DATA RESTORATION: A MANDATORY OPTION FOR CRITICAL COMPUTATIONAL ENVIRONMENTS

CASE STUDY: ZIPRO *VS*. JAR COMPARISON

Vlad-Alexandru GROSU [1]

So far, there are no software-archiving suites available to offer options/services like data restoration or data recovery. This option becomes critical in situations in which licenses are involved (like non-free computational environments) as well as in computation environments where data loss is critical. For such situations, any loss leads to large amount of money spent for recovery. Moreover, there are situations for which this attempt is eventually impossible. This paper intends to circumvent such situations by presenting a practical solution for these kinds of problems. It also takes into consideration some vulnerabilities that .jar format has, from data restoration perspective.

Keywords: data restoration, Zipro, Jar, computational environments

1. Introduction

This paper addresses itself mainly to open-source environments where data manipulation and processing are intensive, namely *embedded systems*. It is especially intended for the development systems that resemble the smartphones (e.g. Beagleboard, Pandaboard or even Raspberry PI). Of course, this attempt can be generalized for the smartphones themselves.

1.1. A new data file type

A new compression/decompression tool was developed - called *zipro* - having in mind some special computational environments: the embedded systems. The file's extension that *zipro* deals with is *.zha* [1]. This is a completely new data file type in the IT field.

1.2. Contributions

As **a novelty** in the IT field, the new developed tool does offer true restoration capabilities. Its capabilities are based on the following pieces of information:

- the original *.zha* archive (about to be restored),
- the meta-data, that gets generated starting from the input file,

[1] Lecturer, Dep. of Electronic Technology and Reliability, University POLITEHNICA of Bucharest, Romania, e-mail: crisvlad74@yahoo.com

- the concept of *symmetrical copy* (backup copy).

This way it offers security proofs, useful against data corruption or data interception that could happen in various types of attacks [2], mainly related to the embedded systems. For instance, within the Android smartphone universe, if several copies of the same file appear within an `.apk [3]`, then an attacker can turn this computational context into a dangerous one. See the design details in section 3.

My approach here is aware of such situations, and in fact doesn't allow several copies of the same information to appear within a `.zha` file. For such computational conditions, dedicated software tools were developed, so that they better address the specific requirements of the embedded systems. The trend is for such media to be classified as *real-time systems*. Some authors [4] say they behave like real-time systems and therefore belong to this class. While some other authors do not accept this classification entirely, unless some changes are made [5].

Whatever the approach, there are well-defined restrictions that must apply for such systems:

1. Fast Task Context Switches,
2. Efficient Native Message Queues,
3. Minimal Operating System Overhead.

And these are not far from what a RTOS should do. Some short but necessary details are presented in the following sections. These justify the philosophy behind the `Zipro (.zha)` archive format, in comparison with the Java Archive.

2. Background

2.1. Smartphones architecture - very short approach

In order to compare the existing facilities in Java Archive, it was necessary to review - in short - the inner software components that qualified themselves as the fundamental pieces in a SoC-like hardware and software environment: the smartphone. A closer look to such systems reveals that they rely on a predefined stack of layers, starting with the lowest level (the kernel level) and ending with the highest one (the OS's main framework that finally supports the entire user interface).

In such environments, all the hardware manufacturers chose to conform to Google's open environment, based on multi-layered architecture. Later on, this architecture became *Android*.

