European Commission
Directorate-General Home Affairs
Prevention of and Fight against Crime Programme

HOME/2010/ISEC/AG/INT/002

ForToo – Forensic Tools against Illegal Use of the Internet

D2: Implementation Report

Workpackage:	WP2: Implementation
Contractual delivery date:	30 Sep 2013
Final draft publication date:	*29 Jan 2014*
Actual delivery date:	*10 Jun 2014*
Leading partner:	FORTH
Contributing partners:	*All Partners*
Editor:	Sotiris Ioannidis
Contributors:	All partners
Internal Reviewers:	Theo Tryfonas
Version:	1.0

Executive Summary:

In this deliverable we present the implementation details for the ForToo toolkit. The toolkit consists of a set of forensic tools that can assist forensic investigations in social networks or mobile devices.

With the support of the Prevention of and Fight against Crime Programme.
European Commission - Directorate-General Home Affairs

This project has been funded with the support of the *Prevention of and Fight against Crime* Programme of the European Commission - Directorate-General Home Affairs. This publication reflects the views only of the author, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

Table of Contents

REVISIONS AND QUALITY CONTROL

Version	**Date**	**Lead contributor**	**Action summary**
0.1	10 Sep 2013	Zacharias Tzermias	Input on Social Forensics toolset
0.2	12 Dec 2013	Panagiotis Andriotis	Steganography component and mobile forensics parts
0.3	27 Jan 2014	Konstantinos Xynos	DEViSE architecture description
0.4	29 Jan 2014	Theo Tryfonas	Review and setting at final draft status
1.0	10 Jun 2014	Theo Tryfonas	Formal submission to PO

1 Introduction

In this document we present the implementation details of each component of the ForToo toolkit. The toolkit consists of a set of forensic tools to help the forensic investigator, when dealing with evidence gathering from social networks and mobile devices. Each component is integrated into a single platform using the DEViSE architecture and visualization tools.

The document firstly presents the DEViSE architecture on chapter 3. Chapter 4 provides implementation details on how to acquire digital forensic evidence from mobile devices, while chapter 5 presents a tool capable of gathering evidence from a variety of social networks. Chapter 6 describes the methodology to detect steganography in JPEG images. Visualization components of the toolkit are presented in chapter 7.

2 DEViSE architecture

An overview of the visualisation architecture was provided in Work Package 1. This provided a highlight of how the architecture's components may fit together in order to develop a visualisation platform. This approach described how smaller, more focused visualisations could be used in the spirit of the UNIX philosophy of building small and letting tools talk to tools.

This section of work details the more practical implementation aspects of DEViSE, in particular the structure of the filesystem and a detailed view of the various XML components of the architecture that facilitate the data exchange. The data exchange is split into three parts, the data exchange between different visualisation tools, and the data exchange between a visualisation tool and the underlying data source, and visualisation integration.

2.1 The DEViSE filesystem

A number of default paths have been created to store the different types of configuration documents needed to run DEViSE visualisation tools. These are standard for all DEViSE applications. The pathnames reflect the Linux naming conventions as most of the development was carried out on this operating system. All these paths are relevant only to the visualisation server; i.e. the computer running the DEViSE visualisation tools.

The system uses the following conventions:

Applications - All applications are stored within /home/viz/TOOL- NAME/. Tools can be configured to operate from an alternate location.

I-DOC - All information documents are stored in /home/viz/xml/idoc/. However, the HistoryManager application also stores an embedded version of the I-DOC within its hierarchical structure.

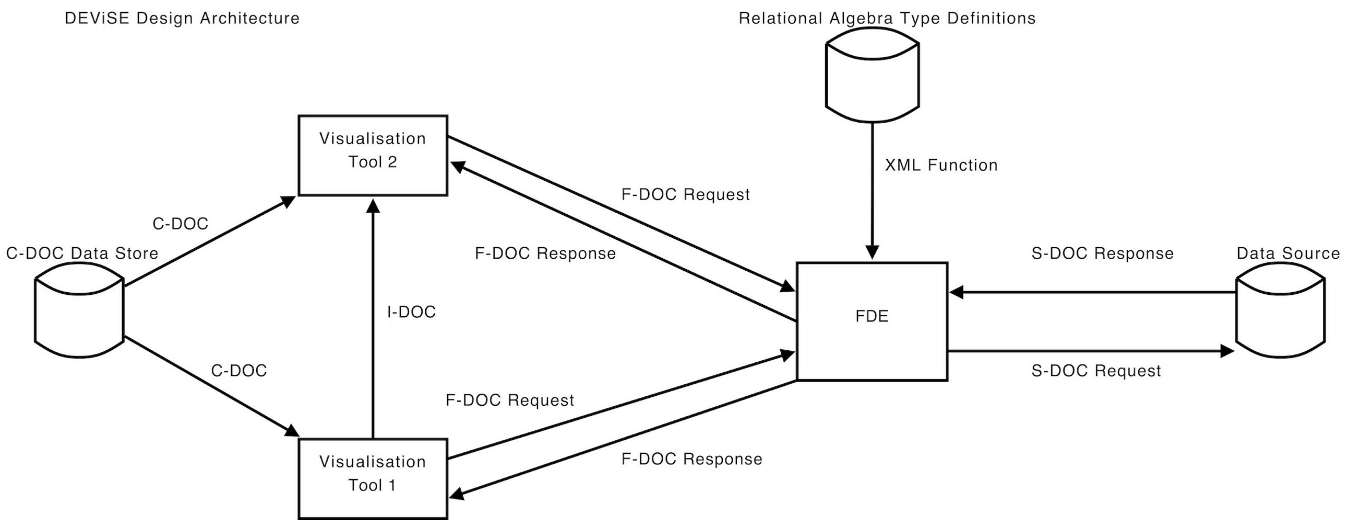
C-DOC - All tool configuration documents are by default stored in /home/viz/xml/toolconf/.

Type Definitions - The FDE data type mappings are stored in /home-/viz/xml/typedef/.

Name Mapping - A mapping between FDE basic data type names and their "human–readable" equivalent are stored in /home/viz/xml/namemap/.

2.2 Data exchange between different visualisation tools

The Figure below represents the implementation architecture of the visualisation system. A number of components were introduced in Work Package 1 and have been implemented as follows.



FDE Implementation in Architecture

2.2.1 Configuration Document (C-DOC)

The following listing is an example of a C-DOC file, in use by the Scattergraph tool. It is currently stored, along with C-DOC's from other tools, in the default hard disk location \home\viz\xml\toolconf. All of the XML elements are required to create a complete C-DOC.

```
<TI vers="0.1">
 <name>Scattergraph</name>
 <description>
   Plots x against y. Good for spotting patterns.
 </description>
 <version>0.0.7</version>
 <location>
   /home/viz/Scattergraph/Scattergraph 0.0.7.py
 </location>
 <input param="*">srcdstip</input>

</TI>
```

The above listing can be broken into the following components:

<TI vers="0.1">

This is the current version of the configuration file. When the C-DOC is referenced, the version number must be present to identify the exact configuration being used. Each tool may have several C-DOC's associated with it due to supporting different input types, but each of these different inputs may have several "versions". There may come a time where a C-DOC containing a particular function type needs to be updated, this tag assists its management. New and existing versions of C-DOCs can be run alongside one another simultaneously, so the effectiveness of revised changes may be compared easily. As new versions of software are notorious for preventing legacy programs from functioning correctly, this simultaneous running allows tools & their configurations, old and new, to continue functioning. From an implementation standpoint, there will be several instances of a tool appearing in an object's menu.

<name>Scattergraph</name>

The <name> tag simply provides the name of the application. It is used in an object's menu to identify the tool and also appears within an I-DOC (see later) to make note whether the tool is being invoked or if it is invoking another. The name should be short, given that it will be inserted in a menu. For more detail about a tool, information should be inserted in the <description> tag.

<description> Plots x against y. Good for spotting patterns. </description>

Provides a short summary on general functionality and expected visual output. This tag has been included in the C-DOC specification so that applications may implement "roll-over" effects on tool names to provide greater insight into an unfamiliar tools' functionality.

<version>0.0.7</version>

The <version> tag identifies for which version of the application this C-DOC is configured. If an application accepts several function types as input, there will be a C-DOC for each function type with the same version number. The version number appears alongside the tool name in an object's menu, to help distinguish between different instances of the same tool.

<location>

/home/viz/Scattergraph/Scattergraph 0.0.7.py

</location>

This is the full path to the application. The default location is /home/viz/tool_name. However the <location> tag allows for storage elsewhere (such as in C:\Program Files\tool_name on Microsoft based operating systems) when non-standard locations are used. As tools are normally run with command-line arguments it is imperative that absolute, not relative, paths are used.

<input param=”*”>srcdstip</input>

The input tag is of particular importance to an object when finding compatible tools, as it contains two items of data, the param attribute and the value (In this instance, srcdstip).

param="*"

This attribute represents the quantity of data the application can visualise. It can either be a fixed amount or an infinite number. In the case of a fixed amount, the desired whole number is applied (e.g. param="6"). Some tools can effectively visualise thousands of events, whereas others can visualise a fixed amount. If an non-fixed quantity is desired, the value is denoted by an asterisk, “*”. When an object menu is generated, the application looks at these two items of data. It checks if the data-type matches and whether an acceptable quantity of events can be provided to the tool referred to by the C-DOC. It is used in conjunction with the <param> tag’s row attribute from the I-DOC (see later).

srcdstip

The value between the two tags is the “data type” the tool can accept as input from another application, in this case, srcdstip, a combination of source and destination ip addresses.

2.2.2 Information Document (I-DOC)

The I-DOC is passed via tools at the command line, using the arguement `-i` or `--input` as shown below.

```
/home/viz/tool/tool 0.0.7.py -i /home/viz/xml/idoc/idoc1191405434.34.xml
```

The following XML listing is an example of an I-DOC.

```
<IDOC>
 <from version=”0.2.5”>Geomap</from>
 <to version=”0.0.7”>Scattergraph</to>
 <param row=”1”>
   <object name=”source destination ip address”>
     <data type=”source computer ip”>
       907212
     </data>
   </object>
 </param>
</IDOC>
```

The application mentioned in the <from> tag is the parent application, i.e. the tool responsible for creating the I-DOC; in the above listing it is the Geomap Tool, version 0.2.5. An I-DOC’s function is to pass event id’s and function names to other tools. Here, the event_id is 907212, the function name is source_computer_ip and the receiving tool is contained within the <to> tag, “Scattergraph”. When Scattergraph initialises, it will run the function named “source computer ip” against event id and will receive the IP address of the attacker from the database.

A complete breakdown of the XML is as follows:

<IDOC>
Stands for “information document”. This identifies this file as belonging to the group that contains data being passed from one tool to another.

<from version=”0.2.5”>Geomap</from>
The name and version of the tool created the I-DOC. In this example, “Geomap”, version “0.2.5” created the I-DOC.

<to version=”0.0.7”>Scattergraph</to>
The name and version of the tool receiving the I-DOC. In this example, “Scattergraph”, version “0.07” will receive the I-DOC.

<param row=”1”>
The <param> tag contains a single attribute, “row”. Row is an integer equal to the number of event id’s within the I-DOC. It is closely related to the param="*" attribute in the <input> tag in a C-DOC. If the C-DOC contains a fixed value, then the I-DOC’s row value should be the same. This is used as a quick redundancy check to make sure that tools which accept fixed amounts of input can accept the I-DOC given to them.

<object name=”source destination ip address”>
The <object> tag contains the type of information that will ultimately be obtained from the database. It is a way of making sure tools which need several different data types to create a visualisation do not get confused when presented with a group of unique identifiers.

<data type=”srcdstip”>907212</data>
This <data> tag is the actual element that contains the data applications will use. The number of <data> elements will equal the number of events that are passed between tools. There are two important items within this tag, the event id and function name.

907212
This text represents the event id of a single event in the data store. This is the only actual data which passes from one tool to another, and is used to purely identify the event from where the data is to be obtained.

type="srcdstip"
The type attribute is the name of a function. The tool which receives the I-DOC will execute this function (in this example, srcdstip) using the event id as input and receive an XML document containing the actual data required.

The above listing is a very basic I-DOC, passing only one <data> element from the Geomap tool to the Scattergraph tool. I-DOC's can contain several <data> elements, each function group stored within a different <object> element. A further I-DOC example containing several elements can be seen below.

```
<IDOC>
 <from version=”0.2.5”>Geomap</from>
 <to version=”1.6”>Piechart</to>
 <param row=”4”>
   <object name=”source destination ip address”>
     <data type=”srcdstip”>907212</data>
     <data type=”srcdstip”>907213</data>
     <data type=”srcdstip”>18</data>
     <data type=”srcdstip”>4654</data>
   </object>
   <object name=”Attack Type”>
     <data type=”attack type definition”>907212</data>
     <data type=”attack type definition”>907213</data>
     <data type=”attack type definition”>18</data>
     <data type=”attack type definition”>4654</data>
   </object>
   <object name=”Date and Time”>
     <data type=”timestmp”>907212</data>
     <data type=”timestmp”>907213</data>
     <data type=”timestmp”>18</data>
     <data type=”timestmp”>4654</data>
   </object>
 </param>
</IDOC>
```

In the above listing a Geomap application passes several events to a Piechart application. Looking at the I-DOC it is possible to make several deductions about the tools involved and their interaction.

The data is passing from a Geomap to a Piechart. To create this I- DOC, an object within the Geomap tool would have been clicked.

The object within Geomap is able to provide at least the three different function types, mainly srcdstip, attack_type_definition and timestmp.

The Piechart tool is able to visualise at least these three data types; they would not be present in the I-DOC if the Piechart was not intended to visualise such data.

The object in Geomap, at the time of interaction, held four unique events (907212, 907213, 18 and 4654).

The Piechart will execute twelve functions. There are four unique event id's and three unique function types in this I-DOC.

The Piechart can visualise four events. There are four event id's listed for each data type and the <param> tag has the value row="4".

With an I-DOC it is possible to find out what data has been passed, where data came from and where it is going to. As I-DOC's are created as XML files, all this information remains after the applications have been closed allowing visuals to be re-opened at a later date.

2.3 Data exchange between visualisation tools and the underlying data source

In work package 1, the data exchange between the underlying data store and visualisations was highlighted. A number of appendices showed some sample queries for data being generated. The components are fully described below.

2.3.1 SOAP XML RPC Request (S-DOC)

The S-DOC is an XML document that is used to query a SOAP server sitting on the database to retrieve results. The FDE generates S-DOC's for basic type definitions; if an application requires data from a known single event, it can call a type definition the FDE can reference to make the S-DOC. If the tool requires a complex query carried out, such as selecting all attacks between two timestamps, the tool needs to create a S-DOC and query the database SOAP server directly. The S-DOC contains a database query encoded in XML which the SOAP server translates and executes on the database. The following listing is an example of a typical S-DOC request to the database SOAP server.

```
<SOAP-ENV:Envelope
xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"SOAP-ENV:encoding
Style="http://schemas.xmlsoap.org/soap/encoding/"/>

 <SOAP-ENV:Body>
   <RELATIONS command="SELECT">
     <REL name="event">
       <ATT name="event id">3</ATT>
     </REL>
   </RELATIONS>
 </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

The listing below is an example of a typical S-DOC response from the database SOAP server. Again, the database mapping can be seen in the code via the <REL> tags. The important changes include the command attribute in the <RELATIONS> tag is now SELECT_RESULTS indicating this XML document contains data retrieved from the database. The first <REL> tag contains two new attributes, "name" and "value". These simply define the unique identifier of this result, in the case of more complex queries that return several items of data each result can be uniquely identified. The <ATT> element is completed with the requested data from the correct relation in the database, in this case a timestamp. Towards the end of the document, there are more <REL> tags with summary information in. "TOTAL RECORDS" and "TOTAL

RESULTS" define how many records were returned in this particular result (1 in this example), and how many different results were returned (1 in this example).

```
?xml version="1.0" encoding="UTF-8"?>
 <RELATIONS command="SELECT RESULTS">
   <REL name="RESULTS ID" value="1">
     <REL name="event">
       <ATT name="timestmp">2007-10-09 11:10:03</ATT>
     </REL>
     <REL name="TOTAL RECORDS">1</REL>
   </REL>
 <REL name="TOTAL RESULTS">1</REL>
</RELATIONS>
```

The various XML elements that create an S-DOC are as follows:

<RELATIONS command="SELECT"> This element identifies the query as a SQL SELECT statement to the SOAP server.

<REL name="event"> This element identifies the table name from which data should be obtained. In this case, the "event" table. Also in the same listing, the code <REL name="source"> can be seen. This is deliberately contained inside the "event" relation as an automatic join occurs between event and source tables. Source's primary key is a foreign key in the event table.

<ATT name="event id">3</ATT> This element provides the comparable WHERE clause in Listing 4.34 for event_id=3. It specifies a column in a table the query should carried out upon.

2.3.2 Function Document (F-DOC)

The Function Document, or F-DOC, is a XML request from a visualisation tool to the DEViSE middleware for data. The F-DOC provides the means for an application to request the execution of a query (i.e. type definition / function) on its behalf on the database and to receive the result of the executed query in a database independent manner.

As the F-DOC uses a SOAP interface to the FDE, the interactions between application and FDE can be divided into the request and the response.

An example of an F-DOC request can be seen below.

```
<SOAP-ENV:Envelope xmlns:SOAP-
ENV=http://schemas.xmlsoap.org/soap/envelope/SOAP-ENV:encodingStyle="http:/
/schemas.xmlsoap.org/soap/encoding/"/>
 <SOAP-ENV:Body>
```

```
    <function>
      <name>source_computer_ip</name>
      <version>1.7</version>
      <eventid>147</eventid>
    </function>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

The main elements in the F-DOC request are as follows:

<name> This element references the type definition / function the application wishes executed.

<version> This is the numbering system used when identifying type definitions. As type definitions are expected to change over time, referencing just the name of the function is not enough. Specifying the version of the function helps prevent problems with legacy type definitions after upgrades have been installed.

<eventid> This references a particular event in the underlying database by its primary key or event id.

An example of an F-DOC response can be seen below.

```
<SOAP-ENV:Envelope xmlns:SOAP-
ENV=http://schemas.xmlsoap.org/soap/envelope/SOAP-ENV:encodingStyle=”http:/
/schemas.xmlsoap.org/soap/encoding/”/>
  <SOAP-ENV:Body>
    <result>
      <relation version=”1.7” name=”source_computer_ip” eventid=”147”>
        <row>
          <data name=”IP”>192.168.204.12</data>
        </row>
      </relation>
    </result>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

The reply from the SOAP server contains the actual data from the data store, packaged in a database independent manner, without the need to reflect on any of the structures that comprise the underlying cloud.

The main elements in the F-DOC response are as follows.

<result> This is the root element from which the return information is packaged within.

<relation> This element contains three attributes, a summary of the values received from the F-DOC request file.

<row> This element is the root for all the returned data elements.

<data> The <data> element contains an attribute and a value. The attribute, “name”, represents the generic data type of the value, in this case “IP”. The value is the actual data the application will visualise, an IP address “192.168.204.12”.

2.4 sqlite Extension to DEViSE

The DEViSE architecture makes use of XML to represent data. XML has the ability to store information in a tree-like structure. This structure lends its self very well when representing relationships between tables. Although the current architecture makes use of Apache’s Cassandra data store, as depicted in WP1, the original system was built upon a relational database.

Modifications were made to include support for sqlite databases which could then be implemented and queried as required. This lets forensic analysts feed in data structures from mobile devices and then query them with the use of the XML notation as specified by DEViSE.

The system is also capable of querying the database in a number of ways and this allows it to represent data in a different manner.

Below, figure A, is a query that is made against the system, that interacts with the sqlite database extracted from an android phone.

```
<RELATIONS command="SELECT">
    <REL name="sms">
        <ATT name="_id"/>
        <REL name="threads">
        </REL>
    </REL>
</RELATIONS>
```

Figure A. - Join query for tables sms and threads

The result of the join query, from figure A, is then presented in figure B where the joining result is presented. This can be verified by the fact that the thread_id from table sms matches with that of _id from table threads.

```
... [cut down for brevity] ...
<REL name="RESULTS_ID" value="6">
    <REL name="sms">
        <ATT name="_id">6</ATT>
        <ATT name="body">Thanks for signing up to My T-Mobile. Your One
Time Pin is: 0000. Just enter this next time you log in and you'll be up
and running.</ATT>
        <ATT name="thread_id">4</ATT>
        <ATT name="address">T-Mobile</ATT>
        <ATT name="date">1390748250000</ATT>
        <REL name="threads">
            <ATT name="_id">4</ATT>
            <ATT name="date">1390748250000</ATT>
        </REL>
    </REL>
    <REL name="TOTAL_RECORDS">2</REL>
</REL>
... [cut down for brevity] ...
```

Figure B - Join result of sms data example

Likewise it is possible to query the tables in reverse order whereby the threads table on _id will be matched with that from sms. The query example is shown in Figure C.

