Evaluating the Authenticity of Smartphone Evidence

Heloise Pieterse, Martin Olivier, Renier Van Heerden

Chapter 3

EVALUATING THE AUTHENTICITY OF SMARTPHONE EVIDENCE

Heloise Pieterse, Martin Olivier and Renier van Heerden

Abstract The widespread use and rich functionality of smartphones have made them valuable sources of digital evidence. Malicious individuals are becoming aware of the importance of digital evidence found on smartphones and may be interested in deploying anti-forensic techniques to alter evidence and thwart investigations. It is, therefore, important to establish the authenticity of smartphone evidence.

This chapter focuses on digital evidence found on smartphones that has been created by smartphone applications and the techniques that can be used to establish the authenticity of the evidence. In order to establish the authenticity of the evidence, a better understanding of the normal or expected behavior of smartphone applications is required. This chapter introduces a new reference architecture for smartphone applications that models the components and the expected behavior of applications. Seven theories of normality are derived from the reference architecture that enable digital forensic professionals to evaluate the authenticity of smartphone evidence. An experiment conducted to examine the validity of the theories of normality indicates that the theories can assist forensic professionals in identifying authentic smartphone evidence.

Keywords: Smartphone forensics, evidence, authenticity, reference architecture

1. Introduction

The 21st century has witnessed the emergence and continuous evolution of smartphones. Smartphones are compact devices that combine traditional mobile phone features with personal computer functionality [22]. The popularity of smartphones is the result of ever increasing functionality provided by the hardware, operating systems such as Google Android and Apple iOS, and their associated applications [28].

The ubiquitous use of smartphones in daily activities has rendered these devices rich sources of digital evidence. This digital evidence is important when smartphones are linked to criminal, civil, accident and corporate investigations.

Digital evidence stored on smartphones, referred to as smartphone evidence, includes information of probative value that is generated by an application or transferred to the smartphone by the user. Malicious individuals are becoming increasingly aware of the importance of smartphone evidence and may attempt to manipulate, fabricate or alter the evidence [15]. In particular, they would attempt to apply anti-forensic techniques and tools to compromise the evidence [11]. Anti-forensics can be described as "attempts to compromise the availability or usefulness of evidence to the forensic process" [16]. It is, therefore important for digital forensic professionals to mitigate anti-forensic actions and to establish the authenticity of smartphone evidence. Authenticity refers to the preservation of evidence from the time it was first generated and the ability to prove that the integrity of the evidence has been maintained over the entire period of time [5, 6, 24]. Authentic smartphone evidence is, thus, evidence that originates as a result of the normal behavior of a smartphone application or user.

Smartphone evidence primarily resides in three components: (i) subscriber identity module (SIM) card; (ii) internal storage; and (iii) portable storage such as a micro SD card [1, 7]. While all these components contain valuable evidence, this work focuses on application-related smartphone evidence that is stored directly on a smartphone. Establishing the authenticity of smartphone evidence requires a better understanding of the applications that create the evidence. Developing a better understanding of smartphone applications can be achieved by designing a reference architecture that captures the common architectural elements and their expected behavior [8].

This chapter introduces a new reference architecture for smartphone applications that models the components as well as the normal or expected behavior of smartphone applications. The reference architecture is designed to enable digital forensic professionals to easily comprehend smartphone applications and to understand how the associated evidence originates. The architecture is used to derive theories of normality for smartphone applications. The theories of normality capture the normal or expected behavior of smartphone applications and assist digital forensic professionals in identifying authentic smartphone evidence and helping eliminate unreliable evidence from being considered in arriving at the final conclusions.

2. Related Research

Evidence on a smartphone can provide a digital forensic professional with valuable insights about the interactions that took place involving the smartphone. Smartphone evidence is, however, vulnerable to change and can be altered, manipulated or fabricated either maliciously or by accident without leaving obvious signs [5, 15]. A digital forensic professional must, therefore, establish the authenticity of smartphone evidence before arriving at the final conclusions.

Many software applications have safeguards, such as audit logs and integrity checks, to ensure that the data is valid and has not been tampered with [30]. Such safeguards could assist forensic professionals in establishing the authenticity of smartphone evidence. However, smartphone applications generally do not have audit logs or similar safeguards. Meanwhile, commercial tools, such as the Cellebrite Universal Forensic Device (UFED) and FTK Mobile Phone Examiner, provide limited support in establishing authenticity [31]. Therefore, new techniques and tools are required to determine the authenticity of smartphone evidence.

Pieterse et al. [26] have introduced an authenticity framework for Android timestamps that enables digital forensic professionals to establish the authenticity of timestamps found on Android smartphones. The framework determines the authenticity of timestamps found in SQLite databases using two methods. The first method explores the Android filesystem (EXT4) for artifacts that indicate potential manipulations of the SQLite databases. The second method identifies inconsistencies in the SQLite databases. The presence of specific filesystem changes and inconsistencies in the associated SQLite databases are indicators that the authenticity of the stored timestamps may have been compromised.

Verma et al. [31] have proposed a technique for identifying malicious tampering of dates and timestamps in Android smartphones. The technique gathers kernel-generated timestamps of events and stores them in a secure location outside the Android smartphone. During a digital forensic investigation, the preserved timestamps can be used to establish the authenticity of the dates and times extracted from the smartphone under examination.