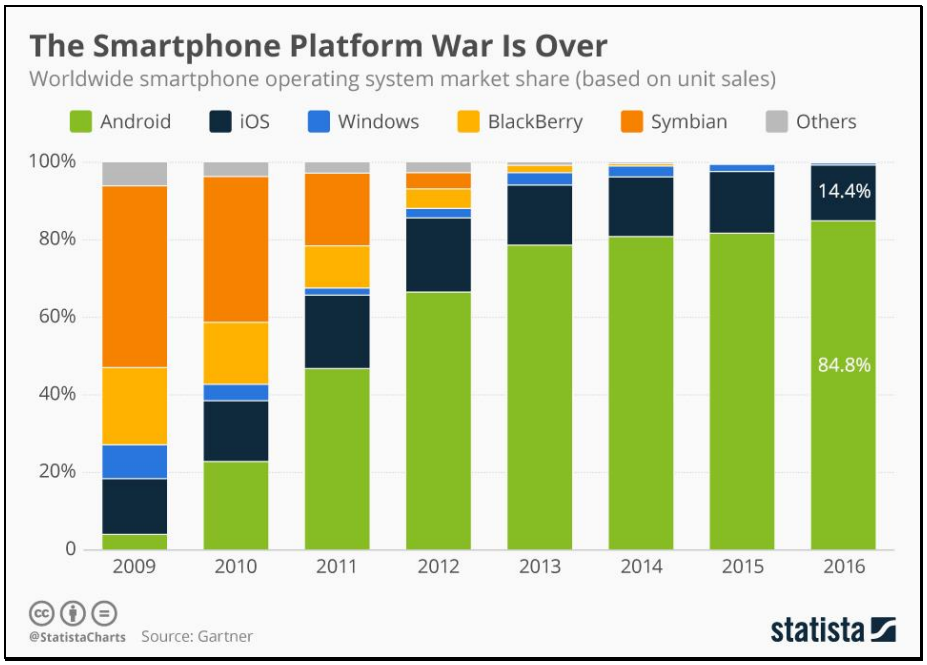

Fig.1. The main actors on the mobile phone market (up to 2016, according to [6])

Finally, it materialized itself into a mature operating system that currently covers around 80% of the smartphones' market [6] in 2015 but its growth is not cooling off in 2016, still growing within a margin of 4-5% (see figure 1).

This OS proved to be well suited for the mobile phones in the *smartphone* class, but this was not the situation at the very beginning. It passed through major software redesign phases until it reached the actual shape. The actors chosen to play the major roles in such a system are [7] (see figure 2):

- C language, because of the Linux kernel and various libraries required by the system. The C standard library was rewritten as *Bionic C* [8]. This was done mainly for three reasons:
  - License: get rid of BSD's GNU Public License (GPL) and move to Lesser GPL (LGPL),
  - Size (memory footprint): the rewritten version is smaller and it needed to be like this,
  - Speed: Bionic is designed for CPUs at relatively low clock frequencies.
- Java Virtual Machine (the older Dalvik virtual machine, completely replaced by ART layer which was introduced as a runtime environment starting with Android 5.0-Lollipop) and Core Java libraries - optimized for low processing power and low memory environments,

- The application framework - collection of managers that finally offers the basic functions of each phone. This include: window manager, activity manager, package manager, content provider,
- Finally, the applications - which are developed using Java language and finally become android *packages* known as `.apk`. This way they are ready for installation onto the mobile terminal (phone).

Maybe one of the most important components that sustain the second and the third elements in the classification above is the Java archive (`.jar`). It is mostly present in the environment's *framework* layer.

3. Application design

3.1. Data restoration: possible drawbacks of Java archive

3.1.1 The JAR meta-information

In many cases, JAR files are not just simple collections of java classes files and/or resources. They are used as building blocks for applications and extensions. Looking at it from the restoration perspective, in the following I can identify three possible drawbacks of a Java archive.

The `META-INF` directory, if it exists, is used to store package and extension configuration data, including security, versioning, extension and services. [9]

Given the data restoration perspective in this article, the main problem here (*the first identified flaw*) is that the meta-information is found ***within*** the archive. Once the *.jar* file gets corrupted the virtual information that such `META-INF` directory would have brought is useless. In the approach presented here (I am referring to the proposed Zipro's `.zha` archive format), in order to maintain a proper self-recognition operation, the meta-information was brought outside of the archive file.

3.1.2 The JAR index

Since its 1.3 version, `JarIndex` is introduced to optimize the class searching process of class loaders for network applications, especially applets. Originally, an applet class loader uses a simple linear search algorithm to search each element on its internal search path, which is constructed from the "`ARCHIVE`" tag or the "Class-Path" main attribute. The class loader downloads and opens each element in its search path, until the class or resource is found. If the class loader tries to find a nonexistent resource, then all the jar files within the application or applet will have to be downloaded. For large network applications and applets this could result in slow startup, sluggish response and wasted network bandwidth. The `JarIndex` mechanism collects the contents of all the jar files defined in an applet and stores the information in an index file in the first *.jar* file on the applet's class path. After the first jar file is downloaded, the applet class loader will use the collected content information for efficient downloading of *.jar* files.

The existing JAR tool is enhanced to be able to examine a list of *.jar* files and generate directory information as to which classes and resources reside in which *jar* file. This directory information is stored in a simple text file named `INDEX.LIST` in the `META-INF` directory within the root of the Jar file. When the class loader loads the root jar file, it reads the `INDEX.LIST` file and uses it to construct a hash table of mappings from file and package names to lists of jar file names.