```
<RELATIONS command="SELECT">
    <REL name="threads">
        <ATT name="_id"/>
        <REL name="sms">
        </REL>
    </REL>
</RELATIONS>
```

Figure C - Join conducted using threads and then sms

Figure D demonstrates the new join's result. This allows the analyst to easily extract data in the same format as that used by DEViSE in the Apache Cassandra data store.

```
... [cut down for brevity] ...
<REL name="RESULTS_ID" value="6">
    <REL name="threads">
```

```
        <ATT name="_id">4</ATT>
        <ATT name="date">1390748250000</ATT>
        <REL name="sms">
            <ATT name="_id">6</ATT>
            <ATT name="body">Thanks for signing up to My T-Mobile. Your One
Time Pin is: 0000. Just enter this next time you log in and you'll be up
and running.</ATT>
            <ATT name="thread_id">4</ATT>
            <ATT name="address">T-Mobile</ATT>
            <ATT name="date">1390748250000</ATT>
        </REL>
    </REL>
    <REL name="TOTAL_RECORDS">2</REL>
</REL>
... [cut down for brevity] ...
```

Figure D - Result of the join on threads and sms

Since the tables can be easily exported they can then be processed and stored in the Cassandra data store for future querying by the visualisation tools. The Cassandra data store also becomes the centralised storage system for all the forensic artefacts collected over a larger time period. This can then be used for future investigations which could span large data sets or even searches across multiple cases, if needed.

2.5 NFAT Extension to DEViSE

The DEViSE system was also extended to include network captures. This was done by capturing network traffic, converting it into PDML, and then converting PDML into the DEViSE format. The final format then allows the analyst to insert the converted network capture into the Apache Cassandra data store. Once in the data store it is possible to query multiple frame packets.

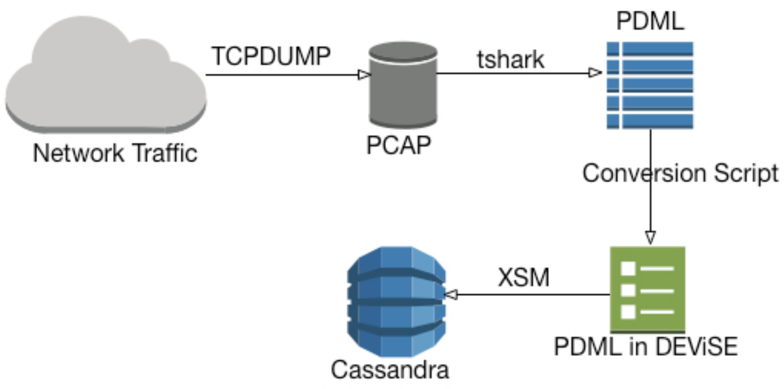


Figure E – Network Traffic storage

Figure E demonstrates the different phase that the captured network traffic go through before being inserted into the data store.

Since each individual frame is inserted into the data store in a structured way it is possible to query on a number of fields and extract parts or all of the data from the data capture. Original captures are kept in order for the findings to be cross-referenced.

3 Mobile Forensics Component

The databases we are providing for visualization are mentioned at the following list:

- /data/com.android.email/databases/EmailProvider.db:
Tables of interest: Account, HostAuth, Message. Here we can find information about the usernames (and maybe passwords) the person under investigation uses to connect to email providers. Also, the emails stored in the device can be found there.

- /data/com.android.providers.contacts/databases/contacts2.db:
Tables of interest: accounts, contacts, data, raw_contacts, speed_dial, status_updates. In this database we will find names and telephone numbers stored in the device or the SIM or Google through the Google account of the user.

- /data/com.android.providers.telephony/databases/mmssms.db:
Tables of interest: pdu, sms, threads. This is the database where all the information considering the sms activity can be found.

- /data/com.facebook.katana/databases/fb.db:
Tables of interest: connections, friends_data, search_results. Since most of the facebook activity is not recorded on the device these tables could provide information about friend’s data and search results. Also they will reveal the timestamp of the last syncing.

- /data/com.google.android.gsf/databases/talk.db:
Tables of interest: accounts, contacts, messages. Talk used to be the default chat application the Android devices used to offer by the time of this report writing. Text messages and account info are the possible outcomes from this database.

- /data/com.twitter.android/databases/*twitter_User_id*.db:
Tables of interest: lists, messages, statuses, user_metadata, users. This is the Twitter database. Data coming from here can be integrated with the social network tool.

- /data/com.skype.raider/files/*Skype_username*.db:
Tables of interest: CallMembers, Calls, Chats, Contacts, Conversations, Messages, Voicemails. Like Twitter this is a database that holds information about the Skype activity of the user. It can be used in conjunction to the social network component.

3.x Mood Analysis from SMS and Instant Messengers

3.x.1 Introduction

Recently we witnessed a remarkable technological bloom on mobile communications and things have changed considering the mobile networks, the devices and the users themselves. Our needs for processing power and usability have grown exponentially along with the capabilities of flagship devices that will be equipped with 64bit processors in 2014. Mobile phones upgraded from simple telephone devices to sophisticated mini computers. The forms of communication between users also have changed but short text messages (SMS) are still one of the most popular services mobile phones provide. In addition, modern smartphones can also send emails or provide the functionality to the user to chat with friends using instant messengers. We can keep notes, update our status in social media or comment other peoples' statuses. Therefore, we store in these devices valuable textual information depicting our feelings for other people.

A forensic analysis usually includes the examination of all electronic devices the suspect uses. The analysts try to extract as much information as they can for the person into question revealing various artifacts. In digital forensics we are interested in recovering deleted files and presenting the extracted data in an efficient way. Text messages are an integral part of our communications and therefore could potentially provide valuable evidence during a forensic investigation. The tool we developed is able to analyze text messages and calculate a score, which describes the mood precipitation the message leaves to the user. Thus, it is acting as a behavioural pattern-retrieving component.

3.x.2 Design Aspects

We already mentioned in D1 - 3.3.1 (Andriotis et al., 2012) that we can perform forensic analysis to smartphones and tablets running the Android Operating System, utilizing open source tools (http://developer.android.com) and basic Linux commands like 'dd' to gain a physical image of the internal memory and its various partitions. In the data partition we will primarily be able to see the data that all applications store internally in SQLite databases once we mount the image to an Unix-like system. Such data are collected every time we use the devices and they are responsible to provide their functionality. The appropriate folder where the SMS data are stored in the data partition is the:

/data/com.android.providers.telephony/databases/mmssms.db.

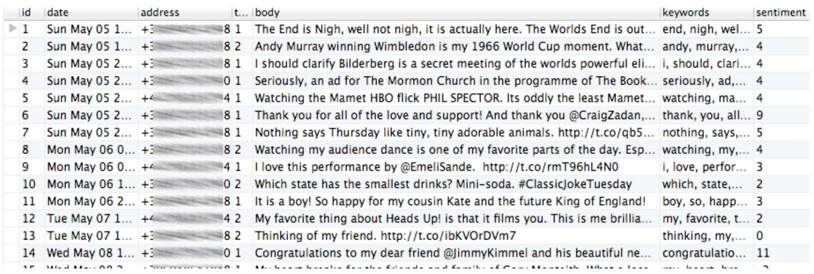
When the forensic examiner mounts the partition on a computer, a database browser, looking at the SMS table of the database, can view the sent and received SMS. Our tool uses text-mining methods to present the sentiment state of entities exchanging messages through the phone. We use a simple bag-of-words approach in order to achieve the mood classification of an SMS. Our research revealed that methods used to extract information from Twitter feeds could be also used to perform SMS mood analysis (to be presented at IFIP W.G. 11.9 on Digital Forensics, Vienna, January 2014).

The tool we developed uses the Apache Lucene Library (http://lucene.apache.org) to perform sentiment analysis on SMS and, in addition, brings the opportunity to the analyst to search fast and efficient for keywords in the database using indexing. Indexing is the method that web search titans employ to perform fast and accurate searches and deliver reliable results. In other words, our tool works like Google or Bing for the database under examination in order to deliver faster results than an SQL query.

The design concept behind the implementation of the tool is simple. First we transfer all the requested content from the SQLite database (which is the database under investigation that was found in a device) to a MySQL database running in our computer. Apache Lucene will produce a set of keywords of the individual SMS (without using any stemming) and also will provide the stemmed words for the SMS sentiment analysis. When all data have been processed we utilize again the Apache Lucene Library to create an index of the MySQL database into the developing machine, which will be responsible to answer any text query the analyst might have. The same concepts could be applied for all kinds of text data like emails or chat messages.

The advantages of this searching technique are mainly that the analyst will be able to indirectly query the SQLite database without having any particular knowledge of the SQL language, through the index, like doing a Google web search. One more advantage of this method is that the search is faster than querying the database itself.

For the needs of this report we simulated the messages in the mmssms.db SQLite database with some tweets from our public tweet database. We collected tweets from popular users using the Twitter API (we write a simple PHP script for this cause). After running the tool the forensic analyst will obtain a MySQL database containing the original data from the mmssms.db SQLite database and also the keywords extracted from each message. Each tuple contains the calculated sentiment score. A view of the database is provided below.



id	date	address	t...	body	keywords	sentiment
1	Sun May 05 1...	+3[redacted]8	1	The End is Nigh, well not nigh, it is actually here. The Worlds End is out...	end, nigh, wel...	5
2	Sun May 05 1...	+3[redacted]8	2	Andy Murray winning Wimbledon is my 1966 World Cup moment. What...	andy, murray,...	4
3	Sun May 05 2...	+3[redacted]8	1	I should clarify Bilderberg is a secret meeting of the worlds powerful eli...	i, should, clari...	4
4	Sun May 05 2...	+3[redacted]0	1	Seriously, an ad for The Mormon Church in the programme of The Book...	seriously, ad,...	4
5	Sun May 05 2...	+4[redacted]4	1	Watching the Mamet HBO flick PHIL SPECTOR. Its oddly the least Mamet...	watching, ma...	4
6	Sun May 05 2...	+3[redacted]8	1	Thank you for all of the love and support! And thank you @CraigZadan,...	thank, you, all...	9
7	Sun May 05 2...	+3[redacted]8	1	Nothing says Thursday like tiny, tiny adorable animals. http://t.co/qb5...	nothing, says,...	5
8	Mon May 06 0...	+3[redacted]8	2	Watching my audience dance is one of my favorite parts of the day. Esp...	watching, my,...	4
9	Mon May 06 0...	+4[redacted]4	1	I love this performance by @EmeliSande. http://t.co/rmT96hL4N0	i, love, perfor...	3
10	Mon May 06 1...	+3[redacted]0	2	Which state has the smallest drinks? Mini-soda. #ClassicJokeTuesday	which, state,...	2
11	Mon May 06 2...	+3[redacted]8	1	It is a boy! So happy for my cousin Kate and the future King of England!	boy, so, happ...	3
12	Tue May 07 1...	+4[redacted]4	2	My favorite thing about Heads Up! is that it films you. This is me brillia...	my, favorite, t...	2
13	Tue May 07 1...	+3[redacted]8	2	Thinking of my friend. http://t.co/ibKVOrDVm7	thinking, my,...	0
14	Wed May 08 1...	+3[redacted]0	1	Congratulations to my dear friend @JimmyKimmel and his beautiful ne...	congratulatio...	11

Figure 1: A view of the MySQL database where the data are stored along with the keywords and the calculated sentiment score.

The algorithm we used takes into consideration the existence of emoticons and also evaluates the sentiment strength of a given word. The goal of the proposed framework is to provide to a timeline analysis interface that depicts the emotional inference of each SMS that was found in the smartphone.

The Emotional Chronology could provide indications of the user's emotional state during a specific time interval. The information we could extract by such a viewing might be helpful because we can focus quickly on specific periods of time that could be of interest. For instance, when analysts deal with a case where the person under investigation is always in good mood, and there exists a period of time the suspect seems to have negative emotions, it will be more efficient to start the examination by analyzing the specific timeframe.

The next figure demonstrates the timeline view of the SMS set that was found in the mmssms.db SQLite database, including sent and received messages from all phone contacts. We observe for example, that most of the time the smartphone owner exchanges messages with positive emotional fingerprint. There are although periods of time (e.g. from 12th August until 17th August) when the sentiment score is negative and maybe this is a period that a forensic analyst would like to search more carefully. The proposed timeline view gives the opportunity to the forensic investigator to maintain a generic view of the user's behaviour.

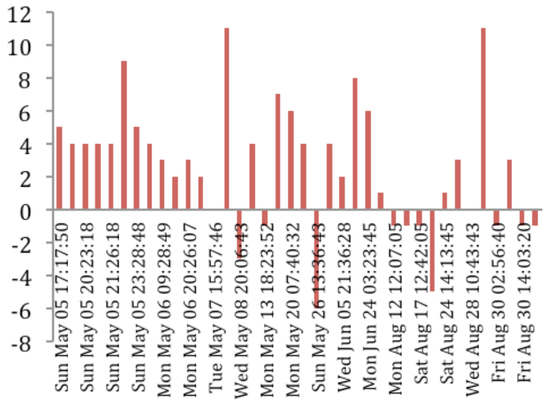


Figure 2: Timeline Emotional Analysis based on SMS found in the device.

The timeline view can be more consistent presenting only the communication between two parts, or indicating only received or sent messages. The next figure depicts the communication between the person under investigation and a single entity (+xxxxxxxxxx38), based on the telephone number stored in the database (left). On the right hand side the timeline view depicts the emotion of messages sent by the person under investigation to all its contacts.

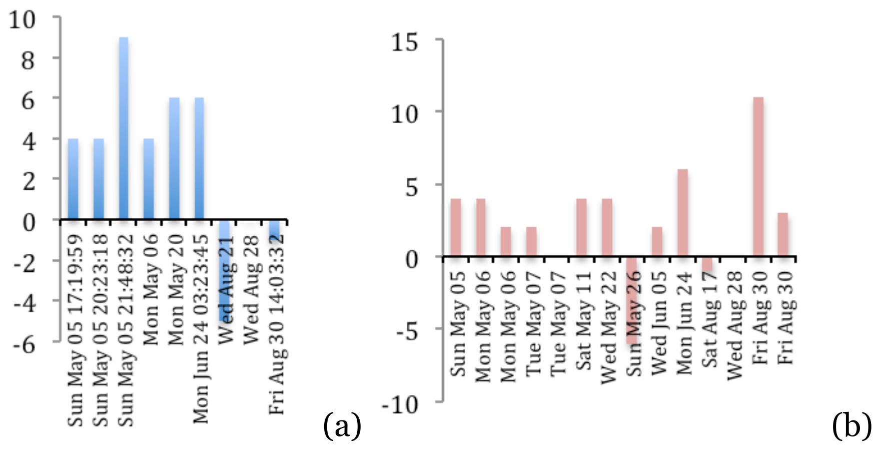
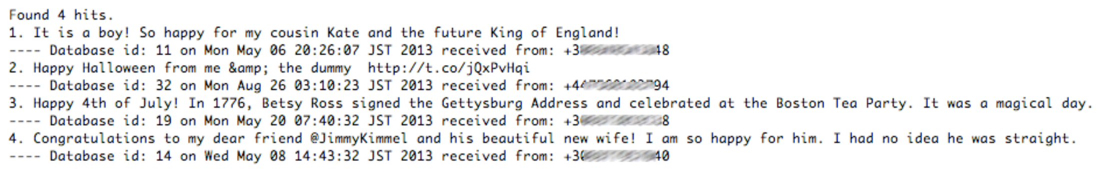


Figure 3: Timeline views highlighting communication between two entities (a) and messages sent by the person under investigation (b).

3.x.3 Searching for keywords

The index of the MySQL database created by the tool is helpful because we provide to the forensic analyst (who is not familiar with the SQL language) the capability to search the database in a way similar to a simple web search. The method of database indexing is more efficient because it provides answers faster than an SQL query. In large datasets (maybe with more interconnected databases) this feature could be critical in order to perform faster and produce more accurate searches. The next figure demonstrates how our tool depicts results obtained after searching the index for the word 'happy'.



```
Found 4 hits.
1. It is a boy! So happy for my cousin Kate and the future King of England!
---- Database id: 11 on Mon May 06 20:26:07 JST 2013 received from: +3[redacted]48
2. Happy Halloween from me &amp; the dummy  http://t.co/jQxPvHqi
---- Database id: 32 on Mon Aug 26 03:10:23 JST 2013 received from: +4[redacted]94
3. Happy 4th of July! In 1776, Betsy Ross signed the Gettysburg Address and celebrated at the Boston Tea Party. It was a magical day.
---- Database id: 19 on Mon May 20 07:40:32 JST 2013 received from: +3[redacted]8
4. Congratulations to my dear friend @JimmyKimmel and his beautiful new wife! I am so happy for him. I had no idea he was straight.
---- Database id: 14 on Wed May 08 14:43:32 JST 2013 received from: +3[redacted]40
```

Figure 4: Searching the index.

There were 4 hits containing the word 'happy' and the tool also returns details like the message id in the original SQLite database, the date the message was sent or received and the contact that communicated the specific message. This format is readable and contains (in a concise sentence) the basic information we could deduce examining the cells of the original database.

Note: This work will be presented at the Tenth Annual IFIP Working Group 11.9 International Conference on Digital Forensics on January 2014 (Vienna). It will be published by Springer Publications at 'Advances in Digital Forensics X'.

3.x.4 Implementation Decisions

We chose the Java Programming Language for the development of the specific tool. As we said in the previous topic, the Apache Lucene Core is written in Java so we decided to implement the tool using the safe environment the language provides. There are also well-tested drivers to provide links among our program, the MySQL and the SQLite databases.

We import the following files to our code:

```
import java.io.File;
import java.io.FileReader;
import java.io.LineNumberReader;
import java.io.StringReader;              // for file and string manipulation
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;                // to handle SQL statements
import java.util.ArrayList;
import java.util.Date;                   // to transform timestamps into human
readable state
import java.util.HashMap;
import java.util.HashSet;
import org.apache.lucene.analysis.Analyzer;  // to handle words with Lucene
import org.apache.lucene.analysis.TokenStream;
import org.apache.lucene.analysis.standard.StandardAnalyzer;
import org.apache.lucene.analysis.tokenattributes.CharTermAttribute;
import org.apache.lucene.util.Version;
import org.tartarus.snowball.ext.PorterStemmer;    // to apply Porter Stem
```

The tool performs the following distinct steps:

- First, we need a MySQL database running on the localhost. We provide the database credentials (username, password).
- Register the JDBC Driver and open a connection with the database.
- Create the TABLE that will hold our data (id, date, address, type, body, keywords, sentiment).

- Connect with Lucene (Version.LUCENE_44) and create internal data structures.
- Open the Analyzer and initialization files (lexica, emoticons list, positive and negative).
- Open a connection with the SQLite database found in the internal memory of the phone – device.
- Extract emoticons and stem the keywords from each message (found in the SQLite database).
- Assign emotional valence of each word and calculate the final sentiment score of each message.
- INSERT INTO the created TABLE (MySQL database) the content of all SQLite tuples and add the extracted keywords and the mood score.
- Close Analyzer and database Connections.

The module we presented will create our content that we can query using the index Lucene will create in the examiner's machine. In order create the index we need the following files.

