UNIVERSITA' degli STUDI di ROMA
TOR VERGATA
Facoltà di Ingegneria

Dottorato di Ricerca in
Informatica ed Ingegneria dell'Automazione
Ciclo XXI

Profiling Mobile Identities

Fabio Dellutri

A.A. 2008/2009

Docente Guida/Tutor: Prof. Giuseppe F. Italiano
Coordinatore: Prof. Daniel P. Bovet

Abstract

The penetration of mobile phones reached 50% of the worldwide population in early 2008 [1]. In the US alone, this percentage will surge past 100% by 2013 [2]. Currently, most purchased mobile phones have enhanced application capabilities and performances, regarding computational resources, connectivity and battery: these mobile devices are referred to *smartphones*, actually cell phones with PDA functionalities which can host custom applications. Due to those enriched capabilities, smartphones can collect large amounts of personal information, which, if analyzed, could reveal important aspects of the owner's identity, such as the kind of relationship with his contacts.

In this work we address the problem of reconstructing the identity profile of the owner, after a smartphone seizure, i.e., the social relationships which is shared by her and her contacts. This goal is achieved by analyzing personal data stored into the device's internal memory, and by correlating it with the Web publicly available information about the owner and her contacts. The resulting social graph is further analyzed through spectral clustering algorithms, in order to find communities of people sharing the same interests.

Each phase of the process is described, and the results obtained are shown. In the interest of practical application, a workflow which disciplines several stages of the profiling process is presented.

Acknowledgements

My thesis would not be complete without some words of thanks to the many friends, colleagues and family members who have supported me during the course of my PhD.

First I would like to thank Pino for the wisdom and guidance he has given me. I have learned from him an enormous amount not only about research, but also about the need for balance in all things. Thanks to Luigi for the hours spent helping with my experiments, for bringing a sense of excitement about new ideas and for understanding the importance of the obtained results. Thanks also to Gianluigi for suggesting some interesting ideas behind this thesis.

I also thank all colleagues who I met through this experience of PhD. It has been a great pleasure to work together. Sharing our ideas has been a great chance of personal and professional growth. I would give special thanks to Gianluca, Vittorio, Lele, Emiliano and Armando: no automatic tool will ever be able to measure our friendship!

Thanks to my family and friends for their love over the years. They have shaped me with support, advice, opportunities and I am ever grateful for everything they have done for me. Words cannot fully express how important you are to me.

Finally, I thank my love Ilaria, who believed in me every day. Without you all this would not have been possible.

Table of Contents

List of Figures

List of Tables

1 Motivation

1.1 Introduction

The mobile phone can be considered the ultimate disruptive technology: in fact, like telephony, radio, television, and the Internet, mobile phones are dramatically changing nearly every aspect of daily life, both inside businesses and in the daily lives of individuals, providing more applications and collecting more private data. The new smartphones (118 million in 2007, Canalys) are mobile phones which have an outstanding computing capability, a sizeable memory, a multi-connectivity (HDSPA, Bluetooth, IR, WLAN) and a multimedia recording system. Moreover, by running a complete operating system (OS) software with a standardized interface and platform for application developers, such devices provide advanced features like e-mail and Internet capabilities and a full keyboard. Such features, as well as the high portability of mobile devices, give the owner the chance to carry a lot of personal data such as contacts, call logs, messages, pictures, video, credit card and bank account codes.

Since these devices collect and store a large amount of personal data into their memories, they can be used as evidence in investigations. Moreover, personal data collected from their storage memories are the starting point to reconstruct the identity of the owner. In this work we will use the following definition for the Identity Profile concept:

> An *identity profile* associated with an individual is given both by the set of people she knows (the *contacts*) and by the type and by the strength of the *relationships* that the individual has with her contacts.

Moreover, such a concept may be extended by the type and by the strength of the relationships that the contacts share between them, introducing the possibility of grouping among them contacts who share similar interests.

1.2 Contributions

This thesis addresses the problem of reconstructing the identity profile of the owner, after a smartphone seizure, by analyzing the personal data stored in the device's internal memory, and by correlating such information with the relationships of the owner and her contacts available on the World Wide Web. In order to achieve this objective, our work focuses on designing a workflow which disciplines several stages of the process, starting with the seizure of the device and ending with the identity profile. Such a workflow was implemented in a software framework which can be used by forensic operators. In particular our contributions cover the following areas:

1. **Smartphone internal memory data seizure and acquisition**: we studied and developed an innovative approach to seize data from the device's internal memory.

2. **Smartphone personal data decoding**: we designed a data reverse engineering methodology, which aims at collecting knowledge about the mobile device's internal database format, and facilitates the development of format-specific parsers which are able to convert smartphone's personal data into a more suitable format.

3. **Mobile identity profiling**: we designed a process to analyze the smartphone's personal data and to show which relationships exist between the owner and its contacts, and, thanks to data collected by the Web search engines, which interests are shared among the contacts.

In the following we will give a short motivation for each of the items above.

1.2.1 Smartphone internal memory data seizure and acquisition

The growing prominence of forensic sciences, in the investigation chain, has led to the broad use of forensic tools to acquire mobile phone internal memory content, to provide evidence of a crime. However, as rule of thumb, the

crime-scene usually offers many different mobile phone/smartphone models, causing the forensic operators to be overwhelmed by using the one-on-one connectors for every single mobile device. Obviously interoperability is not assured for different manufacturer implementations and this leads to a great unpredictability of the effectiveness of the tool on the crime scene. However, the main disadvantage is the partial access of the file system, which relies on the communication protocol. Due to the continuous upgrading of smartphone hardware and software connection interfaces, a tool based on a protocol or connected-via-cable approach could become obsolete in a very short time (less than one year) if not upgraded frequently.

For this reason, we have designed an acquisition methodology which expresses how to seize a device internal memory, and we developed a tool which dumps the internal memory on a SD/MMC (Secure Digital / MultiMediaCard) memory inserted in the available external slot, called MIAT[1] (Mobile Internal Acquisition Tool), making it unnecessary to connect the device to a PC.

1.2.2 Smartphone personal data decoding

After extracting a logical dump of the file system from the mobile device, we need a method to decode and convert the personal data like contacts, messages, logs, calendars, in a more suitable format. Regardless of the device's manufacturer, model and OS version, we developed a methodology to support the data reverse engineering on smartphone file systems. In such way the smartphone's DRE operators will be more flexible in the mobile file format knowledge and they will be supported in developing software parsers for personal data of specific mobile devices and OS.

1.2.3 Mobile identity profiling

The final challenge addressed in this work is to analyze the personal data decoded from the mobile device, by building a model which connects the owner with her contacts and by showing how strong such relationships are. This information is finally complemented with the personal information scattered on the Internet. Nowadays, in fact, vast amount of personal data are left daily on Web pages and are linkable to a person by the *nearly*-unique real-life identifiers like name-surname, or phone numbers or email addresses. Such pages are crawled every day by most search engines and are available for querying

[1] For Symbian OS [3] and Windows Mobile [4]

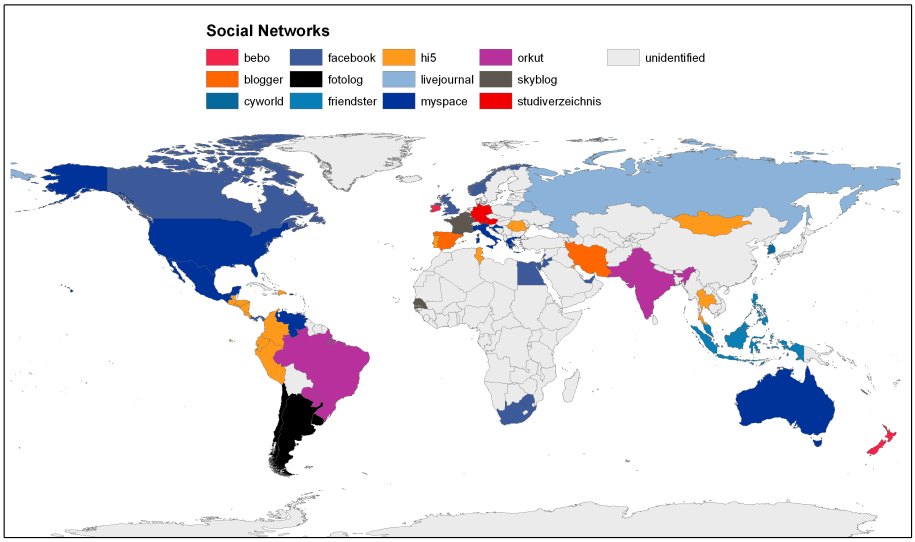


Figure 1.1: Social networks spread (october 2007).

by anyone. The causes of this data spreading may vary: before the advent of Web 2.0 and of the *user-generated content* concept, the small amount of personal data present could be published by the user on their web-sites, or by institutions on their unprotected websites [5]. With the growth of Web 2.0 social networking communities such as Flickr, Facebook, LinkedIn, MySpace, YouTube (see Figure 1.2.3), the users are called to contribute to the communities with their personal assets: they are called to write their current activity, to tell which users are real life friends, to send pictures and videos taken during their life, and so on. Although Web 2.0 companies allow access to profile data only by its owner and, if she agrees, by her friends, in some cases Web search engines are granted access to a little portion of such data. This is justified by the fact that search engines may increment the user visibility to real life friends who have not an account on that community. This increments the chances of the *word-of-mouth* marketing method to make new customers [6]. At the end of 2008, an article appeared in the French magazine "Le Tigre" [7] showing that it is possible to piece together the life of a common person by collecting information left on the Web. In such plethora of personal data it is possible to encounter web pages which report two or more persons' names and surnames, and by

the topic of such web page it can be possible to outline the common ground shared by such people: they could be colleagues, attend the same university or cultivate the same hobbies.

By retrieving people's public information on the World Wide Web, we complete the information about relationships between the phone owner and his contacts, and we are able to extend this investigation to the inter-contact relationships, in order to build a more complete social network. As the resulting profile model is a graph, it lends itself very well to more accurate analysis. In this work we propose two spectral clustering algorithms for finding communities of people in the graph.

1.3 Thesis Outline

This thesis is broadly organised in order of application. A detailed description of each chapter follows below.

Chapter 2 is a survey on mobile phone technology and on the existing mobile operating systems. This chapter also reviews the literature about existing approaches to the mobile forensics problem and resumes available products.

Chapter 3 exposes the framework and the environment we developed in order to implement the profiling workflow.

Chapter 4 describes technical aspects of the design of MIAT, the Mobile Internal Acquisition Tool, and discusses the results obtained from the comparison with other products.

Chapter 5 proposes a wisdom-driven DRE methodological approach to decode smartphone's personal data. The chapter also shows the results obtained.

Chapter 6 presents the Mobile Identity Profiling (MIP) process, which is the core concept of this work.

Chapter 7 summarises the findings of the thesis and considers directions for future work, including ideas for extending the MIP process to multiple device at a time and for automating the entire process through the remote *Forensic Farm*.

2
Background and Related Work

2.1 Introduction

This Chapter describes relevant background knowledge and related work for readers to easily understand the analysis conducted in our work to be presented and discussed in Chapters 4, 5 and 6. This chapter will deal with two different areas: mobile device forensics and identity profiling over the Web. We will firstly review the literature about such areas, then we will give a report of existing device examination products .

2.2 Mobile operating systems

2.2.1 Symbian

Symbian is a joint venture between Nokia, Motorola, Ericsson, Matsushita, and Psion that became independent in June 1998. Symbian was established by leaders in the computing and mobile industries to enable the mass market of communicators and smart phones. Symbian is an open operating system, designed for mobile devices, with associated libraries, user interface frameworks and reference implementations of common tools, produced by Symbian Ltd. Symbian evolved from Psion's EPOC in 1998. Symbian OS is currently owned by BenQ, Ericsson, Panasonic, Nokia, Siemens AG and Sony Ericsson. There are several variations of the Symbian OS that are tailored for different devices. The capabilities of the Symbian OS depend on the device for which it was tailored. Each variation is called a Device Family Reference Design (DFRD). The benefits of Symbian OS are the following:

- Faster time-to-market for platform vendors
- Open, standards-based platform for third-party application developers
- Excellent connectivity
- Advanced design
- Extensibility
- High-performance, 32-bit OS with pre-emptive multitasking
- Long battery life
- Wide industry support and commitment
- Applications that can be designed once and run on multiple devices
- Diversity of devices for consumers

2.2.2 Windows Mobile

Windows Mobile is an operating system that has both a look-and-feel and a programmer API that are similar to Microsoft Windows but which runs in a dramatically reduced footprint. Windows Mobile is the successor operating system to Windows CE. Windows Mobile runs on multiple hardware platforms including Pocket PCs, smartphones, Portable Media Center, and automobiles. These hardware platforms did not always exist from the inception of Windows Mobile. Microsoft Pocket PC, sometimes referred to as P/PC or PPC, is based upon the Windows CE framework. Variants of this operating system include versions such as Pocket PC 2000, Pocket PC 2002, Windows Mobile 2003/2003 SE, 5 and Windows Mobile 6.0. Variants also exist for Smartphones, such as Windows Mobile 2003 Smartphone edition. One of the key benefits of Microsoft's Windows Mobile platform is file format compatibility with the desktop versions of the company's productivity software. Mobile versions of Microsoft software, such as Pocket Word, Pocket Excel, and Pocket PowerPoint, allow individuals to view and edit these files outside of the home and office. Another benefit is integration with Microsoft's cross-platform solution, the .NET Framework. The .NET Framework and its associated class libraries handle things such as memory management, file I/O, and many other functions. The .NET Framework allows programmers to develop code in one

of several .NET languages, such as C and VB.NET. Pocket PCs run a simplified version of the framework called the .NET Compact Framework. In order to maintain synchronization and connectivity with desktop computers, Microsoft developed the ActiveSync program. The user merely has to connect the Pocket PC to the desktop computer in order to synchronize items such as appointments, contact lists, and even multimedia files.

2.2.3 Palm

A Palm is a commonly referred to as a small-scale (hand-held) computer that runs Palm's PalmOS software. The Palm OS platform is an open architecture that provides a basis for third-party developers and original equipment manufacturers (OEMs) to create mobile computing solutions. Palm Computing was founded by Jeff Hawkins, Donna Dubinsky and Ed Colligan. The original purpose of the company was to create handwriting recognition software for other devices (Graffiti). The initial idea for the devices came from Hawkins' habit of carrying a block of wood in her pocket. There are several tools available for the image acquisition and analysis of Palm devices.

EnCase, published by Guidance Software, is a complete cyber forensics software package that handles all steps of the investigative process, from the acquisition to the report creation. The software includes built-in capabilities for performing MD5 hashing, data carving, deleted file recovery, and many other functions. Although traditionally relegated to the realm of desktop computer forensics investigations, EnCase does support the acquisition and analysis of a limited number of Palm devices.

Paraben has a software application that is specifically designed for PDA forensics, PDA Seizure. This comprehensive tool allows PDA data to be acquired, viewed, and reported on, all within a Windows environment. The software comes equiped with quite a few key features. These features include the ability to encrypt saved case files, BlackBerry OS support, built-in recovery of Palm passwords, enhanced viewing on file data, complete physical and logical acquisition for Palm PDA devices, and many more. It has a few drawbacks, in that some of the material acquired from the PDAs is hard to interpret by a person that is not computer savi. Although, on the other hand it has features like a search portion that allows you to enter a search term and PDA Seizure will bring up all files that have that term in them. This allows the investigator to look for case specific information easily and quickly.

2.2.4 BlackBerry

The RIM BlackBerry is a personal wireless handheld device that supports e-mail, mobile phone capabilities, text messaging, web browsing, and other wireless information services. It is most commonly utilized for business purposes. The BlackBerry was first introduced in 1999 by a company called Research In Motion (RIM). The BlackBerry OS provides easy access to applications such as e-mail, to do list, memos, address book, and many other features. Forensics on the RIM platform is complicated by the fact that this is a "push" device, i.e., the RIM server will push device to the PDA whenever the PDA's radio is on and there is data available. RIM devices also feature a remote self-destruct feature. This feature cannot be activated if the radio is turned off, of course. Both of these features mean that you need to be sure that the radio is off when doing a forensic investigation. Depending on the setting, entering a wrong password a certain number of times will wipe the device.

2.2.5 iPhone

The iPhone is an internet-connected multimedia smartphone designed and marketed by Apple Inc. with a flush multi-touch screen and a minimal hardware interface. The iPhone was initially released with two options for internal storage size: 4 GB or 8 GB. On September 5, 2007, Apple discontinued the 4 GB models. On February 5, 2008, Apple added a 16 GB model. All data is stored on an internal flash drive; the iPhone does not contain any memory card slots for expanded storage.

2.3 Literature review

2.3.1 Mobile device forensics

The term "Mobile Forensics" arose when the first PDA was able to store personal data about its owner and, thus, it could be used as evidence during an investigation.

The Netherlands Forensic Institute (NFI) published a workflow for mobile phone forensic examination ([8]). A lot of relevant work about mobile forensics has been carried out by Jansen et al. [9, 10, 11, 12] on behalf of *NIST*. Topics of interest covered by the authors are forensic tools, impediments, procedures and principles, preservation, acquisition, examination and analysis.

All phases are well described and they are currently used as guidelines. The analysis phase lacks a contacts correlation activity.

Zdziarski in [13] describes a method that allows the examiner to perform a bit-by-bit copy of the iPhone's user partition and can provide an MD5 sum to prove the copy was authentic. The method requires modifying a read-only system partition to allow for this technique. Fortunately, this partition remains completely isolated from the partition containing user data and is intended to remain in a factory state throughout the life of the iPhone. This makes it an ideal and forensically sound location to perform the necessary payload installation, without violating user data.

Hoog in [14] reviews the forensic tools available for the iPhone, performs forensic analysis with each tool and reports on the installation, acquisition, reporting and accuracy of each tool.

2.3.2 Mobile Data Reverse Engineering

At the time of writing, we are not aware of other works about how to decode the personal information stored in the smartphone's file system. Commercial forensic products such as Paraben [15], extract a logical dump and they are able to decode it, but they provide neither their source code nor information about the storing formats.

More generally speaking, research in data reverse engineering has been under-represented in the software reverse engineering area because the latter appears to be "more challenging and interesting than data reverse engineering" [16]. Recently the attention to data reverse engineering techniques is increasing, because they represent the key to making several assessments like incorporating legacy data in data warehouses, measuring the software quality, and, more generally, migrating from closed formats toward open standards.

Most of the works found in the literature about DRE deals with DBMS reverse engineering. Since DBMSs provide the functionality to extract initial information about the implemented physical data structure, database reverse engineering has a higher potential for automation than data reverse engineering. Consequently, most existing reverse engineering tools in this area consider information systems that employ a database platform. A relevant work about the auto-reverse engineering of input file is given by Tupni [17], a tool which is able to decode various sets of data, such as record sequences, record types, and input constraints. The tool takes as input a set of different instances of a file format, and an application which is able to read them, then it learns information about the file format by tracing the application and by observing how

it reads the file. Tupni was tested on several common file formats and network protocols, and performs very well. Tupni tracks I/O operations performed by applications running on the device. This is unuseful for our scope since we are interested in tracking the Mobile device OS.

A generic methodological approach has been proposed by Aiken in [18]. The author provides a *DRE analysis template*, which is a "system of ideas for guiding DRE analysis, an overall meta-data-gathering strategy, a collection of measures, and an activity structure that can be used to assess progress toward specific re-engineering goals". Since the mobile environment is populated by a large amount of device manufacturers, operating systems, DBMS and file formats, we think that a methodological approach is more convenient and flexible than a vertical one. Therefore we started from Aiken's work and we followed a similar approach to deal with mobile environments. Thanks to this approach we will be more tolerant if something changes in file formats and OS, when new versions are released.

The most used tool to make DRE is the hex editor. Conti et al. in [19] proposed several interesting techniques to retrieve information from hex dump through visual inspection.

2.3.3 Identity profiling over the Web

In this work we have widely adopted the concept of *identity* as "that part of the self by which we are known to others" ([20]). We fetch up the identity build on the Internet through the analysis of Web pages where the subject appears in conjunction with other contacts of her mobile contact list. Previous work achieved interesting results. Mika et al. in [21, 22], dealing with the problem of "bootstrapping" a Friend-Of-A-Friend (FOAF) based social network, proposed "the traditional Web as source of information about the social networks in a community". So they introduced a system for collecting social network data which fetches data from the traditional Web by mining the index of Google. When the authors wrote (2005), they demonstrated their method through examples of specific community with a significant on-line presence, i.e., a scientific community of computer science. Since social networks spread (see Figure 1.2.3), many "common" users put themself on the Web and, in particular, they entered information about who their friends are. Thus, nowadays, we can extend Mika's experiment to common users, thanks to the part of social networks data that are available publicly on the Web and that is periodically crawled by search engines. A remarkable work about the identity construction on social networks is given by Zhao et al. in [23], where the authors study identity con-

struction on *Facebook* (`http://www.facebook.com`), an emergent social networking site which became most popular among college students in the United States and then spread across other life contexts and countries. Facebook, like other social networking environments, can be considered as a *nonymous* environment[1], because the user is persuaded to publish her real name and typically she knows her friends in the real world too, thus their digital identity is similar to the face-to-face identity. The authors accomplish their study by reporting that Facebook users claim their identities implicitly, because they "show rather than tell". The people they know are the most important data we need to reconstruct the person profile.