In order to find a class or a resource, the class loader queries the hash-table to find the proper jar file and then downloads it if necessary. Once the class loader finds an `INDEX.LIST` file in a particular *.jar* file, *it always trusts the information listed in it*. In my opinion, given the data restoration's point of view, here is another flaw (*the second one* identified within this article). This is particularly dangerous since an attacker may feed in an archive with junk information, or even worse, with information put there on purpose.

One such attack took place by the end of 2012 [10]. In fact, many types of attacks took place over time. Some of them were immediately addressed by Google, while some others were not. A list can be found at [11].

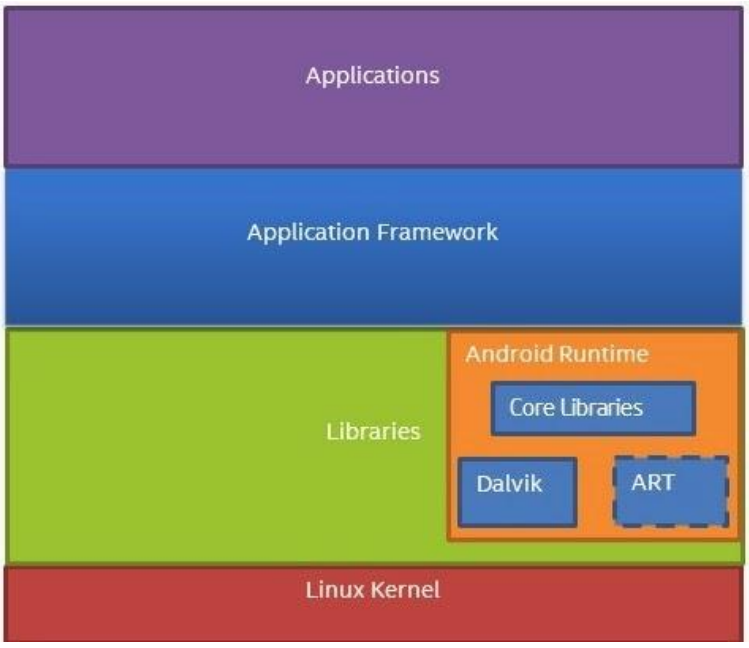

Fig.2. Typical Android architecture: multi-layered approach ([7])

3.1.3 Java archive based on multiple files

Zipro doesn't allow a file to appear multiple times in a given archive. This is checked in the moment of the archive construction. The compression process stops if such a case is encountered.

In case of a *.jar* file, if a mapping is found for a particular java class, but the class loader fails to find it by following the link, an `InvalidJarIndexException` is thrown. When this occurs, the application developer should rerun the JAR tool on the extension to get the right information into the index file. To prevent adding too much space overhead to the application and to speed up the construction of the in-memory hash table, the `INDEX.LIST` file is kept as small as possible. From data restoration point of view **this is yet another limitation** (*the third one*), since this index list is kept *within* the very archive file. In `zipro`'s approach, there are *magic numbers* (used as signatures) spread along the archive file. This classifies `zipro` as a *hybrid compression* technique. I call it hybrid since these pieces of information are stored in the archive *as they are* (in un-compressed format) along with the most compressed part of the archive itself. These particular ASCII combinations help the decompression tool by speeding it up. Given this aspect, this kind of archive cannot necessarily be compared to some other usual (classical) compression techniques, from either the speed or necessary storage space point of view. In the meta-information file (which stays *outside* of the archive itself) there is already a map of the input archive file. If this is a multiple-file based archive then the offsets of the compressed files inside the archive are generated and kept within the meta-information file. As such, any decompression attempt will jump directly to the desired offset within the archive.

3.2. Data restoration technique: Zipro (.zha) approach

After identifying the possible issues of a `.jar` file type in the previous sections, I will show how those drawbacks can be circumvented in `Zipro`'s `.zha file type`, from data restoration perspective. The developed tool herein can identify an input archive as being the right one, based on the meta-information file. As previously stated, the *.zha* archive itself contains some *signature* (specific succession of chars), stored in un-compressed format. Such signature appears at proper places in the file, in order to help both the file identification step as well as the file extraction process. Consequently, the signature appears at the very beginning of .zha archive - for identification purpose - as well as at some positions within the archive, if a multi-archive file is generated - in order to delimit the compressed information of each file. It is this raw information (signature) that gives Zipro's *.zha* its hybrid nature, very useful in the approach proposed in here.

