Analysis of the forensic traces left by AirPrint in Apple iOS devices

Luis Gómez-Miralles, Joan Arnedo-Moreno
Internet Interdisciplinary Institute (IN3)
Universitat Oberta de Catalunya
Carrer Roc Boronat, 117 , 7a planta 08018 Barcelona, Spain
pope,jarnedo@uoc.edu

***Abstract*—Since its presentation by Apple, both the iPhone and iPad devices have achieved a great success and gained widespread popularity. This fact, added to the given idiosyncrasies of these new portable devices and the kind of data they may store open new opportunities in the field of computer forensics. In 2010, version 4 of their operating system (iOS) introduced AirPrint, a simple and driverless wireless printing functionality supported by some network printers. This paper presents an analysis of the traces left by AirPrint and assesses whether it is feasible to recover them in the context of a forensic investigation.**

Keywords: Forensics, iPad, iPhone, iOS, Cybercrime, AirPrint, Apple

I. Introduction

Information technologies have grown rapidly in the last decades, changing the way we live, work, and communicate. Portable devices such as smartphones and tablets have evolved from simple phones and agendas into literally full-fledged, always-online computers that fit our pockets, containing huge amounts of valuable data about us: contacts, calendar, e-mails, photographs, as well as a pile of logs: phone calls, chat, geographic positions, etc.

The practice of digital forensics has needed to adapt quickly to the emerging mobile technologies. We once had a homogeneous personal computer market, mainly dominated by a few different Windows versions, with minor representations of Mac OS or Unix-based systems. Now we find that the most personal devices, the ones that always accompany their users and are more prone to contain sensitive information, run software environments which simply didn't exist a few years ago - namely Android and iOS. Furthermore, because of the competitive nature of the market, with each new version of these systems, new functionalities are added in order to appeal to a greater set of users, and thus become their device of choice. However, some of these new features may manage personal user data and are worth analyzing from a forensic investigation standpoint.

This paper focuses on Apple iOS devices (namely, iPhone, iPad and iPod Touch) and, specifically, their capabilities to print wirelessly to compatible printers using the AirPrint technology, presented by Apple in late 2010 [1]. This paper analyzes this relatively recent feature and determines whether using AirPrint to print a document leaves some kind of trace in the iOS device itself which may be open to subsequent forensic analysis, leaving personal user data exposed without their knowledge. Given the popularity and acceptance of iOS based devices [2], any available process which allows to recover user data becomes especially relevant from both a computer forensics and a privacy concern standpoint.

As far as the authors can tell, there has been no research on how AirPrint works behind the scenes and the forensic traces it may leave. Several authors have reviewed the existing, mostly commercial, forensic investigation tools for iOS based devices [3], [4]. However, analysis of AirPrint activity does not seem to be covered by any of the software solutions available for iOS devices. In fact, it may seem odd that something as basic as document printing has been left out of the forensic analysis of mobile devices so far. One must consider, however, that mobile operating systems have lacked common printing frameworks till quite recently. Apple incorporated AirPrint into iOS in late 2010, whereas Android was not provided printing capabilities until early 2011, through Google Cloud Print [5].

This paper is structured as follows. First of all, in Section II, AirPrint and its mode of operation both from a user's and technical standpoint are presented. The results of the analysis of forensic traces left by AirPrint are shown in Section III. Following, in Section IV, the recoverability of such traces and which kind of useful information may be obtained is assessed. Finally, concluding the paper, Section V summarizes the paper contributions and outlines further work.

II. Description of AirPrint network printing

Briefly explained, AirPrint is an iOS feature that allows applications to send content to printers using the iOS device's wireless connection. Directly quoting Apple [1]: *'AirPrint automatically finds printers on local networks and can print text, photos and graphics to them wirelessly over Wi-Fi without the need to install drivers or download software'*.