Govindaraj et al. [13] have designed iSecureRing, a system for securing iOS applications and preserving dates and timestamps. The system incorporates two modules. One module wraps an iOS application in an additional layer of protection while the other module preserves authentic dates and timestamps of events relating to the application.

All the solutions described above can assist digital forensic professionals in evaluations of smartphone evidence, especially with regard to

authenticity. However, the solutions are either platform-specific or require software to be installed on a smartphone prior to an investigation. Clearly, there is a need for additional solutions that can enable digital forensic professionals to determine the authenticity of smartphone evidence.

A promising solution is to consider the structure and behavior of smartphone applications that create the evidence in question. This can be achieved by modeling smartphone applications using a reference architecture that captures the common architectural elements of applications as well as their behavior [8]. Reference architectures have been specified for several domains, including web browsers [14] and web servers [17]. In the case of smartphone applications, a reference architecture only exists for Android applications [27]. At this time, no generic reference architecture exists for smartphone applications across different platforms.

3. Reference Architecture

A large quantity of smartphone evidence is the result of executing applications. This evidence enables a digital forensic professional to make informed conclusions about application usage. Should the tampering of smartphone evidence not be detected, the digital forensic professional could come up with inaccurate or false conclusions. Therefore, it is important to establish the authenticity of smartphone evidence before attempting to make any conclusions. Identifying authentic smartphone evidence requires the digital forensic professional to have a good understanding of the normal or expected behavior of smartphone applications. Using a reference architecture to model smartphone applications enables the forensic professional to comprehend the structure and behavior of applications and understand how the associated evidence originated.

Designing a reference architecture for smartphone applications requires the evaluation of architectural designs of applications created for various smartphone operating systems. From the architectural designs, common architectural components are identified to create the reference architecture. Finally, the interactions within and between the components are modeled to complete the design of the reference architecture.

3.1 Architectural Designs of Applications

The most common mobile platforms are Google's Android (83% market share) and Apple's iOS (15% market share) [18, 28]. Their popularity is directly related to their functionality, advanced capabilities and numerous third-party applications. Applications designed for these platforms adhere to specific architectural designs to ensure visual attrac-

tiveness and enhanced performance. Examination of the documentation of Android and iOS smartphone applications provides insights into their architectural designs. The combined 98% market share of Android and iOS smartphones has motivated the emphasis on Android and iOS applications in this research.

Android Applications. The visual design and user interface of Android applications are determined by specific themes, styles and structured layouts. A style is a collection of properties that specifies the look and feel of a single view; the properties typically include height, width, padding and color of the view. A theme is a style that is applied to the entire application, enabling all the views to have a similar presentation. Layouts define the visual structure and determine how the views are organized.

Developers use the Extensible Markup Language (XML) to define the theme, styles and layouts for the user interface of an Android application [12]. The user interface facilitates interactions between a user and an application. The interactions are captured by an activity, which contains a window holding the user interface. Activities interact with the user as well as with background processes such as broadcast receivers and services. Broadcast receivers respond to system-wide announcements while services perform longer running operations. Activities, broadcast receivers and services realize the logic of an Android application.

Android applications may require access to persistent data. This access is provided by functions and procedures, captured in storage-specific application programming interfaces (APIs). Android applications have several data storage options: shared preferences, internal storage, external storage and SQLite databases [3]. One or more of the options may be used by an Android application to store data.

Pieterse et al. [27] have proposed a reference architecture for Android applications. The reference architecture has two core components: (i) application activity; and (ii) SQLite database. The application activity component captures the user interface design and logic of the application while the SQLite database component describes the data retention policy [27]. The current reference architecture only models Android applications that use SQLite databases to retain data. Therefore, the architecture is not a complete solution and smartphone applications on other platforms must be investigated to design a reference architecture for smartphone applications in general.

iOS Applications. Developers of iOS applications tend to follow the model-view-controller (MVC) architectural design pattern [21]. The pat-

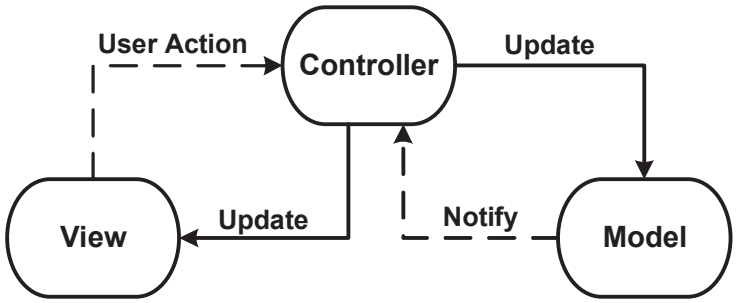

Figure 1. Model-view-controller architecture [19].

tern assigns objects in an iOS application to one of three roles: (i) model; (ii) view; or (iii) controller. As illustrated in Figure 1 [19], the pattern also defines the communications between the objects.

The model object encapsulates the domain data specific to an iOS application [19] and interacts with the physical storage. A view object, which is visible to a user, renders the graphical user interface [19]. View objects are populated with standard user interface elements provided by the UIKit; they often include labels, buttons, tables and text fields [21]. View objects display data from an application model to a user and enable the user to interact with persistent data.