```
import java.io.IOException;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.Statement;            // to handle the SQL connection
import org.apache.lucene.analysis.standard.StandardAnalyzer;
import org.apache.lucene.document.Document;
import org.apache.lucene.document.Field;
import org.apache.lucene.document.StringField;
import org.apache.lucene.document.TextField;
import org.apache.lucene.index.DirectoryReader;
import org.apache.lucene.index.IndexReader;
import org.apache.lucene.index.IndexWriter;
import org.apache.lucene.index.IndexWriterConfig;
import org.apache.lucene.queryparser.classic.ParseException;
import org.apache.lucene.queryparser.classic.QueryParser;
import org.apache.lucene.search.IndexSearcher;
import org.apache.lucene.search.Query;
import org.apache.lucene.search.ScoreDoc;
import org.apache.lucene.search.TopScoreDocCollector;
import org.apache.lucene.store.Directory;
import org.apache.lucene.store.RAMDirectory;
import org.apache.lucene.util.Version;            // Lucene indexing files
```

The basic steps of the indexing procedure are described below:

- Open an Analyzer and describe the MySQL database.
- Create the index and connect to the MySQL database (using the database credentials).
- Create a new Document and add StringFields to the index, looking at the fields of the database.
- Close the IndexWriter.
- Define a query and create a QueryParser to perform the query on the index.
- Search the index and collect the related information.
- Print the results in a concise and informative manner and close the reader.

Finally, we are adding the following jar files to the class path:
(These files can be found in the zip file from
http://www.apache.org/dyn/closer.cgi/lucene/java/4.6.0)

lucene-core-4.4.0.jar
lucene-analyzers-common-4.4.0.jar
sqlite-jdbc-3.7.15-M1.jar
mysql-connector-java-5.1.26-bin.jar
lucene-queries-4.4.0.jar
lucene-queryparser-4.4.0.jar

4 Social Network Forensics Component

This chapter presents the implementation details of the Social Network Forensics component. This component consists of a framework capable of downloading data from social networking sites, correlating accounts from diverse social networks using various heuristics, and visualizing the data in comprehensive forms (graphs, calendar of actions), helping forensic analysis with the extraction of valuable evidence.

4.1 Introduction

The term online social networks (OSNs) is commonly used for referring to services like Facebook, Twitter and Google+, narrowing the true extent of digital social networks. However, many more services can be construed as OSNs, as they reflect sets of people performing digital interactions between them. Such interactions are created through email exchanges, Skype calls, and a wide range of other every-day activities among individuals of a network. Thus, in the context of this work, the term social networks will refer to any online service that creates clusters of people with shared activities or communication, all of which can be sources of valuable information.

With the ever-increasing popularity and use of online social networks, these services are leveraged towards conducting nefarious activities and exhibiting offensive behaviour (e.g., cyber-bullying). Furthermore, even malicious individuals (not only cyber-criminals, but perpetrators of physical world crimes) have also adopted these technologies. This explosive growth rate has, basically, created the first digital generation consisting of people of all ages and backgrounds. People are creating their digital counterparts for interacting with other users, for both recreational and professional reasons, and disclose a vast amount of personal data in an attempt to utilize these new services to the fullest. As a connection in a social network is a representation of social interaction, it also indirectly shows a level of trust between different individuals, in terms of the data they are willing to share with each other. However, the lack of technical literacy among the majority of users has resulted in a naive approach, where the caution demonstrated in social interactions of the physical world has disappeared.

This behaviour has raised the concern of the research community in terms of user privacy and the wide range of threats users expose themselves to, ranging from identity theft to monetary loss. While the amount of personal information disclosed by users [32] or leaked by services [38, 39] is troubling, in certain cases it can prove to have a positive "side-effect". Law enforcement agencies have been able to solve criminal cases after extracting the digital footprints of users, as they contained clues that ultimately led to the discovery of the perpetrators.

Social forensics tools aim to facilitate the discovery of this digital "trail of breadcrumbs", and extract data that can guide criminal investigations towards uncovering valuable information. Such tools differentiate from "traditional" digital forensics tools, aiming to recover potential deleted files or carve artefacts from the volatile memory of a device. The very few existing tools that target social networks tend to be proprietary solutions which not all law enforcement agencies around the world can get access to. Our goal is to provide an extensive open source framework that will assist forensics analysts in this daunting task of collecting evidence.

We have designed and implemented our toolset with the following usage model in mind: the authorities seize the digital devices (be it desktop, laptop or just hard disk drives) of a suspect and wish to acquire and analyze any available information regarding suspect's online activities. The analysis of social network activities presents three major challenges:

1. Acquiring as much data as possible from the suspect's online accounts and relevant local artefacts.
2. Correlating contacts across diverse services
3. Visualizing this extensive collection of data.

Our modular framework contains components for handling all three tasks.

The core functionality of any forensics analysis tool is the extraction of user data. A series of crawling modules has been created, each designed for extracting data from a specific service. When available, public API of each service is leverged to collect available data. In the remaining cases, custom crawlers have been implemented for acquiring the data. The correlation of users across services is a very crucial, yet challenging, aspect of our framework. Correlation component follows a series of techniques for mapping user accounts from different services. Using a method we demonstrated in previous work [44], email addresses extracted from the suspect's accounts can be mapped to Facebook profiles, which are the core sources of information. A similar process conducted in Foursquare, utilizing the search functionality of the official API. Furthermore, we employ data from about.me, a social directory site where users create a profile page with links to their social accounts, to further improve our correlation results. Finally, we also use fuzzy matching techniques for matching user names and email handles collected from different services.

The datasets collected during the data extraction process contain a wide range of different types of information regarding online activities. Existing forensics-related visualization tools usually focus on the depiction of graph-related data. However, various visualization libraries exist, and can handle multiple types of data. As such, we build upon existing libraries and create a visualization framework, designed specifically for visualizing data representing user activities in online social networks and communication services. Furthermore, the massive amount of data necessitates the creation of dynamic viewpoints of varying granularity, that will assist analysts in surveying aggregated statistics, as well as focusing on specific users or interactions.

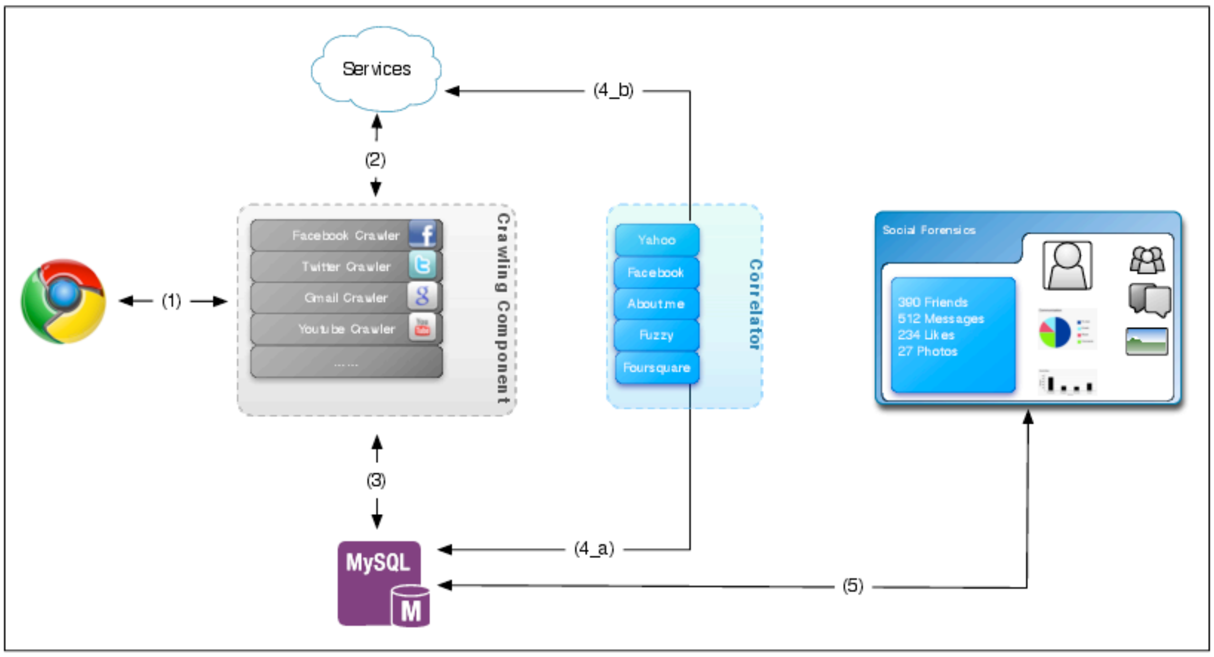


Figure 1: The architecture of the social forensics framework

4.2 Implementation

The core of our framework has been implemented in Python as a collection of components. We have designed it in a modular way in order to incorporate future social networks and services in our dataset. In this section, we provide a high-level overview of our system, describe the role of each component, and present technical details regarding the implementation of some of the components we have created. Figure 1 presents the architecture of our framework and the steps that comprise the whole procedure:

1. The data collection component uses stored session cookies and user credentials to log into the online services as the suspect.
2. Each crawling component extracts as much data possible from each service that the user has an account for.
3. All extracted data is saved into a MySQL database.
4. The account correlator component:
    a. Pulls the account information of the suspect's contacts from the database.
    b. Uses several techniques for correlating the accounts, some of which leverage online services.
5. The data visualization component fetches data from the database asynchronously and dynamically presents the viewpoints requested.

4.2.1 Usage Scenario

We implemented our framework with the following usage scenario in mind. The forensics analysis investigator has acquired the suspect's digital device (or hard disk and connected it to a computer) and connected it to the Internet, since the data extraction and correlation components must connect to online services. Components of our framework can also be leveraged by researchers for collecting, correlating, or inspecting data from social networking experiments collected from social networking experiments.

An important design aspect of our system, was to make its execution as simple as possible. Ideally, the analyst would need only execute a program and everything else would be done automatically. However, due to the requirement of authorizing the crawling modules that use public APIs with the social networks through *OAuth*, a small amount of manual intervention is needed. Specifically, after the system logs into a service, the investigator is prompted to authorize the crawling component for the suspect's profile.

After the authorization phase, everything else is completed automatically. The framework installs a MySQL database and creates a series of tables for storing all the information from the suspect's accounts. The libraries required by the crawling component, for example fbconsole [28] and Tweepy [17], are downloaded and installed automatically. The libraries for the visualization component are included within the web application.

4.2.2 Data collection components

Depending on the targeted service, the corresponding crawling component attempts to extract as much information as possible. In the case of online social services we leverage existing public APIs, if available. Otherwise we create custom crawlers for extracting the data. In this section we provide technical details for certain modules.

Log-in process. Our tool uses the credentials saved in the browser's password manager or existing session cookies, to log into the targeted services as the suspect. Alternatively, the analyst can manually add the suspect's credentials in a configuration file, when no other method of logging in is available for a service. The password managers of Chrome and Firefox utilize a SQLite database as their password storage. Some browsers, like Firefox, encrypt this database using a "master password". In contrast, Chrome does not employ any encryption mechanisms, thus, storing the credentials in plaintext. We implemented a custom password extractor that locates Chrome's SQLite password file in the filesystem, and extracts credentials belonging to social networks and relevant services. Browser session cookies are also stored in SQLite databases. Thus, the same process is followed to extract session cookies.

Facebook. Once logged in, a custom application is installed in the suspect's profile, so the data can be retrieved through Facebook's Graph API [29]. This application has access to any resource available in the profile. After installation, our system leverages the Facebook Query Language (FQL) to extract the data from the user profiles [27]. FQL provides an SQL-like interface for querying user data, and can evaluate multiple queries in a single API call through FQL multi-query requests. Queries are packed as a JSON-encoded dictionary and sent as a single request. The response includes a similar dictionary with the respective results.

Twitter. In order to use the Twitter API, similar steps are followed. An application that has full access to the profile data has to be installed in the suspect's profile. However, Twitter poses an extra overhead during the crawling phase, due to its rate-limiting policy. To avoid potential rate-limiting issues, we add 10-second delay between successive requests. Protected accounts (whose information is only available to selected followers) are collected with the highest priority. Next, we focus on accounts with small volumes of data, i.e., those with the smallest volume of tweets.

Google+. We utilize the official Google Plus API [9] for extracting the data. Through this API and the *OAuth* authorization method, we are able to extract all the public information from the users' profiles, and the contacts from public circles. The information also includes the name of the city where the user resides. We rely on the Google Geocoding API [11] for converting the city to a pair of geographical coordinates.

Foursquare. Our crawling component is built upon a Python wrapper [8] for the official Foursquare API [7]. After the *OAuth* authorization is completed and an authorization token is acquired, the crawler extracts the data through API calls that return the data formatted as *JSON* objects.

4.2.3 Account correlation component

This component has a very important role. As our goal is to collect data from variety of online services, we require a method for correlating the suspect's contacts across services. Several separate modules comprise the component, each leveraging a different service or technique. We can see an overview of the correlation process in Figure 2. The Yahoo module is executed once as its output is not affected by the outcome of the other modules. The remaining modules are executed in a loop, as each module might result in information that can be processed by other modules as well. Thus, the correlation process executes these modules in a round-robin fashion until none of the modules produce new information within an iteration.

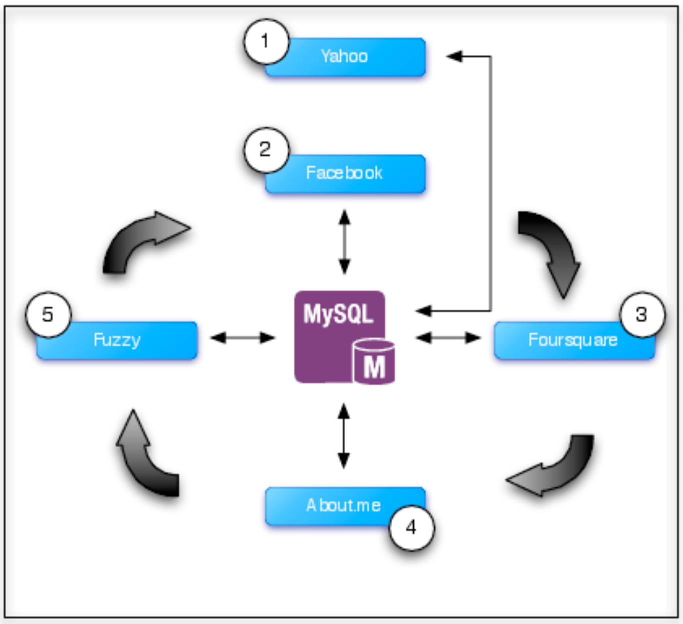


Figure 2: The account correlation process.

Yahoo. The goal of the first module is to extract the email addresses of the suspect's Facebook contacts. Even though Facebook apps are not allowed to obtain the email addresses of the user's contacts [4], there is a method to bypass that restriction.In particular, Yahoo mail allows one to export their Facebook contacts and add them to their email contact book. The output of this process is the Facebook profile names and the email addresses used to setup the profiles. While not all email addresses are returned, as users can change the visibility of their email address, by default the email is returned. In a small case study we conducted, we found that approximately 70% of the users tend to keep the default setting and their email addresses are imported.

Facebook. In previous work [44] we demonstrated how Facebook can be leveraged as an oracle for mapping a user's email address to his online account. A more efficient method for processing large numbers of email addresses was demonstrated in [21]. We follow this technique for mapping all email addresses we have found, that have correspondence with the suspect, to their Facebook accounts (if they have created one with that email address).

Again, while a user can change the privacy settings to be removed from such searches, it is enabled by default and not disabled by the average user.

Foursquare. The official API contains a call that searches for Foursquare accounts based on different types of information, thus, it can also be used as an oracle for correlating user accounts. Specifically, the API call takes as a parameter any one of the following pieces of information and returns the relevant Foursquare account (if it exists): Facebook ID, Twitter username, email address, name, phone number. Consequently, apart from locating a user's Foursquare account, we can also associate disjoint pieces of information we have collected from other services.

About.me. This site offers a platform for users, where they can create a personal page containing links to their profiles on popular social networking services. Using the names extracted in previous steps, we search *about.me* for profiles with the same name and extract the links to their profiles on social services. We then attempt to verify that the account belongs to the same user by comparing the account IDs to any we have correlated previously.

First we leverage the website's search functionality for locating the suspect's contacts that have an abo*ut.me* profile. As the search query results are dynamically rendered through Ajax requests, we scrape the results through PhantomJS [16], a headless webkit that also offers a Javascript API. After obtaining the user profiles, we extract the available links towards social network profiles. Each link to a specific <*network*> is accessible through a unique URL[1]. Similarly, other social directory sites could be used.

Fuzzy matching. Some of the services we extract data from don't provide the email addresses of the account's contacts, which would allow us to deterministically correlate user accounts across services. Furthermore, different email addresses may have been used for different services. To overcome this, we compare information collected from different services and match them based on similarity. While this method follows a ``fuzzy'' approach, we are able to obtain results, as users tend to reuse user names across services, or simple variations of them. For example, a user with Facebook profile under the name "John Doe'" might have an email address handle "*john_doe*", "*johndoe80*" etc. This module also creates synthetic email addresses using certain variations of the given user name (e.g., "*john_doe*",

[1] `http://about.me/content/<username>/<network>`

"*doe_john*") along with the most common email providers (namely "*windowslive.com*", "*gmail.com*", "*yahoo.com*", "*msn.com*", "*hotmail.com*") which are passed to the other modules.

User input. While the above methods yield results, nonetheless, certain accounts may not be correlated with others belonging to the same user. This could be due to users creating multiple accounts under completely different user names. As this correlation can provide invaluable information during the visual inspection of the data by analysts, our visualization component enables the manual correlation of accounts. Specifically, the user can correlate an account from one service with accounts from other services. That information is saved, and the dynamic perspectives will reflect the new associations. Similarly, the user can remove any erroneous correlations made during the automatic correlation procedure by the fuzzy matching module.

4.3 Data Collection

In this section we present a list of the services from which we collect user data, as well as a description of the types of information acquired. For every online social network, we also collect any information that is reachable for every one of the suspect's contacts.

Facebook. This is the main source of information, as it is the most popular online social network, and users tend to reveal a large amount of personal information on it. Our crawling component extracts any of the following information that exists:

- *Personal information* including current location, hometown, education and work information
- *Contact list*: apart from the list of the suspect's contacts, we also collect any custom lists and the contacts contained in each list.
- *Status updates* and any links contained.
- *Chat logs* along with timestamps for each message.
- *Comments* of the suspect on any type of activity, including those of friends.
- *Photos*: links to the photos and information regarding photo albums, photo timestamps and tagged users.
- *Videos* uploaded by the suspect, and videos he has been tagged in.
- *Check-ins:* the places the suspect has checked into, the timestamp and the data and coordinates of the place, along with tagged users.
- *Likes:* activities and articles the suspect has liked.
- *Shares:* pages the suspect has shared.
- *Fan pages* (also checks if the user is an administrator of the page).
- *Events* and the information of the users that participated.
- *Groups* the suspect is a member of, and the information of the other members.
- *Notifications* the suspect has received.
- *User notes*.

- *Contact information*: we also collect all of the aforementioned data from the contacts that is viewable through the suspect's account (e.g., a contact's chat messages are not viewable.)