Apple announced AirPrint in September 2010. Two months later, iOS 4.2 was released for the iPhone, iPad and iPod Touch, being the first iOS version to offer this feature to users. Its standard functionality allows printing only to specific, AirPrint-enabled printers. Nevertheless, as of January 2012, there are more than one hundred AirPrint-enabled printers in the market, from five major vendors (Brother, Canon, Epson, HP and Lexmark) [6]. Apple does not support sharing a common printer via the computer it is connected to, even when

it was possible with some Mac OS X 10.6.5 beta versions; however, it can be done by using software tweaks such as AirPrintActivator [7].

Long before the introduction of AirPrint, different solutions [8], [9] tried to fill in this gap. Usually, such solutions involved iOS applications capable of opening different file formats and sending them to a desktop computer, running a companion application, which would in turn send the document to the printer itself. Some printer vendors developed specific clients, however, none of these solutions were ever widely spread among users. Currently, with AirPrint working out-of-the-box and embedded into all applications, it is hard to believe that new users will consider using a specific, usually paid, application to handle printing, except maybe in some very particular environments, such as cases where the use of these kind of applications was consolidated before AirPrint was launched, or some advanced capabilities are required by power users.

From a user's standpoint, Figure 1 summarizes the printing process, showing the screens it is actually possible to interact with, as seen on an iPhone.

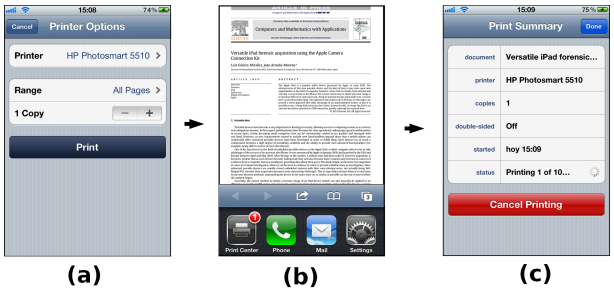

Fig. 1. Step-by-step AirPrint options screen on an iPhone.

In the client side (iOS device), AirPrint-enabled applications contain a "Print" button that, when pressed, will present an extremely simple menu (Figure 1, (a)), with only two or three available options:

1) **Printer:** This option opens a list of all AirPrint-enabled printers found in the local network, showing a "name" and "description" field for each one.
2) **Range:** (optional) Defaulting to "all pages", this option opens a selector which allows the user to choose a range of pages to be printed, rather than all the document.
3) **Copies:** Specifies the number of copies to be printed.
4) Depending on the printer features, additional parameters such as duplex printing can be controlled.
5) **Print:** This button proceeds to send the job to the printer.

The user cannot specify any other kind of information usually available in printing menus, such as paper size or orientation, printing quality, etc. Everything is automatically handled by AirPrint, using some default options. When the user chooses to "Print" the job, the device shows some brief messages (*'Contacting Printer'*; *'Preparing page (...) of (...)'*; *'Sending to Printer'*). However, depending on how long the print job is, these messages may be barely visible or last for several seconds.

After the job has been sent to the printer, the printing menu disappears and the application returns to its previous state. At this point, invoking the list of recently used applications, by double-clicking the device "Home" button, reveals a *Print Center* application (Figure 1, (b)). Unless the user somehow knows this application has started running in background, it may be difficult for him to find it, since no active feedback is provided during the printing process, thus being invisible at casual glance.

Opening the *Print Center* application, the user can see the list of running and pending printing jobs, check their details and cancel them (Figure 1, (c)). When the last job finishes, i.e. the moment the printer ejects the last page, the application closes, and does not appear anymore in the list of recently used applications. As far as it is known, there is no way to open the *Print Center* as a standalone application. It is only executed while there are jobs being printed.

From a technical standpoint, the AirPrint service is known to use the standard IPP protocol at network level for printer management, and Bonjour/Zeroconf [10] for service discovery. A comprehensive description of the printing architecture and its underlaying API in iOS devices can be found in [11].

III. Analysis of Forensic Traces

This section presents the preliminary information that must be considered before more in-depth forensic investigation may proceed. Mainly, assessing which traces are left by the AirPrint subsystem, how they can be discovered, how they behave and which useful information can be obtained from them. All this information was discovered through some basic experiments.

