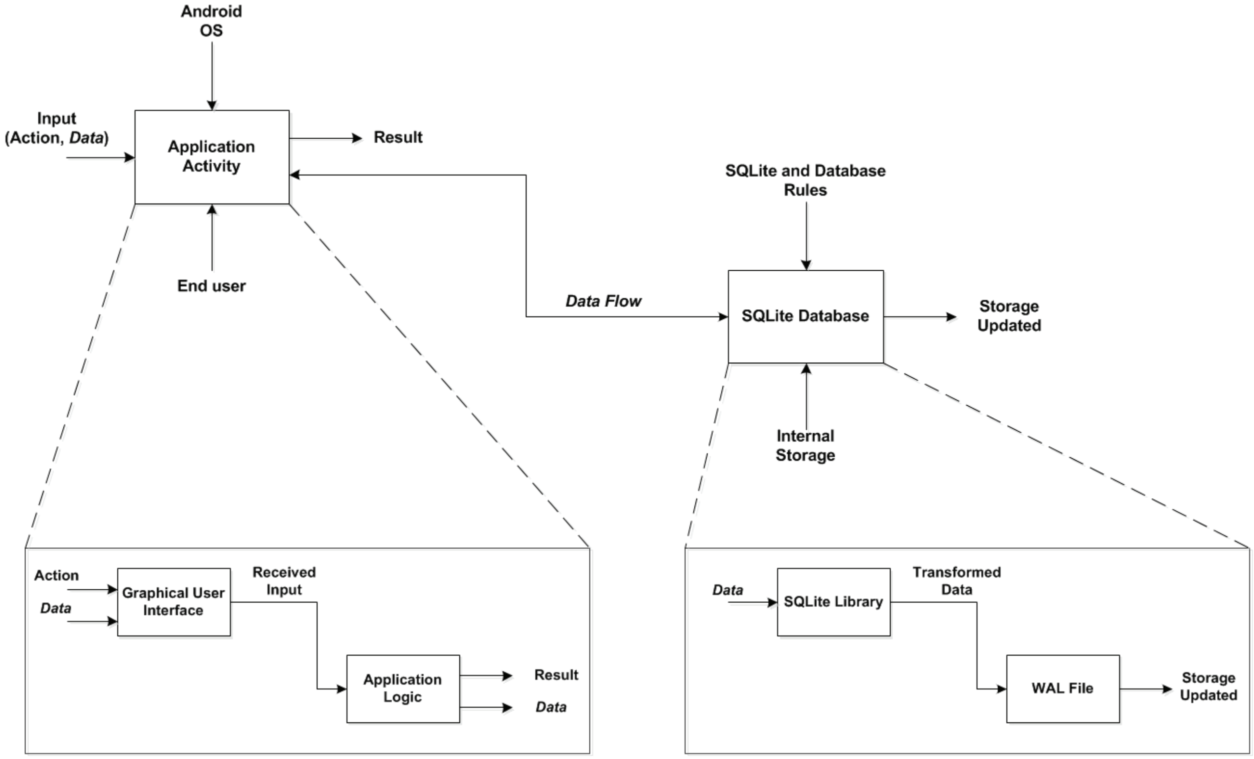


Figure 7: Reference Architecture for Android Applications

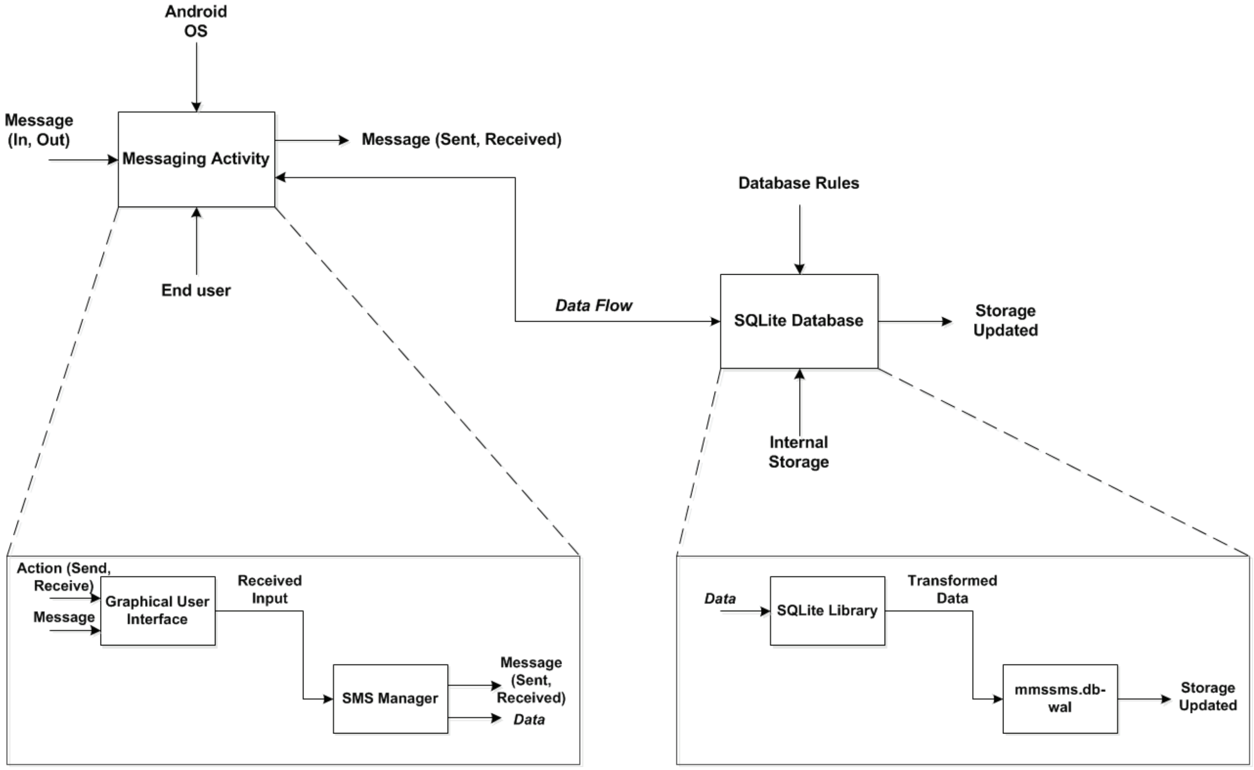


Figure 8: Modelling of the Messaging application according to the Reference Architecture

- Data is expected to flow from the application activity to the SQLite database.
- The data is transformed according to the SQLite or database rules and inserted into the WAL file.

Any findings by the examiner that contradicts the above general characteristics are possible indications of the inaccuracy and unreliability of digital evidence produced by the Android application.

Continuing with the Messaging application as an example, the newly designed reference architecture can now be used to model the application (see Figure 8). The mapping of the Messaging application onto the reference architecture allows for the identification of the core components of the application as well as the flow of data between these components. The modelled architecture of the application can now be used to evaluate the authenticity of the stored data by determining if any of the general characteristics are violated. Firstly, the data is only available in the main database file (*mmssms.db*) and not, as expected, in the WAL file (*mmssms.db-wal*). The omission of the data in the WAL file shows that the data flow between the application activity and the SQLite database has been violated. Secondly, reviewing the usage log of the Messaging application shows that the input was not provided by the end-user. These findings thus confirm the inaccuracy of the digital evidence associated with the Messaging application. It is therefore possible to conclude that the data stored by the Messaging application is possibly changed or altered.

The modelling of Android applications according to the reference architecture thus provides support to examiners involved in ongoing investigations. Using the reference architecture the examiner is capable of establishing the authenticity and accuracy of any digital evidence related to the modelled Android application. The insight offered by the reference architecture will lead the examiner to come to the correct conclusions regarding the investigation, since potentially incorrect digital evidence, such as manipulated timestamps, can be eliminated before concluding the final report. The reference architecture is, however, limited to Android applications only and can't be used to model other software applications.