Of course, one can say that if the meta-information gets corrupted then nothing can be done here as well. This is true, and therefore I've thought about a *three-step restoration model*, consisting of:

- *the symmetrical copy*, which is basically an encrypted backup copy of the input (unaltered) archive file,

- *the meta-data information*, stored in a file, that collects specific information out of the input file,
- *the actual archive file*, meaning the original file, that eventually needs to be restored. It also serves the previous two parts of information.

The restoration process assumes that, at any moment in time, at most two of the previous three parts of information are valid. If and only if *all* these three pieces of information get corrupt (e.g. due to some on-purpose attack) then the restoration cannot take place any longer.

The 'symmetrical copy' is build as follows, in this order:

1. Start scanning the input data (input `.zha` file), one byte at a time.
2. Pass it through a symmetrical encryption algorithm (processing phase).
3. Construct the *symmetrical copy* (the backup file). The locution '*symmetrical copy*' is strongly related to the type of encryption algorithm that I have used here. It can be replaced by other such algorithms, if necessary.

If the symmetrical copy gets corrupted then, based on the input archive and the meta-data, it can be generated once again. Conversely, if the input archive gets altered in any way, such as a read cannot be performed then, based on the symmetrical copy and meta-data it can be restored (perform the decryption step upon the symmetrical copy). Eventually, if the meta-data gets corrupted then, based upon the input archive it can be restored as well. As one can see, this three-step restoration process can stand against various data corruption that could happen.

The software solution and specific implementation decisions do not represent the subject of this article.

4. Mathematical fundamentals of the encryption tool

4.1 The symmetrical copy: the necessity of a correct data restoration

4.1.1 Hill's cipher: mathematical aspects

The encryption algorithm used here is a personal implementation of the Hill's Cipher. This belongs to the class of ciphers based on the *transposition of symbols*. As already stated, the encryption step is useful in the symmetrical copy generation (see section 1.2).This way the symmetrical copy gains in robustness. From the conceptual point of view, also given the scale of some actual project using the tool presented here, I consider that more powerful encryption techniques can be used, if necessary. The target here consists in domestic applications of personal use and not governmental or even higher nature (like military or so). Nevertheless, one has to have in mind that the encryption technique can change if the practical context requires it. The code behind this tool was written with modularity in mind.

Hill's cipher uses two keys [5]:

- the encryption key,
- the decryption key.

Mathematically speaking, both keys are in fact *squared matrices*. Among them, only the decryption key requires matrix calculations. Being an integer based application - the codes of the characters in use belong to - all the computations here are based on *modulo M* arithmetic. The value of *M* is chosen so that it is synchronized with the required input alphabet. [5]

The input consists in the actual file to be restored. Usually, this file is the corrupted archive. An input file can be presented either as a:

- text-based file,
- binary file.

Fortunately, in both situations the available symbols (the set of useful characters) belong to the extended ASCII table. Usually, most of the symbols are alphanumerical. Since extended ASCII table offers `256` useful combinations, my choice for *M* is `256`. At most we can choose for this value a prime number as well, for instance *`M=257`*. This choice further improves security, in the eventuality of cryptographic attacks.

The data encryption approach here doesn't require supplementary processing time (generally speaking, supplementary resources). Given that *`M=256`*, the keys are hard-coded in the source code: both the forward and the backward matrix. Therefore, un-necessary inverse matrix computation was avoided. This has a benefic effect given the processing limitation of any system on chip (SoC). The only operations needed for generation of the encryption keys are the fundamental ones: addition, subtraction, multiplication and division (the last one rarely used).

Furthermore, from the encryption's safety and power points of view, they both rely upon the advantage of hiding the encryption and the decryption keys - by hard-coding them. This way no one knows either the dimension of the encryption key or the size of the input alphabet (*M*).

In order to decrypt the symmetrical copy, the key requires `modulo M` calculation of the inverse of the input matrix's determinant. This operation is known as '*modular multiplicative inverse*' [4]:

$$\frac{1}{\det(A)} \pmod{M} \qquad (1)$$

where:

A - the encryption key (input matrix);

`det(A)` - the determinant of A.