Four common approaches for data persistence are available for iOS applications: (i) user defaults; (ii) property lists; (iii) SQLite databases; and (iv) core data (relational object-oriented model) [20]. The controller object acts as a mediator between the view and model objects [19]. It receives actions from the user and acts accordingly [21]. The controller is also responsible for altering the model object and updating the view object with the changes. The business logic of an iOS application is, thus, realized by the controller object.

Although iOS applications have a different architectural design than Android applications, it is possible to identify certain similar characteristics. These similar characteristics enable the specification of a reference architecture for modeling Android and iOS smartphone applications.

3.2 Reference Architecture Components

Close examination of the architectural designs of Android and iOS applications reveals four similar architectural elements: (i) user interface; (ii) application logic; (iii) data management; and (iv) data storage. Table 1 describes the architectural elements.

The first component is the user interface, which captures the graphical design and presents an interface for user-application interactions. The

Table 1. Architectural similarities of Android and iOS applications.

	Android	**iOS**
Visual Components	Themes, styles and layouts created using XML	View objects containing visual elements created using UIKit
Core Functions and Logic	Activities, broadcast receivers and services	Controller objects
Data Management	Management of stored data using content providers	Management of stored data using model objects
Data Storage	Shared preferences and SQLite databases	User defaults and SQLite databases

interactions allow for the effective operation of the application by permitting the user to perform a limited selection of actions, including the submission of data. This implies that there are common actions, which lead to expected results. The user interface conveys the implemented operations and the received results in a simplistic manner to the user.

The second component is the application logic, which captures the core functions and workflow of the application. The application logic implements the functions responsible for validating, processing and executing the actions and data received from the user interface component. During processing, the data included along with an action can be consumed by the process. Certain actions also cause the data or portions of the data to be kept in their original form and produced as part of the result. After the data is processed, the application logic executes the received action and produces results that are passed to the user interface. Some applications may require all or parts of the data received during an action to be maintained in persistent storage.

The data management component receives data from the application logic component and transforms the data into a suitable format for storage or presentation in the user interface.

The final data storage component stores persistent data and makes the stored data available to applications.

Figure 2 presents the four components of the reference architecture for smartphone applications. The figure shows the architectural ordering of the components and the basic interactions between the components. The current design only provides a high-level overview of the components. In order to obtain a better understanding of the normal or expected behavior of smartphone applications, the reference architecture must capture detailed information about the interactions within and between the architectural components.

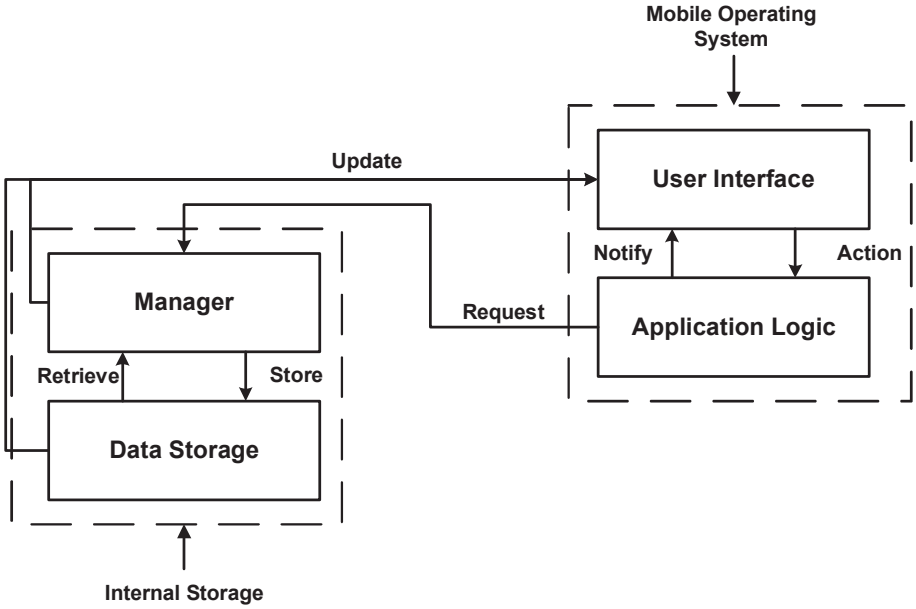

Figure 2. Reference architecture components for smartphone applications.

3.3 Modeling Application Behavior

Modeling smartphones at a fine level of granularity requires a detailed exploration of the behavior of smartphone applications. The internal behavior of smartphone applications is the result of interactions that occur within and between the architectural components. Modeling these interactions provides additional insights into the normal or expected behavior of smartphone applications.

Figure 3 presents a state diagram that models the internal behavior of smartphone applications. The state diagram abstracts the behavior of smartphone applications in terms of four stages:

- **User Interface Stage:** The user interface stage has three states: (i) Idle; (ii) Ready; and (iii) Update. The successful installation of a smartphone application places it the Idle state. An inactive application remains in the Idle state waiting for a user to launch the application or for an external event to occur. An application opened by a user or external event causes it to transition from the Idle state to the Ready state.