2.4 Available products

A collection of surveys on mobile phones and PDA forensics is maintained by NIST. An (un)updated list of available tools is reported in [10, 11, 12]. In order to achieve forensic seizure, closed and open source tools are available. The tools currently in use perform the acquisition of the mobile device internal memory in a remote way: a forensic tool is connected with the target device and, using the OS services, it extracts the data like SMS, MMS, TODO list, pictures, ring tones, etc. This approach has the advantage of minimizing the interaction towards the device and automating the process of seized data interpretation. The main disadvantage relies on the protocol closeness: we are not able to measure any memory alteration caused by data exchange. Moreover, to perform the acquisition process, a few of the available tools degrade the evidence putting an utility file in the device's file system. Moreover, currently available products extract most data from the device, but they do not provide any functionality to further analyze data, as the solution we have studied in this work.

2.4.1 SIM forensics

The data that a SIM card can provide the forensics examiner can be invaluable to an investigation. Acquiring a SIM card allows a large amount of information that the suspect has dealt with over the phone to be investigated. In general, some of this data can help an investigator to determine the phone numbers of

[1]The term "nonymous" is the opposite of "anonymous" and means an on-line relationship which exists in the real world too. This type of on-line relationships is also called "anchored relationships" ([23]).

International Mobile Subscriber Identity (IMSI): A unique identifying number that identifies the phone/-subscription to the GSM network
Mobile Country Code (MCC): A three-digit code that represents the SIM card's country of origin
Mobile Network Code (MNC): A two-digit code that represents the SIM card's home network
Mobile Subscriber Identification Number (MSIN): A unique ten-digit identifying number that identifies the specific subscriber to the GSM network
Mobile Subscriber International ISDN Number (MSISDN): A number that identifies the phone number used by the headset
Abbreviated Dialing Numbers (ADN): Telephone numbers stored in sims memory
Last Dialed Numbers (LDN)
Short Message Service (SMS): Text Messages
Public Land Mobile Network (PLMN) selector
Forbidden PLMNs
Location Information (LOCI)
General Packet Radio Service (GPRS) location
Integrated Circuit Card Identifier (ICCID)
Service Provider Name (SPN)
Phase Identification
SIM Service Table (SST)
Language Preference (LP)
Card Holder Verification (CHV1) and (CHV2)
Broadcast Control Channels (BCCH)
Ciphering Key (Kc)
Ciphering Key Sequence Number
Emergency Call Code
Fixed Dialing Numbers (FDN)
Forbidden PLMNs
Local Area Identitity (LAI)
Own Dialing Number
Temporary Mobile Subscriber Identity (TMSI)
Routing Area Identifier (RIA) netowrk code
Service Dialing Numbers (SDNs)
Service Provider Name
Depersonalizatoin Keys

Figure 2.1: Technical data which can be extracted from the SIM card

calls sent and received, contacts, sms details, sms text. There are many software solutions that can help the examiner to acquire the information from the SIM card. Several products include 3GForensics SIMIS [24], SIMCon [25], or SIM Content Controller, and Paraben Forensics' SIM Card Seizure [26]. Information which can be extracted from the SIM card are reported in Figure 2.1.

2.4.2 TULP2G

The most important open source tool is TULP2G (`http://tulp2g.sourceforge.net`). TULP2G is a forensic software framework developed to make it easy to extract and decode data from mobile phones and SIM cards. It holds on a

modular architecture ([27]) which defines a general examination workflow for investigators and offers an abstract design to developers of so called plug-ins. These plug-ins contain the actual investigation methods. From a user's perspective the advantage of using a framework concept is the "learn once apply everywhere" principle. TULP2G implements the standard modem protocols (e.g., Hayes commands) and OBEX protocol to communicate with the device. Unfortunately, if a smartphone model does not implement these standards, the tool is unusable for the investigation. Currently, TULP2G components have become obsolete, as reported by its authors in the product's Web page: "Current TULP2G plug-ins do not contain state of the art technology for the examination of mobile phones. Most plug-ins were made years ago to demonstrate framework principles. Nowadays a lot of better (commercial and open source) tools exist to assist you in the examination of mobile phones."

2.4.3 Paraben Device Seizure

Among proprietary tools, the Paraben Device Seizure (`http://www.paraben.com`) is one of the most important; it implements specific, proprietary protocols (like D-Bus for Nokia smartphone) but, as for the TULP2G tool, unknown protocols make the acquisition impossible. Recently Paraben Corporation released the "CSI stick" (`http://www.csistick.com`) a portable data gathering and forensic tool, which allows the acquisition of data without using the forensic workstation. This solution, however, still relies on proprietary plugs (currently, Motorola and Samsung).

2.4.4 .XRY

The .XRY tool by MicroSystemation (`http://www.msab.com/en`) adopts a quite similar approach to Paraben, with remote acquisition via hardware specific plugs.

2.4.5 Neutrino

Recently, Guidance Software (`http://www.guidancesoftware.com`) introduced the *Neutrino* forensic acquisition device, which is capable of acquiring data from a device connected through USB cables and able to analyze and correlate data coming both from mobile phone and computer. Moreover, Neutrino is the only device which integrates itself with EnCase package. Neutrino shares the approach with Paraben and MicroSystemation solutions, and it supports

roughly one hundred mobile phone models. It is bought with a subscription service with monthly updates for acquiring newly-supported phones, including any required cables.

3

Profiling system's architecture

3.1 The Profiling Workflow

As mentioned before, the identity profiling activity takes place in a wider process, which can be modeled as a workflow. In Figure 3.1 the entire workflow is depicted.

This process starts from the devices which have been seized. For each of them, depending on the device's manufacturer and model, and on the mobile OS, the correct version of the tool (MIAT) which will acquire the content of the internal memory must be chosen. After acquisition, the device file system's dump is processed by the personal data decoder and the personal data (i.e., contacts, calendar, messages and events log) are decoded and translated in a more suitable XML format. From this point personal data can be imported from the subsequent workflow's components, that will be unaware of the device they come from.

After being extracted and decoded, personal information needs to be analyzed in order to highlight the correlations among data and to help the investigation in increasing the efficiency and effectiveness of the process. The Mobile Identity Profiler (MIP) process, that will be described in further detail in Chapter 6, was developed in order to capture the logical equivalent meta-information that data of a specific mobile device acquisition must contain (e.g., contacts information, calls details, etc.). The tool reorganises collected data using a specific proximity measure, and it also allows to consult the collected data as they are recovered by the decoding tool. The innovative contribution of the MIP component is to reconstruct the interaction between the phone's owner and her contacts, and to reconstruct the relationships among them through the

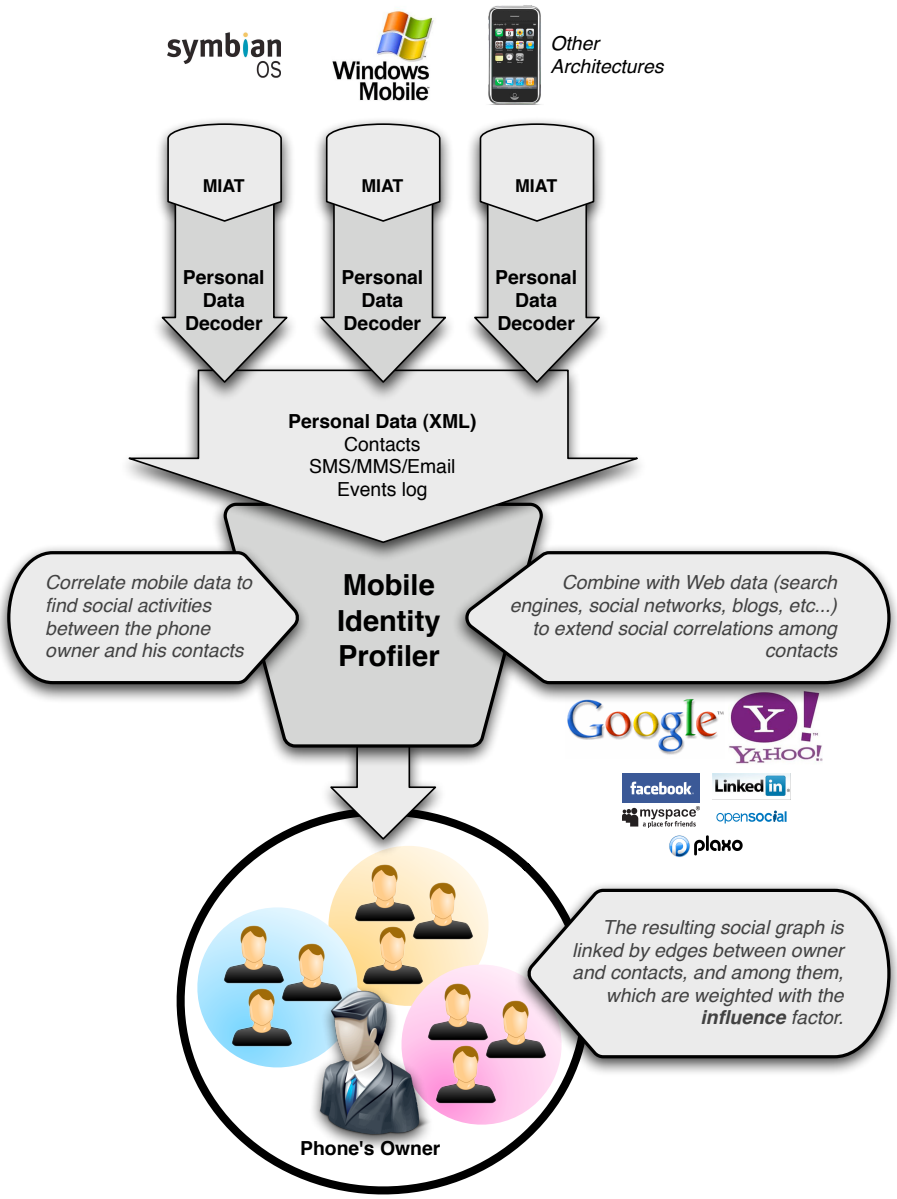


Figure 3.1: The profiling workflow.

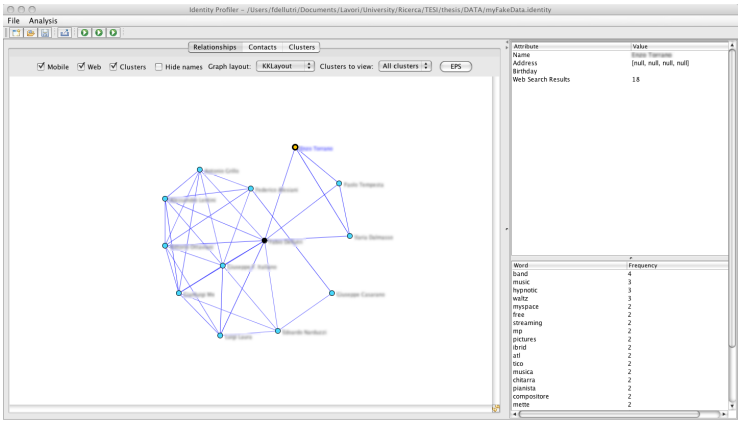


Figure 3.2: The main interface of the analysis tool with example data.

Web analysis. The design of an integrated data analysis tool based on powerful automatic analysis and manipulation data techniques represents a core activity in the process of speeding up the profiling analysis.

3.2 The Identity Profiler Framework

The *Identity Profiler Framework* is a forensic analysis environment designed to orchestrate the software components discussed earlier in this chapter. This environment is able to take a logical dump acquired via MIAT, to decode it into usable data and to analyze them through the profiling activity. The framework is composed of several pieces of software. Besides the component which allows the downloading of MIAT for the specific device and to install it on the external memory, these components also include the data decoding parsers. The core components of the framework are two:

- the collection of analysis tool that implements the MIP process, which will be described in Chapter 6.

- a convenient user interface which allows the operator to easily interact with collected data. In particular, the profile is represented as a graph where the nodes are the contacts and the edges are the connections and

the relationships existing between individuals. In Figure 3.2 is reported a screenshot of the graphical interface.

The framework is written in Java 1.6 and SWING. We adopted an open-source library, JUNG, which provides a common framework for graph/network analysis and visualization. The clustering algorithms are written in C++, using the GNU compiler g++. We used the uBLAS package in the BOOST (`http://www.boost.org/`) library to manage graph and matrices, while we used the library SVDPACKC (`http://www.netlib.org/svdpack/`) to compute the Singular Value Decomposition (SVD) of a matrix.

4

Smartphone's internal memory acquisition

The main results of this chapter have been presented in [4, 28, 29]

4.1 Introduction

In this Chapter we will describe the methodology we developed to seize data contained in the smartphone's internal memory by a logical dump on an external removable memory (if available), then we will present the tools we designed to implement such methodology.

Forensics for mobile devices follows the same rules valid for general purpose hardware. In [30] it is shown the general framework (Figure 4.1); this can be assumed as valid even for mobile equipment forensics. We notice just one very big difference between mobile and general purpose forensics: the acquisition phase. In a mobile device there are three types of memories: Read Only Memory (ROM), which stores the OS boot image as it is later hard reset; RAM Memory containing processes and OS volatile data, and Flash Memory, that stores the user's files, information and documents, and programs non volatile data. Flash memory is the device memory part in which we are interested. This is a chip integrated in the device's motherboard. As Flash memory can be erased/written a limited number of times mobile devices adopt a logging file system to grant the integrity of the transactions executed. In the acquisition phase the SIM-card and the external memory are removed but there is no way to remove the internal memory without damaging the device irreparably, as this is not, like a hard disk, a removable piece-of-hardware. Because of this,

IDENTIFICATION	PRESERVATION	COLLECTION	EXAMINATION	ANALYSIS	PRESENTATION
Event/ Crime Detection	**Case Management**	**Preservation**	**Preservation**	**Preservation**	**Documentation**
Resolve Signature	Imaging Technologies	**Approved Methods**	**Traceability**	**Traceability**	Expert Testimony
Profile Detection	**Chain of Custody**	Approved Software	Validation Techniques	Statistical	Clarification
Anomalous Detection	**Time Synch.**	Approved Hardware	Filtering Techniques	Protocols	Mission Impact Statement
Complaints		**Legal Authority**	Pattern Matching	Data Mining	Recommended Countermeasure
System Monitoring		Lossless Compression	Hidden Data Discovery	Timeline	Statistical Interpretation
Audit Analysis		Sampling	Hidden Data Extraction	Link	
		Data Reduction		Spatial	
		Recovery Techniques			

Figure 4.1: DFRWS Digital Investigation Framework.

the most delicate operation to be done in a mobile environment is the seizure of the internal memory's data.

The Mobile Internal Acquisition Tool (*MIAT*) acquires data directly from the internal memory to an external removable memory (like SD, mini SD, etc.), spawning an acquisition application stored in the same memory card held by the forensic operator. This task is performed without the need of connecting the device to PC. Thanks to this, forensic operators can avoid traveling with luggage full of one-on-one tools for every single mobile device. Even if the NIST guidelines say that "to acquire data from a phone, a connection must be established to the device from the forensic workstation" ([11]), we believe that MIAT approach does not contrast those guidelines, but extends the forensic workstation concept to the removable memory where the MIAT executable file resides. The complete data seizure process is shown in Figure 4.2. In order to acquire the memory content of a GSM, Bluetooth or Wi-Fi enabled mobile device, it is mandatory to shield the device with a Faraday cage [31]. Indeed, new incoming calls, SMS, e-mails, Bluetooth activity, connection status changes or GSM cell switch, could trigger events which may modify some of the file system's objects.

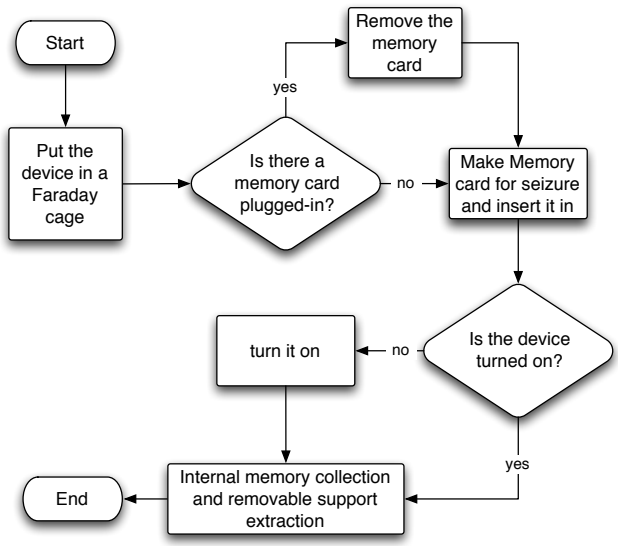


Figure 4.2: Data collection workflow

4.2 Our methodology

The MIAT is an alternative way to seize the internal memory data: it relies on local execution of an application which explores recursively the file system tree and copies each entry to a backup volume like an expansion memory card (Figure 4.3). Before proceeding with the acquisition, the device should be switched off. Then, if the SIM and/or the memory card are inserted in the smartphone, we must remove them to collect the stored data. We note that the SIM card is usually located under the battery, for this reason we must turn off the device. Once the SIM and the memory card have been acquired, we use the host memory card (different from the original memory card found in the device, part of the seizure) for the internal memory seizure: a tool for seizing data is stored in a memory card and the acquisition is performed locally. In order to grant data integrity, during the acquisition process all files and directory are opened in read-only mode to preserve integrity. Moreover, in order to detect further corruptions, MIAT computes an MD5 hash before and after copying each file; MIAT also compiles a log file with all remarkable events (as shown in Table

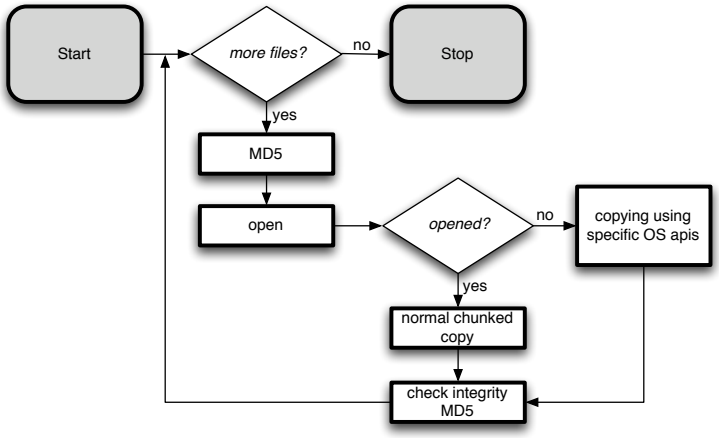


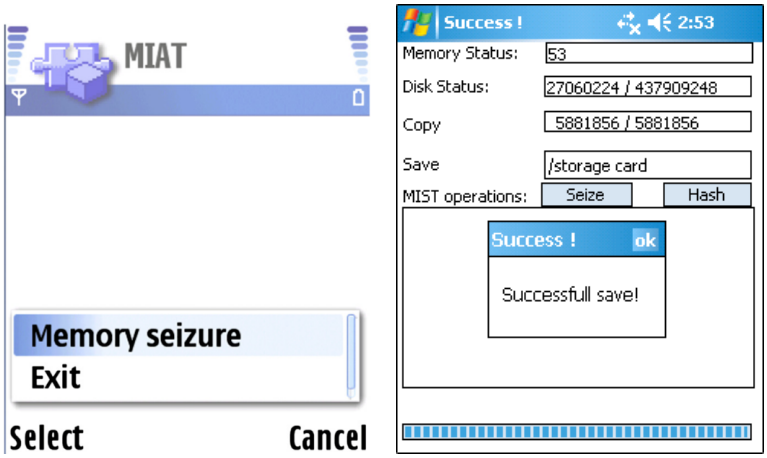
Figure 4.3: How MIAT works.

4.1) in an XML format.

File	Contents
checksum.xml	File size, file typology, file name, MD5 hash, seizure, duration, and creation, access and modification time.
info.xml	Information about the device seized (IMEI, device ID, platform type, model, manufacturer), and about the seizure process (duration, battery consumption date of seizure).
errors.xml	Information about errors that may happen during the process.