It is important to note that, given a modulo M class, not each natural number has an inverse, called *symmetrical* - that belongs to the same very class (mod M), as one can see in the following section.

4.2. Hill's cipher and modulo M arithmetic

In the following, the mathematical background required by the Hill's cipher will be defined. A main conclusion rises here: M must be a prime number in order to simplify the calculations.

Definition 1:

The set:

$$Z_m = \{0, 1, \ldots, m-1\} \quad (2)$$

along with the addition and multiplication operations is called numeration system of the integers *modulo m*.

Proposition 1:

Let m be an integer number, having the property $m>1$.

Then, in the integer numeration system Z_m the following properties hold:

1 For: a, b, c $\in Z_m$, $(a + b) + c = a + (b + c)$

2 For: a, b $\in Z_m$, a + b = b + a

3 For each a $\in Z_m$, $a + 0 = a = 0 + a$

4 For each a $\in Z_m$, there is a unique x $\in Z_m$, called *the opposite* of a, so that: $a+x = 0 = x+a$

5 For: a, b, c $\in Z_m$, $(a \bullet b) \bullet c = a \bullet (b \bullet c)$

6 For: a, b $\in Z_m$, ab = ba.

7 For each a $\in Z_m$, $1 \bullet a = a = a \bullet 1$.

8 For each a $\in Z_m$, with a=0, there is a unique y $\in Z_m$, called *the symmetrical* of a, so that: $a \bullet y = 1 = y \bullet a$

9 For: a, b, c $\in Z_m$, $a \bullet (b + c) = a \bullet b + a \bullet c$

10. In Z_m we have: 1 = 0.

Generally speaking, the 8th property above is not always true. For instance, for Z_4 numeration system we have the following multiplication table (see Table 1). Note that value 2 doesn't have a symmetrical mod 4, since there is no value in Z_4 multiplied by 2 that gives a remainder of 1 (mod 4). Conversely, the value 3 is invertible and its symmetrical is 3 (itself).

Table 1.

Multiplication within *mod 4* numeration system, Z_4.

.	0	1	2	3
0	0	0	0	0
1	0	1	2	3
2	0	2	0	2
3	0	3	2	1

The general theorem that justifies the statement I've made earlier follows: An element in Z_m has a symmetrical (or *multiplicative inverse*) `modulo m` if and only if *a* is relatively prime with *m*, that means 1 is the common divisor of both *a* and *m* (given the division common sense).

As a consequence of this theorem, the Proposition 2 below follows.

Proposition 2:

Unlike Z_4, each element in Z_5 has a symmetrical, as table 2 shows.

Table 2.

Multiplication within *mod 5* numeration system, Z_5.

.	0	1	2	3	4
0	0	0	0	0	0
1	0	1	2	3	4
2	0	2	4	1	3
3	0	3	1	4	2
4	0	4	3	2	1

The distinction between Z_4 and Z_5 is that 5 is prime number whereas 4 is not.

Definition 2:

A closed numeration system with respect to addition and multiplication is called field when all the properties above hold (see Proposition 1).

Based on the distinction between Z_4 and Z_5 and starting from the previous theorem the next proposition follows:

Proposition 3:

Let m be an integer, with $m>1$. Then Z_m is *field* if and only if *m* is a *prime number*. Looking at the previous examples related to Z_4 and Z_5, we have that Z_5 is *field* while Z_4 is not.

For each prime number *m*, the field Z_m is also *finite*. From the implementation point of view, these results suggest how to choose *m*, so that to simplify the calculations: it should be a prime number.

Nevertheless, if the values of the encryption key (input squared matrix) are properly chosen, then an inverse computation is also possible for sets Z_m for which *m* is *not* prime number. This means that the determinant of the matrix has an inverse belonging to the `modulo m` remainders class.

5. Results and discussions

5.1 Appropriate choices for the presented case study

In this case study, the data compression works within extended ASCII table (each character requires 1 byte for encoding). This means a set of 256 possible input combinations. Consequently, the length of the working space can be chosen as: $m=256$, meaning that the Z_{256} set have to be used. The number of column of the input matrix is 4. This choice is based upon the magic number,

which relates to the 'self recognition' operation. Here, this magic number consists in a combination of 4 characters. It is used as a delimiter of the information in the output file resulting after the compression step.