Twitter. We first collect the account's information and contact list. That includes the accounts the suspect follows as well as those following the suspect. We also collect the suspect's tweets as well as any tweets re-tweeted, and all available metadata (e.g. timestamps, location).

Foursquare. We collect the suspect's check-ins along with the corresponding metadata. In particular, we collect the timestamp, the venue's name, *VenueID*, and coordinates of the venue. We also collect user's friends lists, and any links to their profiles on other networks. Unfortunately, due to limits set by Foursquare API and website, only the last 100 check-ins of the suspect's friends can be retrieved.

Skype. We first collect the list of contacts as well as their disclosed information (that may include information about location, gender and date of birth of a contact). Then we extract the history of chat logs and relevant metadata, as well as call history (and duration) and file exchanges. We also attempt to retrieve any exchanged files that are still located on the hard drive.

Gmail. We collect all emails exchanged with the suspect, and extract the email addresses and any names associated with those addresses. For each email we also collect the relevant metadata.

Google. We access the suspect's account in Google and extract the relevant information from Google calendar and Google Docs. Specifically, we collect all calendar entries (which may contain a location, a description, and other users attending), and download documents accessible (we also retrieve information about which other contacts have access to the documents).

Google+. We first collect the suspect's contacts contained in the various "circles" (i.e. contact groups), and the suspect's activities; posts, comments, shares, and "+1"s (similar to likes in Facebook). We extract publicly available data from the accounts of the contacts, as well as any accounts that have commented on the suspect's profile (even if they are not part of one of his circles).

Youtube. We first collect the suspect's information. Then we extract the history of watched videos, and channel subscriptions, playlists, uploaded videos and their comments and favorited videos.

Dropbox. We first locate the Dropbox folder, depending on the suspect's operating system, by retrieving the information from the application data. Then, by traversing the Dropbox directory tree, we extract all the files with their corresponding metadata. We also keep the application data that can be used for other aspects of forensic analysis [6].

5 Steganography Detection Component

5.1 Introduction

First, we present elements of the jpeglib library, which is a significant component of the steganalytic tools we developed. Then, there is an overview of the tools we created. The technical and scientific problems that occurred and the solutions we proposed are described in details. There is also a detailed presentation of the critical decisions we had to make during the implementation phase.

5.2 The jpeglib library

The main characteristics of the tools are accuracy in mathematical procedures and speed in various processes. The C programming language was chosen because it is fast and efficient when it uses mathematical functions. Another reason for this choice is the existence of a very popular library, which is written in C and is called jpeglib. The Independent JPEG Group provides this library and it is freely distributed through the Internet. The eighth version was used for this project and more specifically version 8c. Our involvement in the current project revealed that popular open source steganographic algorithms such as JSteg and Outguess use the same library. These arguments led us to the conclusion that jpeglib is the ideal base for our programs. The library is designed to work under any operating system. We decided to design *stegBennie* for Linux because of the free distribution of the operating system.

Jpeglib consists of various functions that helped us to build a reliable environment where our functions could safely work and provide accurate results. Basically, there are two types of functionality in this library. One part has to do with the methods that are responsible for the compression of a bit stream to a jpeg image. The other part undertakes the decompression procedures from a jpeg image to a bit stream. The basic steps of the decompression of a jpeg image are shown below.

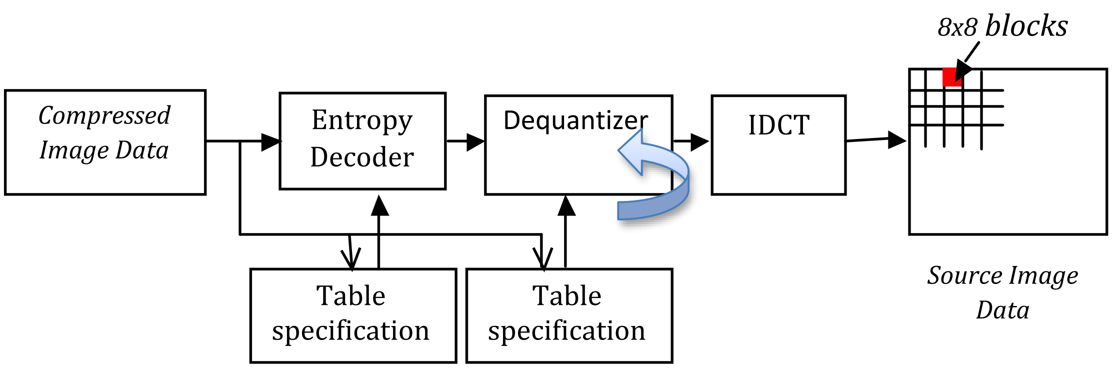


Figure 5: DCT-based decoding process

The blue arrow shows the phase of our interference with the jpeglib. Our decision was to leave the library do its excellent work until we reach to the part of the decompression procedure where we will be able to obtain the quantized DCT coefficients. The library supplies a function to the users for this cause which is called *jpeg_read_coefficients()*. After this step our functions will take care and manipulate the data that exist in the virtual memory. Before the completion of any task, the library will be responsible to clean up the memory by freeing any space that was using while one of our tools was running.

In more details, when *stegBennie* or *compBennie* starts running and during the whole process, jpeglib is responsible for the following tasks:

- Allocate and initialize a jpeg decompression object (although C is not an object oriented language, the developers of the library use object oriented logic).
- Set up normal jpeg error routines. (These routines will produce errors if the source file provides the program with e.g. a png image instead of a jpeg. Basically in this stage we set up the error handler of the library and we will be using it until we finish the decoding of the image.)
- Specify the data source (in our case the data source is always a file).
- Start decompression of image.
- Read file parameters (this will reach the metadata of the image).
- Read the quantized DCT coefficients.
- Access virtual memory.
- Finish decompression (we have to finish the decompression in a smooth fashion and this is done by the library itself).
- Destroy decompress object (clean up the space we used).

By building this environment with well tested and trustful functions we are able to safely extract, copy and manipulate the information that exist in the virtual memory the library creates in order to work efficiently with all image sizes. The README file of the library is a good guide if someone wants to familiarize herself with the standards of the coding style and the maze of procedures and objects that can be created by jpeglib. Also, the structure.txt and libjpeg.txt should be considered as a compulsory guidance for new users. We will not talk about any particular detail of this library here. Further study is left to the reader. We will now present an overview of the two systems we created and talk about their structure and design.

5.3 *StegBennie* and *compBennie*: An overview

For this project we used the standard quantization tables of the Independent JPEG Group (www.ijg.org). These tables can be easily deduced from a two-phase process:

- Define a quality factor QF (from 1 up to 100, 100 is the best quality) and calculate R, where R is a parameter estimated by the following equation:
  $$R = (QF < 50) ? 5000/QF : 200 - QF.$$
- Calculate each coefficient of the quantization matrix M[i] following the equation:
  $M[i] = (R * Q_{50}[i] + 50)/100$, where Q_{50} is the standard matrix.

This in fact, is the process we are following in our function *init_standards()* which initializes the standard arrays for our program. These matrices are been used by various programs such as GIMP and by devices like Android smartphones. The logic behind the construction of our tools is simple and it is described at the following figure.

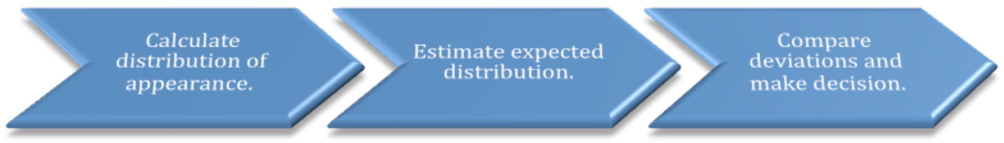

Calculate distribution of appearance.
Estimate expected distribution.
Compare deviations and make decision.

Figure 6: The basic logic structure of stegBennie and compBennie.

This figure says that the basic logic behind *stegBennie* and *compBennie* has to do with the calculation of the appearance of the significant digits of the quantized DCT coefficients of all the components of a colour jpeg image. Then we should estimate their expected distribution and finally compare the deviations between the expected and the calculated distributions. *StegBennie*, which is our main blind steganalytic tool, will use this information to decide if the image is suspicious or not.

As already mentioned, we should be able to predict the distributions of the first digits of the quantized DCT coefficients in order to compare them with the distributions that we will generate from the analysis of a colour image. We should be able to predict how a jpeg image behaves if we know its quality factor. The quality factors that were analyzed for this project are QF = 100, 90, 80, 75, 70, 60, 50. Hence, at this stage (as a proof of concept) we can only investigate jpeg images compressed with those quality factors. Therefore, if during a forensic task, one of our tools tries to analyze an image with a different quality factor than the above, the program exits with a graceful message. The approximation of the behaviour of a colour image was successfully done using *compBennie* and Matlab. It is further discussed at the following topic.

5.3.1 Analyzing the behaviour of a colour jpeg image with *compBennie*

When we are talking about the behaviour of a colour jpeg image we focus our interest on its quantized DCT coefficients and more specifically on the distributions of appearance of their first digits. We mimicked the methodology Fu et al. (2007) used at their research to bring new results that were considering both the luminance and chrominance components of the jpeg images. A new program was developed to help us to collect the distributions of the quantized DCT coefficients. Its name is *compBennie* and consists of two modules: collect and compare.

The standard header file *dirent.h* was used to read any folder. After performing the initialization of the output files and after the setting up of the error handler and the decompression object, *compBennie* reads the quantized DCT coefficients of each image. This information is stored in a virtual memory created by the jpeglib library. Our functions are responsible for the manipulation of the virtual memory. They are using jpeglib's function *access_virt_barray()* for this cause. The core module's functionality is been discussed in more details later. A flow chart will explain what the core module does. Basically, the outcome of its use is an array containing the frequency of the appearance of the first digits of the quantized DCT coefficients of the image (digits 1 to 9). After this step the program writes the distributions of each first digit for the whole folder in separate text files.

The same procedure continues within a loop until there is no other jpeg file in the folder. When the program completes successfully its task we will be provided with nine files containing the percentage of distribution of appearance of each digit of the coefficients for all the jpeg images that exist in the particular folder. It should be underlined that, before we perform this experiment, we should be absolutely sure that the images in the folder we will examine are compressed under the same quality factor. *We are trying to imitate a training scheme at this point and the data we collect should be accurate*. Also, the folder must contain only jpeg images in order to finish the processing without any disturbance. In any other case, the error handler of jpeglib will produce an error and cause program termination.

Apparently, the most important part of all the modes of *stegBennie* and *compBennie* is 'the core module'. It is responsible to perform all the calculations after we access the virtual memory. Its design is simple and is described by the following flow chart.

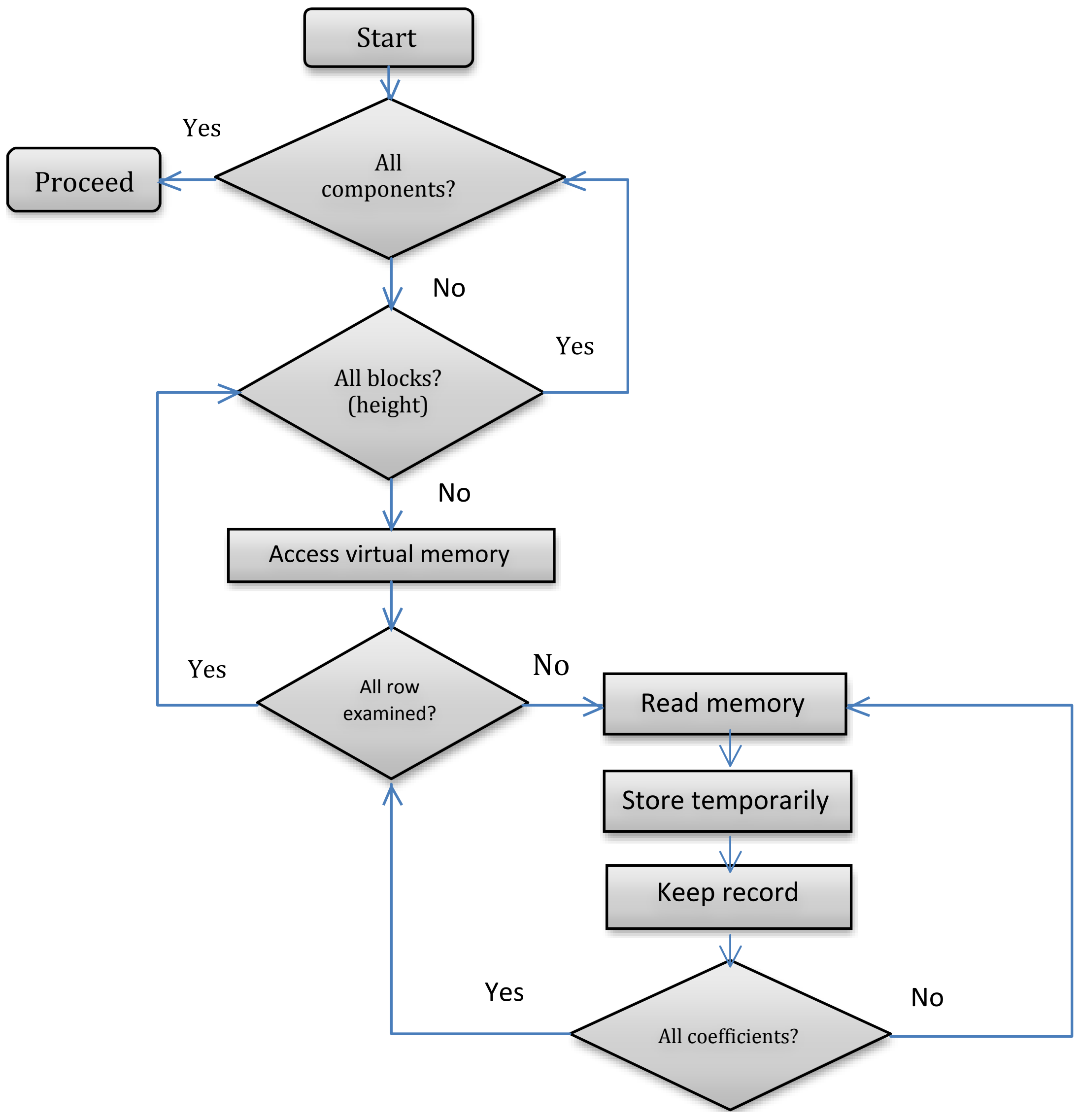


Figure 7: The 'core module'.

Its functionality is based on the way that the quantized coefficients are stored in the virtual memory. At a glance, there are big blocks of data, called MCUs, which contain several blocks of 8x8 integers. The above module is responsible to read all coefficients that exist in all DCT blocks in all MCUs. The latter are virtually stored in rows and columns and the library itself is responsible to provide non-interleaved MCUs and exclude any dummy DCT block that might be stored at the right and bottom side of the inspected image. More information and details can be found in structure.txt in jpeglib.

This is the reason that forces the core module to perform three loops. The first is for each component of the image (luminance and chrominance); the second loop is for the columns and the third for the rows. When we finally reach an 8x8 block of the quantized coefficients, we use a temporary array to copy and store them in a different place, where we can safely remove their first digit. This is been done by the *keep_record()* function. Its task is to extract the first digit of the coefficient and increase the variable that counts the appearance of that digit. If, for example, the coefficient is 235, its first digit is 2 and the counter that keeps the record for 2s will be increased by one. Also, we are counting the total coefficients we have examined. Hence, at the end of the program we will know the appearance rates of each first digit. Their distributions will be calculated as a division of the appearances by the total numbers counted.

5.3.2 UCID and JPEG

After the description of the inner functionality of *compBennie* we will discuss the methodology we used to create a model that can predict the distributions of the quantized DCT coefficients of any colour image. This can be feasible if we prove that equation $p(n) = N \cdot \log_{10}\left(1+\frac{1}{s+n^{q}}\right)$, x = 1, 2, ..., 9 (see D1 - 3.4.2) is still a reliable model that describes the probability of appearance of the first digits of the quantized DCT coefficients of a jpeg image, even if these were collected from all the components of the image. We used the second version of the UCID (Schaefer & Stich, (2003)) for this experiment which contains 1338 uncompressed tiff images and wrote a script that used Matlab's functions *imread* and *imwrite* to compress the images within seconds. Then we used *compBennie* (collect mode) to calculate the distributions of the first digits of the quantized DCT coefficients.

After this step we calculated the mean distributions for each digit. The mean distributions and Matlab's Curve Fitting Toolbox were used to estimate the goodness-of-fit of the generalized Benford's Law. To avoid the calculation of any complex values from Matlab we had to determine the boundaries of the parameters N, s, q. For this reason we defined that:

- - Inf < N < Inf
- 0 < q < 2
- - 0.8 < s < 1

The use of the curve fitting toolbox (Matlab) for all quality factors resulted in the conclusion that gBL is a perfect model which can describe the distributions of the appearance of the first digits of the quantized DCT coefficients of a colour jpeg image in a very satisfactory manner. The following figure illustrates the fitting of gBL (red colour) for the mean distributions of images compressed with quality factor QF = 90. (The distributions are symbolized by the blue dots.)

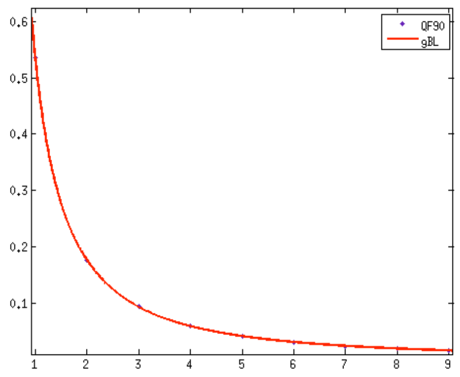


Figure 8: Goodness-of-fit of gBL for colour images compressed with QF90.

The fitting results for all the quality factors we examined are presented at D1 – 3.4.3. In that document we show the values of parameters N, q, s for each quality factor and there is also a column that represents the Sum of Squares Due to Error (SSE). The given table will help us calculate the expected distributions of the appearance of the quantized DCT coefficients. The idea behind this concept is that given the quantization table of the luminance of a jpeg image, we can decide the compression quality that was used during the encoding. Then, we can calculate the distributions of appearance of the coefficients and compare them with the expected distributions. We will be able to estimate the deviations between the distributions (current and expected) and decide if the jpeg image is suspicious or not.

If, for example, the quality factor of the image is 80, the expected distributions will be calculated by the Generalized Benford's Law equation (D1 – 3.4.2). N will be substituted by 1.344, q = 1.685 and s = - 0.376.

The next table will give the expected distributions for this case.

First digits	*Expected distributions (%)*
1	55.83
2	17.61
3	9.01
4	5.58
5	3.85
6	2.83
7	2.19
8	1.75
9	1.43

Figure 9: The expected distributions of first digits when DF = 80.

Calculations like the above will take place while the program is running and are closely connected to the quantization tables that were used during compression. These distributions will give us a clue about how violent an embedding algorithm can be, when it performs steganography.

5.4 The classification module

At this stage of the forensic process we are aware of the expected distributions of the significant digits of the quantized DCT coefficients. The next task we have to accomplish is to measure the impact of steganography on these distributions. We can use the results from this research to develop a function that will be responsible for making decisions. It will be able to state if a jpeg image seems suspicious or not. The needs of the particular step of the project were assisted by an extension of *compBennie*; the compare module.

5.4.1 The compare module

The design of this module shares the same logic with the collect module of *compBennie*. Our primary goal is to successfully and reliably manipulate the data provided by jpeglib and introduce a novel method for image steganalysis using the gBL. For this module the program will use ten files for writing. One file, which is called data.txt, will contain the current and expected distributions of the first digits of the coefficients and also the deviations that occur between them. The other nine files contain the current distributions of each digit for all the inspected images; just like the collect module. We should underline that these files will be stored in a folder the user indicates through the command line. Then, *compBennie* starts the decompression, reads the header of the image and looks for the quantization table of luminance. The next phase is the calculation of the distributions from the core module. After that, the program will compare the quality factor that read at the previous stage and will seek for a match within its known tables. If the table is known (which means that is the quantization table for QF = 100, 90, 80, 75, 70, 60, 50), it will proceed to the next phase, which comprises the calculation of deviations, the storing of data into the ten files and the cleaning of memory from the garbage it produced. If *compBennie* deals with an unknown quantization table, a message will be printed so the user can be informed and no further action will be taken for this image. This procedure will continue until all images will be examined.

CompBennie proved to be the most valuable ally to our attempt to decide which factors we should consider for developing the function that determines whether an image is suspicious or not. We also used it when we had to calculate the deviations of the distributions of the first digits of images that contained hidden data. We did that because it was a compulsory step for our research; we should estimate the damage that various steganographic algorithms cause to the distributions of the first digits of the quantized coefficients. For testing reasons we used free distributed tools; JPHide, Outguess and Vsl.

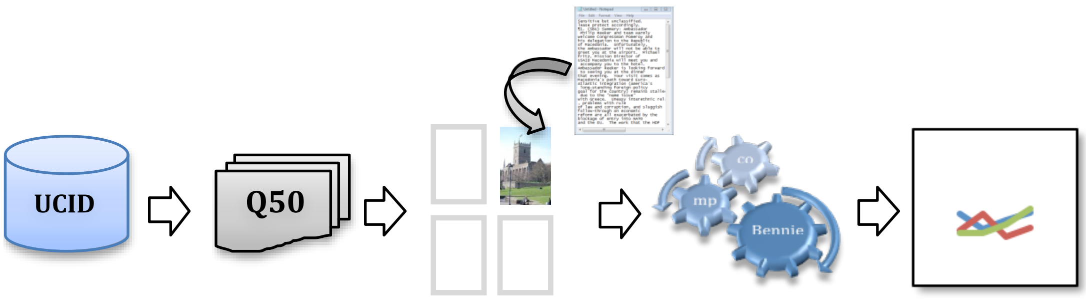
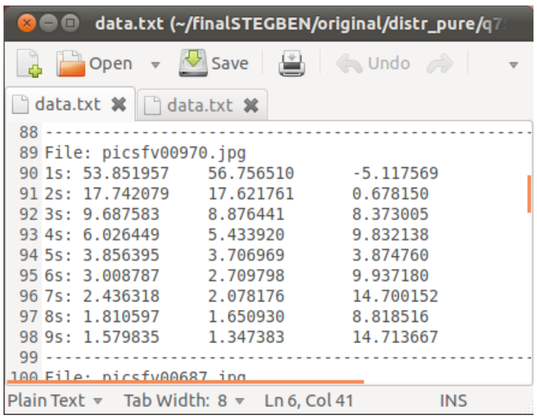
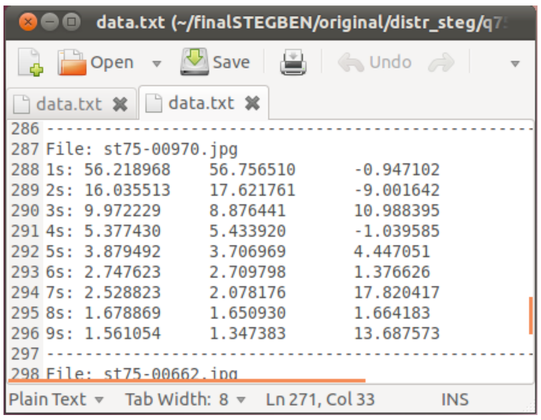


Figure 10: The methodology we used to conduct our experiments.

The figure above follows symbolically the sequence of the steps we made so far and the work that we will now discuss in more details. From the UCID dataset we created seven folders of jpeg files. For this part of the project we randomly chose images from the folders, which (each of them) contain 1338 jpeg files. Various messages were hidden in these images using the three algorithms we have already mentioned. Then, *compBennie* gathered all the information we needed. We used the compare module of *compBennie*. The result of this process was the production of another text file (data.txt), which describes the deviations of the distributions of the first digits of the quantized DCT coefficients for each stego carrier. By doing this, we will have a clear picture of any consequences that these algorithms cause to the distributions of the first digits. The next figures show the output of *compBennie* for a pure jpeg image and the output for the same image, which contains a hidden text file embedded by JPHSWIN, respectively.