5. DISCUSSION AND FUTURE WORK

The exploratory experiments performed while composing this paper showed that the timestamps found in SQLite databases can be manipulated by following the technique described in Section 3.2. This technique is currently the most plausible technique to manipulate timestamps in SQLite databases. Although other techniques can be designed, the inability to directly alter the data in the WAL file and the unavailability of the *sqlite3* command utility on an Android smartphone will limit the capabilities and impact the efficiency of other techniques.

To establish the authenticity of timestamps found in SQLite databases and detect the potentially manipulated timestamps, this paper introduced two categories of heuristics. The first group of heuristics determine the authenticity of timestamps by evaluating the file system for specific changes and the second group identifies inconsistencies in the SQLite databases. These heuristics are independent of an Android smartphone and does not require any prerequisites to be installed on the Android smartphones prior the investigation. The purpose of the heuristics is to give examiners an indication of whether the timestamps collected in SQLite databases were tampered with. The results presented by the heuristics allow the examiner to establish the authenticity of the timestamps. Based on the authenticity of the timestamps, the examiner can decide to either include or disregard the evidence, associated with the evaluated timestamps, in the investigation. The heuristics are therefore capable to save crucial time during the investigation and allow the examiner to arrive at correct and accurate conclusions. The experiments provided throughout this paper showed that all of the identified heuristics are capable of providing the examiner with the necessary support to establish the authenticity of timestamps in SQLite databases.

The current collection of heuristics only focuses on detecting the manipulation of timestamps found in SQLite databases. Manipulation of timestamps can, however, occur at multiple locations. The time zone settings of an Android smartphone, which can be set incorrectly or be changed (intentionally or unintentionally), can influence the accuracy of the timestamps found in SQLite databases. Besides the time zone settings, the actual time of the Android smartphone can also be manually adjusted by disabling the automatic time synchronisation feature. Manual adjustments of the time will impact the timestamps that are generated for the traces stored in the SQLite databases when certain events occur on the Android smartphone. It is therefore necessary to incorporate the evaluation of the time zone and time settings of an Android smartphone.

The availability of the identified heuristics can, however, be influenced by external factors and therefore this paper also introduced a reference architecture for Android applications. The newly introduced reference architecture serves as a template to model a diverse collection of Android applications. To allow for such diversity, the reference architecture captures only the essential components found in modern implementations of Android applications and identifies the relationships between these components. The simplistic design of the reference architecture clearly and concisely describes the role of each component, allowing for easy comprehension of the modelled Android application. The design of the reference architecture is flexible, providing the ability to model Android applications at different levels of complexity. Depending on the architecture of Android applications, the components of the reference architecture can be combined or expanded to more closely represent the modelling of a specific Android application. The reference architecture therefore serves as a valuable template for examiners to gain a better understanding of Android applications. Modelling any Android application according to this reference architecture allows examiners to easily comprehend the related digital evidence and provide the necessary insight to determine the evidence's authenticity and accuracy.

Future work will therefore continue to expand this research, exploring different avenues to further establish the authenticity and accuracy of digital evidence. Firstly, the existing collection of heuristics will be extended by identifying additional heuristics that can establish the authenticity and accuracy of timestamps. Such heuristics will include the evaluation of the time zone and time settings of a seized Android smartphone. Secondly, the current focus is only on determining the authenticity and accuracy of timestamps. It is thus necessary to broaden the focus and also include other forms of digital evidence. Lastly, the existing reference architecture will be extended to include the modelling of other forms of applications software. The reference architecture will also be validated using mathematical notation to ensure the architecture applies to more complex scenarios.