Based on this technique, each file can be easily identified as a native `.zha` file. This is done by:

- checking if the file's signature is valid (this information stays within the input file);
- checking the digital footprint based on MD5 number [11] (this information stays outside the `.zha` file presented as input).

This delimiter qualifies the resulting `.zha` file as a hybrid format. The term *hybrid* here means that the output compressed file (the output archive file) contains both binary information (as a result of the compression step) as well as clear, unaltered ASCII information, identifiable throughout these magic numbers. **This is the original approach** proposed in my PhD thesis. Going back to Hill's algorithm, if someone wants to use other cryptographic techniques, then these techniques can be embedded into the software project proposed here thanks to the modularity approach. Any change should be based on LGPL license this project is based upon.

6. Conclusions

First of all, it is clear that *.jar* format hasn't been designed with data restoration in mind. Otherwise, most likely all the above identified flaws wouldn't have been there any longer. Secondly, the restoration operation helps a lot when data corruption appears: it saves both time and money. This option should be taken into consideration when dealing with sensible environments like mobile terminals or critical computational situations (e.g. research and development, scientific laboratories, military applications etc.). The proposed solution takes into considerations such aspects. It lets the final user to eventually change the encryption technique, making it more reliable if required.

The extra storage space needed by the 'symmetrical copy' of the proposed solution tends not to be an issue. Lately, one can choose among a plethora of storage options. Another reason is that the storage space is not expensive anymore. Much storage room is now available in less physical space. Besides, such situations can be foreseen and therefore a proper setup should be taken into consideration in the first place. I can state that the final user (the customer) must be an educated user, being aware about the choices he/she is about to make.

Backup facilities must be in place where supplementary copies must be synchronized with the main data processing steps. This expresses as another form of the redundancy, which is helpful in so many situations. Of course, this backup copy is the result of a process that should take place at certain moments in time, usually subsequent to the major processing steps in a given environment. I

consider that it could be performed using specific time slots, properly chosen so that will not interfere with the main activities of a given computation system. However, this step can be performed periodically, e.g. while in maintenance/backup activities or even during overnight activities, all of them not affecting the main process. Consequently, the solution proposed here is suitable for *periodic tasks*, programmed using a time based process (like the `cron` tool for **nix* systems). However, if no new changes appeared in the necessary input files (after a check over the input fed data - e.g. based on `md5sum` binary tool) then the backup is skipped, no further processing time being required.

Given the reasons identified and presented in this article, I consider that nowadays the data restoration option must always be a valid and valuable choice, or even mandatory where possible.

REFERENCES

[1] *C. Maia, L. Nogueira, L. M. Pinho*,"Evaluating Android OS for Embedded Real-Time Systems*"*, in Proceedings of the 6th International Workshop on Operating Systems Platforms for Embedded Real-Time Applications, Brussels, Belgium, July 2010, pp. 63-70

[2] *W. Mauerer, G. Hillier, J. Sawallisch, S. Honick, S. Oberthur, "*Real-Time Android: Deterministic Ease of Use", Siemens AG, Siemens Corporate Research and Technologies, published on http://www.embedded.com/ on Feb. 2014

[3] *D. Gentry*, "The Six Million Dollar LibC", Nov. 2008

[4] *Murray Eisenberg*, Hill Ciphers and Modular Linear Algebra, Nov. 3, 1999

[5] *Vlad-Alexandru Grosu*, Hill's Cipher: Analysis of the Cryptographic Computational Times in the Eventuality of a Brute-Force Attack, posted in IJISC - Volume 2, Issue 2, published December 2013 (see: http://www.ijisc.com/articles/hills-cipher-analysis-of-the-cryptographic-computational-times-in-the-eventuality-of-a-brute-force-attack/

[6] ***https://www.statista.com/chart/4112/smartphone-platform-market-share/ (last accessed 30.06.2017)

[7] ***http://opensourceforgeeks.blogspot.ro/2015/02/difference-between-dalvik-and-art.html (last accessed 05.07.2017)

[8] ***http://www.h-online.com/open/news/item/Second-Android-signature-attack-disclosed-1918061.html (accessed on 05.06.2014)

[9] ***http://www.saurik.com/id/17 (accessed on 05.06.2014)

[10] ****http://seclists.org/fulldisclosure/2013/Mar/140 (accessed 05.06.2014)

[11] *** MD5 description (RFC 1521), www.ietf.org/rfc/rfc1521.txt (accessed 05.06.2014)