Table 4.1: Files generated during the Seizure Process

The main advantage of this approach is to access the whole file system, then to seize the files stored in non conventional location (for example pictures hidden between system files). The tool uses the standard mobile OS APIs to access the file system (like Open, Read and Write), which is typically invariant during the OS's evolution. The data seized on the removable storage support can be accessed with a common MMC o SD reader, and analysed with ad-hoc tools. The adoption of this methodology means saving hardware tools like USB cables specific for each device or additional equipment like notebook PC to perform the acquisition; the forensic workstation is now the seized Mobile Equipment (*ME*) with a supplementary SD/MMC memory card with MIAT



(a) (b)

Figure 4.4: These figures show screenshot version of MIAT we developed. In (a) MIAT for Symbian. In (b) MIAT for Windows Mobile.

onboard.

The difference between MIAT and some other forensic tools (e.g., TULP2G) is that the former interacts directly with the Operating System while the latter requires an intermediary (located in the mobile phone) in charge of managing the messages sent by remote forensic tool to the mobile phone. Since the intermediary code is almost always closed, the second case makes it impossible to verify how the intermediary code is written. Whenever the source code is not available, obviously, it is impossible to state by code reading if the tool respects integrity and the other required fundamental forensic properties.

We implemented MIAT for Symbian and Windows Mobile platforms. The Figures 4.4(a-b) show the screenshots.

4.3 Implementation

4.3.1 Symbian

MIAT for Symbian, presented in [3], was developed to support and to test the methodology described above. Symbian is an operating system derived from

the Epoc operating system; Symbian OS supports a wide range of device categories with several user interfaces, including Nokia S60, UIQ and the NTT DoCoMo common software platform for 3G FOMATM handsets. The commonality of Symbian OS APIs enables development that targets all of these phone platforms and categories. In order to produce executable code which does not need of any other software layer (e.g., a JVM to interpret the bytecode) MIAT application was originally developed in C++, the native language of the Symbian OS. Note that since there are many versions of each combination OS/UI, there is a different SDK for each combination. You have to build application for specific devices using the appropriate SDK of the correct version of the target phone. To improve the operational efficiency of the forensics operator, different versions of MIAT were compiled based on the various SDK. The specific required version of MIAT is recognized by the IMEI number as mentioned in the methodology above.

Most relevant files are locked by system processes, many files on the system are always open and locked by system processes. For example the file Contacts.cdb, which contains the database of contacts, is locked by PhoneBook that is the address book process. In the past ([3]) we made use of the OS Backup service to perform seizure of locked data. Such service is an utility allowing the backup of the memory contents, even if these contents are locked. An application or a service can register itself and the files which locks. The Backup Server notifies a backup request to registered applications, so they can release the lock temporarily. Once the file had been seized, the MIAT application could notify this to Backup Server and then the system process could re-acquire the lock. In a recent work ([32]), we adopted a further alternative way to get access to locked files. This way is accomplished by the Symbian RFs API method ReadFileSection that allows a file to be read without opening it. By this method it is possible to seize the entire file system tree including files which have a persistent lock on; furthermore this strategy preserves integrity because the access is established in read-only mode, guaranteed by the OS.

4.3.2 Windows Mobile

Besides the Symbian version, we developed a version of MIAT for Microsoft Windows Mobile Edition and we presented it in [4].

The term *PocketPC* refers to a Microsoft specification that sets various hardware and software requirements for a handheld-sized computer (*PDA*, Personal Digital Assistant) that runs the Windows Mobile operating system. As reported in Table 4.2, many tools perform forensic operations on a PDA. How-

ever, as Ayers et al. assert in [11, 12, 33], the only NIST certified tool on a PocketPC is Paraben's *PDA Seizure*. This tool performs data seizure of internal memory in a remote way. Actually, the forensic tool is connected to a device by cradle or USB cable and, through the Microsoft's ActiveSync protocol, it extracts data such as user's files, call logs, SMS, MMS, TODO list, etc. This approach has the advantage of minimizing the interaction towards the device and automating the process of seized data interpretation. The main disadvantage relies on the protocol closeness: we are not able to measure any memory alteration caused by data exchange. Moreover, to perform the acquisition process, PDA seizure degrades the evidence putting a dll file in the device's file system.

	Palm OS	PocketPC	Linux PDA
pdd	Acquisition	NA	NA
Pilot-Link	Acquisition	NA	NA
PDA Seizure	Acquisition, Examination, Reporting	Acquisition, Examination, Reporting	NA
EnCase	Acquisition, Examination, Reporting	NA	Examination, Reporting
POSE	Examination, Reporting	NA	NA
dd	NA	NA	Acquisition

Table 4.2: PDA forensic tools

In Windows Mobile 2003 PocketPC and earlier, the device's memory was split in two sections: a ROM section, containing all operating system core files, and a RAM section aimed at keeping the user storage (Storage Memory) and the memory space for running applications and their data (Program Memory). The user can choose the amount of memory to be reserved to Storage Memory and then to the Program Memory. The RAM chip was built on a volatile memory scheme, so a backup battery was required to keep the RAM circuitry powered up, even if the device was just suspended. In case battery power supply went down, all the user's data were lost. Such a scenario forced the user to recharge the battery within a time limit of 72 hours (as mandatory by Microsoft to devices manufacturers).

Since Windows Mobile 5, memory architecture has been redesigned to implement a non-volatile user storage. Currently, the memory is split in two sections (see Figure 4.5): the RAM holds running processes data, whereas the ROM keeps core OS code and libraries (called modules), the registry, databases and user's files. Such memory, also called Persistent Storage and contained

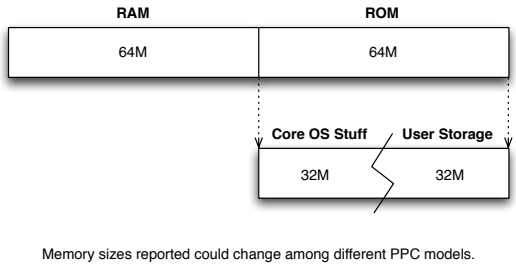


Figure 4.5: Windows Moble 5.0 memory architecture.

within a flash memory chip, can be built using many different technologies (see [34]):

- **XIP model**, based on NOR memory and volatile memory, this technology enables a device to store modules and executables in XIP (execute-in-place) format and allows the operating system to run applications directly from ROM, avoiding coping them first in the RAM section. NOR memory has poor writing performance.

- **Shadow model**, which boots the system from NOR and uses a NAND for the storage. This model is power-expensive, because the volatile memory requires it to be constantly powered on.

- **NAND store and download model**, which reduces costs replacing NOR with OTP (one-time programmable) memory model.

- **Hybrid store and download model**, which mixes SRAM and NAND, covering them with a NOR-like access interface (to support XIP model).

Windows Mobile 5 and above place most of the applications and system data in the Persistent Storage. Core OS files, user's files, databases and registry are seen by applications and users in the same file system tree, which is held and controlled by the FileSys.exe process. Such a process is also responsible for handling the Object Store, which maps objects like databases, registry and user's files in a contiguous heap space. The Object Store's role is to manage the stack and the heap memory, to compress and to expand files, to integrate ROM-based applications and RAM-based data. For a comprehensive explanation

about how Windows Mobile uses the Object Store and manages linear flash memory, see [35, 36].

The strategy for storing data is based on a transactional model, which ensures that a store is never corrupted after a power failure while data is being written. Finally, the Storage Manager manages storage devices and their file systems, offering a high-level layer over storage drivers, partition drivers, file system drivers and file system filters.

Filename	Location	Description
System.hv	/Documents And Settings/	System registry hive.
User.hv	/Documents And Settings/default/	User registry hive for default user.
Default.vol	/Documents And Settings/	Object store replacement volume for persistent CEDB databases. This file contains MSN contacts
Mxip_system.vol, Mxip_lang.vol, Mxip_notify.vol, Mxip_initdb.vol	/	Metabase volumes, including language-specific data and storage for notifications.
Cemail.vol	/	Default SMS and e-mail storage.
Pim.vol	/	Personal Information Manager (PIM) data, such as address book, schedules, SIM entries, call logs.

Table 4.3: Windows Mobile relevant files

Unlike old Symbian smartphones, where the user is forced to remove the battery supply to remove the memory card, in a standard PocketPC it is possible to plug-in a memory card (typically an SD) while the device is powered-on (*hotplug*). This is a great chance for collecting data which, otherwise, could be altered if the device was turned off before the seizure process. Moreover, we developed the application using a native C++ approach, fulfilling the requirement of having a tool to be launched from an external memory card, without the need of a pre-installed runtime environment (like java virtual machine), nor the need to install the tool on the device.

4.4 Results and conclusions

A first intrinsic result obtained by MIAT philosophy is that as it is an open source software, is it possible to state by code reading if the tool respects integrity and the other required fundamental forensic properties. Since other

application (Paraben, TULP2G) codes are almost always closed, this makes it impossible to verify how their intermediary code is written.

In order to evaluate the performance, we compared MIAT for Symbian and WM with Paraben products. As a measure, we chose to detect which files were modified after the data seizure. We noticed that MIAT (both versions) performs as well as Paraben in coverage and in integrity. The results of the comparison are reported in Tables 4.4 (Symbian) and 4.5 (Windows Mobile). Moreover, in both versions for Symbian and for Windows Mobile, MIAT is slower than Paraben in seizure times. Those are constrained by device type and filesystem density. Moreover, MIAT copies the filesystem in a logical way, instead Parabens seems to get a copy of the device ROM at a lower level, by putting a library in the device's filesystem.

Furthermore, MIAT can lower the effort for law enforcement agencies (LEAs), due to the non-technical skills required for the forensic operators; hence, we expect the MIAT and the proposed methodology to speed up the forensic acquisition process, especially when the amount of devices to analyse can overwhelm the high tech crime units. Currently the MIAT tool is being experimented by a LEA, in order to establish if it could be proposed as a repeatable seizure tool in respect of Italian laws.

Future developments of MIAT methodology will include (but are not limited to) the OS DRM mechanism, in order to maintain the highest level of device portability on Symbian phones and the development of an analysis farm, where to forward all the seized images (from the crime scene) in order to provide a real time standard analysis of the mobile equipment.

File	Reboot	Seiz.	File	Reboot	Seiz.
100056c6.ini	B		101f6df0.ini	B	B
AlarmServer.ini	B	P	Applications.dat	B	M
backupdb.dat	B	P	btregistry.dat	B	
cbtopicsmsgs.dat	B		CntModel.ini	B	P
CommonData.D00	B	P	DRMHS.dat	B	
ECom.lang	B		HAL.DAT	B	
LocaleData.D05	B	P	nssvasdatabase.db	B	B
ScShortcutEngine.ini	B	P	smssmssegst.dat	B	
System.ini	B				

B Means that a change happens for both tools
P Means that a change happens for PARABEN
M Means that a change happens for MIAT

Table 4.4: MIAT for Symbian and Paraben's Device Seizure comparison, and their hashes consistency. This table also shows the event which trigger changes.

File	Paraben	MIAT-WM
/Documents And Settings/default.vol	–	–
/Documents And Settings/system.hv	–	–
/Documents And Settings/default/user.hv	–	–
/Windows/*.dll	–	–
/mxip_notify.vol	√	⋆
/cemail.vol	√	⋆
/mxip_system.vol	√	√
/mxip_lang.vol	√	√
/pim.vol	⋆	√

– file not copied
⋆ file copied but its hash does not match
√ file copied and hash matches

Table 4.5: MIAT for Windows Mobile and Paraben's PDA Seizure comparison, and their hashes consistency.

5

Smartphone's data reverse engineering

The main results of this chapter have been presented in [37]

5.1 Introduction

After extracting the file system's logical dump from a smartphone, we need a method to decode personal data stored within several mobile DBMS files and to make them available to other applications. Such DBMS files contain actual and obsolete data, i.e., old or deleted entities; this occurs because the mobile OS, for performance reasons, defers the deletion as long as possible, e.g., when the free space available in the file system is not enough. Often mobile DBMS can be accessed via APIs provided by mobile OS manufacturers, but they are prevented from accessing those data, therefore they are not useful in the forensic environment. Therefore we chose a Data Reverse Engineering (DRE) approach to retrieve and decode the storing format. In the traditional architectures (PCs and mainframes) the DRE was studied as business solution either for the control of data handled via legacy applications or in order to reconstruct deteriorated data. Developed models are too generic for mobile environments [18], or they aims at discovering mainly the data model [38, 39, 40], or have been studied to address vertical problems like extracting data from COBOL, DB/2 [41] or Access. For our scope, we are not interested in discovering the data model because we know *a priori* which data we are looking for (e.g., all the user controllable data attributes like contact's name and surname or SMS text), and we do not care about the relational structure. Moreover,

a great facility given by a methodological DRE application, is that, when file formats change, after re-applying the methodology we are able to update our knowledge about how data are stored.

In this chapter we will propose a methodology allowing smartphone's DRE operators to be more flexible in the mobile file formats knowledge. As a matter of fact, the mobile phone environment is composed of a plethora of manufacturers and operating systems, each of them is released in several versions which stores data in different formats. Handling such heterogeneity through a methodological approach is an important asset for mobile forensics and, more generally, for smartphones' businesses.

As a case study we applied these methods to the Symbian OS, and we obtained several results, including the mapping between a given data and its location into the file system, the obsolete data recovering, and the Symbian personal databases format reversed. The obtained results (see Section 5.3) show that our methodology can be successfully applied to environments which are different from our forensic starting point. The methodology helps to decode databases files and to develop ad-hoc parsers; data extracted by such parsers can be easily converted and used to perform tasks such as user profiling, device syncing or data recovery.

5.2 Our step-by-step Methodology

Smartphone's operating systems save personal data in many DBMS tables which are stored in binary files. Often the format of such files is not public and the tools available to read them rely on an operating system native API (if they run on the device) or on a porting of their code (if they run on a PC), and they can not extract data in a forensic way, i.e., they do not retrieve information of high investigative value as deleted or modified data. Therefore, the solution is to interpret the binary file directly in order to give a structure to the internal data. Initially the problem was addressed through the comparison of multiple files of the same type, relying on the analyst's ability in the intuitive interpretation of the data content. In such way the analysis of data was often confused and led to performing redundant operations without any result. Therefore, in order to preserve obsolete data, we chose to design a methodology for the binary file interpretation, which was able to decode the information required without performing redundant operations. Furthermore, the methodology will help to retrieve the data alterations and deletions.

Our main contribution is to propose a wisdom-driven DRE methodologi-

cal approach to decode smartphone's personal data, that are stored in several DBMS-managed files; with this work we provide the tools to reach the following targets:

- understand where information is stored in the mobile device's file system;
- retrieve and decode personal actual and obsolete data;
- develop a suitable parser.

A flow-chart of the methodology is shown in Figure 5.1.

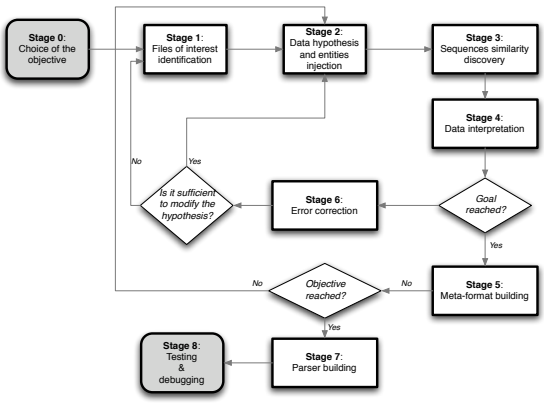


Figure 5.1: The methodology flow

Stage 0 aims at choosing which kind of information (the *objective*) we want to find and how it can be decoded. An objective is composed by one or more *goals*. We may think of an objective as an entity (e.g., a contact, or a call log, or a SMS) composed of one or more fields (e.g., for a contact, the first name, the last name, the phone number, etc.), which are the *goals* of our objective. Stage 1 aims at identifying which files (*file of interest*) could contain data (our goal) we wish to decode. With Stage 1, the methodology enters in a iterative process which allows to understand the binary format of data by comparing different versions of it.

In Stage 2 some assumptions about the data type are made. Such assumptions lead the choice of sample instances of entities to be inserted into the device's databases. Instances are stored as records which are contained in one or

more binary files. If required, the hypotheses made in Stage 2 will be refined in Stage 6 and the instances may change. The number of instances inserted will determine the number of comparisons among binary records, that will affect the precision of next Stage. Stage 3 deals with the binary files' content formatting, in order to make the data instances inserted in Stage 2 identifiable and comparable. Usually, we try to group similar zones within the same sample binary file, and among different sample binary files, and then proceed to the interpretation.

Formatting must take into account the data interpreted successfully in previous iterations, in order to cut them off (i.e., data already analyzed) from the study of a new format. The Stage 4 comprises two sub-tasks: the first deals with identifying candidate bytes sequences, and the second aims at decoding the candidate bytes sequences. The identification of candidate bytes sequences is performed by removing all the sequences that do not match with the hypothesis of the Stage 2. The second task tries to find the connection between the data inserted in Stage 2 (the instances) and its binary representation. As depicted in Figure 5.1, the methodology iterates through Stage 1, 2, 3, 4, and 6 (error correction) until a goal is reached, i.e., the information about the format of a entity's field is exhaustive and a mapping between the field and its binary storing format is found. The fifth Stage simply annotates in a meta-format all mapping information found. If the joining of all meta-format found allows the decoding of the entire objective (the information needed) identified in Stage 0, the methodology goes to Stage 7. At this Stage a piece of software able to decode automatically the now-exposed file format will be designed and implemented. All collected knowledge about the format turns into a set of software requirements. This process must be repeated for each file marked as *file of interest*. Such a piece of software will be tested at Stage 8.

In the following sections we will describe each methodology Stage.

5.2.1 Stage 0: Choice of the objective

Before starting, we must to choose from which data we want to start the decoding process. We define as *objective* the type of personal data (e.g., contacts, SMS, email, calendar, events log, etc.) we want to find into the device's file system and to decode the binary format. An objective can be seen as the set of "atomic" goals that must be completed in order to reach the objective. For instance, in order to decode the contacts (the *objective*), after having detected in which file (or files) they are stored, we have to find how the contact's data elements (*goals*) are encoded. Such goals are attributes such as name, surname,

mobile phone number, e-mail, street address, etc.

Let an objective Γ be a set such that it contains the list of goals we want to reach.

$$\Gamma = \gamma_1 \dots \gamma_n$$

In this Stage we can only define roughly an approximation of Γ: thanks to information about the objective's data format that we will learn progressively in the next Stages, we will be able to refine Γ with more accurate goals.

5.2.2 Stage 1: Files of interest identification

Given the objective chosen in the previous Stage, this step aims at identifying files to be analyzed and decoded in next Stages. Mobile devices save personal data in database files stored persistently in the file system. To identify the files containing the information we are looking for, we first need to cause a lot of changes inside these files in order to make them identifiable. These changes are objective-dependent: if we are looking for contacts, we will generate activity like contact insertion; if we are looking for events log, we will make calls, sim-changes, and send and receive SMS. Each of these operations generates an entity (E) which will be stored as one or more records in the file system. Each entity E is a set composed by $m \in N$ attributes (ϵ).

For each goal $\gamma_i \in \Gamma$ there is a set of attributes $\epsilon_j \in E$ such that, after discovering the encoding of each ϵ_j in the set, the goal γ_i will be reached.

We define Ω as the sequence $\{E_1, \dots, E_n\}$ of entities we have to insert in the device in order to modify all the files involved in the given objective.

The value of n depends on the objective's type and on how its entities are stored. Then, n can only be supposed as the process starts, but it could be refined over the methodology's iterations if needed.

For instance, let E be a contact's card: each $\epsilon_i \in E$ will be an attribute such as name, surname, date of birth, phone number, email address, and so on.

As a best practice, there is the need to fill every $\epsilon_i \in E$ attribute in order to modify all possible files involved in Γ.

Let A be the fileset (in our case, the whole device's file system) before performing the Ω operation set on the device. Let B br the fileset after performing Ω operation set. The application T of operations set Ω on the device is:

$$T_\Omega : A \rightarrow B$$

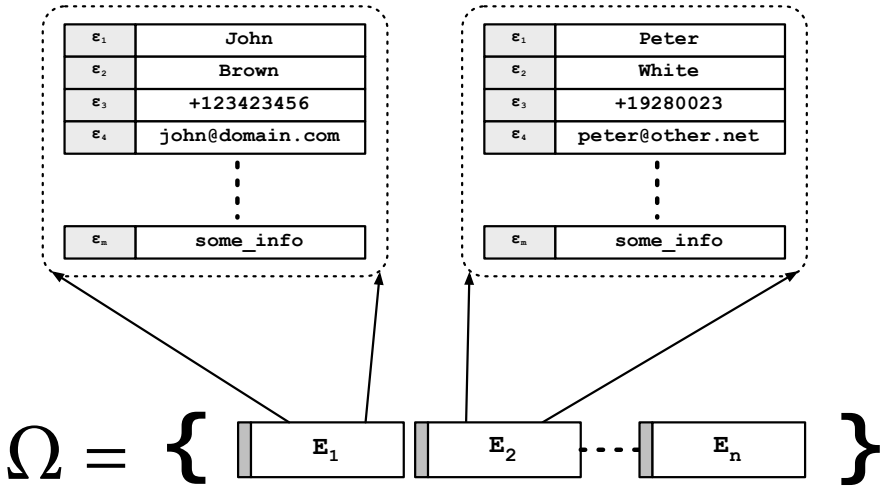


Figure 5.2: The format of the Ω operations sequence. In this figure is shown an example with contacts discovery as objective

Let *diff* denote the function which computes the differences between two filesets. The fileset C, which contains only files modified by the T application, is:

$$C = \mathit{diff}(B, A)$$

C may contain garbage data, since other operations may occur when the user performs T. Then, we must "clean" C, searching and deleting all irrelevant data. Let *clean* denote the function which cleans a fileset of garbage data. The fileset Φ is:

$$\Phi = clean(C)$$

5.2.3 Stage 2: Data hypotheses and entities injection

After the insertion of the Ω entities, the Φ set tells us which files have been modified, but it still does not give us information about how the ϵ_i are encoded in the storage.