6. CONCLUSION

Evidence found, in the form of traces, on smartphones form an important asset of digital investigations. The timestamps associated with the traces allow the examiner to construct a timeline of events. Such a timeline often forms the basis for further investigation and has the ability to provide answers to certain questions. Due to the importance of timestamps, it is necessary for examiners to be able to verify their authenticity. Collected timestamps might be incorrect due to tampering and without additional verification; the timestamps will lead the examiner to make unreliable conclusions. To verify the authenticity of timestamps found in SQLite databases, this paper introduced a collection of heuristics that can be categorised into two distinct groups. The first group of heuristics identifies the presence of certain changes in the Android file system, which is indicator of the manipulation of the SQLite databases. The second group of heuristics subsequently focuses on the individual SQLite databases and the identification of inconsistencies in these databases. The availability of these heuristics are, however, susceptible to external factors and therefore a reference architecture for Android applications was also introduced to further establish the authenticity and accuracy of digital evidence. The challenges addressed in the paper were to show that (a) timestamps can be manipulated in SQLite databases, (b) identifying that the authenticity of these timestamps has been compromised and (c) overcoming the limitations of techniques used to identify the compromised timestamps. Challenge (a) was addressed by showing the process that must be followed to successfully manipulate timestamps in SQLite databases, challenge (b) was addressed by using the identified heuristics and challenge (c) was addressed by introducing a newly designed reference architecture for Android applications. The current paper provides preliminary evidence that the suggested approach shows potential and future work will focus on expanding this research.

REFERENCES

[1] L. Goasduff and J. Rivera: "Gartner says smartphone sales surpassed one billion units in 2014," 2015. [Online]. Available: http://www.gartner.com/newsroom/id/2996817. (Accessed: Apr. 7, 2015).

[2] S. Radicati: "Mobile statistics report, 2014-2018," The Radicati Group Inc., Tech. Rep., 2014.

[3] R. Harris: "Arriving at an anti-forensics consensus: examining how to define and control the anti-forensics problem," *Digital Investigation*, Vol. 3, pp. 44-49, 2006.

[4] R. Verma, J. Govindarai and G. Gupta: "Preserving dates and timestamps for incident handling in Android smartphones," *Advances in Digital Forensics X*, Vol. 433, pp. 209-225, 2014.

[5] J. Govindaraj, R. Verma, R. Mata and G. Gupta: "Poster: iSecureRing: Forensic ready secure iOS apps for jailbroken iPhones," *Proceedings: 35th IEEE Symposium on Security and Privacy*, 2014.

[6] M. Song, W. Xiang and X. Fu: "Research on architecture of multimedia and its design based on Android," *Proceedings: 2010 International Conference on Internet Technology and Applications*, pp. 1-4, 2010.

[7] M. Faheem, N.-A. Le-Khac and T. kechadi: "Smartphones Forensic Analysis: A Case Study for Obtaining Root Access of an Android Samsung S3 Device and Analyse the Image without an Expensive Commercial Tool," *Journal of Information Security*, Vol. 5, pp. 83-90, 2014.

[8] Open handset alliance: "Android," 2015. [Online]. Available: http://www.openhandsetalliance.com/android overview.html. (Accessed: Apr. 9, 2015).

[9] C. Maia, L.M. Nogueira and L.M. Pinho: "Evaluating Android OS for embedded real-time systems," *Proceedings: 6th International Workshop on Operating Systems Platforms for Embedded Real-Time Applications*, pp. 63-70, 2010.

[10] B. Speckmann: "The Android mobile platform," Ph.D. dissertation, Department of Computer Science, Eastern Michigan University, Michigan, 2008.

[11] C. Zimmermann, M. Spreitzenbarth, S. Schmitt and F.C. Freiling: "Forensic analysis of YAFFS2," *Sicherheit*, pp. 59-69, 2012.

[12] J. Lessard and G. Kessler: "Android forensics: simplifying cell phone examinations," *Small Scale Digital Device Forensics Journal*, Vol. 4, No. 1, pp. 1-12, 2010.

[13] H.J. Kim and J.S Kim: "Tuning the ext4 filesystem performance for Android-based smartphones," *Frontiers in Computer Education*, Springer Berlin Heidelberg, pp. 745-752, 2012.

[14] F. Freiling, M. Spreitzenbarth and S. Schmitt: "Forensic analysis of smartphones: The Android Data Extractor Lite (ADEL)," *Proceedings of the Conference on Digital Forensics, Security and Law*, pp. 151-160, 2011.