```
--------------------------------------------------------
File: picsfv00970.jpg
1s: 53.851957     56.756510       -5.117569
2s: 17.742079     17.621761       0.678150
3s: 9.687583      8.876441        8.373005
4s: 6.026449      5.433920        9.832138
5s: 3.856395      3.706969        3.874760
6s: 3.008787      2.709798        9.937180
7s: 2.436318      2.078176        14.700152
8s: 1.810597      1.650930        8.818516
9s: 1.579835      1.347383        14.713667
--------------------------------------------------------
File: picsfv00687.jpg
```



```
--------------------------------------------------------
File: st75-00970.jpg
1s: 56.218968     56.756510       -0.947102
2s: 16.035513     17.621761       -9.001642
3s: 9.972229      8.876441        10.988395
4s: 5.377430      5.433920        -1.039585
5s: 3.879492      3.706969        4.447051
6s: 2.747623      2.709798        1.376626
7s: 2.528823      2.078176        17.820417
8s: 1.678869      1.650930        1.664183
9s: 1.561054      1.347383        13.687573
--------------------------------------------------------
File: st75-00662.jpg
```

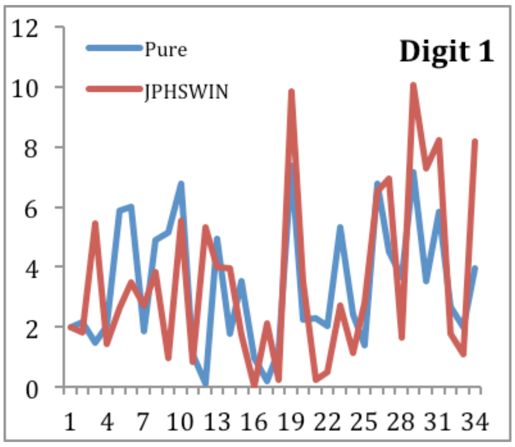
Figure 11: CompBennie: Output for pure image (left) and stego carrier (right).

From the comparison of the third columns of the above figures, it is obvious that the stego carrier causes some disturbances to the distributions of the first digits. Digits 1, 2, 4, 6 and 8 changed their probability of appearance when the message was embedded. To be more precise, all the distributions are altered but the deviations of the five digits we mentioned before seem to be more violent.

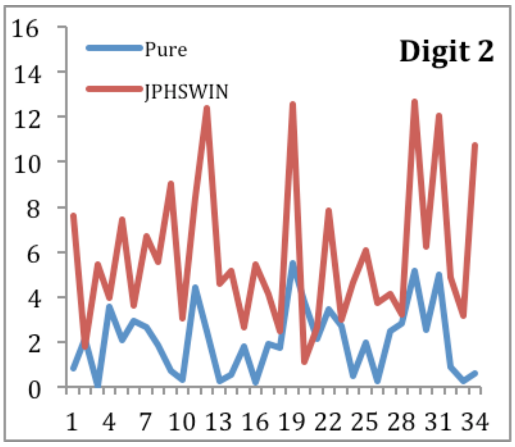
5.5 The final decision

At this stage of the project we tried to verge on the issue of finding a reliable indicator that could safely reveal the suspicious image. We focus our interest on the deviations of the distributions of the first digits of the quantized coefficients of pure images and stego carriers. The stego carriers are the same jpeg images but they contain messages embedded by JPHSWIN.

We carefully examined about 480 images compressed with different quality factors. The following figures illustrate the deviations of the distributions of the first digits of images compressed with quality factor 75 (more information about all the digits can be found at D1 – 3.4.3 – Fig. 10). The blue line indicates the deviations that emerged from the inspection of pure images and the red line indicates the deviations for the same images after applying steganography on them.



(a)



(b)

Figure 12: Deviations in distributions of qDCT coefficients.

The horizontal axis of the preceding figures represents the images we examined and the vertical axis states the percentage (%) of the deviations of the distributions of the examined digit. The overview we got by examining the figures we formed from images that were compressed by various quality factors was similar to what we can see on figure 21.

A first attempt to analyze the data we collected and the differences between the deviations of the distributions of the pure images and the stego carriers showed that often the disturbances were not very crucial. But in most of the cases there were some differences between the deviation of an image and the deviation of the stego carrier that were larger than 5%. We counted the number of the digits that present this difference for all images and concentrated the results into the next table.

QF	***1***	***2***	***3***	***4***	***5***	***6***	***7***	***8***	***9***
50	8.82	50.0	5.88	52.94	14.71	47.06	20.59	52.94	38.24
60	17.65	55.88	11.76	55.88	29.41	44.12	11.76	32.35	17.65
70	8.57	55.88	14,29	61.76	17.65	23.53	5.88	50.0	17.65
75	2.86	29.41	11.76	47.06	2.86	17.65	14.71	38.24	8.82
80	0	47.06	20.59	32.35	5.88	14.71	8.82	35.29	8.82
90	0	17.65	35.29	20.59	5.88	8.82	14.71	17.65	5.88

Figure 13: Percentage of digits with differences larger than 5%.

The contents represent the percentage (%) of images whose difference of deviations was greater than 5%. We can see that the digits with the roughest alteration of their deviations are digits 2, 4, 6 and 8. This is an indication that persuades us to focus our interest on these particular digits. Furthermore, previous figure (b) reveals a characteristic of digit 2 that no other digit seems to have.

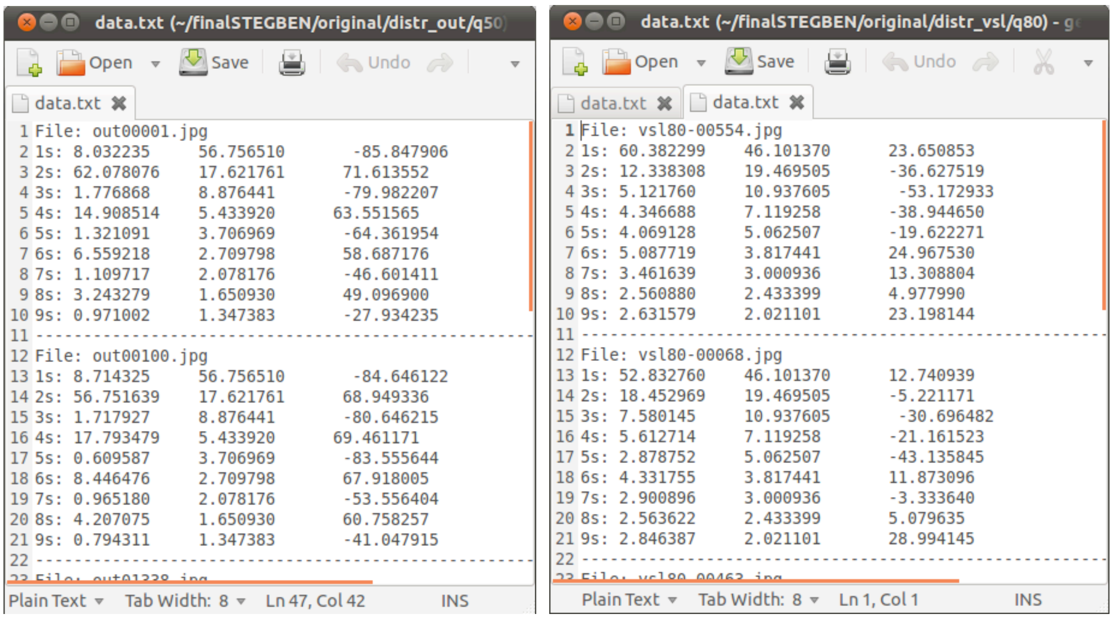
When we examined pure images the deviations of digit 2 were very stable. The range of these deviations was quite convenient and usually varied from 0 to 3 or 4 per cent. Except from that, the deviations of digit 2 after the embedding of a message on the same images behaved in a similar fashion, but this time the deviations exceed the threshold of 4 per cent. We cannot see the same attitude at digit 1 for example. Here, the deviations are within a small range but we can see that the two lines have not the same behaviour compared to the two lines of figure (b). In figure (b) we can see that the blue line is almost always below the red line. Thus, in most of the cases, we expect that an image that contains a hidden message will present deviations higher than a certain threshold T. In the specific example the threshold could be T = 3. It becomes more interesting if we say that the thresholds for the quality factor we examined vary from 3 to 4. It seems that only digit 2 presents this stable behavior.

Taking these findings into consideration we concluded that the most stable and reliable indicator for a suspicious image to be revealed is the deviation of digit 2. If this deviation exceeds a specific threshold, which depends on the quality factor of the compression of the examined image, we can conclude that the image is suspicious. We approximated these values statistically for each compression quality and presented them in D1. (See Figure 11 - D1 – 3.4.3).

5.5.1 Outguess and Vsl

As already mentioned in previous paragraphs the same experiments were performed to jpeg images using Outguess and Vsl as the embedding algorithms. We analyzed the data using the same methodology and discovered that the impact of steganography on the distributions of the first digits was dramatic, as it can be seen in next figure. We also confirmed that the deviations of digit 2 were smooth and the thresholds of table 7 were sufficient and capable to detect a suspicious jpeg image.

A closer look to the effects of the application of steganography with Outguess and Vsl revealed that both algorithms change the quantization tables of the images when they embed a message into their internal structure. Outguess always use the quality factor of 75 and Vsl uses the table Q_{100}. As a consequence, every time *compBennie* and *stegBennie* investigate stego carriers made by Outguess or Vsl, they consider that the quality factors are 75 and 100, respectively. This fact causes the expected distributions of the inspected images to be significantly different than the calculated. For example, if the original image had a quality factor of 60 and we use Vsl to hide data in it, the original expected distribution of digit 2 would be 17.515325% but the expected distribution that our tools consider is 19.469505% because they read in the header of the image that the quality factor is 100. This usually results in large differences between the current and expected distributions and sometimes these deviations can be critically major (see the following figure).



```
File: out00001.jpg
1s: 8.032235      56.756510        -85.847906
2s: 62.078076     17.621761       71.613552
3s: 1.776868      8.876441        -79.982207
4s: 14.908514     5.433920       63.551565
5s: 1.321091      3.706969        -64.361954
6s: 6.559218      2.709798        58.687176
7s: 1.109717      2.078176        -46.601411
8s: 3.243279      1.650930        49.096900
9s: 0.971002      1.347383        -27.934235
-------------------------------------------------------
File: out00100.jpg
1s: 8.714325      56.756510        -84.646122
2s: 56.751639     17.621761       68.949336
3s: 1.717927      8.876441        -80.646215
4s: 17.793479     5.433920       69.461171
5s: 0.609587      3.706969        -83.555644
6s: 8.446476      2.709798        67.918005
7s: 0.965180      2.078176        -53.556404
8s: 4.207075      1.650930        60.758257
9s: 0.794311      1.347383        -41.047915
-------------------------------------------------------
```

```
File: vsl80-00554.jpg
1s: 60.382299     46.101370       23.650853
2s: 12.338308     19.469505       -36.627519
3s: 5.121760      10.937605        -53.172933
4s: 4.346688      7.119258        -38.944650
5s: 4.069128      5.062507        -19.622271
6s: 5.087719      3.817441        24.967530
7s: 3.461639      3.000936        13.308804
8s: 2.560880      2.433399        4.977990
9s: 2.631579      2.021101        23.198144
-------------------------------------------------------
File: vsl80-00068.jpg
1s: 52.832760     46.101370       12.740939
2s: 18.452969     19.469505       -5.221171
3s: 7.580145      10.937605        -30.696482
4s: 5.612714      7.119258        -21.161523
5s: 2.878752      5.062507        -43.135845
6s: 4.331755      3.817441        11.873096
7s: 2.900896      3.000936        -3.333640
8s: 2.563622      2.433399        5.079635
9s: 2.846387      2.021101        28.994145
-------------------------------------------------------
```

Figure 14: Deviations caused from Outguess (left) and Vsl (right).

Our research revealed that Outguess leaves the quantization table of quality 75 as a fingerprint or signature. The same goes for Vsl which turns the quality factor of the stego carriers to 100. We used these fingerprints when we built the decision making function of our programs. If we try to investigate an image, which has a quality factor of 75 or 100 and the deviation of digit 2 is really large, we can deduce that Outguess or Vsl was used, respectively.

5.6 *StegBennie*

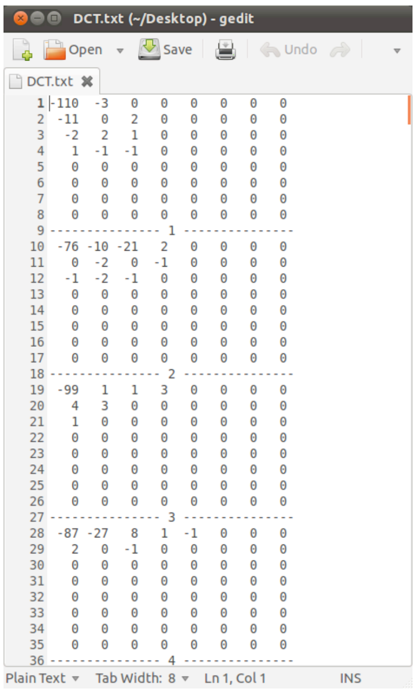
The study on the behaviour of the first digits of the quantized DCT coefficients of colour jpeg images and the analysis of the data we gathered from their distributions and deviations resulted in the development of our universal steganalytic tool which we called *stegBennie*. This tool uses the characteristics of the distributions of digit 2 and it is a new approach to the problem of steganalysis of colour jpeg images. *StegBennie* will apply a statistical attack on a jpeg image using the generalized Benford's law and will estimate whether it is a suspicious image or not.

The design of *stegBennie* should now look familiar. We basically extended the functionality of *compBennie*, which is the program we used to collect information about the distributions and deviations of the first digits. *StegBennie* has three modes; full, file, folder.

The file mode expects from the user to input an image. The outcome will be a short analysis of the image displaying the quality factor, the counted digits, the distributions (current and expected) of first digits and the percentage of their differences. Finally, *stegBennie* will decide if the inspected image is suspicious using the facts we presented in this chapter.

Small variations of the basic logic of the program are introduced by the full mode and the folder option. The former does the same job with the file option but it also creates a text file that contains a list of the examined DCT blocks.

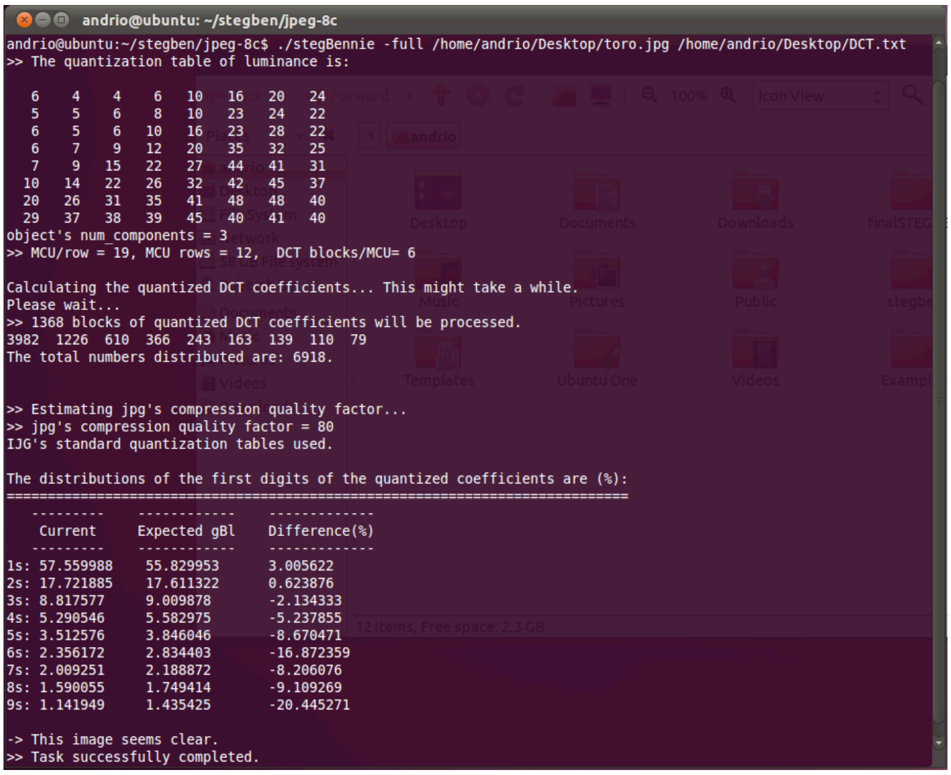
The next figure shows a screenshot of such a file where the DCT coefficients can be clearly seen.



```
-110  -3   0   0   0   0   0   0
 -11   0   2   0   0   0   0   0
  -2   2   1   0   0   0   0   0
   1  -1  -1   0   0   0   0   0
   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0
--------------- 1 ---------------
 -76 -10 -21   2   0   0   0   0
   0  -2   0  -1   0   0   0   0
  -1  -2  -1   0   0   0   0   0
   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0
--------------- 2 ---------------
 -99   1   1   3   0   0   0   0
   4   3   0   0   0   0   0   0
   1   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0
--------------- 3 ---------------
 -87 -27   8   1  -1   0   0   0
   2   0  -1   0   0   0   0   0
   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0
--------------- 4 ---------------
```

Figure 15: A screenshot of the examined DCT blocks.

We also provide a screenshot of the main output of the "full' and "file" options.



```
andrio@ubuntu:~/stegben/jpeg-8c$ ./stegBennie -full /home/andrio/Desktop/toro.jpg /home/andrio/Desktop/DCT.txt
>> The quantization table of luminance is:

   6    4    4    6   10   16   20   24
   5    5    6    8   10   23   24   22
   6    5    6   10   16   23   28   22
   6    7    9   12   20   35   32   25
   7    9   15   22   27   44   41   31
  10   14   22   26   32   42   45   37
  20   26   31   35   41   48   48   40
  29   37   38   39   45   40   41   40
object's num_components = 3
>> MCU/row = 19, MCU rows = 12,  DCT blocks/MCU= 6

Calculating the quantized DCT coefficients... This might take a while.
Please wait...
>> 1368 blocks of quantized DCT coefficients will be processed.
3982  1226  610  366  243  163  139  110  79
The total numbers distributed are: 6918.


>> Estimating jpg's compression quality factor...
>> jpg's compression quality factor = 80
IJG's standard quantization tables used.