In Stage 2 we will perform three tasks:

1. We make assumptions about the possible ϵ_i format. The Λ set represents the collection of assumptions we made at this Stage. Λ is composed by assumption about *data type*, *size* and *predictability*. The latter indicates if we can control the value of ϵ_i. Possible values of *predictability* can be the following:

   - *controllable*: the attribute correponds to any input field and the user can fully control its value. An important property, for the methodology application, is that controllable attributes can be stored more than once in the device, and the corresponding byte sequence is always the same. In contacts case, controllable attributes are input fields like name, surname, phone number, etc. If we hit the right type and size, we will be able to predict the binary (hexadecimal) version of the data.
   - *uncontrollable*: the attribute does not correspond to any input field and the user is prevented from handling its value; there is no way to predict the binary version of the data. In the contacts case, the contact's ID is an uncontrollable attribute, because it is transparently assigned by the system.
   - *pseudo-controllable*: the attribute does not correspond to any input field and the user is prevented from handling its value, but it can be partially predictable in its binary version. For instance, if we store two contacts in the same day, the year/month/day part of the insertion date (the 6 most meaningful bytes, for 8-bytes date format) will be the same for both of them.

2. Once the assumptions at the previous point have been made, we generate a set Ω' of sample entities which have all attributes but the i-th set to NULL:

$$\Omega' = \left\{ \begin{pmatrix} \epsilon_1 = NULL \\ \dots \\ \epsilon_i = v_1 \\ \dots \\ \epsilon_m = NULL \end{pmatrix}, \dots, \begin{pmatrix} \epsilon_1 = NULL \\ \dots \\ \epsilon_i = v_k \\ \dots \\ \epsilon_m = NULL \end{pmatrix} \right\}$$

   where $|\Omega'| = k$, $\epsilon_i = v_a \in \{v_1 \dots v_k\}$, $\epsilon_j = NULL$, $\forall\ j \neq i$. Values v_a will be chosen as they will be easily identified trough all file bytes

```
00 11 F2 66 B8 58 2A E1 00 01
60 02 00 00 00 00 00 1B 00 00
00 63 02 30 06 31 31 39 02 0C
00 00 00 46 00 DD 6E 82 BD 58
2A E1 00 01 60 03 00 00 00 00
00 1B 00 00 00 63 02 30 06 31
31 39 02 0C 00 00 00 46 00 9C
7F 98 C3 58 2A E1 00 01 60 04
00 00 00 00 00 46 00 00 00 63
02 30 06 31 31 39 02 0C 00 00
00 46 00 C1 EB 9C C9 58 2A E1
00 01 60 05 00 00 00 00 00 46
00 00 00 63 02 30 06 31 31 39
02 0C 00 00 00 46 00 FB 96 CA
CF 58 2A E1 00 01 60 06 00 00
00 00 00 C8 00 00 00 63 02 30
06 31 31 39 02 0C 00 00 00 46
```

(a)

```
00 11 F2 66 B8 58 2A E1 00 01
60 02 00 00 00 00 00 1B 00 00
00 63 02 30 06 31 31 39 02 0C
00 00 00 46 00 DD 6E 82 BD 58
2A E1 00 01 60 03 00 00 00 00
00 1B 00 00 00 63 02 30 06 31
31 39 02 0C 00 00 00 46 00 9C
7F 98 C3 58 2A E1 00 01 60 04
00 00 00 00 00 46 00 00 00 63
02 30 06 31 31 39 02 0C 00 00
00 46 00 C1 EB 9C C9 58 2A E1
00 01 60 05 00 00 00 00 00 46
00 00 00 63 02 30 06 31 31 39
02 0C 00 00 00 46 00 FB 96 CA
CF 58 2A E1 00 01 60 06 00 00
00 00 00 C8 00 00 00 63 02 30
06 31 31 39 02 0C 00 00 00 46
```

(b)

```
00 11 F2 66 B8
58 2A E1 00 01 60 02
00 00 00 00 00 1B
00 00 00 63 02 30 06 31 31 39
02 0C 00 00 00 46
00 DD 6E 82 BD
58 2A E1 00 01 60 03
00 00 00 00 00 1B
00 00 00 63 02 30 06 31 31 39
02 0C 00 00 00 46
00 9C 7F 98 C3
58 2A E1 00 01 60 04
00 00 00 00 00 46
00 00 00 63 02 30 06 31 31 39
02 0C 00 00 00 46
...
...
```

(c)

Figure 5.3: These figures show an example of a DBMS binary file before and after the Stage 3. In (a) the sample file after making pairs of calls of the same duration (Stage 2). In (b) equal sequences highlighted. In (c) the formatted file $\hat{\Phi}'$

in the next Stages. A good choice for v_a values critically influences the subsequent steps; in the early iterations of the methodology, v_a should be chosen with values that, disposed in the Ω' entity sequence, follow a periodical repetitive pattern (e.g., AABB, ABAB, AAAA, etc.). Thanks to this approach, in the next Stages we will be able to retrieve them through a pattern similarity matching, avoiding ambiguities in the modified file's zones caused by insertion side effects.

3. Finally, we have to insert Ω' entities into the device through an application $T_{\Omega'}$, and then we have to perform a new file system dump in order to analyze the files generated via the insertion performed in the previous task.

The output of this Stage is Λ and Φ', the set composed by all files containing Ω' entities.

5.2.4 Stage 3: Sequences similarity discovery

The goal of this Stage is to get the Φ' fileset, containing the sample entities, and to find all sequences of bytes which present the same similarities as the attributes of Ω' entity set inserted in Stage 2. In the previous Stage, we injected entities which shared one or more attributes among them. The attributes of

entities was injected following a pattern, like pairs of calls with the same duration or contacts with the same fields. In this Stage we have to highlight the file's byte sequences which are equal among them. In the call duration example, if we made c pairs of calls with the same duration, we will find c equal pairs of byte's sequences in the events log file. Therefore, if the assumptions of the previous Stage were correct, the current step simplifies the interpretation tasks in the next Stages reducing the file's complexity.

The Stage 3 process iterates through the following steps:

1. Discard file zones which are not directly affected by the operations in Stage 2;

2. Identify attribute separation flags;

3. Identify, highlight and separate similar byte sequences.

In Figure 5.3 is shown an example in which we are going to format the event log file to detect the storage format of the voice call duration. In the previous Stage we made pairs of calls of the same duration (Figure 5.3a). All the useless information (metadata, index and tables) was discarded and similar zones were looked for in accordance with the methodology described.

Once similar zones are identified (Figure 5.3b), they have to be formatted in the same way to enable the next Stage to refine the identification and to understand which file parts were changed after the Ω' entities insertion. We must separate the user added information from other file data (Figure 5.3c). A good file formatting is given by isolating different file zones from similar ones, and then by isolating flags.

The output of this Stage is the $\hat{\Phi}'$ containing the formatted fileset.

5.2.5 Stage 4: Data interpretation

Stage 4 is composed of two steps; the *candidate sequence identification* and the *candidate sequence interpretation*.

The **candidate sequences** are sequences of bytes, stored in the $\hat{\Phi}'$ fileset, in which we are likely to find the data we are looking for. $\Sigma_{\Gamma,\Lambda}$ is the set of candidate sequences for a given objective Γ, and under a given assumption Λ.

The **candidate sequence identification** relies on the hypothesis about attribute data properties made in Stage 2, and it deals with simplifying the sequence, deleting all non-relevant data. In particular:

- If the data is **constant** it is always stored in the same format, so the formatted files containing the data can be simplified by removing all the different bytes; if the data's size is equal to the size in the assumptions made, such data is added to $\Sigma_{\Gamma,\Lambda}$;
- If the data is **variable** probably the storing format will be always different, so all the equal formatted files parts can be removed to simplify. If the data size is equal to the size in the assumptions, such data is added to $\Sigma_{\Gamma,\Lambda}$;
- If the data is **pseudo-variable** the storing format will be partially constant and partially variable; we have to look for the constant parts of the file and, then, we can look at the proximity of the constant zone in an area with its size equal to the hypothesis. Then the sequence is added to $\Sigma_{\Gamma,\Lambda}$.

If $\Sigma_{\Gamma,\Lambda} = \emptyset$ or $|\Sigma_{\Gamma,\Lambda}|$ is *large* (unmanageable quantity), in order to reduce the number of resulting candidate sequences, we have to analyse the results and understand how to change the Λ assumptions made in Stage 2 (through Stage 6). Once the assumptions are modified and the new Ω' entities are inserted in the device (reiteration through Stages 2, 3 and 4), the precision of this Stage will improve.

When we reach a manageable size of $|\Sigma_{\Gamma,\Lambda}|$, the **candidate sequence interpretation** task can start. In this step we consider Λ to better understand which part of the *candidate sequence* represents the data we are interested in. We look at the Ω' sequence of operations and check if the sequence does match in the *candidate sequence* set. If the sequence of attended values of attributes in Ω' is the same in the $\Sigma_{\Gamma,\Lambda}$, the sequence is ready to be interpreted. As the database files are usually in hexadecimal format and the target data are in a different format (e.g., string, decimal format), it is necessary to transform data in a common format (e.g., decimal).

The last step to be performed is to compare data contained in the database with the data inserted in Ω' entity sequence and, if those match, the storage format is saved and the next Stage starts.

5.2.6 Stage 5: Meta-format building

After the data decoding in Stage 4, we need to store the information collected in a intermediate format. This Stage should be seen as a "methodology intermediate status saving", which helps the operator to choose the next γ goal to process, and to refine it if required. Before compiling the meta-format, this

Stage requires the compilation of a "formats table". In such a table a list of data discovered at Stage 4 is reported, and for each data the following metadata are shown:

Field Name Is a text placeholder associated with the data. This label will be substituted to the data value in the $\hat{\Phi}'$, in order to make its retrieval easier.

Size The size of data, expressed in bytes.

Description Other information, useful to the parser building Stage, like: which information is held by the field, type of data, endianess, suggestions for the automatic data localization, etc.

Example An example value of the field.

Each discovered data needs a row in the table. An example of formats table is shown in 5.4a.

After compiling the formats table, the meta-format file will be equivalent to the sample binary file purged from non-relevant bytes. Data such as headers, indexes, etc, can be deleted if they are not relevant for the purposes of the objective. The first step to be performed is to identify, for each entry in the table, the values with which the data is manifested into the meta-format file (figure 5.4b) and to replace them with the related labels in the table (figure 5.4c). In this way all relevant data in the meta-format file will be replaced by placeholders that will be easily detected at the parser building Stage.

The example shown in Figure 5.4 takes into account a contacts file containing two records with following fields: name, surname and company.

After this Stage, the given binary file could be automatically interpretable, if all the following conditions are satisfied:

1. The meta-format's data and values not yet identified have a static size, so they can be ignored. In this case the parser is able to skip them automatically;

2. All required meta-format's data and values are identified;

3. If after having tried different hypotheses of Ω', the identified zones in the meta-format did not change at all, then the meta-format file and the formats table are *stable*.

5.2.7 Stage 6: Error correction

This Stage will be performed if the current γ_i was not reached (e.g., Stage 4 was unable to find a correct interpretation for the ϵ_i representation) and it is mandatory to re-iterate the methodology. The error leading to this Stage can be caused by two cases. In the following list we show the actions to be performed in the next iteration:

1. $\Sigma_{\Gamma,\Lambda} = \emptyset$ or $|\Sigma_{\Gamma,\Lambda}|$ is *high* (unmanageable quantity): if there are no candidate sequences or there are too many, some backtracking needs to be performed to obtain a manageable number of candidate sequences. Some actions may be useful to do this:

   (a) **Changing the assumed data size**. This implies reformatting the Φ', building up a new $\hat{\Phi}'$. If $\Sigma_{\Gamma,\Lambda} = \emptyset$ and we are looking for matching sequences, we need to decrease the size. Two different big sequences might contain two matching smaller sequences. On the other hand, if we are looking for non-matching sequences, the size needs to be increased. In the case where $|\Sigma_{\Gamma,\Lambda}|$ being *high*, if we are looking for matching sequences we need to increase the size, and decrease it for non-matching sequences.

   (b) **Modifying** Ω', adding or deleting entities, or changing the ϵ_i values. A new Ω' could give as output more accurate results. The changes should be done according to the feeling of the operator, this is the hardest part of the whole process and the operator's skills play the starring role.

   (c) **Verifying** Φ' **correctness**. Verify that the file we are looking into is the right one (the required information may reside in another file).

2. If the interpretation of candidate sequence did not decode any information about the storing format:

   (a) **Changing the assumed data size**.

   (b) **Modifying** Ω'. If an ambiguity among different candidate sequences happened, modify Ω' in order to restrict the change to less bytes;

   (c) **Changing the data type**. Changing the data type might help the decoding from hex.

If none of the above cases apply, or the suggested changes did not lead to a correct data interpretation, we need to review the current γ_i goal in Stage 1. Each reached γ reduces the space of assumptions we are free to choose to build Λ (and Ω' as a consequence) for other γ.

5.2.8 Stage 7: Parser building

This Stage takes as input all collected knowledge about the given binary file format. The operator should be able to write a program that reads data from the logical dump of the smartphone and converts them in a XML format. It is mandatory to implement a quality monitor that measures the number of entries in which the parser encounters problems. The ratio $r = \frac{F}{T}$ between the number of failures (F) and the total number of entries (T) will be an indicator of the need to perform additional methodology's iterations. The threshold below which r is acceptable depends on the required accuracy.

5.2.9 Stage 8: Testing and debugging

In this phase the parser produced in the last Stage will be applied on several logical dumps, in order to test it and to debug it over real cases. In this Stage the r values of the current parser it will be verified and will be established if the implementation precision is sufficient or not.

5.3 Results

In order to verify and to refine the methodology's Stages, we took the Symbian S60 operating system as a case study. Applying the methodology produced the results we are going to show in this section.

File of interest - Stage 1 helped us to find a list of files containing SMS, MMS, contacts, and all user's personal data, which are shown in Table 5.1.

Symbian personal data files format - Thanks to the methodology we have been able to reverse engineer the Symbian S60 DBMS file format. We applied the methodology to the contacts list, to the calendar, to the text/multimedia messages and to the phone's event log (which contains calls, sent and

Case Study	Information	Detailed Information
Logdbu.dat	Event Log	SMS previews, MMSs, e-mails, calls, video calls, PRSConnection, SIM/MC change.
Calendar	Memo	Daynotes, meetings, anniversaries
Contacts.cdb	Contacts	Contacts information
Mail folder	SMS/MMS/Email	Sender, receiver and body

Table 5.1: Symbian files of interest

received SMS/MMS preview and SD card ad SIM changes). The complete format is explained in Appendix A.

Obsolete data - Among information identified and retrieved in the case study, we were able to find obsolete data which were not purged from the file system. The DBMS resources optimization strategy, in fact, reduces the high-cost of DB's modify/delete operations by flagging them as "obsolete": for these reasons the modify/delete operations are scheduled as late as possible, and the circumstance when they are performed varies depending the kind of file. For instance, in the Symbian case, in Contacts.cdb the deleting operations are performed when the `Compress()` syscall is invoked. Operating system tasks and third-party software as well can invoke this function, and they are able to know whether or not to perform compression by invoking `CompressRequired()` (see [42]). Let S the disk total space, F the free disk space, and W the amount of disk space wasted; the boolean function returns true if:

$$\begin{aligned}&(W > 64K) \vee (W > 16K \wedge W > \tfrac{1}{2S}) \vee \\ &(W > 16K \wedge F < \tfrac{1}{20S}) \vee \\ &(W > 16K \wedge F < 16K)\end{aligned}$$

After a compression is performed, the contacts are rearranged, the space wasted by obsolete records is recovered and there is no way to recover obsolete data. If the seizing operation occurs before the compression was invoked, we will find a database file that will contain all data since last compression. Often users are not aware that data they deleted are still stored in their smartphone, therefore the value of obsolete information recovered is valuable from a forensic perspective. For case studies related to the contacts, calendar and event log, enough information was decoded

in order to reconstruct the owner communication history. In the case study of messages (SMS, MMS and emails, stored in the `/System/Mail` folder) we were not able to find erased data, because OS purges immediately deleted messages to optimize the available storage.

Unexpected information - A part of data attributes are not controllable by the user, i.e., she can not insert them into the system explicitly, thus we were not conscious of their presence. During Stage 4, the nature of our methodology helped us to retrieve such "hidden" information, as the record's ID and its creation date. Such an important result enforces the methodology effectiveness, since it is able to detect more goals than the identified ones in Stage 0; from the forensic point of view, such a result completes the scene by adding more information useful to trace the user's activity. In our case study, some unexpected information helped us to better understand the data model, thus the application's behaviour.

We applied the methodology to more than 50 device dumps. At the beginning, the first dumps we studied came from Nokia N70 devices[1], but we realized that the knowledge we had about the S60 format was still incomplete since the parser was unable to decode an older phone's dump (Nokia 7610). After applying a few iterations of the methodology, we built a parser able to interpret the new format.

[1] Equipped with Symbian OS v8.1a, S60 Platform Second Edition, Feature Pack 3

Field Name	Size	Description	Example
ID	4	Int, Bigend	`B6 03 00 00`
NAME_LEN	1	Int, Littleend	`0E`
NAME	$(\frac{NAME\_LEN}{2})$	String	`43 6C 61 75 64`
...	...	...	...

(a)

```
B6 03 00 00
0E
43 6C 61 75 64 69 61
0A
44 72 61 67 6F
10
55 6E 69 72 6F 6D 61 32
09 13 00 10
```

(b)

```
ID
NAME_LENGHT
NAME
SURNAME_LENGHT
SURNAME
COMPANY_NAME_LENGHT
COMPANY_NAME
CXF1
```

(c)

Figure 5.4: This three figures depict an example of the application of Stage 5 on a file containing the phone's address book. *(a)* A table with pseudo data type, got as output by Stage 4. *(b)* The meta-format file before Stage 5. *(c)* The meta-format file after Stage 5.

6 The MIP process

The main results of this chapter have been presented in [43]

6.1 Introduction

In this chapter we describe the *Mobile Identity Profiling* (MIP) process, which aims at reconstructing a user's profile by combining the smartphone's data analysis with social relationships data found on the Web. Such a process is split into three stages: the *Smartphone Data Analysis*, the *Web Data Analysis* and the *Clustering Analysis*. The goal of the process is to build a smartphone owner's social network, namely the *profile graph*, and to find all sub-graphs (*clusters*) which represent the social groups within the graph.

6.2 Smartphone Data Analysis

The decoding phase aims at generating parsers which can export data in xml-format, or that can be integrated directly in the analysis application (in order to simplify the workflow). Data decoded as contacts list, sms list, event log and calendar entries list can be hard to analyse manually by a human operator, because she has to correlate their unique identifier, in order to reconstruct situations or conversations (or more precisely, the importance of relationships) between a device's owner and their contacts.

The *Smartphone Data Analysis* performs such tasks automatically; it is composed of four sub-phases[1]: *File Analysis*, *Contact Analysis*, *Event Analysis* and

[1] In this work we do not deal with the calendar analysis.

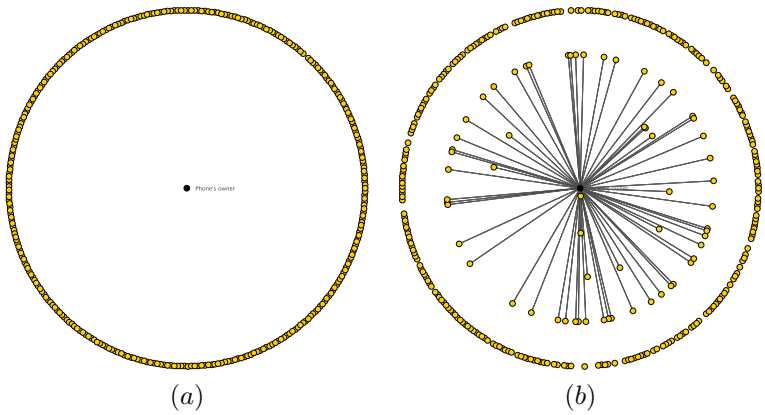


Figure 6.1: The graph representation of contacts (a) and their relationships with the phone's owner (b), which are revealed by the number of calls and number of sms/mms.