  An application in the Ready state can accept an action from a user, the same application (internal action) or another application (external action). An application transitions from the Ready state

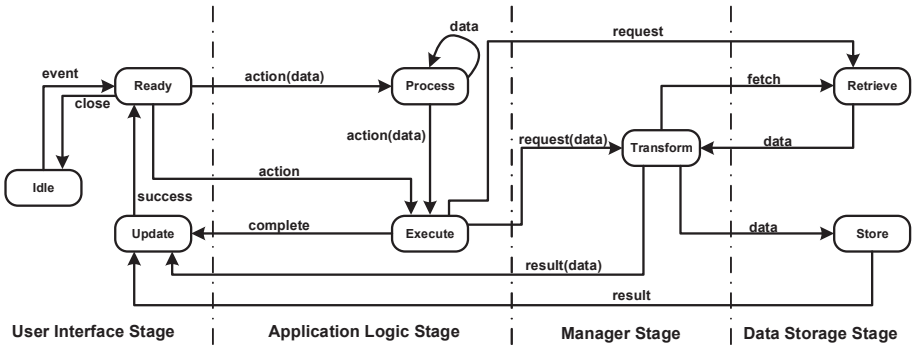

Figure 3. Internal behavior of smartphone applications.

to the Process state upon receiving an action that includes data. An action involving no data causes the application to transition directly to the Execute state.

After an action completes, an application transitions to the Update state. In the Update state, the user interface is updated based on the completed action; this can involve updating the data or elements displayed by the interface. After this is completed, the application returns to the Ready state. Note that an application transitions to the Idle state only when it is closed by a user.

- **Application Logic Stage:** The application logic stage has two states: (i) Process; and (ii) Execute. An application transitions to the Process state upon receiving an action that includes data. The Process state is responsible for separating the action from the data and processing the data accordingly. The processing may involve data validation or the application of security measures such as encryption and encoding. Data processing may involve multiple iterations and the application transitions to the Execute state only after the processing has completed.

  An application in the Execute state executes a received action. If the action involves no data, the application transitions to the Update state after completing the action. An action involving data can cause several outcomes. First, the action may completely consume the data and cause a transition to the Update state. If the data or portion of the data have to be retained in storage, the application transitions to the Transform state. Whether or not an action involves data, it can require data that is maintained in data storage. In order to retrieve the data, the application moves

to the Retrieve state. After the data is received, the application transitions to the Update state.

- **Manager Stage:** The manager stage has a single state called Transform. An application moves to the Transform state from the Execute or Retrieve states after receiving data. In the Transform state, the received data is converted into the desired form. The application then transitions to the Store state in order to store the data. To complete an action that requires transformed data to be retrieved from storage, the application transitions to the Update state.

- **Data Storage Stage:** The data storage state has two states: (i) Retrieve; and (ii) Store. An application moves to the Store state after it accepts transformed data from the Transform state; it then proceeds to store the data in a database or file. After the data is stored, the application transitions to the Update state. The Retrieve state fetches data from storage and the application transitions to the Transform state to transform the data into an acceptable form.

3.4 Exploring an Android Application

The proposed reference architecture provides an abstraction of smartphone applications, enabling a digital forensic professional to easily comprehend the applications and their associated evidence. From this abstraction, the following general characteristics regarding smartphone applications can be identified:

- Only a smartphone application can access and/or update the stored data.

- Data stored by a smartphone application is only accessible via an executed action.

- Data displayed by the user interface directly corresponds to the stored data.

- Changes to data stored by a smartphone application can only occur after an action is received.

- An action can only be provided by a human user, the current smartphone application (internal action) or some other smartphone application (internal action).

- A smartphone application only accepts a fixed set of actions.

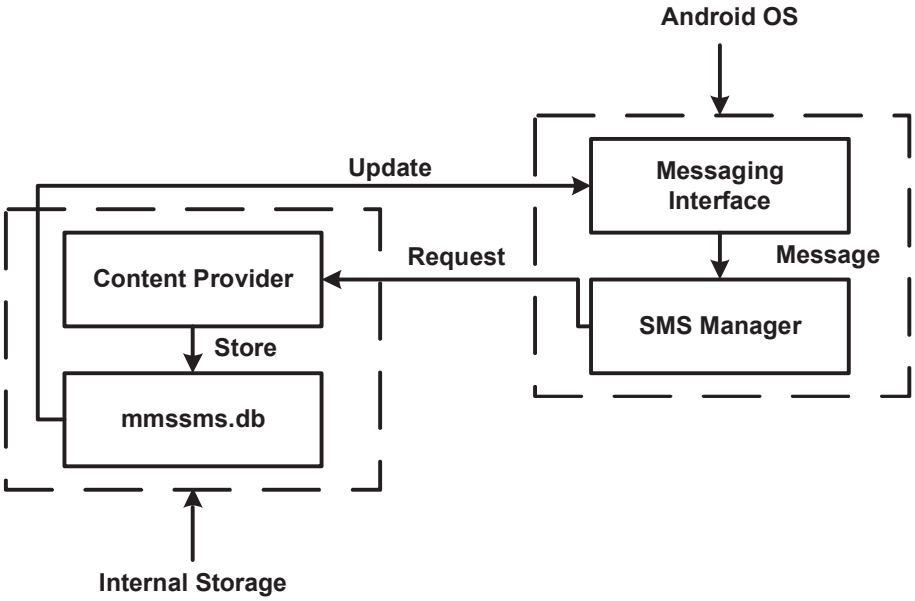

Figure 4. Modeling Android's default messaging application.

- An action in the fixed set leads to an expected result.