A. Preliminary setup

The following equipment was used for all the analysis and tests throughout this paper:

1) iOS devices: iPhone 3G (8 GB, iOS 4.2.1, multitasking enabled) and iPhone 4 (16 GB, iOS 4.3.3 and 5.0.0), both jailbroken with `redsn0w`.
2) AirPrint-enabled printer: HP Photosmart 5510.
3) Two laptops with 802.11 wireless connectivity.

The iPhone 3G was used so that it would be possible to dump and analyze the filesystem without being affected by the filesystem encryption policy enforced on newer devices. Nevertheless, Bedrune and Sigwald published details about iOS data protection, and shortly after they released the tools and source code capable of breaking this encryption [12]. The device was jailbroken in order to gain root access, and install an SSH server and basic UNIX tools.

Given that iOS enforces the device to run only code signed by Apple (downloaded from the App Store), the process of jailbreaking was used in the tests to bypass that restriction in order to have full access to the devices and be able to run shell commands on them. The jailbreak process is exempted from prosecution under the anti-circumvention section of the

U.S. Digital Millenium Copyright Act [13], and it has been very useful for forensic research in the past [14], [15].

The AirPrint feature depends on the multitasking capabilities of the device. In fact, Apple disabled these feature for some devices in iOS 4 alleging performance issues. However, it is possible to enable them during the jailbreak process using the `redsn0w` tool [16]. By doing so, the chosen device became the perfect testbed for the experiments: a small device (its 8 GB take about 3 hours to transfer via wi-fi), with no encryption, that supports AirPrint. The iPhone 4 was used to confirm that some of the findings still applied on newer devices and under iOS version 5.

For both environmental and budget issues, the best efforts were made to reduce costs and waste. Paper was reused by printing once and again over the same pages, and the life of print cartridges was extended well beyond the limits of readability. Some printers complain on low ink and refuse to continue printing even when they are still giving pretty decent results; luckily, the chosen model kept printing for a long time after the cartridge was empty, and only then it presented a message warning about the warranty issues when continuing printing with empty cartridges. Nevertheless, it is probably obvious that it was necessary to throw away quite a lot of paper during the experiments, as well as quite a few ink cartridges. Anyway, resources were recycled as much as possible.

B. Forensic traces left by AirPrint

Once a root shell is executed, it is possible to invoke several commands before, during and after printing, comparing the results in order to look for remarkable differences. The following commands were executed in the iOS devices:

1) `find / -type (b,c,d,f,l,p,s)` for listing, respectively, all the block special devices, character special devices, directories, regular files, symbolic links, FIFOs, and sockets, in the filesystem.
2) `netstat -an -f inet` for listing any current network connections. This would show active client-server activity, as well as inactive servers awaiting for incoming petitions.
3) `ps aux` for getting information about running processes.

By reviewing the lists of files and directories generated with the `find` commands explained above, it was observed that, when a device prints via AirPrint for the first time, the following folder is created:

```
/var/mobile/Library/com.apple.printd/
```

The moment a document is sent to printing, a new file named `1.pdf` is created under this folder. After running additional tests, it was confirmed that this PDF file exists in disk only while the document is being printed. The moment the printer ejects the last page and considers the job finished, the PDF file is deleted. This is also the moment at which the *Print Center* application disappears from the list of recently used applications.

Printing some documents and copying the resulting temporary PDF files to another location before their deletion, and then examining them, is enough to find that these files are regular PDF files with the same content that is being sent to the printer (no matter whether it was originally in PDF format or not). Hence, an obvious trace is being left in the filesystem, and it reflects exactly what was printed.

It must be noted that, in some of the preliminary tests, before the physical printer was available, a virtual PDF printer was setup in a Mac computer and shared, making it look like an AirPrint printer. It worked as expected, it was possible to print to it from an iPhone, as well as an iPad. However, the printing of the document (in this case, the generation of a file in the hard drive of the Mac) was much shorter than the actual printing of a page through a real printer, with real ink and paper. This greatly reduces the chances of the temporary files being flushed to physical disk from the buffer cache in the iOS device before deletion, and hence their chances of recoverability. Therefore, to obtain accurate results, the experiments need to be performed with a real printer.