Messages Analysis.

The **File Analysis** phase will load in the framework each file contained in the logical dump, and will organize them by their MIME-Type and run the decoding tool over personal data files.

The **Contact Analysis** phase will merge together duplicate contacts information, highlighting those which provide incomplete data and which may represent potential source of noise for the next Web analysis.

The **Event Analysis** will mine the phone's log in order to reconstruct the user's activity. Although the event type may depends on the device type, events always belong to the following macro-classes: voice calls, data calls, sms/mms sent or received, SIM change, SD change. Voice calls and sms/mms logs are useful for the reconstruction of the phone owner's social activity, in fact they are used to determine the strength of a bond between the owner and each contact.

The **Messages Analysis** completes the event analysis by extending it to all sms/mms that have been deleted from event log but could still persist in the saved sms/mms list.

After these analysis sub-phases have been completed, the *profile graph* G is built and the information collected is organized and stored inside it. Such data structure will give us the chance to represent the social network given by:

Algorithm 1: Communication links building algorithm

```
Input    : A set of contacts C, the owner o, the graph G
foreach element c of C do
    if one or more calls exist between c and o then
        x ← number of calls (sent/received) between c and o
    if one or more sms/mms exist between c and o then
        y ← is the number of sms/mms (sent/received) between c and o
    w ← x + y
    if w > 0 ∧ E = {c, o} ∉ G then create E
    w(E) ← w ;                          /* Assigns w as weight of E */
max ← Max_W(G) ;          /* Maximum weight among graph's edges */
foreach element E of G do
    r(E) ← 1 − w(E)/max ;          /* The link's lenght (or radius) */
```

- phone interaction between the owner and the contacts,
- web relationships between the owner and the contacts, and among them.

Let $G = (V, E)$ an *undirected*[2] graph, comprising a set $V = \{1, \ldots, n\}$ of vertexes or nodes together with a set E of edges or lines, which are 2-element subsets of V. Each edge has a value $w \in [0 \ldots 1]$ called "weight". In our case, the vertex V represent the contacts found on the phone and stores all its data, while an edge $E = \{u, v\}$ stores the phone interactions (*mobile-edge*) and/or web relationships (*web-edge*) between the contacts u and v. The weight of E indicates the importance of the link. We used a graph representation which puts the phone's owner in the center of a circle, where its contacts are drawn along the circumference (Figure 6.1a). After the smartphone data analysis, the graph is enriched with edges from the phone's owner and the contacts we are able to trace a link between the owner and all contacts with whom she has communicated. The weight of these links is computed trivially as the sum between the number of calls (sent or received) and sms/mms (send and received) between the owner and the contacts. Such weight is mainly used to compute the proximity between the vertex (the edge length or radius), that will be used to represent graphically these relationships in the graph by bringing the most frequently contacted people near to the owner (Figure 6.1b). The algorithm

[2] A graph in which edges have no orientation, i.e., they are not ordered pairs, but sets $\{u, v\}$ (or 2-multisets) of vertexes.

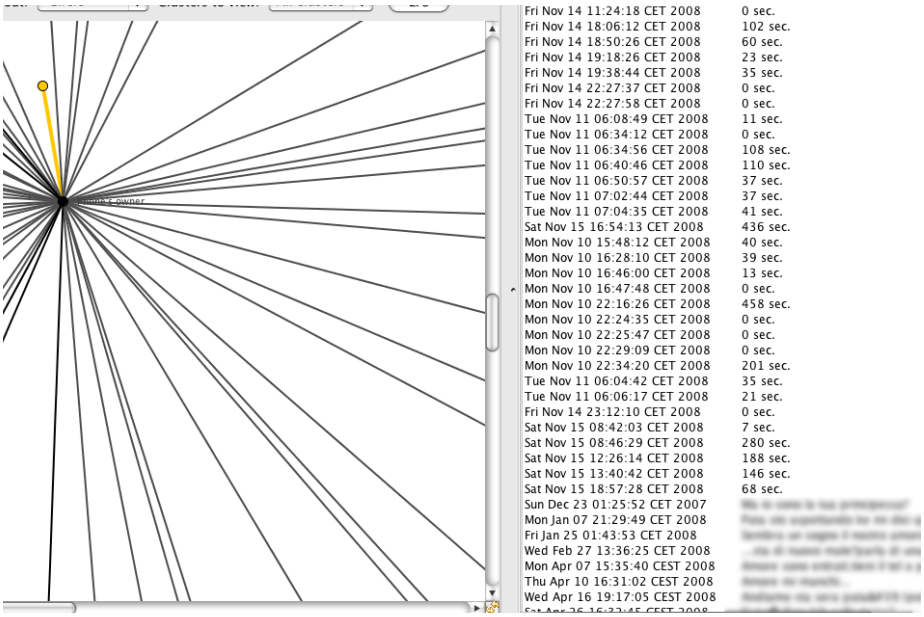
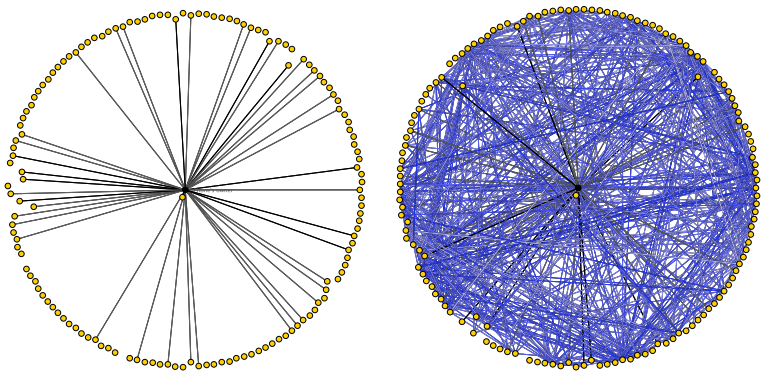


Figure 6.2: After the smartphone analysis, the details about the calls and sms/mms are accessible simply by clicking on the edge between the owner and the desired contact.

which traces those links and computes their attributes is shown in Algorithm 1. The framework also provide facilities which allow the operator to simply click on an edge and retrieve the list of calls and sms/mms between the ends. An example of this functionality is shown in Figure 6.2.

6.3 Web search analysis

The goal of the *Web search analysis* component is to find the social network between the phone owner and her contacts, and among them, by retrieving people's public information on the World Wide Web. As mentioned before, we follow the approach of Mika et al. [21] to retrieve relationships from search engine records. In this section we will describe in greater detail the relationships retrieving algorithm and the techniques used to estimate the web-edges



(a) (b)

Figure 6.3: The graph representation of contacts before (a) and after (b) the SESORR execution. Black links represents relationships extracted from the mobile phone (mobile-edges). Blue links represents the relationships extracted from the Web (web-edges).

weight.

6.3.1 SESORR Algorithm

In order to reconstruct relationships among a phone's contacts, we have used the huge amount of data collected by search engines over the years to obtain relational network data. Our approach is to submit all possible pairs of names and surnames to the search engine and to retrieve the results, i.e., the pages where the two pairs $\langle name, surname \rangle_{i,j}$ co-occur, by counting the number of pages found (hits) and, for each of them, by saving the title and the short description returned by the search engine. Moreover, it counts non-stop words contained in titles and description for further analysis. Serving this purpose, we designed the *SESORR* (Search Engine SOcial Relationships Retrieving) algorithm; the pseudocode is shown in Algorithm 2. As preliminary examination, SESORR submits the query

$$\langle name, surname \rangle \vee \langle surname, name \rangle$$

for each contact and stores the results in the G nodes data structures. In such way it is able to discard from subsequent queries the contacts which are not present on the Web (i.e., the query returned a resultset $R = \emptyset$). Finally, for each

pair of contacts i, j, SESORR submits the following query:

$$(\langle name_i, surname_i \rangle \vee \langle surname_i, name_i \rangle) \\ \wedge \\ (\langle name_j, surname_j \rangle \vee \langle surname_j, name_j \rangle)$$

and stores results. Name and surname pairs are sent to the search engine by enclosing them within quotation marks: in such way the search engine is forced to retrieve only pages which contain the adjacency of the search terms. The piece of software which implements SESORR is able to contact both Google and Yahoo. After the SESORR execution, the profile graph is enriched by web-

Algorithm 2: SESORR Algorithm

Input : A profile graph G, a search engine S
Output : The profile graph G, enriched by the web-edges

```
/* Phase 1:  Contacts web search                                  */
foreach c ∈ V(G) do
    send c name and surname to S
    retrieve the resultset R_c as list of ⟨title, description, url⟩
    store R_c in c
    remove stop-words from titles and descriptions
    analyze word frequencies
/* Phase 2:  Pair of contacts web search                          */
foreach a ∈ V(G) do
    foreach b ∈ V(G) do
        if a ≠ b and R_a ≠ ∅ and R_b ≠ ∅ then
            send a and b name and surname to S
            retrieve the resultset R_{a,b} as list of ⟨title, description, url⟩
            store R_{a,b} in E(a, b) ∈ E(G)
            remove stop-words from titles and descriptions
            analyze word frequencies
```

edges between owner and their contacts, and among contacts. An example is reported in Figure 6.3.

For each web-edge, SESORR merges the titles and the descriptions of each resultset entry in a single string. It firstly purges the symbols and the stop words from the string, then it computes the occurring frequency of each remaining word. Such words are called *keyword* and the keyword-frequency lists are stored in the web-edge data structures and are displayed to the user when

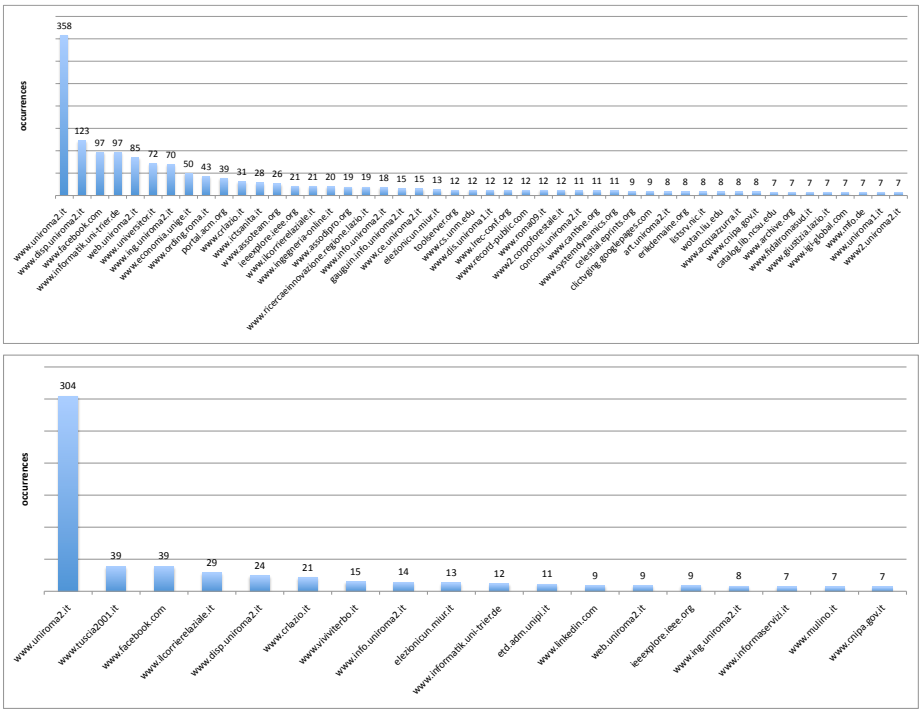


Figure 6.4: Frequency distribution of URLs (domains) providing relationships. Each graph refers to a distinct profile.

she clicks on the respective graph edges. Given a web-edge, the list of keywords and their frequencies provides a kind of "semantic vision" of the relationship and the user is able to figure out a meaning of the relationship at glance.

Moreover, besides title and description, SESORR stores each URL in the result set. By calculating the frequency with which each URL occurs over all relationships on a single profile, SESORR also provides a distribution of frequency of domains related to the profile and its contacts (see Figure 6.4).

6.3.2 Web-edge weight estimation

In order to measure a Web-edge weight, i.e., how similar are two contacts between which an Web-edge exists, we define a function $\sigma(e) \in [0, 1]$ which measures the similarity between u and v individuals. In the semantic Web area, the similarity between two classes is assessed by observing the number of instances that these classes share, their individual number of instances, and the total number of instances they contain. The most frequently used metrics are the following:

Jaccard index between two sets X and Y is defined as the ratio between the size of the intersection and the size of the union of the two sets being compared:

$$\sigma(X, Y) = \frac{|X \cap Y|}{|X \cup Y|}$$

where $|.|$ denotes the size, i.e., the number of elements in the set. $\sigma(X, Y) = 0$ when the sets are disjoint; $\sigma(X, Y) = 1$ when the sets are identical. This is very intuitive measure for the given setting as it measures the relative overlap of the two sets. There is also a probabilistic interpretation, namely the probability that a random instance from the union is in the intersection of the two sets [44]. Independently the **Dice coefficient** has been proposed, which is equal to the Jaccard index multiplied twice:

$$s = 2\sigma(X, Y) = \frac{2|X \cap Y|}{|X \cup Y|}$$

For further information see [45].

Normalized Google Distance (NGD) was introduced by Cilibrasi et al. [46] and it takes advantage of the number of hits returned by Google to compute the semantic distance between concepts. The concepts are represented with their labels which are fed into the Google search engine as search terms. Given two search terms x and y, the the normalised Google distance between x and y, $NGD(x, y)$, can be obtained as follows:

$$NDG(x, y) = \frac{\max\{\log f(x), \log f(y)\} - \log f(x, y)}{\log M - \min\{\log f(x), \log f(y)\}}$$

where $f(x)$ is the number of Google hits for the search term x, $f(y)$ is the number of Google hits for the search term y, $f(x, y)$ is the number of Google hits for the tuple of search terms xy, and M is the number of web pages indexed by Google, i.e., approximately ten billion pages.

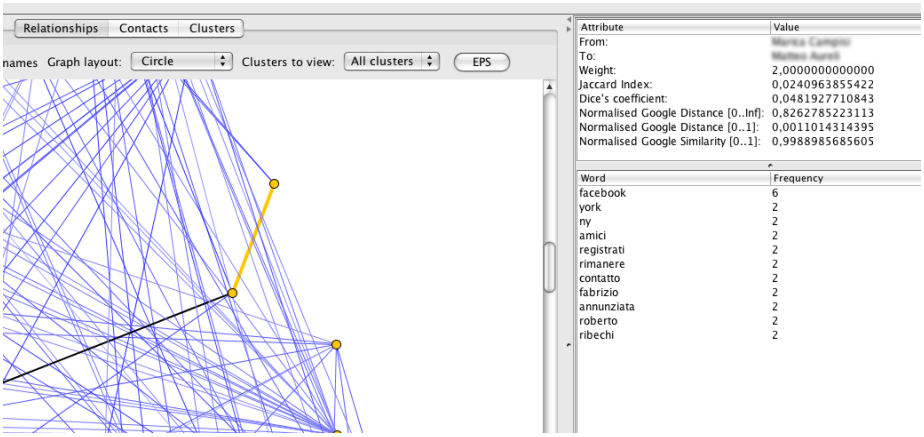


Figure 6.5: When user clicks on an Web link (left-side), the framework shows information about indexes (right-side, top) and word frequencies found in the Web search result set.

In our case, we want to compute the Jaccard similarity ($\sigma_J(e)$) to an edge $e = \langle u, v \rangle$: let $X = W_u$ the list of Web results of individual u, $Y = W_v$ the list of Web results of individual v, $X \cap Y = W_e$ the list of Web results of edge e. The Jaccard similarity can be obtained as following:

$$\sigma_J(e) = \frac{|W_e|}{|W_u| + |W_v|}$$

If we compute $\sigma_J(e)$ only if $|W_e| > 0$, $|W_u| > 0$ and $|W_v| > 0$ because $|W_e| = |W_u \cap W_v|$, thus $\sigma_J(e)$ will have not singularities. Finally we observe that $0 \leq \sigma_J(e) \leq \frac{1}{2}$. Thus, in order to be used as similarity weight, the Jaccard index has to be *normalized*. The Dice's similarity (σ_D) was introduced for such purpose:

$$\sigma_D(e) = 2\sigma_J(e) = \frac{2|W_e|}{|W_u| + |W_v|}$$

To compute the Google similarity ($\sigma_G(e)$), we compute the normalized google distances for each edge $e = \langle u, v \rangle$, assigning $f(x) = |W_u|$, $f(y) = |W_v|$ and $f(x, y) = |W_e|$:

$$\sigma_G(e) = \frac{\max\{\log|W_u|, \log|W_v|\} - \log|W_e|}{\log(10^6) - \min\{\log|W_u|, \log|W_v|\}}$$

In our experiments, Pearson's correlation between the Jaccard index and the NGD shows that their linear relationship is equal to 0.4. For this reason, and in order to have a more flexible tool which is able to adapt itself to the user experience, our software give the faculty to choose which of the above three indexes to use to measure the weight of the web-edge.

6.4 Clustering

As the final analysis stage, we want to identify subgroups of contacts sharing similarities. Generally speaking, the goal of clustering is to group together similar elements and thereby to identify the skeleton structure of the input data.

In this work we have employed clustering techniques to split the phone owner's social graph into small subgraphs (clusters). We chose *spectral* algorithms because i) they are general and versatile, and ii) they proved to perform effectively in the identification of locally dense subgraphs that are sparsely inter-connected, also known as the paradigm of intra-cluster density versus inter-cluster sparsity (see [47]). For each cluster, we expect to find nodes which share strong similarities among them. Spectral methods have been frequently adapted, for examples see [48, 49, 50, 51, 52, 53]. While most of these spectral approaches have been deeply analyzed from a theoretical point of view, to the best of our knowledge only few methods that involve eigenvalue decomposition (of the adjacency or related matrices) have been experimentally studied and evaluated. An algorithm that embeds the graph using eigenvectors of the normalized adjacency matrix has been tested as an example in [48]. We will see that the tested algorithms performed effectively; however, the lack of a large amount of available phone data prevented us from performing a broader assesment of clustering algorithms.

6.4.1 Notation

Graphs and matrices

Let $A = (a_{i,j})_{1 \leq i \leq n; 1 \leq j \leq n} \in R^{n \times n}$ be an $n \times n$ square matrix over the real numbers; its i-th row (j-th column) is denoted by $A_{(i)}$ ($A^{(j)}$). We can represent an n-node graph by an n-by-n matrix. The weighted adjacency matrix $A_w(G)$

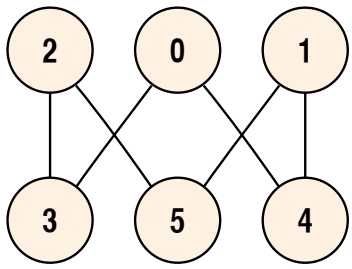


Figure 6.6: Sample undirected graph.

is a square matrix of order $|V|$ and its entries are defined by:

$$[A_w(G)]_{u,v} := \begin{cases} w & \text{if } (u,v) \in E \\ 0 & \text{otherwise} \end{cases}$$

A matrix A is called *symmetric* if $A^T = A$. If the graph G is undirected, $A_w(G)$ is symmetric. For instance, the undirected graph shown in figure 6.6 is described by the following symmetric matrix:

$$A = \begin{pmatrix} 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 \end{pmatrix}$$

Eigenvalues and eigenvectors

Eigenvalues are often introduced in the context of linear algebra or matrix theory. Given a linear transformation, an *eigenvector* of that linear transformation is a nonzero vector which, when that transformation is applied to it, may change in length, but not direction. For each eigenvector of a linear transformation, there is a corresponding scalar value called an *eigenvalue* for that vector, which determines the amount the eigenvector is scaled under the linear transformation. In the real-world applications, eigenvalues and eigenvectors appear

as preferential axes of rotation of a rigid body, resonance frequencies, main stress directions, and so on. The theory concerning eigenvalues and eigenvectors is also called *spectral theory*. Let A a square $n \times n$ matrix. A column vector x_r is an *right-eigenvector* if:

$$A \cdot x_r = \lambda x_r$$

while a *left-eigenvector* is a row-vector x_l that is:

$$x_l \cdot A = x_l \lambda.$$

We call λ an eigenvalue of A. The eigenvalues of a graph are defined as the eigenvalues of its adjacency matrix. The set of eigenvalues of a graph is called a graph spectrum [54].