To illustrate the value and generality of the reference architecture, Android's default messaging application is modeled according to the reference architecture. Android smartphones are equipped with basic messaging functionality. A user can employ the default messaging application that is pre-installed on a smartphone to send and receive text or multimedia messages [9].

Figure 4 shows how the reference architecture is used to identify and model the core components of the application. The messaging interface enables users to view, delete, send and receive text or multimedia messages. The SMS manager, which contains the workflow logic of the messaging application, is implemented by SmsManger, an Android class that manages the messaging operations [2]. The SmsManger class uses public methods to implement the requested actions. The management functions transform data into a suitable format for storage, which includes the creation of timestamps and the extraction of additional information such as the service center number. After it is transformed, the data is retained in the `mmssms.db` SQLite database.

The behavior of the messaging application is illustrated by sending a new text message. The interactions involved in sending a text message involve three phases. In the first phase, human user opens the messaging

application on the Android smartphone. Upon receiving the open event, the application transitions from the Idle state to the Ready state and receives an internal action to retrieve all the stored messages. The received action causes the application to transition to the Execute state. In the Execute state, the action is evaluated, which requests the retrieval of the stored messages from the SQLite database (`mmssms.db`) and causes the transition to the Retrieve state. In the Retrieve state, the stored messages are fetched and the application transitions to the Transform state to correctly format the messages for visual presentation. Next, the application transitions to the Update state to update the user interface in order to display the messages. Finally, the application returns to the Ready state where it waits for a new action.

In the second phase, the user provides a new action by selecting the option to create a new text message. The application transitions to the Execute state and, because the action does not request the retrieval or storage of data, the action completes. Next, the application transitions to the Update state and updates the user interface accordingly, enabling the user to enter one or more recipients and write the new text message. Finally, the application returns to the Ready state.

During the third and final phase, the user enters the recipient(s) and the text message. Pressing the send button generates an action that includes the data entered by the user, causing the application to move to the Process state. In the Process state, the data is validated (i.e., length of the phone number and text message) before the application transitions to the Execute state. In the Execute state, the text message is sent, but the message must be recorded in the `mmssms.db` database. Therefore, the application proceeds to the Transform state where the data is formatted correctly, after which the application transitions to the Store state. The application then transitions to the Update state to update the user interface and show that the text message was sent. Finally, the application returns to the Ready state.

Modeling Android's default messaging application according to the reference architecture enables a digital forensic professional to identify valuable information about the application. Specifically, the architecture serves as a valuable template for a forensic professional to gain a good understanding of the normal or expected behavior of the messaging application. Although modeling the messaging application can offer insights about the origin of evidence related to the smartphone application, as discussed in the next section, additional criteria are required to efficiently identify authentic smartphone evidence.

4. Theories of Normality

Despite its utility, the information provided by modeling a smartphone application is insufficient to establish the authenticity of the related evidence. Seven theories of normality are specified to assist digital forensic professionals in evaluating the authenticity of smartphone evidence. The theories capture the normal or expected behavior of smartphone applications and can assist forensic professionals in identifying authentic smartphone evidence. The following seven theories of normality stem from the research conducted when designing the reference architecture:

- **Data Correspondence:** Many smartphone applications include actions that retrieve or store data in persistent storage such as a database. This data is made accessible to a user via the user interface of the application. Unauthorized changes made to stored data may not be immediately reflected in the user interface because of cached data. Authentic smartphone evidence requires the stored data to correspond to the data presented by the user interface. Should the application allow for bi-directional communications (i.e., text messaging or telephone calls), the stored data must also correspond to the data stored on the other smartphone involved in the communications (if the other smartphone is available for examination).

- **Data Storage Consistency:** Smartphone applications have several options for storing persistent data, one of the most popular is an SQLite database [4, 10]. Authentic smartphone evidence should have consistent database records. A consistent record in a SQLite database is one that is listed correctly when ordered according to the auto-incremented primary key and a field containing a date or timestamp.

- **File System Consistency:** Files containing stored data have specific permissions and owners that allow/restrict modifications to the data. When a file is created for the first time, the responsible application is given ownership of the file and is assigned the necessary read/write permissions. Authentic smartphone evidence requires file permissions and ownership to remain unaltered.

- **Smartphone Reboot:** Tampering with smartphone evidence may require a system reboot for the changes to be reflected on the smartphone and on the user interface of the smartphone application [27]. A system reboot is generally performed after a file containing stored data has been modified. A timestamp associ-

ated with a system reboot that follows soon after the modification of the file is a possible indicator of evidence tampering.

- **Presence of Anti-Forensic Tools:** Anti-forensic tools for smartphones can be used to destroy, hide, manipulate or prevent the creation of evidence [29]. Smartphone applications, such as File Shredder (Android) or iShredder (iOS), can be used to destroy data; data can be hidden using StegDroid or MobiStego (both Android) applications. Eliminating the presence of anti-forensic applications on a smartphone limits the possibility of evidence tampering.

- **Smartphone Rooting/Jailbreaking:** Data stored by a smartphone application is inaccessible to users. Access to the application and the data can be obtained by rooting (Android) or jailbreaking (iOS) the smartphone [23, 25]. Although rooting or jailbreaking is not a direct indication of data tampering, a rooted or jailbroken smartphone lacks the additional protection measures against data tampering that are required to ensure evidence authenticity.