C. Properties of the AirPrint temporary files

From the execution of the different tests, the following rules were observed:

1) For every print job sent via AirPrint, a file with the name of `1.pdf` is created in the directory `/var/mobile/Library/com.apple.printd/`
2) This file is in PDF format, containing the document sent to the printer. This observed behavior is consistent across internal iOS applications (Mail, Safari) as well as third party ones (GoodReader, Papers, Keynote...) The only exception found is the iOS Photos app, which seems not to generate any temporary files on disk when printing, thus not leaving these traces.
3) The file `1.pdf` is deleted as soon as the printing job finishes. The timing observed indicates that this happens not just after finishing the task of submitting the job to the physical printer, but after the document has been completely printed.
4) When one job is being printed, subsequent jobs arriving to the queue generate files named `2.pdf`, `3.pdf`, etc. The behavior observed suggests that in iOS 4 the counter resets as soon as the queue is empty (if a new job arrives later it will be named `1.pdf` again), whereas in iOS 5 the counter seems to keep increasing (each new job gets a higher number even if the queue is empty) until the device reboots.
5) When a job asks for more than one copy of the same document, the temporary PDF file contains only one copy of it. There is probably a command, sent from the device to the printer, indicating the number of copies wanted.
6) When a page range is specified, the temporary PDF file contains only this page range. There is an exception when some applications print files that are themselves PDFs, which is studied later in Section III-D.

D. PDF metadata of the temporary files

Using a standard PDF reader, the metadata contents inside different temporary files generated by AirPrint were extracted and compared them. Generally, all the temporary PDF files created by AirPrint can be identified as such by their metadata: they all show the same 'PDF producer' entry (*'iPhone OS x.y.z Quartz PDFContext'*, with *x.y.z* being the iOS version number), and the creation and modification dates both indicate the date and time when the document was sent to the printer (see Figure 2). Therefore, knowing what has been printed from a device looks as simple as recovering deleted PDF files from it and focusing on those with the strings *'iPhone OS x.y.z Quartz PDFContext'*. Moreover, the creation and modification dates contained inside those PDF files in the form of PDF metadata will indicate precisely the printing date and time.

Only one exception was found to this behavior. When printing PDF files, i.e. when the content to be printed is itself in PDF format, some applications behave like described earlier, but others (including iOS built-in applications such as Safari or Mail) actually copy the original PDF file to the `com.apple.printd` directory as `1.pdf`, instead of generating a new PDF file with fresh metadata. In these cases, PDF metadata cannot be used to tell whether one given file is a trace from AirPrint printing: the only peculiar thing about that PDF file is the fact that it resides under such directory.

Considering this, an eventual automated tool aimed at recovering the traces left by AirPrint should go further than just carving for PDF files: it should correctly interpret the internals of the HFSX filesystem and tell whether each recovered file existed within the `com.apple.printd/` directory, or somewhere else.

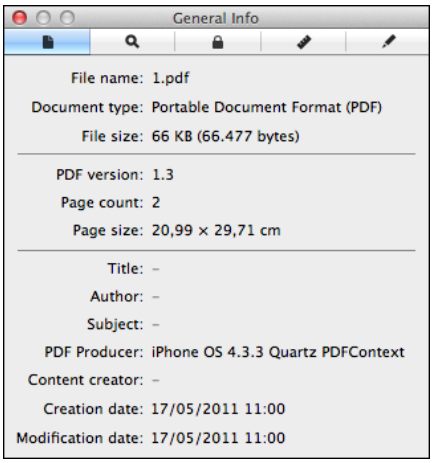

Fig. 2. Metadata of one temporary PDF file generated by AirPrint, as shown by the OS X 'Preview' tool.

IV. Recoverability of AirPrint traces

Given that the temporary PDF files generated by AirPrint only exist in the filesystem for a limited time and are deleted when the document has been printed, the possibility that, depending on disk scheduling and other factors, these files might never be actually flushed to the physical disk must be considered. In that case, they are unrecoverable by a subsequent forensic analysis. This section studies such possibility and assesses the probability that a given trace may be actually recovered at a later time.