[15] S. Jeon, J. Bang, K. Byun and S. Lee: "A recovery method of deleted record for SQLite database," *Personal and Ubiquitous Computing*, Vol. 16, No. 6, pp. 707-715, 2012.

[16] SQLite: "About SQLite," 2015. [Online]. Available: https://www.sqlite.org/about.html. (Accessed: Apr. 14, 2015).

[17] SQLite: "The SQLite Database File Format," 2015. [Online]. Available: https://www.sqlite.org/fileformat2.html. (Accessed: Sept. 15, 2015).

[18] P. Patodi: "Database recovery mechanism for Android devices," Ph.D. dissertation, Indian Institute of Technology, Bombay, 2012.

[19] A. Caithness: "The forensic implications of SQLite's write ahead log," 2012. [Online]. Available: http://www.cclgroupltd.com/the-forensic-implications-of-sqlites-write-ahead-log/. (Accessed: Sept. 15, 2015).

[20] SQLite: "Write-Ahead Logging," 2015. [Online]. Available: https://www.sqlite.org/wal.html. (Accessed: Sept. 15, 2015).

[21] A.S. Sharma, M.S. Malhi, M. Singh and R. Singh: "CSLA - An application using Android Capstone project," *Proceedings: 2014 IEEE International Conference on MOOC, Innovation and Technology in Education*, pp. 382-385, 2014.

[22] A.D. Schmidt, H.G. Schmidt, L. Batyuk, J.H Clausen, S.A. Camtepe, S. Albayrak and C. Yildizli: "Smartphone malware evolution revisited: Android next target?," *Proceedings: 4th International Conference on Malicious and Unwanted Software (MALWARE)*, pp. 1-7, 2009.

[23] Android Developers: "Android Debug Bridge," 2015. [Online]. Available: http://developer.android.com/tools/help/adb.html. (Accessed: Sept. 22, 2015).

[24] Stackoverflow: "How to use ADB in Android studio to view an SQLite DB," 2013. [Online]. Available: http://stackoverflow.com/questions/18370219/how-to-use-adb-in-android-studio-to-view-an-sqlite-db. (Accessed: Sept. 22, 2015).

[25] R. Johnson and A. Stavrou: "Resurrecting the read_logs permission on Samsung devices," *Proceedings: Black Hat Asia 2015*, pp. 1-25, 2015.

[26] Android Developers: "DropBoxManager," 2015. [Online]. Available: http://developer.android.com/reference/android/os/DropBoxManager.html. (Accessed: Sept. 22, 2015).

[27] M. Kaart and S. Laraghy: "Android forensics: Interpretation of timestamps," *Digital Investigation*, Vol. 11, No. 3, pp. 234-248, 2014.

[28] H. Pieterse and M. Olivier: "Smartphones as distributed witnesses for digital forensics," *Advances in Digital Forensics X*, Vol. 433, pp. 237-251, 2014.

[29] E. Ugboma: *Learn Database Programming using SQL of MS Access 2007*, BookSurge Publishing, 2009.

[30] W. Eixelsberger, M. Ogris, H. Gall and B. Bellay: "Software architecture recovery of a program family," *Proceedings: 1998 International Conference on Software Engineering*, pp. 508-511, 1998.

[31] A. Grosskurth and M.W. Godfrey: "A reference architecture for web browsers," *Proceedings: 21st IEEE International Conference on Software Maintenance*, pp. 661-664, 2005.

[32] A.E. Hassan and R.C. Holt: "A reference architecture for web servers," *Proceedings: Seventh Working Conference on Reverse Engineering*, pp. 150-159, 2000.

[33] D.A. Marca and C.L. McGowan: *SADT: structured analysis and design technique*, McGraw-Hill, Inc., 1987.

[34] M.E. Dickover, C.L. McGowan and D.T. Ross: "Software design using: SADT," *Proceedings: 1977 Annual Conference*, pp. 125-133, 1977.

[35] S. Lee: "Creating and Using Databases for Android Applications," *International Journal of Database Theory and Application*, Vol. 5, No. 2, pp. 99-106, 2012.