- **Application Usage:** The internal behavior of a smartphone application, illustrated in the state diagram, shows that only actions can create or alter stored data. Users (humans, the smartphone application itself or another smartphone application) are the only entities capable of providing actions; therefore, their presence must be confirmed. Verifying that a user created or altered the data increases its authenticity.

The seven theories of normality indicate whether or not the evidence produced by a smartphone application is the result of normal or expected behavior of the application. A digital forensic professional can, therefore, use the theories of normality to evaluate the authenticity of smartphone evidence.

An experiment was conducted to confirm the validity of the seven theories of normality. The experiment involved the tampering of text messages produced by Android's default messaging application. The manipulation of the text messages involved the following steps

- **Step 1:** Root the test Android smartphone (Samsung Galaxy S5 Mini running Android version 4.4.4).

- **Step 2:** Copy the `mmssms.db` and `mmssms.db-wal` SQLite database files that contain all the text messages to the `/sdcard/` location on the Android smartphone and then to a computer.

```
<pkg name="eu.chainfire.supersu">
    <comp name="eu.chainfire.supersu.PromptActivity" lrt="1469541224910"/>
</pkg>
```

Figure 5. Confirmation of SuperSu application use in `usage-history.xml`.

- **Step 3:** Use SQLite Expert Personal to alter the text messages.
- **Step 4:** Remove the `mmssms.db` and `mmssms.db-wal` SQLite database files from the Android smartphone using the `rm` command.
- **Step 5:** Copy the altered `mmssms.db` SQLite database file to the Android smartphone and move the file to the `/data/data/com.android.provider.telephony/databases/` location.
- **Step 6:** Change the permissions of the `mmssms.db` SQlite database file using the command `chmod 666 mmssms.db`.
- **Step 7:** Reboot the Android smartphone.
- **Step 8:** Unroot the Android smartphone.

In the experiment, the seven theories of normality were used to evaluate the text messages and determine whether or not the messages originated as a result of the normal behavior of the messaging application. First, the installed applications on the Samsung Galaxy S5 Mini were viewed. No anti-forensic applications were installed on the smartphone. Traditional root applications, such as SuperSU and Superuser, were also not present on the smartphone. However, their absence is not a definite indicator that the smartphone was not rooted; this is because a root application could have been uninstalled or root could have been removed. Examination of the `/data/system/usage/usage-history.xml` file, which contains log entries showing when the user last used an application, revealed that the SuperSu application was previously installed on the smartphone.

Figure 5 presents a snippet of the `usage-history.xml` file. Conversion of the timestamp revealed that the SuperSu application was last used on `26/07/2016 15:53:44 GMT+2:00`. The log entry offers a positive indication that the smartphone was rooted.

The `usage-history.xml` file in Figure 6 also shows that the default messaging application (identified by the `com.android.mms` package name) was last used on `23/07/2016 14:09:44 GMT+2:00` (Figure 6).

Figure 7 shows the timestamps of the SQLite database files associated with the default messaging application. The timestamps contradict the log entry in the `usage-history.xml` file. In fact, the timestamps of the

```
<pkg name="com.android.mms">
    <comp name="com.android.mms.ui.ConversationComposer" lrt="1469275784131"/>
    <comp name="com.android.mms.settings.NotificationSettings" lrt="1426085288917"/>
    <comp name="com.android.mms.settings.CheckDefaultSmsAppsActivity" lrt="1411617622059"/>
    <comp name="com.android.mms.settings.EntrancePrefActivity" lrt="1440400537806"/>
    <comp name="com.android.mms.settings.MultimediamessagesSettings" lrt="1440400533905"/>
    <comp name="com.android.mms.ui.ClassZeroActivity" lrt="1444779525868"/>
    <comp name="com.android.mms.ui.ForwardMessageActivity" lrt="1466240005345"/>
    <comp name="com.android.mms.ui.ReservationMessageManager" lrt="1390041980198"/>
    <comp name="com.android.mms.prioritysender.AddGlanceListActivity" lrt="1451841909782"/>
    <comp name="com.android.mms.cover.MissedMsgActivity" lrt="1429775515108"/>
```

Figure 6. Confirmation of SMS application use in `usage-history.xml`.

```
root@kminilte:/data/data/com.android.providers.telephony/databases # ls -l
-rw-rw---- radio    radio     1273856 2016-06-09 13:03 mmssms.db
-rw-rw---- radio    radio       32768 2014-01-18 12:35 mmssms.db-shm
-rw-rw---- radio    radio     4152992 2014-01-15 09:22 mmssms.db-wal
-rw-rw---- radio    radio       32768 2014-09-24 10:00 nwk_info.db
-rw-rw---- radio    radio       25136 2014-01-01 02:01 nwk_info.db-journal
-rw-rw---- radio    radio      188416 2014-01-18 12:35 telephony.db
-rw------- radio    radio           0 2014-01-01 02:24 telephony.db-journal
```

Figure 7. Original timestamps in the `mmssms.db` SQLite database.