Singular Value Decomposition

A *singular value decomposition* (SVD) of A is given by:

$$A = U \cdot S \cdot V$$

where:

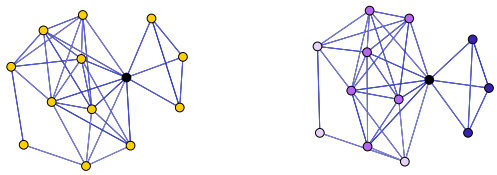
U is an $m \times m$ orthogonal matrix, whose column vectors $[u_1 \dots u_m]$ are called the *left singular vectors*, they are eigenvectors of $A \cdot A^T$, and therefore are orthonormal[3];

V is an $n \times n$ orthogonal matrix, whose column vectors $[v_1 \dots v_n]$ are called the *right singular vectors*, they are the eigenvectors of $A^T \cdot A$, and therefore are orthonormal too;

S is an $m \times n$ diagonal matrix with nonnegative elements on the diagonal (and with the same rank as A).

Such a decomposition exists for each matrix A. Conventionally, the singular values which are the nonzero diagonal elements of S are ordered in non-increasing fashion. For further information see [55]. Please note that S is uniquely determined, and the same does not hold for the matrices U and V.

[3] A square matrix M is orthonormal if it is regular ($det(M) = 0$) and if $M^T = M^{-1}$; for an orthonormal matrix M it holds $M^{-1} \cdot M = M^T \cdot M = I$. Also $det(M) = \pm 1$



(a) (b)

Figure 6.7: A small example network. (a) Before the clustering. (b) After clustering through Spectral ($k = 3$). The black node is the smartphone owner.

6.4.2 Clustering algorithms

In this section, we briefly describe the spectral-based clustering approaches we will benchamark in order to choose the best one for our purposes. They are, namely: Spectral and Full SVD. They are standard and frequently used methods for general data clustering. All algorithms have in common that the SVD is performed using the adjacency matrix of the graph and the clusters are assigned based on the singular vectors. We selected these methods for our benchmark mainly for two reasons: first, all of them are simple and easy to implement, and second, these techniques form the basis of more complex methods, e.g., [50] that presented a variation of Fast SVD using *adaptive sampling*.

Generally speaking, SVD is a dimensionality-reduction technique under the constraint of maintaining contained variance. Furthermore, SVD is closely related to the k-means-clustering problem for an extensive discussion on this refer to [51, 56, 57, 58].

Spectral

Algorithm 3: Spectral Algorithm

Input : An $m \times n$ matrix A, an integer k.
Output : An partition of the m rows (i.e., nodes) of the matrix A into k clusters.

compute the SVD of matrix A
find the top k *right* singular vectors $v_1, \ldots, v_k$
let C be the matrix whose j-th column is given by Av_j
place row (node) i in cluster j if C_{ij} is the largest entry in the i-th row of C

The *Spectral* algorithm is essentially a projection onto the first k right singular vectors. Its pseudocode is given in Algorithm 3. In [59, 52] it was called Spectral Algorithm I and served Kannan et al. for their study of spectral graph clustering. The intuition of this technique is that the matrix A describes the location of m points in an n-dimensional space. The projection onto the subspace defined by the top k right singular vectors gives the best k-rank approximation of A. In the context of graph clustering the input matrix A is just the adjacency matrix of the considered graph. The different dimensions in this space are then interpreted as clusters and the assignment of nodes is done by choosing that dimension/cluster that has the largest corresponding entry.

Full SVD

Drineas et al. studied in [51] k-means and its continuous version. While the discrete version is known to be $\mathcal{NP}$-hard, the latter can be solved efficiently using a projection onto the top k left singular values. Similar to the Spectral algorithm, the cluster assignment is a discretization of the continuous solution. We refer to this method as *Full SVD* in order to avoid ambiguities with the SVD computation which is the core of all these algorithms. Its pseudocode is given in Algorithm 4. The clustering algorithms (both the Spectral and the Full SVD)

Algorithm 4: Full SVD

Input : An $m \times n$ matrix A, an integer k.
Output : A partition of the m rows (i.e., nodes) of the matrix A into k clusters.

compute the SVD of matrix A
find the top k *left* singular vectors $u_1, \ldots, u_k$
let C be the matrix whose jth column is given by u_j, i.e., the first k columns of U
place row i in cluster j if C_{ij} is the largest entry in the i-th row of C

output a matrix C which has on the rows the nodes (contacts) indexes, and on the columns the clusters indexes. The matrix cells represent intuitively the weight of how a contact belongs to a cluster. We choose to assign a node to the cluster with the maximum absolute value. In Figure 6.8 is shown a screenshot with the C matrix details and, for each contact, the chosen cluster.

6.4.3 Clustering quality indices

In order to give an evaluation of obtained clustering, we followed the approach of Brandes et al. [60, 48], who used a set of quality indices that measured the

Clusters: 6

Contact	Cluster 0	Cluster 1	Cluster 2	Cluster 3	Cluster 4	Cluster 5
[illegible]	1.97113E-17	-0.401179	-8.25476E-17	-0.551163	9.41226E-16	-1.62784E-16
[illegible]	0.236249	2.00926E-15	-0.16099	6.62031E-13	0.587447	-0.267262
[illegible]	0.462952	3.84945E-15	-0.372254	1.29902E-12	0.0896404	0.133775
[illegible]	0.336419	2.74591E-15	-0.105736	9.43641E-13	0.298778	0.543252
[illegible]	0.32444	2.48783E-15	0.13332	9.10858E-13	-0.378125	0.338583
[illegible]	0.097132	9.14943E-16	-0.207542	2.72095E-13	0.346742	-0.0160145
[illegible]	0.340234	2.68625E-15	0.838445	9.53527E-13	0.282402	-0.178824
[illegible]	0.293121	2.33637E-15	-0.134461	8.22844E-13	-0.220197	-0.343508
[illegible]	0.364713	2.93871E-15	-0.178423	1.02384E-12	-0.265351	-0.561804
[illegible]	0.409015	3.25145E-15	0.102843	1.14806E-12	-0.309427	0.187237
[illegible]	2.21585E-17	-0.419997	-9.06043E-17	-0.599129	1.02831E-15	-2.27765E-16
[illegible]	3.09952E-17	-0.814038	1.74454E-16	0.580744	4.78592E-15	-7.39229E-16

Figure 6.8: Contact-to-cluster assignment.

density of the clusters and the sparsity of the connections between the clusters. Their collection consisted of *coverage*, *performance*, *intra-* and *inter-cluster conductance*. The problem of finding (non-trivial) graph clusterings optimizing any one of these measures is $\mathcal{NP}$-hard [48]. Given a clustering $\mathcal{C}$, in the following we denote with $m(\mathcal{C})$ the number of intra-cluster edges, and with $m(\bar{\mathcal{C}})$ the number of inter-cluster edges; thus $m = w(\mathcal{C}) + w(\bar{\mathcal{C}})$. In analogy, the weight of intra-cluster edges is $w(\mathcal{C})$, and the weight of inter-cluster edges is $w(\bar{\mathcal{C}})$; obviously $w(E) = w(\mathcal{C}) + w(\bar{\mathcal{C}})$.

Coverage

The *coverage* of a clustering $\mathcal{C}$ is defined as the ratio of the sum of the weights of the edges contained inside clusters and the total weight of all edges, i.e., $w(E)$. Intuitively, the larger the value of coverage the better the quality of the clustering. Note that coverage is non-decreasing when merging two clusters as the weight of edges inside clusters can only increase but never decrease. Thus it favors coarse clusterings and has its (trivial) maximum of 1 for the 1-clustering. Coverage was also used in [59, 52].

$$coverage(\mathcal{C}) := \frac{w(\mathcal{C})}{w(E)} = \frac{w(\mathcal{C})}{w(\mathcal{C}) + w(\bar{\mathcal{C}})}$$

Performance

The idea of *performance* is to measure "correctly interpreted pairs of nodes" in a graph. A pair of nodes is correctly interpreted by a clustering if both nodes

belong to the same cluster and are adjacent or if they are not adjacent and belong to different clusters. Analogously to coverage, performance counts the number of correctly interpreted pairs and is then normalized by the total number of node pairs. It was originally introduced in [61] and is closely related to the editing distance of a clustering, i. e., the minimum number of edges that have to be deleted or inserted in order to transform the graph into a set of disjoint complete subgraphs. Although it is similar to coverage, it additionally evaluates the sparsity between clusters and can thus favor fairly different clusterings than coverage. Generally speaking, for sparse graphs performance often prefers fine clusterings.

$$performance(\mathcal{C}) := \frac{w(\mathcal{C}) + \sum_{\{v,w\} \notin E, v \in C_i, w \in C_j} 1}{\frac{n(n-1)}{2}}$$

Intra- and Intercluster Conductance

The *conductance* measures the quality of a cut and the weight of edges in either of the two induced subgraphs [60, 48]. The conductance $\varphi(G)$ of a graph G is then the minimum conductance value over all cuts of G. For a clustering $C = (C_1, \ldots, C_k)$ of a graph G, the intracluster conductance $\alpha(\mathcal{C})$ is the minimum conductance value over all induced subgraphs $G[Ci]$, while the intercluster conductance $\delta(\mathcal{C})$ is the maximum conductance value over all induced cuts $(C_i, V \backslash C_i)$. For a formal definition of the different notions of conductance, let us first consider a cut $C = (C, V \backslash C)$ of G and define *conductance* $\varphi(C)$ and $\varphi(G)$ as follows:

$$a(C) := \sum_{v \in C} \sum_{cw \in V \{v,w\} \in E} \omega(\{v,w\}) = 2 \sum_{e \in E(C)} \omega(e) + \sum_{f \in E(C, V \backslash C)} \omega(f)$$

$$\begin{aligned} \varphi(C) &:= \begin{cases} 1, & C \in \{\emptyset, V\} \\ 0, & C \notin \{\emptyset, V\} \text{ and } \overline{\omega}(\mathcal{C}) = 0 \\ \frac{\overline{\omega}(\mathcal{C})}{\min(a(C), a(V \backslash C))}, & \text{otherwise} \end{cases} \\ \varphi(G) &:= \min_{CV} \varphi(C) \end{aligned}$$

Based on the notion of conductance, we can now define intra- $\alpha(\mathcal{C})$ and intercluster conductance $\delta(\mathcal{C})$.

$$\begin{aligned}
\alpha(\mathcal{C}) &:= \min_{i \in \{1,\ldots,k\}} \varphi(G[C_i]) \\
\delta(\mathcal{C}) &:= \begin{cases} 1, & \text{if } C = \{V\} \\ 1 - \max_{i \in \{1,\ldots,k\}} \varphi(C_i), & \text{otherwise} \end{cases}
\end{aligned}$$

6.5 Results

All spectral clustering algorithms we proposed in this chapter take as input the k parameter, i.e., the number of clusters. Although an operator can feel free to specify any $0 \leq k \leq n$, where n is the number of nodes in G, the clustering indexes are heavily affected by the the number of clusters, as reported in Figure 6.9. This Figure shows, for each clustering algorithm and for each web-edge weight metric, the performance of clustering quality indexes to the variation of k parameter. The estimation of best-k is an open research topic. A rule of thumb is given by Mardia et al. in [62], in which the authors suggest that an approximation of best-k is:

$$k' = \sqrt{\frac{n}{2}}$$

with n the number of vertices.

Moreover, as shown in Figure 6.10, both Spectral and SVD algorithms produce a certain percentage of empty clusters if $k > k'$. In Figure 6.11 is shown a graph with 218 contacts and 1242 web-edges. This graph has been clustered via SVD clustering, with GND as web-edge weight metric and with $k = k'$. The figure also shows some clusters sampled from the the graph. By looking at each cluster's web-edge keywords, we have been able to gather the area of interest shared by individuals in the cluster.

In our specific case, we believe that there is no exact value of best-k that could be estimated *a priori*, and that this parameter changes case-by-case. We give the investigator all the instruments that can help her understand which are reasonable values of k, and to develop a common sense to assess which are the best values of clustering algorithms for each case.

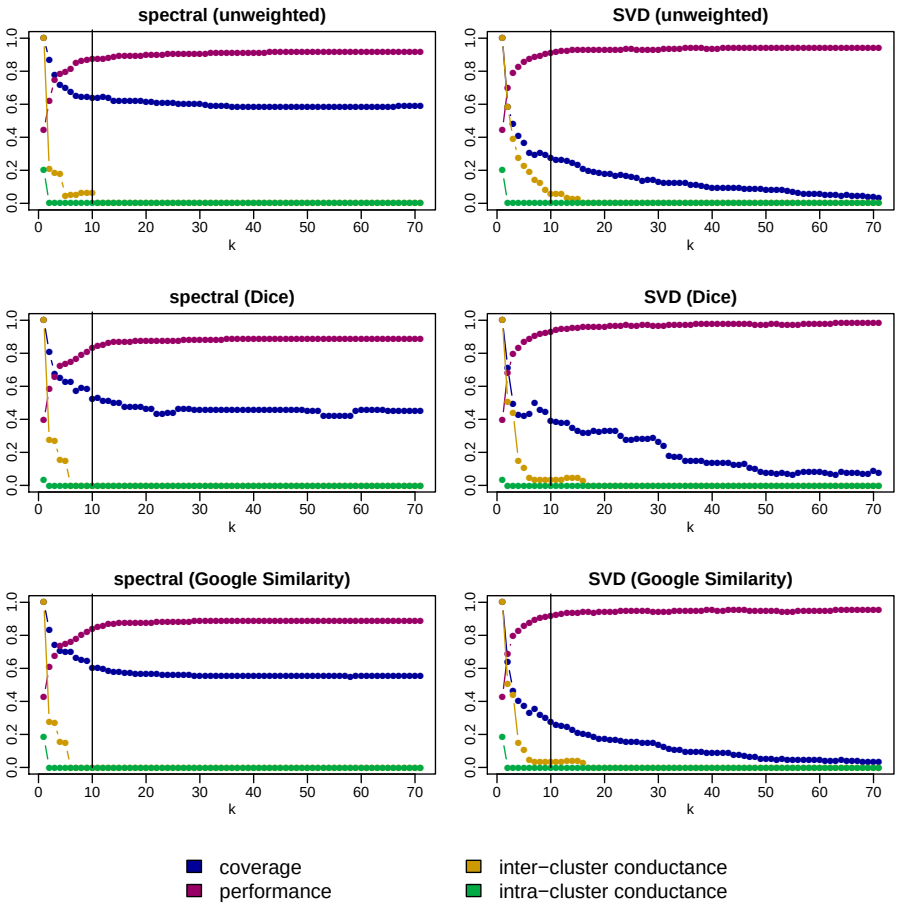


Figure 6.9: Clustering metrics trends. The profile graph used in this example has 218 contacts and 1242 web-edges. $k' = 10$ and its value is shown by the black vertical line.

6.6 Limitations of our method

Our profiling method relies on information stored in the smartphone and its precision depends by the quantity and quality of such data.

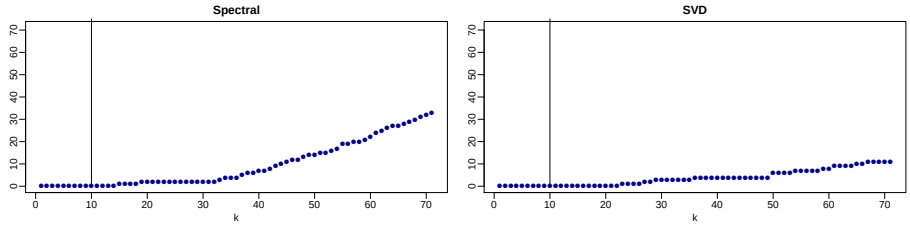


Figure 6.10: Spectral and SVD number of empty clusters to the variation of k parameter. The profile graph used in this example has 218 contacts and 1242 web-edges. $k' = 10$ and its value is shown by the black vertical line.

Since the method we used to find a person on the Web, and her relationships with other phone contacts, relies on her first name and last name, precision is strongly dependent on the care used by the owner when she inserted each first name and last name. Sometimes only the names or the nicknames of a contact are inserted (e.g., most intimate contact); after submitting such weak identity to a search engine, this will produce no or useless results. To deal with this aspect, the framework performs a pre-processing of all contact entries (e.g., hightlights entries which have name or surname missing) and suggests the operator identifies the contacts (where possible) and enters their correct names and surnames.

An obvious limitation deals with the time frame that we can reconstruct. The event log stored in the device is limited to a fixed size which restricts the vision of user activity to the latest. Also the size of stored sent and received sms/mms, if set, will limit precision.

Another obvious limitation of our method is that it relies on the page and bag-word counts returned by querying the search engines index with name and surname of two contacts; we do not download the pages themselves for further processing, because of the large amount of time involved in this process. We are unable to apply any automatic disambiguation technique in order to reduce cases of homonyms. On the other hand, the bag-word frequency related to each edge of the graph is a valid help for the human operator to further analyze and to validate such as relationship.

Moreover, our method ignores the fact that a contact's name can occur more than once on a given Web page. This approximation is justified, as Zhu et al.

demonstrated in [63]: page counts and name counts are highly correlated on a log-log scale.

Another source of noise is the fact that search engines[4] will sometimes return pages that do not include the search term at all. This can happen if the search term is contained in a link to the page (but not on the page itself).

[4]We noticed that this phenomenon happens on Google and Yahoo, but not Altavista[64]

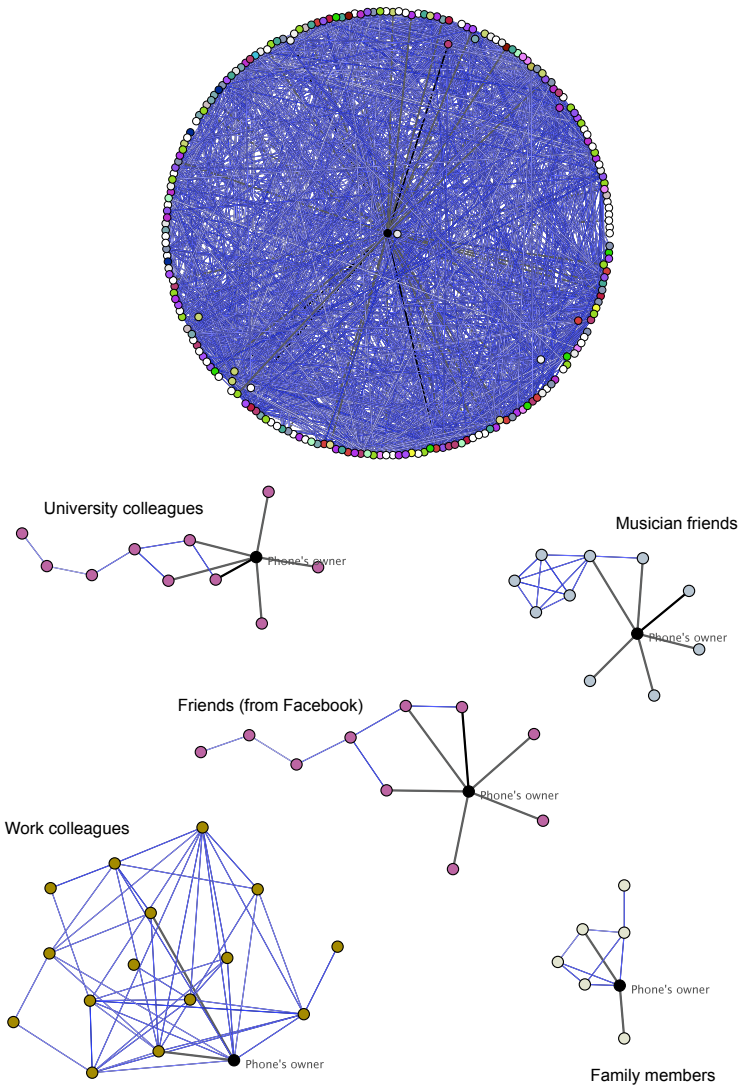


Figure 6.11: An example of clustering after applying the SVD method. At the top, an overview of the entire graph. Each cluster has a colour associated, then each node is coloured according with the cluster colour which it belongs to. At the bottom, some cluster subgraphs extracted from the main graph.

7
Conclusions and Future Work

In this work we have presented a complete workflow which allows the investigator to forensically extract the internal memory data contained in the smartphones, to decode them, and to analyze such information in order to reconstruct the importance of relationships between the owner and its contacts, and, thanks to the data publicly available on the Web, to reconstruct the intra-contacts relationships. The resulting graph, namely the Identity Profile, is further analyzed with spectral clustering techniques coming from the graph theory, and several groups of contacts sharing similar interest are found.

The MIP process illustrated in Chapter 6 analyzes a single device at a time. In order to have a more comprehensive profiling, we plan to extend this analysis to more devices coming from different owners linked to the same case. In this way, correlations between different personal datasets may come to light and this may open a more complete social network graph. Another benefit of this approach relies on the fact that if a contact belongs to two or more personal datasets, its name and surname give a greater chance of disambiguation and the limitations due to manual names refining can be mitigated.