The distributions of the first digits of the quantized coefficients are (%):
============================================================================
   ---------    ------------    -------------
    Current     Expected gBl    Difference(%)
   ---------    ------------    -------------
1s: 57.559988    55.829953      3.005622
2s: 17.721885    17.611322      0.623876
3s: 8.817577     9.009878       -2.134333
4s: 5.290546     5.582975       -5.237855
5s: 3.512576     3.846046       -8.670471
6s: 2.356172     2.834403       -16.872359
7s: 2.009251     2.188872       -8.206076
8s: 1.590055     1.749414       -9.109269
9s: 1.141949     1.435425       -20.445271

-> This image seems clear.
>> Task successfully completed.
```

Figure 16: The basic output of stegBennie.

The folder option is most powerful and useful among the others. This feature of *stegBennie* will provide the opportunity to examine a folder, which consists of jpeg images. The steganalyst must be sure that only jpeg images are contained in the examined folder in order to successfully complete the task of folder investigation. In any other case, the error handler of jpeglib will produce an error and the report for the contents of the folder will not be completed. The report will present *stegBennie*'s judgment about the behaviour of the first digits of each image in the folder. It will also inform the user about any image that cannot be examined because of its unknown quantization table.

The following figure is an outline of the basic parts of the 'file' mode of *stegBennie*. The basic difference between this and the flow charts we have already seen is the *make_decision()* function. The logic, as it can be seen, is exactly the same with the logic we used earlier to develop *compBennie*. 'Full' and 'file' modes do not need to open any folder and this is why the design looks easier to read. 'Folder' mode's flow chart seems quite identical to the 'compare' module of *compBennie* except that there is a *make_decision()* function at the end of the first loop before the program writes into files about the decision that *stegBennie* made. For this reason we will only provide the flowchart of the 'file' mode of *stegBennie* (see next figure).

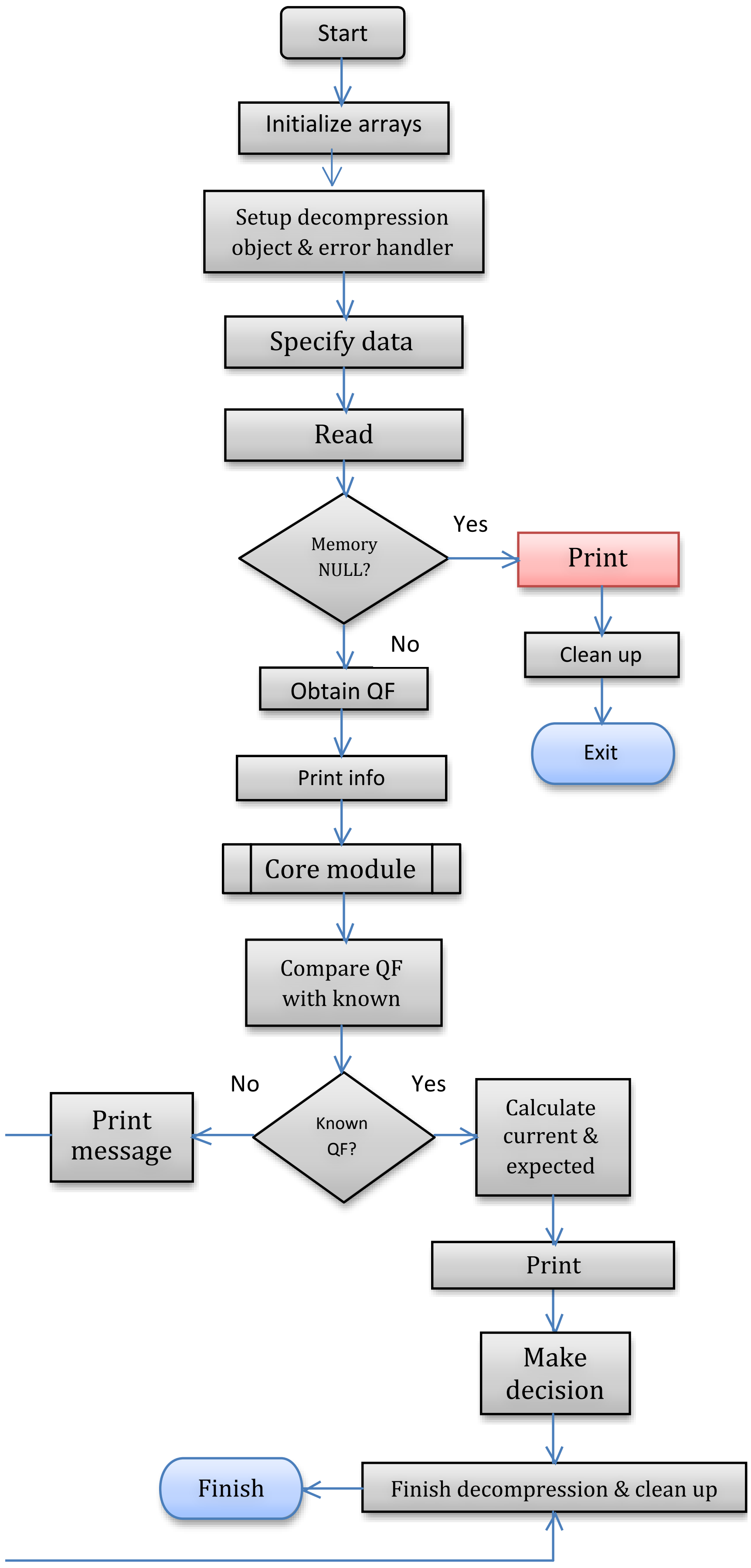

Start
Initialize arrays
Setup decompression object & error handler
Specify data
Read
Memory NULL?
Yes
Print
Clean up
Exit
No
Obtain QF
Print info
Core module
Compare QF with known
No
Yes
Known QF?
Print message
Calculate current & expected
Print
Make decision
Finish
Finish decompression & clean up

Figure 17: File module of stegBennie.

We will discuss briefly the design of the file module. First, we initialize the arrays we will use; a 1x9 array that counts the appearance of each digit and an 8x8 block, which is used as a temporary storage array. Then, the functions of jpeglib prepare a stable environment where we can calculate the distributions of the first digits and their deviations from the expected. We have to set up the error routines with *jpeg_std_error()* and then initialize the decompression object with *jpeg_create_decompress()*. The next step is to specify the file we want to decompress and read the metadata of the image; these tasks are done by *jpeg_stdio_src()* and *jpeg_read_header()*, respectively. Now, the system is ready to read the quantized DCT coefficients by using the *jpeg_read_coefficients()* function. Note that the functions of jpeglib always start with *jpeg*.

If all coefficients have been read successfully, *stegBennie* will look for the quantization table of the luminance and will print some information about the quantity of MCU blocks of the image. After this step, the core module will calculate the number of the first digits of the blocks. (The only difference between the file and the full module is that, at the current stage, the core module will have to do a printing task when it reads the coefficients of each block). Then, *stegBennie* will calculate the quantization tables for the known quality factors using *init_standards()*. This step was made because we wanted the program to be expandable and be able to work with more quality factors at the future.

The final phase of the process is the comparison of the quality table of the image with our virtual database of tables using *compare_quality()*. If this array is a perfect match with an entry in the database, we can proceed. Otherwise, the program informs the user that it is not able to decide for this image and exits. If *stegBennie* examines an image with a known quality factor it will get the expected and the current distributions of the digits and, at the end, it will make a final decision about the image. This step involves three functions; *gbl_distribution()*, *compare_distr()* and *make_decision()*.

Every time the tool exits or when it successfully completes a task, it will free the memory that was used, close any opened files and destroy the decompression object. In the folder module of *stegBennie* the decompression object will be destroyed only when all the images that the folder contains have been examined. This is a feature that the decompression object provides and is extremely powerful because it gives the opportunity to reuse it for multiple decompressions and save a lot of valuable time while the program is running.

It has been done clear that the use of the core module is vital for *stegBennie*. The following example gives an overview of the calculations it performs and its simple logic. First of all there are some basic arrays that have been initialized to zero as we said previously; they are called *distribution* and *curr_block*. The following figure shows their initial state before the examination of the first block of coefficients. We assume that block D (next figure), which holds quantized DCT coefficients, is the first block in virtual memory that will be examined by *stegBennie*. After the reading of block D is completed, *stegBennie* copies the 8x8 block into a temporary array which is called *curr_block*. We made this choice because it seems safer to manipulate our own array than the original from the virtual memory.

distribution:	0	0	0	0	0	0	0	0	0

$$curr\_block = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \quad D = \begin{bmatrix} -75 & 20 & 12 & 5 & 1 & 1 & 0 & 0 \\ -14 & -18 & -7 & -4 & -1 & 0 & 0 & 0 \\ 5 & 13 & 7 & 2 & 1 & 0 & 0 & 0 \\ -7 & -6 & -4 & -2 & 0 & 0 & 0 & 0 \\ 4 & 4 & 2 & 0 & 0 & 0 & 0 & 0 \\ -2 & -2 & -1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Figure 18: Basic data types.

The examination of *curr_block* (which now is identical to D) starts with the top left coefficient. This is the DC coefficient and it is ignored. We agreed on this exclusion following the logic of the previous work of Fu et al. (2007). A possible explanation for this exclusion from the gathering of first digits is that, if we look the DCT blocks of an image, it is quite common to see identical blocks, which contain only the DC coefficient. This might change the distributions drastically. To continue with the example, next digit in row is 20. An integer division by 10 will give 2 as the first digit. *StegBennie* will increase by one the variable that counts the number of the examined coefficients and the second column in the *distribution* array. The tasks of extracting the first digit and distributing in the *distribution* array are done by *calculate()* and *distribute()* functions, respectively. The next number in row is number 12. The first digit is 1, the counter and the first column of *distribution* array will be increased by one and so on, until every coefficient will be examined. The algorithm skips zeros and negative numbers because Benford's law stands for natural numbers only. Since all AC coefficients of the block have been examined, the *curr_block* array is set to zero and the same procedure will be repeated for all DCT blocks. The *distribution* and the counter stay untouched because we need them for the calculation of the percentage of appearance of each first digit.

The *make_decision()* function depends on the quality factor of the examined image. Its first concern is to set the threshold T that will use to decide if the deviation of distribution of digit 2 is greater than T. If deviation is greater than T, the image will be flagged as suspicious. If the deviation is greater that a specific limit and the quality factor of the compression is 75 or 100, *stegBennie* will inform that Outguess or Vsl was probably used to hide data in this image. Sorting the deviations of digit 2 of the pure images and calculating the maximum value that occurred defined the limit. This kind of value became the limit for each quality factor. Finally, if *stegBennie* deals with images with the highest quality factor and the deviation of digit 2 is less than the limit, the output will inform the user that the decision might be inaccurate.

This paragraph completes the detailed description of the work we did in order to extend the previous results that could predict the behaviour of grey scale jpeg images and develop a program, which is able to detect a stego carrier. The tools we developed should be easy to use and should need only an input path and an output path. The completion of this project had one major prerequisite; to study the behaviour of colour jpeg images and especially the distributions of the first digits of the quantized DCT coefficients. We should also confirm the validity of the gBL and propose a method that could distinguish a suspicious image.

5.7 Details on Software Design

The following figure show the dependencies of the files we created.

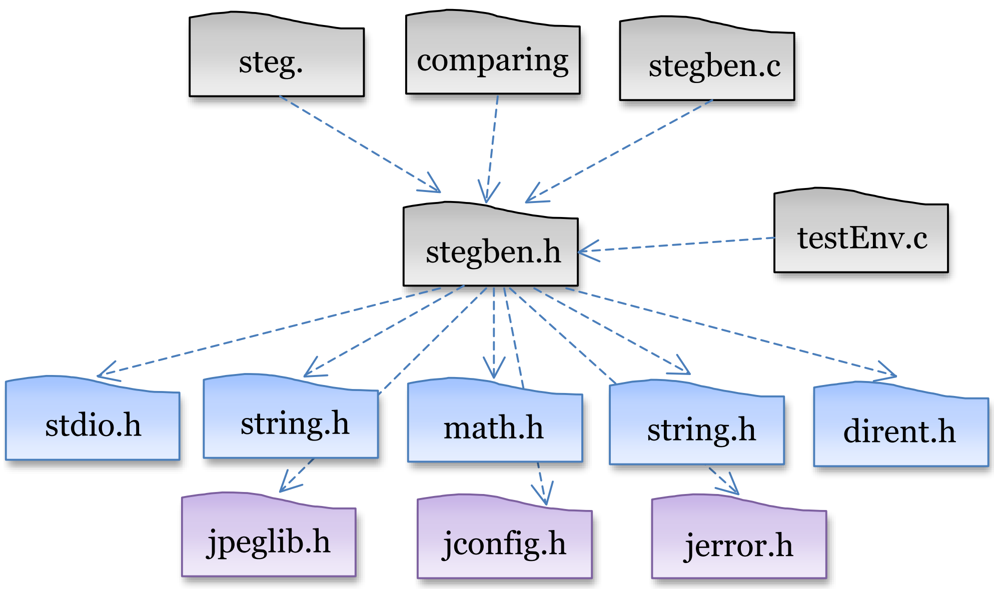


Figure 19: Dependencies of files with standard and library header files.

We did not use the whole jpeglib but only the files we needed. We created only four major files for this project and one for testing purposes (*testEnv.c*). *Stegben.h* is the header file for our tools, *steg.c* contains the main function for *stegBennie*, *comparing.c* is the main function for *compBennie* and *stegben.c* includes all the functions we had to write during the development phase. The following figure describes the functions in *stegben.c* that are been used by *steg.c* and *comparing.c*. Also there is a list with the functions of *stegben.c* and the functions we borrowed from the library.

steg.c
intro_steg()
usage_info_steg()
full_module()
file_module()
folder_module()

comparing.c
intro_comp()
usage_info_comp()
collect_module()
compare_module()

Jpeglib
jpeg_std_error()
jpeg_create_decompress()
jpeg_stdio_src()
jpeg_read_header()
jpeg_read_coefficients()
access_virt_barray()
jpeg_finish_decompress()
jpeg_destroy_decompress()

stegben.c
alloc_vector()
alloc_block()
alloc_dvector()
calculate()
distribute()
keep_record()
init_benford()
compare_distr()
compare_distr_dir()
gbl_distribution()
compare_quality()
compare_quality_silence()
compare_quality_dir()
init_standards()
print_info()
print_info_steg()
print_info_comp()
intro_steg()
intro_comp()
usage_info_comp()
usage_info_steg()
make_decision()
make_decision_folder()
collect_module()
compare_module()
full_module()
file_module()
folder_module()
test_environment()

Figure 20: Description of function use.

5.7.1 Implementation and testing

During developing stages the waterfall software development model was used. The following figures present schematically an overview of the lifecycle of the project and the phases of the software development.

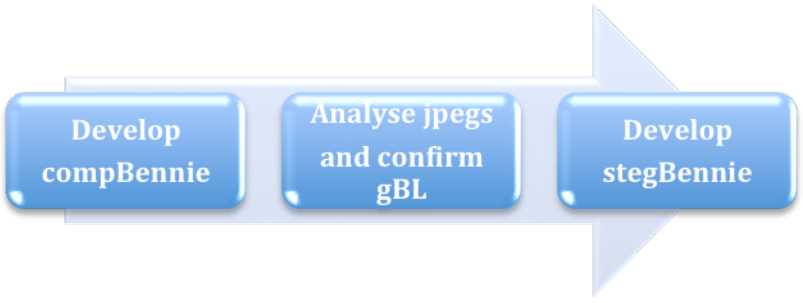


Figure 21: Lifecycle of the project.

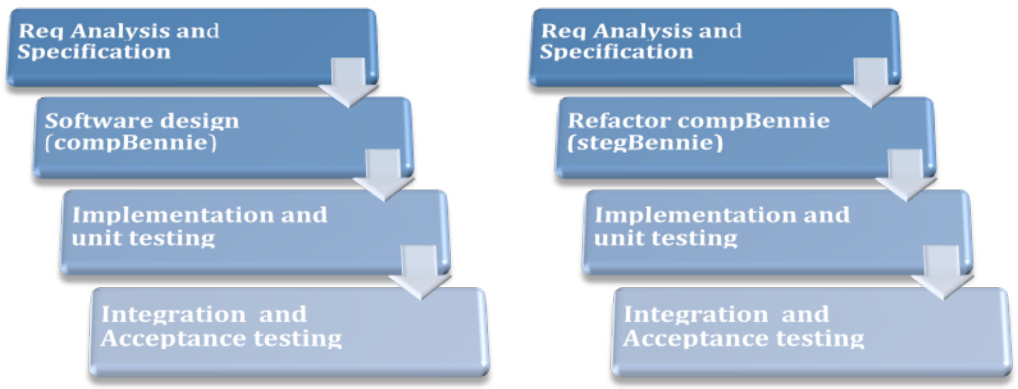

Req Analysis and Specification
Software design (compBennie)
Implementation and unit testing
Integration and Acceptance testing
Req Analysis and Specification
Refactor compBennie (stegBennie)
Implementation and unit testing
Integration and Acceptance testing

Figure 22: Development phases (compBennie and stegBennie).

The testing in both cases was a two-phase process. First, for *compBennie*, we did unit testing at the collect and the compare modules. Then, we integrated the two modules into one single program and performed the acceptance testing for *compBennie*. The same procedure was done for the three parts of our main tool, *stegBennie*. We tested separately the performance of 'full', 'file' and 'folder' module and then we unified the three different components to give to the steganalytic tool full functionality and a range of choices. Also, we tested the ability to create the decompression object and read the coefficients of a jpeg image, which was the starting point of each module. For this reason we provide the *testEnv.c* file which uses the *test_environment()* function. More information about testing, installing and using the two tools is provided in READTHIS.txt file and in the Appendix B of this report. Jpeglib should be already installed in order to proceed with the installation of *compBennie* and *stegBennie*. The installation process involves the testing of the library itself; thus, we do not have to write any testing script for the library.

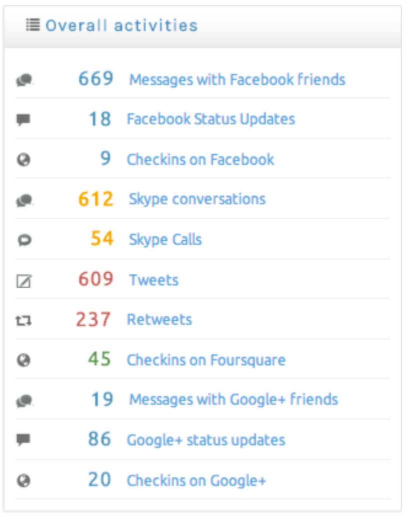
6 Visualization components

6.1 Visualization of Social Network interactions

This section describes various methods for visualizing data collected using the methodology described on chapter 4. The plethora of services that can be used by suspects require a grouping of this disjoint information into a unified set, where actions across services are correlated (e.g. what type of communication does the suspect have with user X from all services). Furthermore, the abundance of available information necessitates the ability to shift focus to specific activities (e.g., status updates on Facebook), and interactions (e.g., users with the largest amount of shared activities with the suspect). Thus, we provide the analyst with dynamic “perspectives” of varying granularity, with aggregated correlations as well as fine-grained views of the collected data. We have several viewpoints for creating the different perspectives.

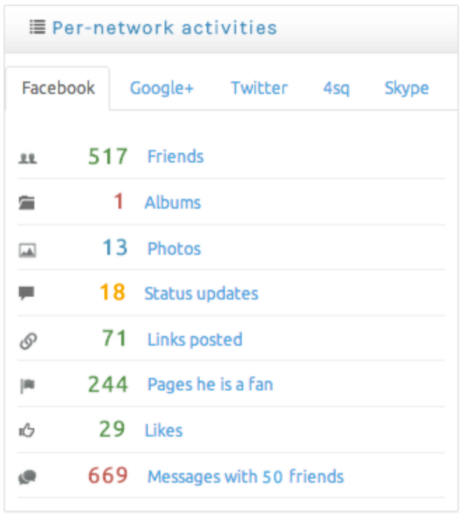