SQLite database files (`mmssms.db` and `mmssms.db-wal`) indicate that the application was last used on `26/07/2016 16:01:00 GMT+2:00`.

```
root@kminilte:/data/data/com.android.providers.telephony/databases # ls -l
-rw-rw-rw- root     root      1302528 2016-07-26 15:57 mmssms.db
-rw-rw-rw- radio    radio       32768 2016-07-26 16:01 mmssms.db-shm
-rw-rw-rw- radio    radio        4152 2016-07-26 16:01 mmssms.db-wal
-rw-rw---- radio    radio       32768 2014-09-24 10:00 nwk_info.db
-rw-rw---- radio    radio       25136 2014-01-01 02:01 nwk_info.db-journal
-rw-rw---- radio    radio      188416 2016-07-26 16:01 telephony.db
-rw------- radio    radio           0 2014-01-01 02:24 telephony.db-journal
```

Figure 8. Changed timestamps in the `mmssms.db` SQLite database.

Closer inspection of the SQLite database files in Figure 8 indicate changes to the file permissions and ownership. To confirm the consistency of the SQLite database records, the database records were viewed and the records were found to be listed correctly. It was also discovered that the records stored in the SQLite database corresponded to the text messages displayed on the user interface.

Finally, the log files associated with a system reboot were examined. Figure 9 indicates that a system reboot occurred shortly after the SQLite database was modified.

The specific findings – inconsistent usage of the default messaging application, filesystem inconsistencies, subsequent rebooting and the rooting of the smartphone – lead to the conclusion that the text messages

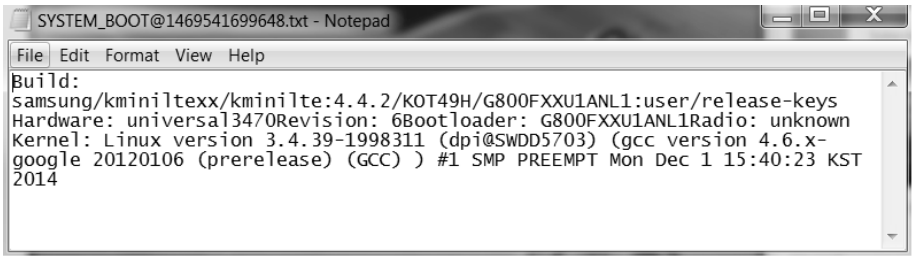

Figure 9. Confirmation of reboot on `26/07/2016 16:01:39 GMT+2:00`.

stored on the Android smartphone may have been tampered with and that the authenticity of the text messages cannot be established.

5. Discussion

The proposed reference architecture for smartphone applications allows for the abstraction of a diverse collection of Android and iOS applications. To support the diversity, the reference architecture captures the essential components of applications and identifies the behaviors of the architectural components. The simplistic design clearly and concisely describes the role of each component, enabling the easy comprehension of modeled smartphone applications. The design is also flexible, providing digital forensic professionals with the ability to model smartphone applications at different levels of complexity. Using the reference architecture, forensic professionals can swiftly obtain a better understanding of the normal or expected behavior of smartphone applications as well as the smartphone evidence related to the applications. The reference architecture is limited to Android and iOS applications, but it is readily extended to model applications that run on other operating systems. However, although the reference architecture offers insights into the internal behavior of applications, it is not sufficient to establish the authenticity of the related smartphone evidence.

The seven theories of normality derived from the reference architecture capture the normal or expected behavior of smartphone applications. Digital forensic professionals can use the theories of normality to evaluate smartphone evidence. Based on an evaluation, a forensic professional can decide whether to consider or disregard the smartphone evidence. The experiment conducted as part of this research demonstrates that the theories of normality provide a forensic professional with the support needed to determine whether or not evidence originated as a result of the normal behavior of a smartphone application. While the theories of normality cannot directly pinpoint the tampering of smart-

phone evidence, they can assist in eliminating unreliable evidence. Using the theories of normality in smartphone investigations is expected to save digital forensic professionals valuable time and help them reach correct and accurate conclusions.

6. Conclusions

The popularity and rich functionality of smartphones have required digital forensic professionals to examine large quantities of smartphone evidence. However, the integrity of smartphone evidence can be compromised by anti-forensic tools, malware and malicious users. It is, therefore, necessary to establish whether or not smartphone evidence is the result of the normal or expected behavior of smartphone applications. The reference architecture described in this chapter models the components of smartphone applications and their expected behavior. The reference architecture helps derive seven theories of normality that assist digital forensic professionals in evaluating the authenticity of smartphone evidence. An experiment involving the manipulation of evidence produced by Android's default messaging application validates the use of the normality theories. Indeed, the experiment demonstrates that the normality theories provide significant investigatory assistance to digital forensic professionals while enabling them to identify unreliable evidence so that it can be eliminated when arriving at the final conclusions.

Future research will engage the theories of normality to create a smartphone evidence classification model that will enhance the ability to establish the authenticity of evidence. The classification model will also be evaluated against authentic and manipulated smartphone evidence.

References

[1] M. Al-Hadadi and A. AlShidhani, Smartphone forensics analysis: A case study, *International Journal of Computer and Electrical Engineering*, vol. 5(6), pp. 576–580, 2013.

[2] Android Developers, SmsManager (`developer.android.com/reference/android/telephony/SmsManager.html`), 2015.

[3] Android Developers, Storage Options (`developer.android.com/guide/topics/data/data-storage.html`), 2016.

[4] M. Bader and I. Baggili, iPhone 3GS forensics: Logical analysis using Apple iTunes Backup Utility, *Small Scale Digital Device Forensics Journal*, vol. 4(1), 2010.