The framework idea takes place in a wider concept, the remote *Forensic Farm*. In such a scenario, when the device acquisition is completed, the seized image is forwarded to the Forensic Farm, where all the data are collected and analysed. The Forensic Farm is a laboratory specialised in *Data Decoding* and *Data Analysis* of the mobile equipment's internal memory copies. When a new internal memory copy is received from the forensic operator's device, the Forensic Farm's software starts the framework and the data decoding operation begins; the output of this operation consists of all the information residing on the smartphone's internal memory copy but in an standard format, easy to be consulted, linked by the forensic investigator. As this phase fin-

ishes, the analysis phase starts. In this step the software extracts *standard* correlations among all the information extracted in the data decoding stage (e.g., contacts sending SMSes, with related content and details, incoming and outcoming calls, calendar events related), reconstructs the relationships among contacts through the Web analysis and identifies the clusters. After the smartphone's file system has been analysed the results are sent back to the operator, who receives via 3G/4G the results of the Forensic Farm's work on her mobile equipment and she can decide what to do directly at the crime scene.

A

The Symbian S60 format

A.1 Address book

Type flag	Meaning	Type flag	Meaning
04024008	fax	14020001	H_fax
2402C003	job	04140280	home
24028004	mobile(work)	1C02C00C	nickname
0402000D	video	1402400D	video (home)
0C02C00D	wv user ID	0402C008	url
04024009	po.box	04028009	extension
0402400A	city	0402800A	state
14024002	extension (home)	14028002	street (home)
14024003	state (home)	14028003	country (home)
24024006	street (work)	24028006	postal code (work)
24024007	country (work)	3C02000B	DTMF
Type flag	**Meaning**	**Type flag**	**Meaning**
2402C004	W_fax	04028007	general
0402C007	mobile	1402C000	mobile(home)
04028008	?	04020008	pager
2402800D	video (work)	24020004	work
14028001	url(home)	24024005	url(work)
0402C009	street	0402000A	postal code
0402C00A	country	14020002	po.box (home)
1402C002	postal code (home)	14020003	city (home)
2402C005	po.box (work)	24020006	extension (work)
2402C006	city (work)	24020007	state (work)
3402800B	note		

Table A.1: Possible values for the rows of table "DATA_TYPE_TABLE". They describe the type of attributes present in the "DATA_BLOCK". (Symbian S60 v2)

Contacts and their data are stored in the `Contacts.cdb` file (located under `C:\System\Data`). During the methodology iterations, we found that

contacts data were fragmented and spread across the entire file. In fact, after a contact update, Symbian preserves the old contact entry and appends new one at the end of the file with the same ID but fresh data. When the system performs a DB compression, obsolete entries are purged. After a first analysis, we found that data could grouped in three macro-areas (*parts*, see Table A.2).

For each contact, the three parts are connected because each of them shares the same contact ID. The first part stores metadata about each contact and a block containing attributes like phone/fax/mobile numbers, snail mail address and notes:

```
D2 64
10 00 00 00
FF 09 13 00 10 FF FF FF FF
20 30 30 65 31 32
30 30 66 66 61 39
64 31 32 37 36
76 12 9D FA 0F 20 E1 00
76 12 9D FA 0F 20 E1 00
04 00 00 00 00 00
00 00 1F
1D
04 00 00 00 04
14 02 00 00 00 00 00 00 00 00 00 00
04 02 C0 07 00 00 00 00 00 00 00 00
04 00 00 00 00
1A
20 00
33 33 38 38 37 36 35 34 32 33
```

```
VD
ID
CXF1
20 30 30 65 31 32
30 30 66 66 61 39
64 31 32 37 36
EDIT_DATE
CREATION_DATE
04 00 00 00 00 00
00 00 1F
TYPE_TABLE_LEN
      |
TYPE_TABLE
      |
04 00 00 00 00
DATA_BLOCK_LEN
FIELD_FLAG
DATA_BLOCK
```

The second part stores contact's name, surname and company:

```
10 00 00 00
12
50 61 70 E0 20 43 65 6C 6C
09 13 00 10
```

```
ID
NAME_LENGHT
NAME
CXF1
```

The third part stores email addresses:

```
1C
32
00 00 00 03
10 00 00 00
24
6F 64 6F 6D 65 6E 69 63 40
6C 69 62 65 72 6F 2E 69 74
```

```
EMAIL_LENGHT
EMAIL_ID
EMAIL_FLAG
ID
EMAIL_ADDRESS_LEN
EMAIL_ADDRESS
            |
```

A.2 Calendar

Calendar entries are stored in `Calendar` file (located under `C:\System\Data`). A calendar's entry belongs to one of the following categories: *anniversary*, *meeting* or *note*. A sample of calendar's entry, an anniversary without alarm, is shown below:

```
03
0F 00 00 00
0A
52
02 00 00 00
01 00 00
A4 28 52 03
05
AA 28 A2 AC
01 00 00 01 00
08 00 00
20
41 6E 6E 69 76 65 72 73
61 72 79 41 6F 66 66
0E 20 29
AA 28
AA 28
```

```
FT
VS
AF
BL
ID
 Flag1
CD (GG MM)
 05
AD A2 AC
 Flag3
 Flag2
TL
 Text
  |
ET
SD
ED
```

An example of anniversary with the alarm setted to on is shown here below:

```
03
0F 00 00 00
1A
52
02 00 00 00
01 00 00
A4 28 5B 03
05
AA 28 A2 AC
01 00 00 01 00
3C
43 61 6C 65 6E 41 6C 61
72 6D 53 6F 75 6E 64 32
09 01 00
08 00 00
1E
41 6E 6E 69 76 65 72 73
61 72 79 41 6F 6E
0E 20 29
AA 28
AA 28
```

```
FT
VS
AF
BL
ID
 Flag1
CD (GG MM)
 05
AD AT
 VS2
ANL
ATXT
  |
09 01 00
 Flag2
TL
 Text
  |
ET
SD
ED
```

An example of meeting with the alarm set to off is shown below:

```
00
0F 00 00 00
0A
50
02 00 00 00
01 00 00
A4 28 C6 02
01
A6 28
A8 28
01 00 00 00 00
08 00 00
1A
4D 65 65 74 41 6F 66 66 52 64 61 79
0E 20 29
A6 28 64 05
A6 28 91 05
```

```
FT
SF 00 00 00
AF
50
ID
Flag1
CD (CCMM)
RF
ERD
A8 28
01 00 00 00 00
Flag2
TL
Text
ET
SD
ED
```

An example of meeting with the alarm set to on is shown below:

```
00
0F 00 00 00
1A
50
02 00 00 00
01 00 00
A4 28 34 03
01
A5 28
A6 28
01 00 00 00 00
3C
43 61 6C 65 6E 41 6C 61
72 6D 53 6F 75 6E 64 AF
05 00 00
08 00 00
18
4D 65 65 74 41 6F 6E 52 64 61 79
0E 20 29
A5 28 EC 01
A5 28 90 03
```

```
FT
SF 00 00 00
AF
50
ID
Flag1
CD (CCMM)
RF
ERD
A6 28
01 00 00 00 00
RTN_LEN
RTN
 |
05 00 00
Flag2
TL
Text
ET
SD
ED
```

Some meetings are saved in a different way, we call this kind of entries *special meetings*; here below is shown a special meeting with the alarm set to off and without repetition.

```
00
0F 00 00 00
08
50
02 00 00 00
01 00 00
A4 28 7A 03
08 00 00
1C
53 6D 65 65 74 41 6F
66 66 52 6F 66 66
0E 20 29
A6 28 64 05
A6 28 91 05
```

```
FT
SF 00 00 00
AF
50
ID
Flag1
CD (CCMM)
Flag2
TL
Text
 |
ET
SD
ED
```

An example of special meeting with the alarm set to on and repetition set to off is shown below:

```
00
0F 00 00 00
18
50
02 00 00 00
01 00 00
A4 28 9C 03
3C
43 61 6C 65 6E 41 6C 61
72 6D 53 6F 75 6E 64 AF
05 00 00
08 00 00
1A
53 4D 65 65 74 41 6F
6E 52 6F 66 66
0E 20 29
A6 28 94 02
A6 28 C1 02
```

```
FT
SF 00 00 00
AF
50
ID
Flag1
CD (CCMM)
RTN_LEN
RTN
 |
05 00 00
Flag2
TL
Text
 |
ET
SD
ED
```

Notes are stored in a similar format as special meeting. An example of note is shown below:

```
02
0F 00 00 00
08
52
02 00 00 00
01 00 00
A4 28 6F 03
08 00 00
10
44 61 79 4E 6F 74 65
0E 20 29
A5 28
A5 28
```

```
FT
VS
AF
BL
ID
Flag1
CD {ID2}
Flag2
NL
Text
ET
SD
ED
```

A.3 Events log

Events and status changes are stored in `Logdbu.dat` file (located under `C:\System\Data`), and can belong to the following categories: sms , mms, voice and data calls, SIM changes. In the last case, the event is stored as an sms, so we will not examine it. Details about the fields are reported in Table A.4.

An example of SMS is shown in the following:

```
03
BA A3 EB EB B6 2C E1 00
FF
B2
60 16 00 00
1C
52 61 6D 6F 6E 61 20 4D
6F 72 65 74 74 69
04 00 05 00 33
80
44 4F 4D 41 4E 49 20 53 45 52 41 20
54 49 20 49 4E 56 49 54 4F 20 41 44
20 55 53 43 49 52 45 20 49 4E 53 49
45 4D 45 20 58 20 55 4E 41 20 43 45
4E 41 2C 6F 76 76 69 61 6D 65 6E 74
65 20 6F 67
1A
2B 33 39 XX XX XX XX
XX XX XX XX XX XX
18
00
00 00 00 01 00 00
00 00 00 00 00 00
00 00 00 00 00 00
00 01 00 00 00 A4
```

```
03
DATE
FF
NAME_FLAG
  |
60 16 00 00
NAME_LENGTH
NAME
04 00 05 00 33
MESS_LENGTH
  |
  |
MESS
  |
  |
  |
NUMBER_LENGHT
NUMBER
  |
18
DIRECTION
00 00 00 01 00 00
00 00 00 00 00 00
00 00 00 00 00 00
00 01 00 00 00 A4
```

A voice call example is shown below:

```
00
1F 66 4A EF C8 2C E1 00
01
70
63 16 00 00
1C
52 61 6D 6F 6E 61 20 4D
6F 72 65 74 74 69
01
00
00 00 00 00
67 02 36
1A
2B 33 39 XX XX XX XX
XX XX XX XX XX XX
20 03 00 00 02 A6
17 00 00 A4
```

```
00
DATE
01
NAME_FLAG
63 16 00 00
NAME_LENGHT
NAME
  |
01
DIRECTION
CALL_TIME
67 02 36
NUMBER_LENGHT
NUMBER
  |
20 03 00 00 02 A6
17 00 00 A4
```

An MMS example is reported below:

```
05
41 47 86 12 05 2A E1 00
01
F0
00 00 00 00
0E
54 69 6D 20 6D 6D 73
00 00 07 00 00 00
02 00
30 08
36 34 31 38 2C 37 32 37
5A
```

```
05
DATA
01
F0
00 00 00 00
0E
PROVIDER
00 00 07 00 00 00
02 00
NS
NUMBER
5A
```

A data call, or data traffic log entry, may belong to two different categories which are related to the type of storage format used: *mms-type* and *sms-type*. The text body of sms or mms is used to store a single packet content. An example of sms-type data call is reported below:

```
03
C1 36 5F 21 08 2A E1 00
FF
A2 03 00 00 00 03
00 04 00 33
7E 50 65 72 20 75 74 69 6C 69 7A 7A
61 72 65 20 69 6C 20 73 65 72 76 69
7A 69 6F 20 64 65 76 69 20 61 74 74
69 76 61 72 65 20 41 6C 69 63 65 20
4D 41 49 4C 20 65 20 61 73 73 6F 63
69 61 72 65
0A
34 39 30 30 31
18 00 00
```

```
03
DATA
FF
A2 03 00 00 00 03
00 04 00 33
  |
  |
PACKET_CONTENT
  |
  |
  |
SEP_MES_NUM
34 39 30 30 31
FLAG_END
```

A.4 SMS

SMS are stored in the first folder (assuming that the folders are ordered alphabetically) in `/System/Mail` folder. An example of received message is reported in the following table:

```
68 3C 00 10 68 3C
        ...
00 10 00 00 00 00
25 3A 00 10
0C 52 69 63 65 76
0E
20 29 34 18
00 10 45 04 01 00
01 00 00 00 02 00
01
00 00 00 00 00 00
00 01 00 00 00
FA 54 17 46 F2 29 E1 00
28
33 34 39 34 36 37 37 31 34 36 34
44
69 73 74 65 66 61 6E 6F 20 41 6C 65
00 00 00 00 00 02
00 00 00 02 03 00
00 00 01 00 00 00
00 00 00 00
F1 5A 15 41 F2 29 E1 00
02 91
34
2B 33 39 33 32 30 35
38 35 38 35 30 30
15 00 81
28
33 34 39 34 36 37 37 31 34 36
```

```
68 3C 00 10 68 3C
        ...
00 10 00 00 00 00
flag1
Text
ET
Received_Flag_Mar
00 10 45 04 01 00
01 00 00 00 02 00
Received_Flag
00 00 00 00 00 00
00 01 00 00 00
Date
SNL
Number
NL
Name
00 00 00 00 00 02
00 00 00 02 03 00
00 00 01 00 00 00
00 00 00 00
SCRD
SCF
SCNL
SCN
  |
ESNF
ESNL
ESN
```

An example of sent message is reported in the following table:

```
68 3C 00 10 68 3C
        ...
00 10 00 00 00 00
25 3A 00 10
10
49 6E 76 69 61 74 6F
0E
20 29 34 18
00 10 F5 02 01 00
00 00 00 00 00 00
00
00 02 03 00 00 00
01 00 00 00 01 00
00 00
00 0C 09 22 F2 1C 32 E1 00
00 91
34
2B 33 39 33 34 39 32
30 30 30 38 39 38
04 91
34
2B 33 39 33 34 39 34
36 37 37 31 34 36
00 00
00 93 2D 4B F2 29 E1 00
```

```
68 3C 00 10 68 3C
        ...
00 10 00 00 00 00
flag1
TL
Text
ET
Received_Flag_Mar
00 10 F5 02 01 00
00 00 00 00 00 00
Received_Flag
00 02 03 00 00 00
01 00 00 00 01 00
00 00
Date
flag2
UNL
UN
  |
RNF
RNL
RN
  |
00 00
SCRD
```

Field Name	Size (Bytes)	Description	Example
VD	2	(Uncertain) Used by DB indexing	D2 64
ID	4	Contact identifier (stored as little-endian)	10 00 00 00
CXF1	9	Flag. The first byte have to be equal to the last four. The second byte depends on Symbian version; values can be 09, 0B, 10. "13 00 10" is constant.	FF 09 13 00 10 FF FF FF FF
EDIT_DATE (ED)	8	17 bytes-offset from CXF1. Represents the number of microseconds from year zero.	76 12 9D FA 0F 20 E1 00
CREATION_DATE (CD)	8	Represents the number of microseconds from year zero.	76 12 9D FA 0F 20 E1 00
TYPE_TABLE_LEN (TTL)	1	41 bytes offset from CXF1. Stores the lenght (in bytes) of the TYPE_TABLE	1D
TYPE_TABLE (TT)	TTL	Table of 12-bytes lenght rows, describing the types of the corrisponding data in the DATA_BLOCK. The first 5 bytes are the table start flag. The last 5 bytes indicate the end table flag. The first 12-bytes row does not contain useful information. For further information about data types, see Table A.1.	04 00 00 00 04 14 02 00 00 00 00 00 00 00 00 00 00 04 02 C0 07 00 00 00 00 00 00 00 00 04 00 00 00 00
DATA_BLOCK_LEN (DBL)	1	Stores the size in bytes of DATA_BLOCK	1A
FIELD_FLAG (FF)	2	Flag which is repeated as many times as the number of fields in DATA_BLOCK.	20 00
DATA_BLOCK (DB)	$\frac{DBL-2FF-1}{2}$	Stores contact's information, according to fields type described in TYPE_TABLE. Each field is separated by 00.	33 33 38 38 37 36 35 34 32 33
ID	4	Contact identifier (stored as little-endian) - The same as above.	10 00 00 00
NAME_LENGHT (NL)	1	The size in nibbles of the name field.	0E
NAME	$\frac{NL}{2}$	The contact's name	43 6C 61 75 64 69 6F
SURNAME_LENGHT (SL)	1	The size in nibbles of the surname field.	08
SURNAME	$\frac{SL}{2}$	The contact's surname	43 65 71 61
COMPANY_NAME_LENGHT (CNL)	1	The size in nibbles of the company field.	0C
COMPANY_NAME (CN)	1	The contact's company	44 72 2E 77 68 79
CONTACT_END	4	Flag. Denotes the end of a contact's details. The first byte depends on symbian version (09, 0B, 10, as in CXF1 field). The other bytes are constant.	09 13 00 10
EMAIL_LENGHT (EL)	1	The size in nibbles of the email address block.	1C
EMAIL_ID	$\frac{EL}{2}$	The ID of email address	32
EMAIL_FLAG	4	Flag.	00 00 00 03
ID	4	Contact identifier (stored as little-endian) - The same as above.	10 00 00 00
EMAIL_ADDRESS_LENGHT (EFL)	1	The size in nibbles of the email address string.	24
EMAIL_ADDRESS	$\frac{EFL}{2}$	The email address string.	6F 64 6F 6D 65 6E 69 63 40 6C 69 62 65 72 6F 2E 69 74

Table A.2: This table lists all contact's data which can be found in the Contacts.cdb. Since data are located in three logical file areas, the table is split in three parts.

Field Name	Size (Bytes)	Description	Example
FT	1	Indicates the event type: if 00 is a Meeting if 02 is a daynote if 03 is an anniversary.	00
VARIABLE_SEQUENCE(VS)	4	A four bytes variable secuence if the entry represents a Meeting and the first byte is 10 this indicates the meeting needs to be processed in a different way	0F OO OO OO
ALARM_FLAG (AF)	1	This byte indicates if the alarm is set to ON (1A for normal events 18 for Special Meetings) or OFF (0A for normal events 08 for Special Meetings).	0A
BODY_LENGTH (BL)	1	Represents the length of the in nebbles of the following part of the entry.	52
ID	4	Calendar entry identifier (stored as little-endian)	02 00 00 00
Flag1	3	Indicates a calendar entry in this area	10 00 00
CREATION_DATE (CD)	4	Stores the creation date of the Calendar entry, is composer by GG and MM.	A4 28 52 03
DAY (GG)	2	Represents the day part of the CD field.	A4 28
MONTH (MM)	2	Represents the month part of the CD field.	52 03
REP_FLAG (RF)	1	Appears only for meeting type entryes; indicates if the repetition of the meeting is daily (value 01), weekly (value 02), montly (value 03).	01
REPEAT_UNTIL	2	Appears only for meeting type entryes; indicates the date until the event has to be repeated.	A5 28
ANNIVER_DATE (AD)	2	Stores the date of the event, is an integer counting the number of days since 1-1-1980. This field appears only if the entry type is Anniversary.	AA 28
ALARM_TIME (AT)	2	If the alarm is set to on stores the information about the alarm time, else is unused. For the day note this field does not appear.	A2 AC
VAR_SEQ	5	is a variable secuence, in case of Anniversary the first 3 bytes are 01 00 00 if the anniversary's alarm is set to off the lasttwo are 01 00 may vary but their value is always lass than 32.,	01 00 00 01 00
AL_NAME_LEN (ANL)	1	Indicates the size in nibbles of the ALARM_NAME field	3C
AL_NAME (AN)	$\frac{ANL}{4}$	Stores a text field indicating the ringtone name for the alarm	43 61 6C 65 6E 41 6C 61 72 6D 53 6F 75 6E 64 32
Flag2	3	is a flag characterizing a calendar event	08 00 00
TEXT_LENGTH (TL)	1	Indicates the size in nibbles of the TEXT field	20
TEXT	$\frac{TL}{2} - 1$	Stores the text field of the calendar entry	41 6E 6E 69 76 65 72 73 61 72 79 41 6F 66 66
END_TEXT (ET)	3	Is the end flag of the TEXT field the value is always 0E 20 29	0E 20 29
START_DATE (SD)	2	Stores the starting date of the entry if is a note or an anniversary else it does not appear	A5 28
START_DATE_M (SDM)	4	Stores the starting date of the entry if is a meeting else it does not appear	A5 28 EC 01
END_DATE (ED)	2	Stores the ending date of the entry if is a note or an anniversary else it does not appear	A5 28
END_DATE_M (EDM)	4	Stores the ending date of the entry if is a meeting else it does not appear	A5 28 90 03

Table A.3: This table lists all calendar entries such as Notes Meetings Anniversaries stored in the Calendar file.