Our goal is to provide a modern visualization platform that will offer a variety of data representations. Therefore, the visualization component is developed as a web application. The vast amount of data, dictated an event-driven and asynchronous approach, where data is fetched upon request. The front-end issues AJAX requests using the jQuery framework which is also responsible for data manipulation. Data visualization is performed using the *D3.js* [2] Javascript library. This library provides a broad variety of visualization and schematics like calendar views and graph layouts among others. Moreover, we leverage the Google Maps Javascript API [10] to render location-based information on a map.

Aggregated. Here we present aggregated statistics regarding the most interesting activities from all the services. With one glance, the analyst can see which services the suspect mainly uses, and what data is available. In Figure 3 we see screenshot regarding some of the aggregated statistics presented in this viewpoint. Specifically, we can see the most important types of data across services and a more detailed description of activities per service, respectively.


Overall activities

669	Messages with Facebook friends
18	Facebook Status Updates
9	Checkins on Facebook
612	Skype conversations
54	Skype Calls
609	Tweets
237	Retweets
45	Checkins on Foursquare
19	Messages with Google+ friends
86	Google+ status updates
20	Checkins on Google+

(a) User granularity


Per-network activities

517	Friends
1	Albums
13	Photos
18	Status updates
71	Links posted
244	Pages he is a fan
29	Likes
669	Messages with 50 friends

(b) Service granularity

Figure 3: Aggregated statistics provide statistics information at a user-granularity and service-level granularity.

Service. Here we focus on a specific service, and present aggregate statistics regarding the users activities. A list presents the contacts that have had the most communication with the suspect. Next, as shown in Figure 4, the structure of the social graph is depicted as well as any interconnections between all contacts. The size of each node is based on the number of connections the contact has. Thus, the analyst can immediately recognize heavily connected users or outliers. The graph can plot contacts of a specific service as shown on this figure, or a combined view of all services where the contact's of each service have a common color. Each graph node represents a user, and when clicked presents the contact's name and photo. Furthermore, a contact search function dynamically detects and highlights nodes in the graph, allowing investigators to quickly identify contacts of interest inside the graph.

Figure 5 presents a screenshot of a graph that visualizes the total communication between the suspect and online contacts. The amount of shared activity defines the width of the connector. Thus, users having greater amout of communication can be easily identified and scrutinized. Moreover,upon clicking the connector, a window is shown, presenting any shared activities between two edges of the connector.

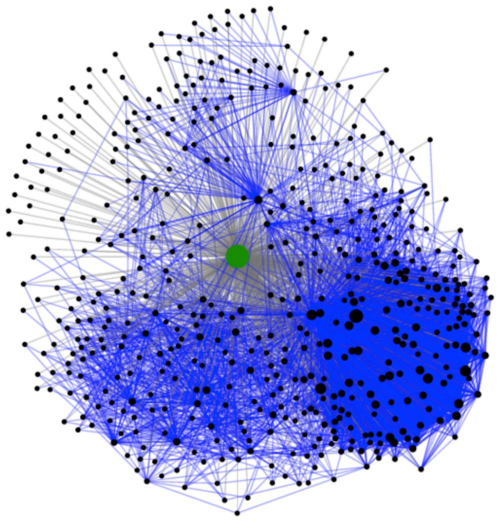


Figure 4: An example plot of the suspect's social graph. The suspect is depicted with the green node. The size of a node is defined by its degree of connectivity. Edges toward the suspect's node are grey, while edges between contacts are blue.

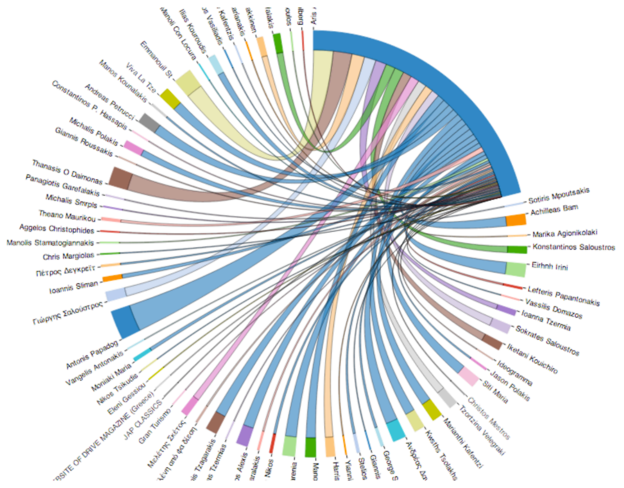


Figure 5: This graph plots the overall communication between the suspect and his contacts. This visualization facilitates the recognition of important contacts, as the volume of communication determines the width of the connection between suspect and contact.

User. A very important viewpoint is that which focuses on a specific user. Once the analyst has identified online contacts that might be of interest, he can use one of two perspectives. First, one can select the contact and be redirected to an aggregated statistics viewpoint, containing all information available regarding the actions of that contact across all services (based on the number of accounts that have been associated during the correlation phase). Second, the analyst can choose to focus only on the shared activities the contact has with the suspect across all services. That includes, chat messages, emails sent, wall posts, shared photos, etc. The coarse-grained perspective presents aggregated statistics, while the more fine-grained perspective allows to focus on a specific type of activity. In both viewpoints, the investigator can ultimately view all individual activity and communication resources, e.g., exchanged messages, pages "liked", or Skype calls. Furthermore, the viewpoints can be dynamically configured to visualize data from one or all services.

Timestamp. A very important factor for visualizing relevant data, is the factor of time. Every perspective contains a color-coded calendar which depicts the amount of conducted activity of a user user on a specific date. A segment of this calendar is shown in Figure 6. Light-green squares depict days where the user had few activities where red squares represent days with high activity. Blank squares indicate that the user did not exhibit any activity on that particular date. However, the analyst might wish to focus on the activities of the suspect during a specific time period which is of interest. As such, certain viewpoints can dynamically change and allow one to focus on a specific time window.

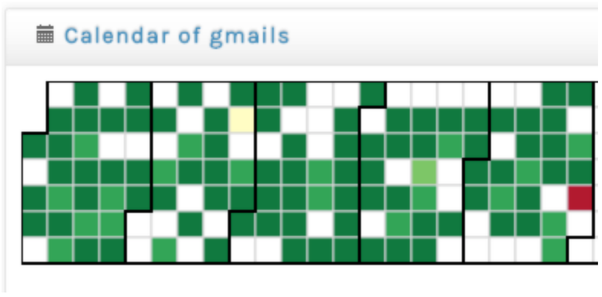


Figure 6: Extract of the calendar element, depicting the email exchange activity of the suspect over a period of five months.

Content. A word cloud provides the analyst with a quick view of the most common words contained in the suspect's communication, which can be across services or focused on a specific service or user. Thus, recurring motives and topics can easily be spotted. In the case of Twitter, we also create a word cloud with the hashtags (i.e., topics) of the suspect's tweets as we show in Figure 7. This can also reveal subjects that the suspect tends to comment on (e.g. politics, religion) and can be relevant to the analyst's investigation. Clicking on one of the terms will fetch all the messages, emails, tweets or posts containing the term. Furthermore, we also follow a more targeted approach, by employing the list of keywords that the US Department of Homeland Security searches for in social networks [15]. Specifically, we search for 377 keywords belonging to 9 categories, ranging from terrorism to drug-related incidents. The number of occurrences are broken down per-category and per-service. By clicking on the respectful information, the analyst is presented with the resources that contain the keywords.

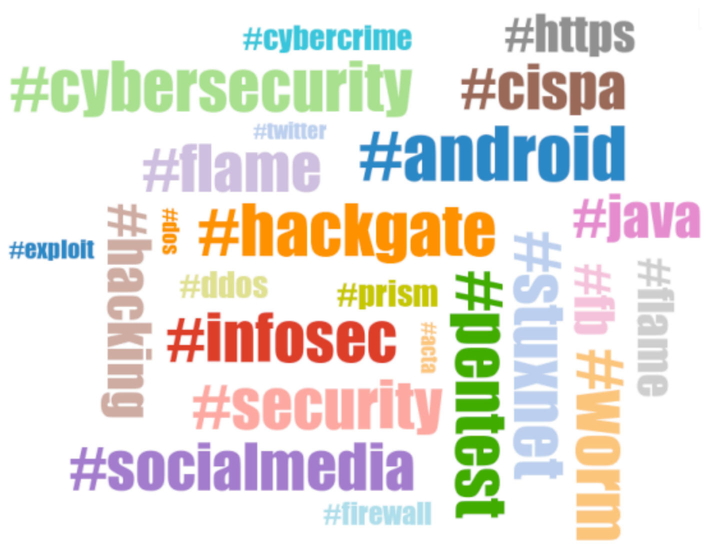
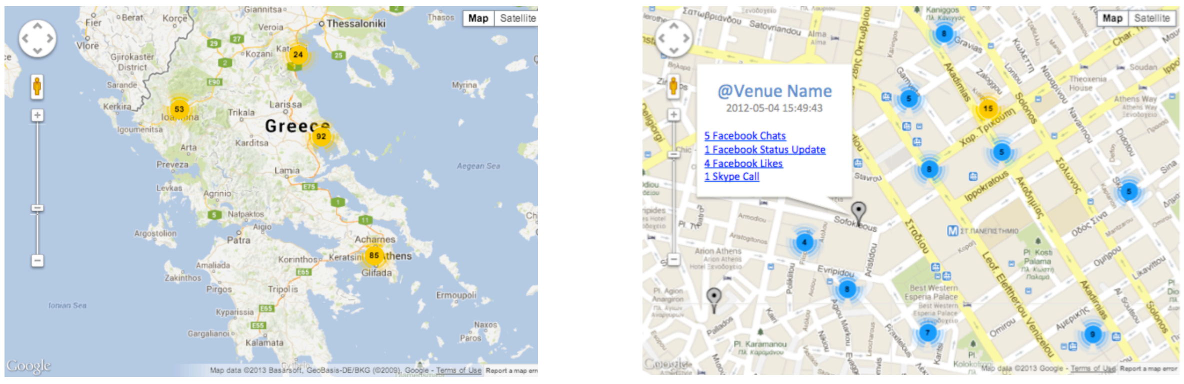


Figure 7: The word cloud shows the words most frequently contained in the suspect's communications. Here we see an example created from a user's Twitter hashtags (topics).

Location. A very important piece of information is the suspect's location. Using information from the suspect's check-ins and residence we plot a map with the locations he has visited, and also visually annotate the amount of times each location has been visited. Furthermore, the analyst can also define a time window, within which all of the suspect's activities are correlated with that location. For example, with a time window of one hour, by clicking on the location marker, a window will inform of all the activities (e.g. chat, Skype calls) the suspect conducted up to one hour after the check-in. Thus, the analyst can associate important activities to specific locations or even search for patterns of activities at certain locations.

In Figure 8(a) we can see an example screenshot showing the aggregated check-ins at a city-level granularity. Figure 8(b) depicts a closer view of a specific region, with the information window for a specific check-in. The window presents the name of the venue, the check-in timestamp and a series of activities that have been completed within a one hour time window. All elements are clickable for presenting the resources of interest.

Furthermore, as a specific period might be of interest, we can plot the check-ins conducted during a specific period of time. Also, the investigator can select a contact, a distance X and a time duration T , and the map presents any check-ins that the suspect and the contact conducted with a time difference up to T at venues that have a maximum distance of X.


@Venue Name
2012-05-04 15:49:43
5 Facebook Chats
1 Facebook Status Update
4 Facebook Likes
1 Skype Call

(a) Aggregated view **(b) Close-up view**

Figure 8: Two views of the map plotting the suspect's check-ins. (a) An aggregated city-level view. (b) The details of a specific check-in and the associated activities.

Photographs. A valuable resource of information in criminal investigations are photos found in social networks, as demonstrated in the case of the Vancouver riots [41], where vandals were identified through photos posted in social networks. The investigator can select to view all the photos collected from the suspect's profiles. Any available user tag information is also presented, and statistics show the contacts with the most common photos with the suspect (based on tag information).

6.2 Timeline Analysis Toolkit

In this section we look at how our timeline analysis tool (TAT) may be used to provide context in the analysis of call logs from an actual police investigation.

During the night of 17/9 to 18/9, just before midnight, a man was stabbed to death after being involved in an argument with a group of people. While the confrontation was in progress, the police were called and arrived shortly after the victim had been stabbed but while he was still alive. The victim, before, expiring, identified the assailant to the police officers. The attacker was then arrested and a cell phone found on his person was confiscated.

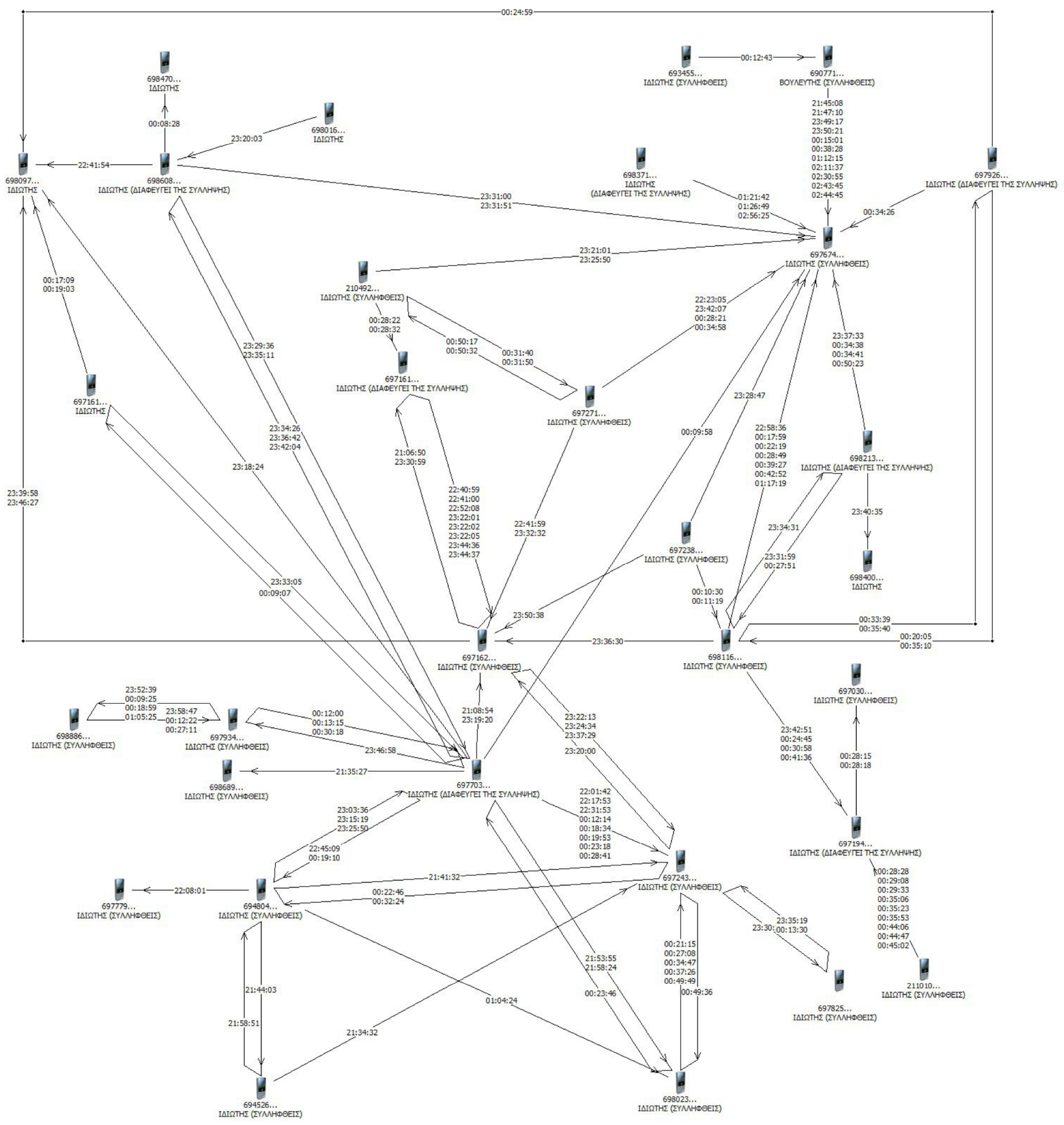


Figure 1: Map of communications made between suspects on the night of the incident [1]

The police speculated that the attack was orchestrated by an organized crime gang and in order to prove their hypothesis, proceeded to examine the call detail records (CDRs) of the attacker's cell phone as well as those who called or were called by that cell phone. This resulted in the graph, shown above.

Each node is a cell phone, the directed links indicate who initiated the call and the numbers overlaying each link show the time that call was placed (but unfortunately not the duration of the call). Please note that the last three digits of the displayed cell phone numbers have been removed to protect the identity of the parties involved.

This graph conveys the extent of the communications that took place from 9 pm till 3 am the night of the murder, but little else. For example it is very difficult to tell

[1] http://tvxs.gr/sites/default/files/article/2013/40/139778g-sxediagramma-tilefonimaton-2_0.jpg

which calls took place before the incident and may, therefore, indicate collusion between the communicating parties.

We used our timeline analysis tool (TAT) to produce the graph shown below, which arranges the same CDR data in a timeline. Each cell phone is a horizontal line in the table and arrows connecting cell phones indicate calls. Here, too, the direction of the arrow indicates who initialed the call. The green vertical line in the middle of the image shows the approximate time of death to help the user differentiate between calls that took place before the incident and those that occurred afterwards. This line may be moved by the user in order to highlight a different time of reference.

The TAT tool expects its input data to be in the form of an Excel file and the order of the telephone numbers on the y-axis depends on the ordering of the entries in the spreadsheet. This allows for various orderings of the numbers (e.g. earliest call first).

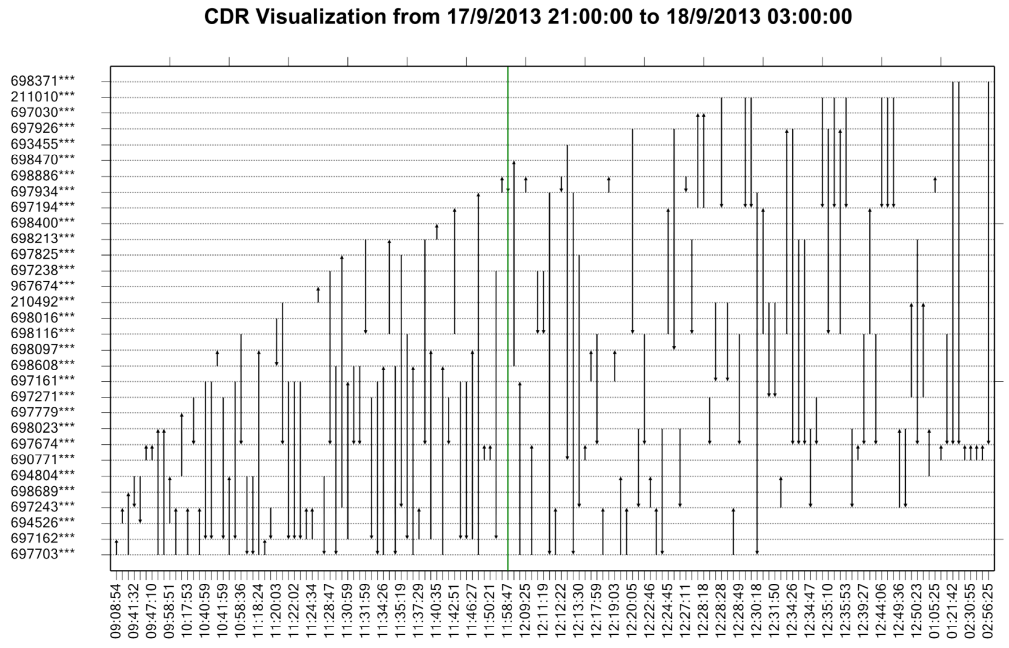


Figure 2: Compressed timeline of calls.

The system also allows the user to zoom in on a subset of the time line in order to be able to see the calls in greater detail. In addition, the user is able to define aliases for cell phone numbers (e.g. place the name of the owner next to the cell phone number).

By selecting a specific cell phone from the y-axis, we can highlight the calls made from, or to, that cell phone (outgoing calls are marked in red, while incoming are marked in blue).

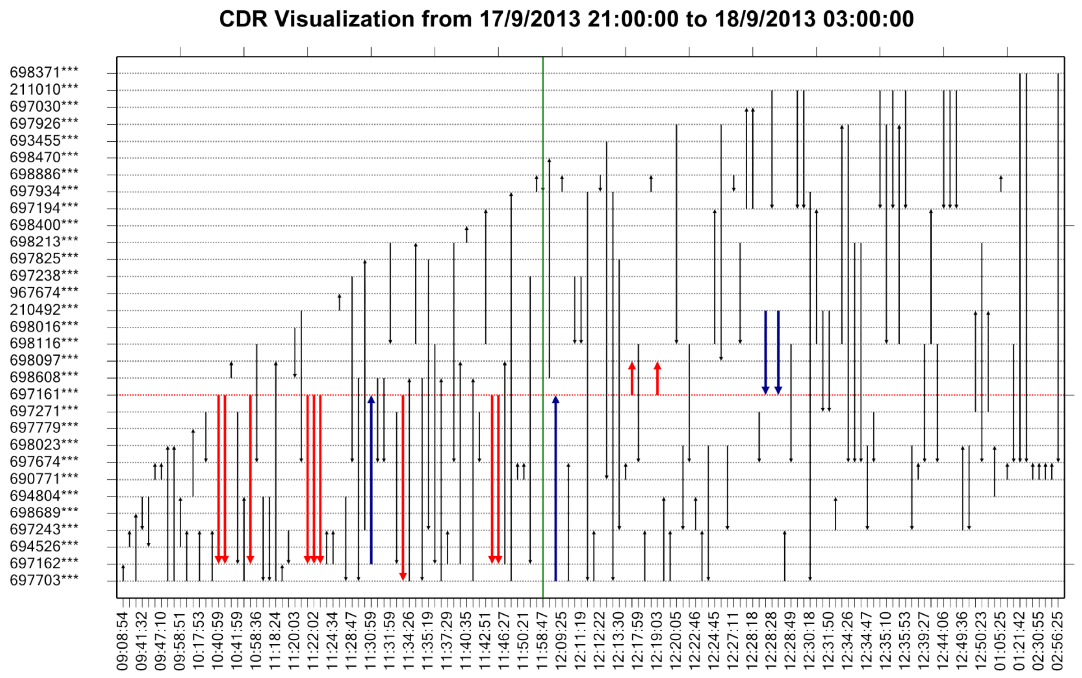


Figure 3: Calls made by selected cell phone are highlighted.

To avoid clutter, we can hide calls unrelated to the number we are interested in. We isolate the call records of specific cell phones by simply selecting the line corresponding to that cell phone. In this case the other communications are removed from the screen, while the list of numbers that were in direct communication with the selected cell phone is displayed in a popup window (shown in figure 6 below).

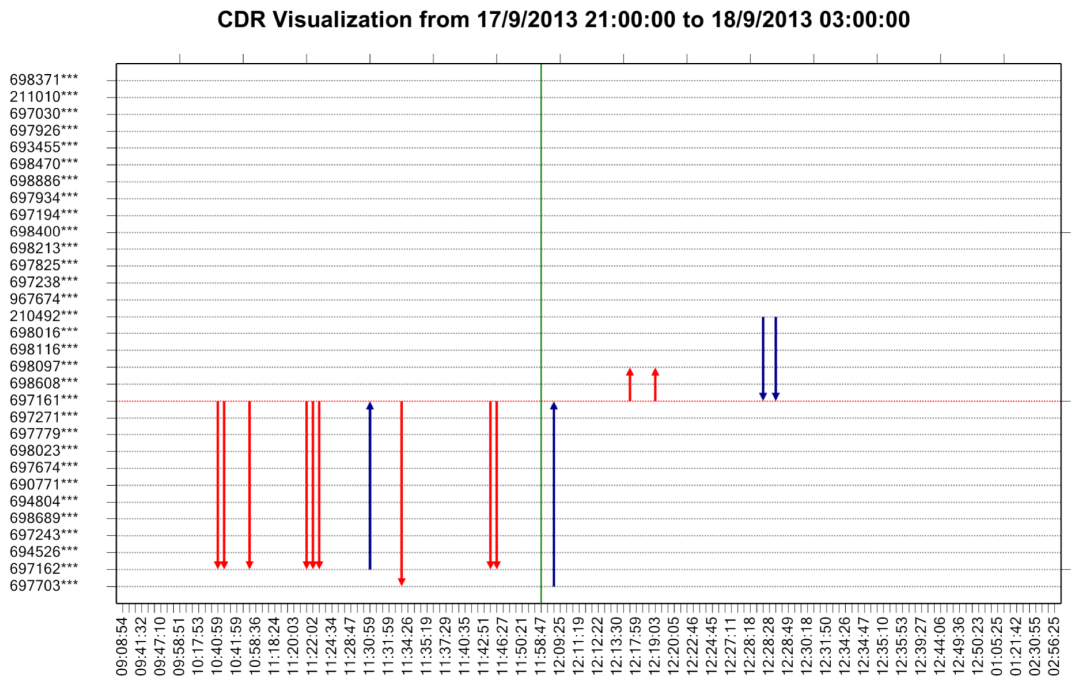


Figure 4: Calls made by selected cell phone are shown exclusively.

Finally, we can display together call records of multiple cell phones. In this case, calls made between numbers outside the selected group are not shown. In Figure 5 we have added name tags to three numbers and we have asked the system to display only calls related to these numbers. We have also selected “true timeline” (x-axis shows time in scale) versus “compressed timeline” (all calls are placed at an equal distance from each other, regardless of how much time may have passed between the placement of these calls) which was used in the earlier figures.

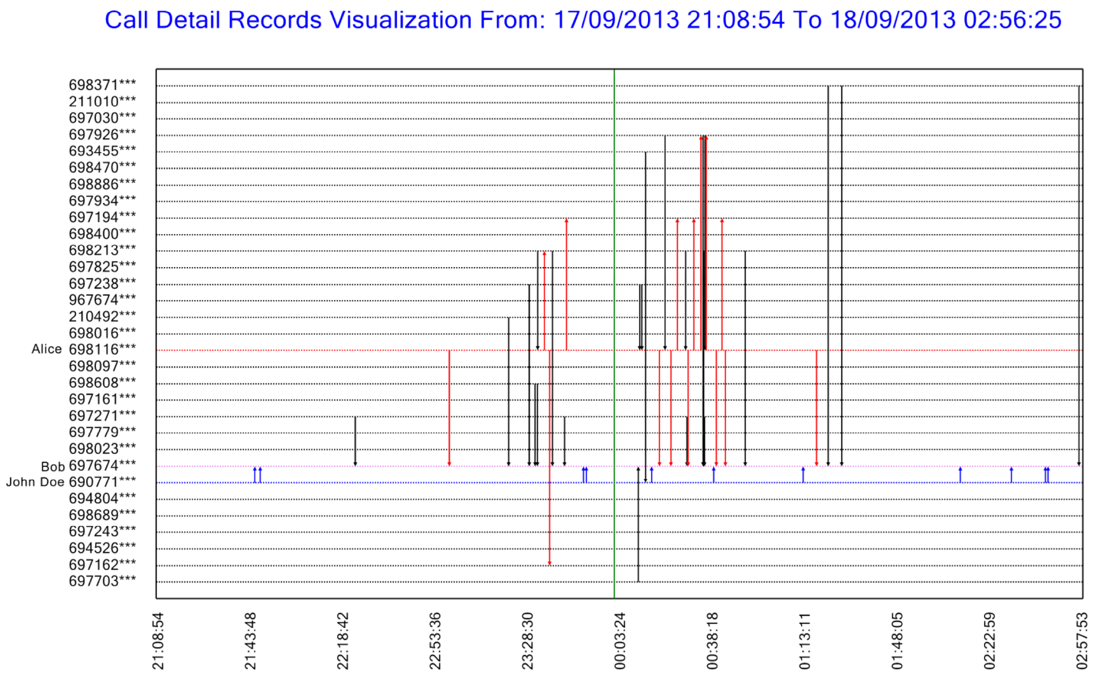
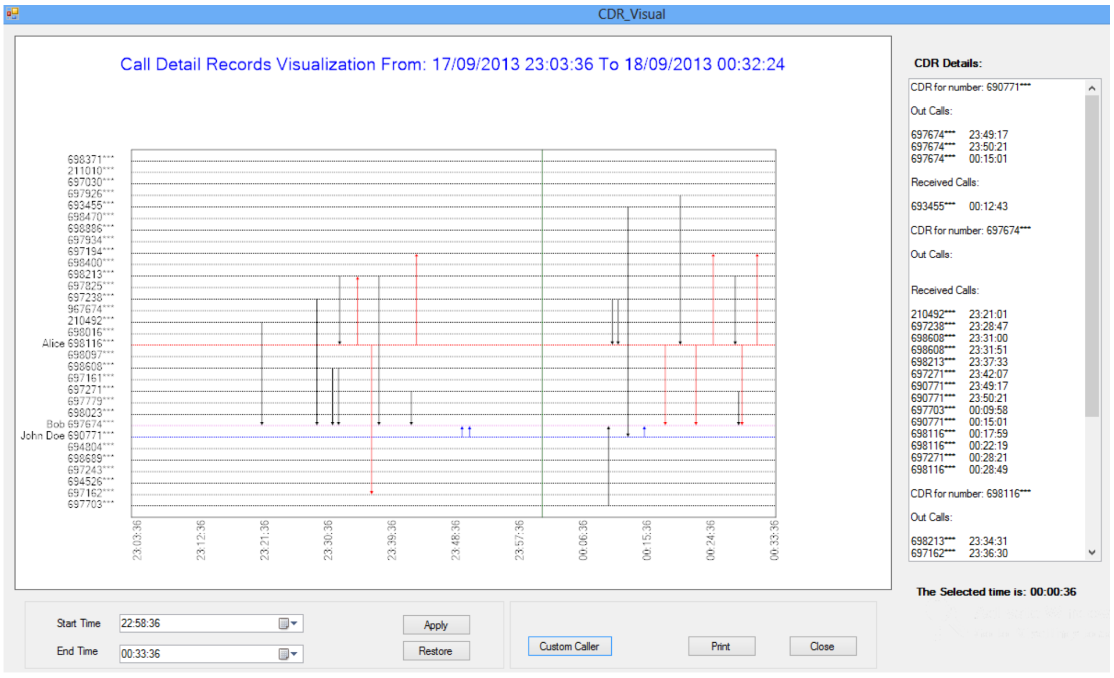


Figure 5: Group of three cell phones are displayed in “true timeline” mode. This mode may cause clutter in cases where multiple calls are made within a short time interval (e.g. at 00:38:18).

Each of the selected numbers is allocated a colour (e.g. Alice is red, Bob is violet, and John Doe is blue), so that calls initiated by Alice are shown as red arrows, and John Doe’s initiated calls are shown as blue arrows. Notice that there are no violet arrows because Bob did not initiate any calls during the monitored time interval.



CDR Details:

```
CDR for number: 690771***

Out Calls:

697674***    23:49:17
697674***    23:50:21
697674***    00:15:01

Received Calls:

693455***    00:12:43

CDR for number: 697674***

Out Calls:


Received Calls:

210492***    23:21:01
697238***    23:28:47
698608***    23:31:00
698608***    23:31:51
698213***    23:37:33
697271***    23:42:07
690771***    23:49:17
690771***    23:50:21
697703***    00:09:58
690771***    00:15:01
698116***    00:17:59
698116***    00:22:19
697271***    00:28:21
698116***    00:28:49

CDR for number: 698116***

Out Calls:

698213***    23:34:31
697162***    23:36:30
```

The Selected time is: 00:00:36

Figure 6: The entire visualization window showing a zoomed-in view of all communications that occurred between the selected handsets on the night of the incident.

The user may also zoom-in on specific time intervals so that busy periods can be viewed in greater detail. In Figure 6 we see a screen dump of the entire visualisation window of the TAT application with all the information available to the user.

Notice that in the window on the right hand side of the figure, the system displays the numbers that called or were called by the selected cell phones.

6.2.1 Analysis

The system is still undergoing improvements, but it has already shown its power by allowing undisclosed numbers to be associated with real persons. For example, reading in the papers that, say, Bob was in frequent communication with John Doe starting from before 10pm and then again after the incident till between 2:30 and 3 am, we can use the TAT system to zoom in on the specified times and identify the cell phone numbers that are likely to correspond to Bob and John Doe.

In a similar manner we can proceed to unravel the remaining threads of the communications that took place on the night of September 17th.

7 Visualisation Integration

This section addresses how new visualisations may be integrated into the DEViSE architecture. It is intended as a technical specification to clarify how different visualisation tools can exchange information.

7.1 Inter-Tool Information Exchange

The exchange of data from one tool to another makes use of three components:

- The C-DOC or Configuration Document
- The I-DOC or Information Document
- Interactive "objects" within an a visualisation tool

These components were introduced in work package 1; C-DOC is an XML file which stores the configuration of a visualisation tool, I-DOC is an XML file which acts as the transport mechanism for transferring data between tools (passed as a command line argument). Objects are context sensitive areas programmed into an application. An object represents one or more function types and holds one or more events. When clicked the object will find compatible applications that can visualise the data it holds.

The purpose for having XML files passed from one tool to another is to remove any association between tools, replacing all interactions with interoperable XML. In this fashion, as long as a computer language has an XML parser implementation, tools can be written in different programming languages and interact solely via the XML interface.

The general process for invoking a tool has been implemented as follows.
Immediately after an object is clicked in an application, a number of steps are carried out to find compatible applications. These steps are:

1. The application obtains other tools' C-DOC's. All C-DOC's are stored in one location (i.e. folder on a hard disk).
2. The application retrieves the entry within each C-DOC for the input other tools can accept. The <input> tag within the C-DOC contains the function a tool can carry out.
3. The application retrieves the quantity of input a tool can visualise (again, from the <input> tag).
4. A comparison is made between the data types & quantity found in the tool configuration with the same details retrieved from the object.
5. A menu is generated listing all tools matched by data type & quantity (including, if applicable, the initial tool itself).
6. When a tool is selected from the menu, an I-DOC is created. This I-DOC contains the names of functions and the unique identifier of each event. The I-DOC file is passed to the selected tool as a command line argument.
7. The receiving tool parses the I-DOC and runs each function with the unique identifier to retrieve information from the database, which it then uses to create a visualisation.

8 Tool and Platform Integration

The forthcoming deliverable D3 will deal with the integration of the aforementioned tools into meaningful use cases, where we will demonstrate how useful they can be in real life circumstances. It is anticipated that at least one demonstrator will be developed showcasing the linking of tools' output with the DEViSE platform. Those use cases will be validated by practitioners from our Police partners prior to the formal release of the corresponding code via open source routes. The exact format of dissemination of the code will be decided at a later stage (e.g. via github).

9 Bibliography

1. Anatomy of Facebook. https://www.facebook.com/notes/facebook-data-team/anatomy-of-facebook/10150388519243859
2. D3.js – Data Driven Documents http://d3js.org
3. Department of Law, State of New Jersey. Melanie McGuire found guilty of murder in 2004. http://www.nj.gov/oag/newsreleases07/pr20070423a.html
4. Facebook developers: Email permissions. https://developers.facebook.com/docs/reference/login/email-permissions/
5. Facebook social plugins. https://developers.facebook.com/docs/plugins/
6. Forensic focus: Dropbox forensics. http://www.forensicfocus.com/Content/pid=429/page=2/#database
7. Foursquare API Endpoints. https://developer.foursquare.com/docs/
8. Foursquare wrapper. https://github.com/mLewisLogic/foursquare
9. Google+ API. https://developers.google.com/+/api/
10. Google Developers - Google Maps JavaScript API v3. https:// developers.google.com/maps/documentation/javascript/
11. Google Geocoding API. https://developers.google.com/maps/documentation/geocoding/
12. The guardian: NSA prism program taps in to user data ofapple, google and others. http://www.guardian.co.uk/world/2013/jun/06/us-tech-giants-nsa-data
13. IACP center for social media, 2012 survey results. http://www.iacpsocialmedia.org/Resources/Publications/2012SurveyResults.aspx
14. jQuery. http://jquery.com
15. List of searched keywords http://animalnewyork.com/2012/the-department-of-homeland-security-is-searching-your-facebook-and-twitter-for-these-words/
16. PhantomJS: Headless WebKit with JavaScript API. http://phantomjs.org/
17. tweepy. https://github.com/tweepy/tweepy
18. Adu-Oppong, F., Gardiner, C. K., Kapadia, A., and Tsang, P. P. Social circles: Tackling privacy in social networks. In symposium on Usable Privacy and Security (SOUPS) (2008)
19. Al Mutawa, N., Baggili, I., and Marrington, A Forensic analysis of social networking applications on mobile devices. Digital Investigation 9 (2012), S24–S33
20. Andriotis, P., Oikonomou, G., and Tryfonas, T. Forensic analysis of wireless networking evidence of android smartphones. In Information Forensics and Security (WIFS), 2012 IEEE International Workshop on (2012), IEEE, pp. 109–114
21. Balduzzi, M., Platzer, C., Holz, T., Kirda, E., Balzarotti, D., and Kruegel, C. Abusing social networks for automated user profiling. In RAID (2010), pp. 422–441

22. Bugzilla. Bug 609361 - SVG takes ages to render. Is ok in other browsers, like Opera or Safari https://bugzilla.mozilla.org/show_bug.cgi?id=609361

23. Bugzilla. Bug 791699 - Slow setting of attributes on SVG elements due to time spent in region operations on this molecular dynamics simulation https://bugzilla.mozilla.org/show_bug.cgi?id=791699

24. Chu, H.-C., Deng, D.-J., and Park, J. H. Live data mining concerning social networking forensics based on a facebook session through aggregation of social data. Selected Areas in Communications, IEEE Journal on 29, 7 (2011), 1368–1376

25. Correa, C., and Ma, K.-L. Visualizing Social Networks. Springer, 2011, ch. Chapter 11: pp. 307-326.

26. Craiger, J. P., Swauger, J., and Marberry, C. Digital evidence obfuscation: recovery techniques. In Defense and Security (2005), International Society for Optics and Photonics, pp. 587–594.

27. Facebook. Facebook Query Language (FQL) Reference https://developers.facebook.com/docs/reference/fql/

28. Facebook. fbconsole. https://github.com/facebook/fbconsole

29. Facebook. Graph API https://developers.facebook.com/docs/reference/api/

30. Forbes. Solving a teen murder by following a trail of digital evidence http://www.forbes.com/sites/kashmirhill/2011/11/03/solving-a-teen-murder-by-following-a-trail-of-digital-evidence/

31. Garfinkel, S. L. Forensic feature extraction and cross-drive analysis. Digital Investigation: The International Journal of Digital Forensics & Incident Response 3 (Sept. 2006), 71–81

32. Gross, R., and Acquisti, A. Information revelation and privacy in online social networks. In Proceedings of the 2005 ACM workshop on Privacy in the electronic society (New York, NY, USA, 2005), WPES '05', ACM, pp. 71–80

33. Hall, P. A., and Dowling, G. R. Approximate string matching. ACM Computing Surveys (CSUR) 12, 4 (1980), 381–402

34. Heer, J., and Boyd, D. Vizster: Visualizing online social networks. In Proceedings of the Proceedings of the 2005 IEEE Symposium on Information Visualization (Washington, DC, USA, 2005), INFOVIS '05', IEEE Computer Society, pp. 5–

35. Huber, M., Mulazzani, M., Leithner, M., Schrittwieser, S., Wondracek, G., and Weippl, E. Social snapshots: digital forensics for online social networks. In ACSAC (2011), pp. 113–122.

36. Jin, L., Takabi, H., and Joshi, J. B. Towards active detection of dentity clone attacks on online social networks. In Proceedings of the first ACM conference on Data and application security and privacy (New York, NY, USA, 2011), CODASPY '11' ACM, pp. 27–38

37. Kontaxis, G., Polakis, I., Ioannidis, S., and Markatos, E. P. Detecting social network profile cloning. In Pervasive Computing and Communications Workshops (PERCOM Workshops), 2011 IEEE International Conference on (2011), IEEE, pp. 295–300.

38. Krishnamurthy, B., and Wills, C. E. On the leakage of personally identifiable information via online social networks. In WOSN '09: Proceedings of the 2nd ACM

	workshop on Online social networks' (New York, NY, USA, 2009), ACM, pp. 7–12
39.	Krishnamurthy, B., and Wills, C. E. Leakage in Mobile Online Social Networks. In Proceedings of the 3rd Workshop on Online Social Networks (WOSN 2010) (June 2010)
40.	Mao, H., Shuai, X., and Kapadia, A. Loose tweets: an analysis of privacy leaks on twitter. In Proceedings of the 10th annual ACM workshop on Privacy in the electronic society (NewYork, NY, USA, 2011), WPES '11', ACM, pp. 1–12.
41.	Mashable. Vancouver fans riot as canucks lose stanley cup. http://mashable.com/2011/06/15/vancouver-hockey-riot/
42.	Mulazzani, M., Huber, M., and Weippl, E. Social network forensics: Tapping the data pool of social networks. In Eighth Annual IFIP WG 11.9 International Conference on Digital Forensics (1 2012)
43.	Navarro, G. A guided tour to approximate string matching ACM computing surveys (CSUR) 33, 1 (2001), 31–88
44.	Polakis, I., Kontaxis, G., Antonatos, S., Gessiou, E., Petsas, T., and Markatos, E. P. Using social networks to harvest email addresses. In Proceedings of the 9th Annual ACM Workshop on Privacy in the Electronic Society (WPES) (2010), ACM, pp. 11–20.
45.	Shen, Z., Ma, K.-L., and Eliassi-Rad, T. Visual analysis of large heterogeneous social networks by semantic and structural abstraction. IEEE TRANSACTIONS ON VISUALIZATION AND COMPUTER GRAPHICS 12, 6 (2006), 1427–1439
46.	Tang, J., Sun, J., Wang, C., and Yang, Z. Social influence analysis in large-scale networks. In Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining (New York, NY, USA, 2009), KDD '09', ACM, pp. 807–816
47.	The Concord Consortium. Serious Performance Regression in Firefox 18 and newer. http://blog.concord.org/serious-performance-regression-in-firefox-18-and-newer
48.	Andriotis, P., Oikonomou, G., & Tryfonas, T. (2013). JPEG steganography detection with Benford's Law. *Digital Investigation*.
49.	Fu, D., Shi, Y. Q., & Su, W. (2007). A generalized Benford's law for JPEG coefficients and its applications in image forensics. *SPIE Electronic Imaging: Security, Steganography, and Watermarking of Multimedia Contents, San Jose, CA, USA*.
50.	Schaefer, G., & Stich, M. (2003). UCID: an uncompressed color image database. *Electronic Imaging 2004*, 472-480.