[5] E. Casey, *Digital Evidence and Computer Crime: Forensic Science, Computers and the Internet*, Academic Press, Waltham, Massachusetts, 2011.

[6] F. Cohen, *Digital Forensic Evidence Examination*, Fred Cohen & Associates, Livermore, California, 2009.

[7] K. Curran, A. Robinson, S. Peacocke and S. Cassidy, Mobile phone forensic analysis, in *Crime Prevention Technologies and Applications for Advancing Criminal Investigations*, C. Li and A. Ho (Eds.), IGI Global, Hershey, Pennsylvania, pp. 250–262, 2012.

[8] W. Eixelsberger, M. Ogris, H. Gall and B. Bellay, Software architecture recovery of a program family, *Proceedings of the Twentieth International Conference on Software Engineering*, pp. 508–511, 1998.

[9] W. Enck, M. Ongtang and P. McDaniel, On lightweight mobile phone application certification, *Proceedings of the Sixteenth ACM Conference on Computer and Communications Security*, pp. 235–245, 2009.

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

[11] S. Garfinkel, Anti-forensics: Techniques, detection and countermeasures, *Proceedings of the Second International Conference on i-Warfare and Security*, pp. 77–84, 2007.

[12] M. Goadrich and M. Rogers, Smart smartphone development: iOS versus Android, *Proceedings of the Forty-Second ACM Technical Symposium on Computer Science Education*, pp. 607–612, 2011.

[13] J. Govindaraj, R. Verma, R. Mata and G. Gupta, iSecureRing: Forensic-ready secure iOS apps for jailbroken iPhones, poster paper presented at the *IEEE Symposium on Security and Privacy*, 2014.

[14] A. Grosskurth and M. Godfrey, A reference architecture for web browsers, *Proceedings of the Twenty-First IEEE International Conference on Software Maintenance*, pp. 661–664, 2005.

[15] M. Hannon, An increasingly important requirement: Authentication of digital evidence, *Journal of the Missouri Bar*, vol. 70(6), pp. 314–323, 2014.

[16] R. Harris, Arriving at an anti-forensics consensus: Examining how to define and control the anti-forensics problem, *Digital Investigation*, vol. 3(S), pp. S44–S49, 2006.

[17] A. Hassan and R. Holt, A reference architecture for web servers, *Proceedings of the Seventh Working Conference on Reverse Engineering*, pp. 150–159, 2000.

[18] International Data Corporation Research, Smartphone Growth Expected to Drop to Single Digits in 2016, Led by China's Transition from Developing to Mature Market, According to IDC, Press Release, Framingham, Massachusetts, March 3, 2016.

[19] T. Iulia-Maria and H. Ciocarlie, Best practices in iPhone programming: Model-view-controller architecture – Carousel component development, *Proceedings of the International Conference on Computer as a Tool*, 2011.

[20] B. Jacobs, iOS from Scratch with Swift: Data Persistence and Sandboxing on iOS, *Envato Tuts+* (`code.tutsplus.com/tutorials/ios-from-scratch-with-swift-data-persistence-and-sandboxing-on-ios--cms-25505`), December 25, 2015.

[21] M. Joorabchi and A. Mesbah, Reverse engineering iOS mobile applications, *Proceedings of the Nineteenth Working Conference on Reverse Engineering*, pp. 177–186, 2012.

[22] A. Kubi, S. Saleem and O. Popov, Evaluation of some tools for extracting e-evidence from mobile devices, *Proceedings of the Fifth International Conference on the Application of Information and Communication Technologies*, 2011.

[23] J. Lessard and G. Kessler, Android forensics: Simplifying cell phone examinations, *Small Scale Digital Device Forensics Journal*, vol. 4(1), 2010.

[24] M. Losavio, Non-technical manipulation of digital data, in *Advances in Digital Forensics*, M. Pollitt and S. Shenoi (Eds.), Springer, Boston, Massachusetts, pp. 51–63, 2005.

[25] C. Miller, Mobile attacks and defense, *IEEE Security and Privacy*, vol. 9(4), pp. 68–70, 2011.

[26] H. Pieterse, M. Olivier and R. van Heerden, Playing hide-and-seek: Detecting the manipulation of Android timestamps, *Proceedings of the Information Security for South Africa Conference*, 2015.

[27] H. Pieterse, M. Olivier and R. van Heerden, Reference architecture for Android applications to support the detection of manipulated evidence, *SAIEE Africa Research Journal*, vol. 107(2), pp. 92–103, 2016.

[28] A. Prasad, Android to rule smartphone market with 85% share in 2020 says IDC report, *International Business Times*, March 5, 2016.

[29] I. Sporea, B. Aziz and Z. McIntyre, On the availability of anti-forensic tools for smartphones, *International Journal of Security*, vol. 6(4), pp. 58–64, 2012.

[30] L. Thomson, Mobile devices: New challenges for admissibility of electronic evidence, *Scitech Lawyer*, vol. 9(3), 2013.

[31] R. Verma, J. Govindaraj and G. Gupta, Preserving dates and time-stamps for incident handling in Android smartphones, in *Advances in Digital Forensics X*, G. Peterson and S. Shenoi (Eds.), Springer, Heidelberg, Germany, pp. 209–225, 2014.