Field Name	Size (Bytes)	Description	Example
		COMMON PART	
START_DATA_FLAG (SDF)	1	Indicates a date starting at next byte, this flag combined with the EDF indicates what kind of data are stored in the session. (03 DATA FF indicates an SMS, GPRS traffic or 'DATAMESSAGE' MMS, 05 DATA 01 MMS recived from the operator or GPRS traffic to the operator, 00 DATE 01 indicates incoming and outgoing calls)	03
DATE	8	Stores the date in wich the operation has been performed. The date in stored in big endian format.	BA A3 EB EB B6 2C E1 00
END_DATA_FLAG (EDF)	1	Is located with an offset of 8 after the SDF and indicates that a date finishes here.	FF
NAME_FLAG (NF)	1	Is located with an offset of 1 after the EDF and if the entry refers to a contact present in the address book (for the SMS the value is B2 if the message is to/from a contact in the address book for calls the value can be 70 if present else 60).	B2
NAME_LENGTH (NL)	1	If the contact is present in the address book is located with an offset of 4 after the NF and indicates the length in nibbles of the subsecuent field $NAME$.	1C
NAME	$\frac{NL}{2}$	If the contact is present in the address book is located with an offset of 1 after the NL and stores the name of the contact stored in the address book.	52 61 6D 6F 6E 61 20 4D 6F 72 65 74 74 69
		SMS PART	
MESS_LENGTH (ML)	1	If the contact is present in the address book is located with an offset of 5 after the $NAME$ field and indicates the length in nibbles of the subsecuent field $MESS$, else is st with an offset of 5 after NF.	80
MESS	$\frac{ML}{2}$	Is located with an offset of 1 after the ML and stores the message sent/recived.	44 4F 4D 41 4E 49 20 53 45 52 41 20 54 49 20 49 4E 56 49 54 4F 20 41 44 20 55 53 43 49 52 45 20 49 4E 53 49 45 4D 45 20 58 20 55 4E 41 20 43 45 4E 41 2C 6F 76 76 69 61 6D 65 6E 74 65 20 6F 67 1A
NUMBER_LENGTH (NUL)	1	Is located with an offset of 1 after the $MESS$ and indicates the length in nibbles of the subsecuent field $NUMBER$.	1A
NUMBER	$\frac{NUL}{2}$	Is located with an offset of 1 after the NUL and stores the number of the sender/recipient of the message.	2B 33 39 XX XX XX XX XX XX XX XX XX XX
DIR	1	Is located with an offset of 2 after the $NUMBER$ and stores the information about the direction of the data stored in the section (value 00 indicates a sent message else the value will be 02).	02
		CALL PART	
DIRECTION	1	If the contact is present in the address book is located with an offset of 2 after the $NAME$ and stores the information about the direction of the data stored in the section (value 00 indicates an exiting call else the value will be 02).	02
CALL_TIME (CT)	4	Is located with an offset of 2 after the DIR field and stores the information about the duration of the call, data is atored in big endian format.	00 00 00 00
NUMBER_LENGHT (NUL)	1	Is located with an offset of 4 after the CT and indicates the length in nibbles of the subsecuent field $NUMBER$.	1A
NUMBER	$\frac{NUL}{2}$	Is located with an offset of 1 after the NUL and stores the number of the sender/recipient of the message.	2B 33 39 XX XX XX XX XX XX XX XX XX XX
		MMS PART	
PROV_LEN (PL)	1	is located with an offset of 5 after the EDF and indicates the length in nibbles of the subsecuent field $PROVIDER$.	0E
PROVIDER	$\frac{PL}{2}$	Is located with an offset of 1 after the PL field and stores the information about the mms service provider's name.	54 69 6D 20 6D 6D 73
NUM_START (NS)	2	Is located with an offset of 4 after the CT and indicates the length in nibbles of the subsecuent field $NUMBER$.	30 08
NUMBER	$\frac{NUL}{2}$	Is located with an offset of 1 after the NUL and stores the number of the sender/recipient of the message.	36 34 31 38 2C 37 32 37

Table A.4: This table lists all event entries such as SMS, MMS, voice and data calls, SIM change.

Field Name	Size (Bytes)	Description	Example
		COMMON PART	
flag 1	4	If this flag is in the starting part of the file or at offset 5 the file does not contain SMS so there will be no need to parse it.	25 3A 00 10
REC_FLAG_MARK (RFM)	4	Received Flag Marker indicates a recived message	20 29 34 18
REC_FLAG (RFM)	1	Starts at byte 13 after the (RFM). If its value is 01 then the message is recived else if the value is 00 the message is a sent message.	10
TEXT_LEN (TL)	1	Generally is just after the flag 1 indicates the message's text length.	10
SPEC_MES (SM)	1	If appears after TEXT_LEN indicates a message from a special number.	02
TEXT	$\frac{TL}{2} - 1$	Stores the text of the SMS message.	49 6E 76 69 61 74 6F
END_TEXT (ET)	1	Indicates the end of the message text.	0E
		RECEIVED MESSAGE	
DATE	8	This field starts 12 bytes after the recived flag (REC_FLAG).	FA 54 17 46 F2 29 E1 00
SEND_NUM_LEN (SNL)	1	This is an otiponal field: appears only if the sender's number is stored in the address book. Indicates the length of the sender NUMBER field.	28
NUMBER	$\frac{SNL}{4}$	Stores the number of the sender if the sender appears in the address book.	33 34 39 34 36 37 37 31 34 36 34
NAME_LENGTH (NL)	1	This is an otiponal field: appears only if the sender's number is stored in the address book. Indicates the length of the following NAME field.	44
NAME	$\frac{NL}{4}$	Stores the name of the sender if the sender appears in the address book.	69 73 74 65 66 61 6E 6F 20 41 6C 65
SERV_CENT_REC_DATE ($SCRD$)	8	Is stored with an offset of 23 bytes after the name field.	F1 5A 15 41 F2 29 E1 00
SERV_CENT_FLAG (SCF)	2	Indicates that the message service center's number starts here.	02 91
SERV_CENT_NUM_LEN ($SCNL$)	1	Indicates the length of the following SERV_CENT_NUM field.	34
SERV_CENT_NUM (SCN)	$\frac{SCNL}{4}$	Stores the number of the messge service provider.	2B 33 39 33 32 30 35 38 35 38 35 30 30
EFF_SERV_CENT_FLAG ($ESCF$)	3	Indicates that the effective message service center's number starts here.	15 00 81
EFF_SERV_NUM_LEN ($ESNL$)	1	Indicates the length of the following EFF_SERV_NUM field.	28
EFF_SERV_NUM (ESN)	$\frac{ESNL}{4}$	Stores the effective number of the SMS message service provider.	33 34 39 34 36 37 37 31 34 36
		SENT MESSAGE	
DATE	8	Is stored with 14 bytes offset from the end of REC_FLAG.	00 0C 09 22 F2 1C 32 E1 00
Flag 2	2	Is a flag indicating the presence of a recived message.	00 91
UNDEF_NUMB_LEN (UNL)	1	Indicates the length of the following UNDEFINED_NUMBER field.	34
UNDEFINED_NUMBER (UN)	$\frac{RNL}{4}$	It is not clear which number does this field stores. Maybe the number of the sender's message service provider.	2B 33 39 33 34 39 32 30 30 30 38 39 38
RECIVER_NUMB_FLAG (RNF)	2	This flag indicates the presence of the sender's number in the next bytes.	04 91
RECIVER_NUMBER_LEN (RNL)	1	Indicates the length of the following RECIVER_NUMBER field.	34
RECIVER_NUMBER (RN)	$\frac{RNL}{4}$	Stores the reciver's number.	2B 33 39 33 34 39 34 36 37 37 31 34 36
SERV_CEN_REC_DATE ($SCRD$)	8	Stores the reciving date for the message service provider, it is stored with an offset of 2 bytes from the end of RECIVER_NUMBER.	00 93 2D 4B F2 29 E1 00

Table A.5: This table lists all fields characterizing an SMS.

Bibliography

[1] RNCOS Industry Research Solutions. Global mobile phone penetration to reach 50% in 2008.
http://www.rncos.com/Blog/2008/05/Global-Mobile-Phone-Penetration-to-Reach-50-in-2008.html.

[2] SNL Financial. 10-year wireless projections.
http://www.snl.com/press/20070823.asp.

[3] Gianluigi Me and Maurizio Rossi. Internal forensic acquisition for mobile equipments. In IEEE Computer Society Press, editor, *4th Int'l Workshop on Security in Systems and Networks (SSN2008), Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS)*, 2008.

[4] Fabio Dellutri, Vittorio Ottaviani, and Gianluigi Me. Forensic acquisition for windows mobile pocketpc. In *Proceedings of the Workshop on Security and High Performance Computing Systems, HPCS 2008, Nicosia, Cyprus June 3-6, 2008*, pages 200–205.

[5] Michael Hampton. Personal data for 28,000 navy personnel found on public web site.
http://www.homelandstupidity.us/2006/06/24/personal-data-for-28000-navy-personnel-found-on-public\-web-site/.

[6] Donna L. Hoffman and Thomas P. Novak. How to acquire customers on the web. *Hardvard Business Review*, (78):179–188, 2002.

[7] Par Raphael Meltz. Marc l***, Le Tigre.
http://www.le-tigre.net/Marc-L.html.
November-december 2008.

[8] Netherlands Forensic Institute. Workflow for mobile phone forensic examinations.
http://www.holmes.nl/MPF/FlowChartForensicMobilePhone\Examination.htm.

[9] Wayne Jansen, Aurélien Delaitre, and Ludovic Moenner. Overcoming impediments to cell phone forensics. In *HICSS '08: Proceedings of the Proceedings of the 41st Annual Hawaii International Conference on System Sciences*, page 483, Washington, DC, USA, 2008. IEEE Computer Society.

[10] Rick Ayers and Wayne Jansen. Guidelines on cell phone forensics. Technical report, NIST, 2007.

[11] Rick Ayers, Wayne Jansen, Ludovic Moenner, and Aurelien Delaitre. Cell phone forensic tools: An overview and analysis update. Technical report, NIST, 2007.

[12] Rick Ayers and Wayne Jansen. Pda forensic tools: An overview and analysis. Technical report, NIST, 2004.

[13] Jonathan A. Zdziarski. *iPhone Forensics - Recovering Evidence, Personal Data, and Corporate Assets*. O'Reilly, 2008.

[14] Andrew Hoog. iphone forensics, annual report on iphone forensic industry. Technical report, chicago-ediscovery.com chicago-ediscovery.com chicago-ediscovery.com chicago-ediscovery.com, 2009.

[15] Paraben Corporation. Paraben device seizure. `http://www.paraben.com`.

[16] Hausi A. Muller, Jens H. Jahnke, Dennis B. Smith, Margaret-Anne Storey, Scott R. Tilley, and Kenny Wong. Reverse engineering: a roadmap. In *ICSE '00: Proceedings of the Conference on The Future of Software Engineering*, pages 47–60, New York, NY, USA, 2000. ACM Press.

[17] Weidong Cui, Marcus Peinado, Karl Chen, Helen J. Wang, and Luis Irun-Briz. Tupni: automatic reverse engineering of input formats. In *CCS '08: Proceedings of the 15th ACM conference on Computer and communications security*, pages 391–402, New York, NY, USA, 2008. ACM.

[18] P. H. Aiken. Reverse engineering of data. *IBM Systems Journal*, 37(2):246–269, 1998.

[19] Gregory Conti, Erik Dean, Matthew Sinda, and Benjamin Sangster. Visual reverse engineering of binary and data files. In *VizSec '08: Proceedings of the 5th international workshop on Visualization for Computer Security*, pages 1–17, Berlin, Heidelberg, 2008. Springer-Verlag.

[20] David L. Altheide. Identity and the definition of the situation in a mass-mediated context. *Symbolic Interaction*, 23(1):1–27, 2000.

[21] Peter Mika. Bootstrapping the foaf-web: An experiment in social network mining, 2004.

[22] Peter Mika. Flink: Semantic web technology for the extraction and analysis of social networks. *Web Semantics: Science, Services and Agents on the World Wide Web*, 3(2-3):211–223, October 2005.

[23] Shanyang Zhao, Sherri Grasmuck, and Jason Martin. Identity construction on facebook: Digital empowerment in anchored relationships. *Comput. Hum. Behav.*, 24(5):1816–1836, 2008.

[24] 3gforensics. Simis 3g: Forensic sim card software. `http://www.3gforensics.co.uk/products/simtools/simis3g`.

[25] SIMCon forensics. Sim card recovery. `http://www.simcon.no/`.

[26] Paraben. Sim card seizure. `http://www.paraben-forensics.com/catalog/product_info.php?products_id=289`.

[27] Jeroen van den Bos and Ronald van der Knijff. Tulp2g – an open source forensic software framework for acquiring and decoding data stored in electronic devices. *International Journal of Digital Evidence*, 2005.

[28] Rosamaria Bertè, Fabio Dellutri, Antonio Grillo, Alessandro Lentini, Gianluigi Me, and Vittorio Ottaviani. Fast smartphones forensic analysis results through miat and forensic farm. *International Journal of Electronic Security and Digital Forensics (IJESDF), Inderscience*, 2008.

[29] Rosamaria Bertè, Fabio Dellutri, Antonio Grillo, Alessandro Lentini, Gianluigi Me, and Vittorio Ottaviani. *Handbook of Electronic Security and Digital Forensics*, chapter A Methodology for Smartphones Internal Memory Acquisition, Decoding and Analysis. Worldscience, 2008.

[30] Peter Stephenson. Using a formalized approach to digital investigation. *Computer Fraud & Security*, 2003:17–20(4), July 2003.

[31] CA) Leyland, Walter E. (Riverside. Lightweight portable emi shielding container, August 1992.

[32] Alessandro Distefano and Gianluigi Me. An overall assessment of mobile internal acquisition tool. *Digital Investigation*, 5(Supplement 1):S121–S127, 2008.

[33] Rick Ayers, Wayne Jansen, Ludovic Moenner, and Aurelien Delaitre. An overview and analysis of pda forensics tool. *Digital Investigation*, 2 (2):pp. 120–132, 2005.

[34] Michael Santarini. Nand versus nor. *EDN*, October 2005.

[35] Microsoft. Linear flash memory devices on microsoft windows ce. `http://www.microsoft.com/technet/archive/wce/plan/flashce.mspx`.

[36] Microsoft. The windows ce 5.0 object store. `http://msdn2.microsoft.com/en-us/library/ms885891.aspx`.

[37] Fabio Dellutri, Vittorio Ottaviani, Daniele Bocci, Giuseppe F. Italiano, and Gianluigi Me. Data reverse engineering on a smartphone. *Submitted to the 5th International Conference on Security and Privacy in Communication Networks (SecureComm 2009)*, 2009.

[38] Chen and Associates. *Reverse-DBMS (Access. 2.0) for Windows Reference Manual Version 3.0*, 1994.

[39] Roger H. L. Chiang. A knowledge-based system for performing reverse engineering of relational databases. *Decis. Support Syst.*, 13(3-4):295–312, 1995.

[40] Kathi Hogshead Davis. August-ii: a tool for step-by-step data model reverse engineering. *Reverse Engineering, 1995., Proceedings of 2nd Working Conference on*, pages 146–154, Jul 1995.

[41] Jean Henrard, Didier Roland, Anthony Cleve, and Jean-Luc Hainaut. Large-scale data reengineering: Return from experience. In *WCRE '08: Proceedings of the 2008 15th Working Conference on Reverse Engineering*, pages 305–308, Washington, DC, USA, 2008. IEEE Computer Society.

[42] Contacts Database (CContactDatabase). Symbian developer library. `http://www.symbian.com/Developer/techlib/v70docs/SDL_v7.0/doc_source/reference/cpp/ContactsModel/CContactDatabaseClass.html`.

[43] Fabio Dellutri, Marco Gaertler, Robert Görke, Giuseppe F. Italiano, and Luigi Laura. Experimental validation of svd-based graph clustering. 2009. Unpublished Manuscript.

[44] Paul Jaccard. Etude comparative de la distribution florale dans une portion des alpes et des jura. *Bull Soc Vaudoise Sci Nat*, 37:547–579, 1901.

[45] Gerald Salton, editor. *Automatic text processing*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1988.

[46] Rudi L. Cilibrasi and Paul M. B. Vitanyi. The google similarity distance. *IEEE Transactions on Knowledge and Data Engineering*, 19(3):370–383, March 2007.

[47] Marco Gaertler. Clustering. In Ulrik Brandes and Thomas Erlebach, editors, *Network Analysis: Methodological Foundations*, volume 3418 of *Lecture Notes in Computer Science*, pages 178–215. Springer, February 2005.

[48] Ulrik Brandes, Marco Gaertler, and Dorothea Wagner. Engineering Graph Clustering: Models and Experimental Evaluation. *ACM Journal of Experimental Algorithmics*, 12(1.1):1–26, 2007.

[49] David Cheng, Ravi Kannan, Santosh Vempala, and Grant Wang. A Divide-and-Merge Methodology for Clustering. *ACM Transactions on Database Systems*, 31(4):1499–1525, 2006.

[50] Amit Deshpande, Luis Rademacher, Santosh Vempala, and Chen Wang. Matrix Approximation and Projective Clustering via Volume Sampling. In *Proceedings of the 17th Annual ACM–SIAM Symposium on Discrete Algorithms (SODA'06)*, pages 1117–1126, 2006.

[51] Petros Drineas, Alan M. Frieze, Ravi Kannan, Santosh Vempala, and V. Vinay. Clustering Large Graphs via the Singular Value Decomposition. *Machine Learning*, 56(1-3):9–33, 2004.

[52] Ravi Kannan, Santosh Vempala, and Adrian Vetta. On Clusterings: Good, Bad, Spectral. *Journal of the ACM*, 51(3):497–515, May 2004.

[53] Daniel A. Spielman and Shang-Hau Teng. Spectral Partitioning Works: Planar Graphs and Finite Element Meshes. In *Proceedings of the 37th Annual IEEE Symposium on Foundations of Computer Science (FOCS'96)*, pages 96–106, October 1996.

[54] Fan R. K. Chung. *Spectral Graph Theory (CBMS Regional Conference Series in Mathematics, No. 92) (Cbms Regional Conference Series in Mathematics)*. American Mathematical Society, February 1997.

[55] Gene H. Golub and Charles F. Van Loan. *Matrix Computations*. John Hopkins University Press, 3rd edition, 1996.

[56] Chris H. Q. Ding and Xiaofeng He. K-Means Clustering via Principal Component Analysis. In *Proceedings of the twenty-first international conference on Machine learning*, volume 69, pages 29–39. ACM Press, 2004.

[57] Chris H. Q. Ding, Xiaofeng He, and Horst D. Simon. Nonnegative Lagrangian Relaxation of K-Means and Spectral Clustering . In *Proceedings of the 16th European Conference on Machine Learning*, volume 3720 of *Lecture Notes in Computer Science*, pages 530–538. Springer, 2005.

[58] Chris H. Q. Ding, Xiaofeng He, and Horst D. Simon. On the Equivalence of Nonnegative Matrix Factorization and Spectral Clustering. In *Proceedings of the fifth SIAM International Conference on Data Mining*, pages 606–610. SIAM, 2005.

[59] Ravi Kannan, Santosh Vempala, and Adrian Vetta. On Clusterings - Good, Bad and Spectral. In *Proceedings of the 41st Annual IEEE Symposium on Foundations of Computer Science (FOCS'00)*, pages 367–378, 2000.

[60] Ulrik Brandes, Marco Gaertler, and Dorothea Wagner. Experiments on Graph Clustering Algorithms. In *Proceedings of the 11th Annual European Symposium on Algorithms (ESA'03)*, volume 2832 of *Lecture Notes in Computer Science*, pages 568–579. Springer, 2003.

[61] Stijn M. van Dongen. *Graph Clustering by Flow Simulation*. PhD thesis, University of Utrecht, 2000.

[62] Kanti V. Mardia, J. T. Kent, and J. M. Bibby. *Multivariate Analysis*. Academic Press, New York, 1979.

[63] Xiaojin Zhu and Ronald Rosenfeld. Improving trigram language modeling with the world wide web, 2001.

[64] Frank Keller and Mirella Lapata. Using the web to obtain frequencies for unseen bigrams. *Comput. Linguist.*, 29(3):459–484, 2003.

[65] Fabio Dellutri, Salvatore Di Blasi, and Giuseppe F. Italiano. Transparent file protection in on-demand computing. In Hamid R. Arabnia, editor, *In Proceedings of the 2007 International Conference on Grid Computing & Applications, GCA 2007, Las Vegas, Nevada, USA, June 25-28, 2007*, pages 116–122. CSREA Press, 2007.

[66] Fabio Dellutri, Giuseppe F. Italiano, and Gianluigi Me. Authentication in wireless mesh networks environment. Unpublished Manuscript.

[67] Fabio Dellutri, Gianluigi Me, and Maurizio A. Strangio. Local authentication with bluetooth enabled mobile devices. In *ICAS-ICNS '05: Proceedings of the Joint International Conference on Autonomic and Autonomous Systems and International Conference on Networking and Services*, page 72, Washington, DC, USA, 2005. IEEE Computer Society.