The are several factors that increase the chances of the temporary PDF files being flushed to disk, making them potentially recoverable in the future. Some of these factors are:

- Documents that take a long time to be printed because they have many pages or because they contain graphics. Note that, when sending a document to the printer, the user can control very few options: printer, page range, number of copies - nothing else. There is neither an option for printing in 'draft mode', nor one for using only the black cartridge, meaning that everything sent via AirPrint is printed in full color, good quality... and may require quite a lot of time to finish.
- Documents that are sent while there are other printing tasks running.
- Periods of printing interruptions due to the need of human interaction, such as the printer running out of paper or out of ink.

A. Test scenario

In order to test the recoverability of AirPrint traces in the form of deleted temporary files, a series of experiments was run. The goals of these experiments were twofold. First of all, to determine whether the temporary files generated by AirPrint are actually written to the physical disk at some point, and thus may be recoverable after deletion. In addition, to asses whether, even if the previous case is true, each of those temporary files would still be recoverable after generating more of them (i.e. what are the chances that the AirPrint temporary files overlap each other in disk, always overwriting the same space and thus making each other unrecoverable).

Some steps in the testing process involved printing documents, while others were just aimed at simulating some casual user activity in the device (mail browsing, software update, reboot). For those tests that involve printing, 10 documents of a given kind are printed in each test. Five of the tests involve printing, which means that after completing all the tests, 50 documents had been printed.

After each of the tests, a dump of the iPhone filesystem was obtained. For those tests that involved printing, the dump process was not started until the printer had finished printing all of the submitted jobs sent.

Given that the chosen device was an iPhone, the disk image was transferred on-the-fly via wi-fi to an external computer, using the method proposed by Zdziarski , based on the `dd` and `netcat` commands [14]. However, in the case of iPad devices, the image could be transferred much faster to USB storage using a method based on the Apple Camera Connection Kit [17].

Note that the set of tests was carried out in a cumulative way, meaning that given any dump $dumpN$, it could contain

traces of not only the temporary files created during test $testN$, but of all of the temporary files created during the earlier tests as well. Given that there was a total of 50 documents printed, but 10 of them correspond to the Photos application, which does not generate the AirPrint temporary files (as explained earlier in section III-C), the latest dumps should have a maximum of 40 recoverable artifacts.

A detailed description of the whole testing process follows:

1) Use Safari to print 4 web pages and 6 PDF files. Obtain $dump1$. From this filesystem dump, it should be possible to recover up to 10 temporary PDF files generated by AirPrint.
2) Use GoodReader to print 10 PDF files. Obtain $dump2$. This introduces 10 new temporary files, thus there should be a maximum of 20 recoverable AirPrint artifacts in this dump (10 from the previous test, and 10 more from this test).
3) Use iOS built-in Photos app to print 10 photos. Obtain $dump3$. The number of maximum recoverable AirPrint artifacts remains at 20, given that, as explained earlier, printing from the Photos application does not generate temporary files - an exception not observed in any other application.
4) Use iOS built-in Mail app to print 5 e-mails and 5 attachments. Obtain $dump4$. There should be a maximum of 30 recoverable AirPrint artifacts (20 from earlier tests, plus 10 more created during test).
5) Turn the device off, wait for 30 seconds, turn it on. Obtain $dump5$. This does not generate any new AirPrint temporary files, thus at this point the number of maximum recoverable artifacts remains at 30.
6) Use Safari to print 6 DOC and 4 XLS files. Obtain $dump6$. These 10 new temporary files increase the number of maximum recoverable artifacts to 40.
7) Open the Mail app, download messages and large attachments (totaling 15 MB), turn the device off, wait for one minute, and turn it on. Obtain $dump7$. This does not generate any new temporary files, thus the number of maximum recoverable artifacts remains at 40.
8) Open the App Store. Install available software updates (iBooks and GoodReader). Obtain $dump8$. Again, this does not generate any new artifacts.

B. Recovering the traces

Each filesystem dump was analyzed in order to determine whether the temporary files generated by AirPrint at each stage were recoverable, both immediately after their generation, and at later stages.

In order to recover the traces left by AirPrint, the *file carving* [18] technique was chosen. This is a data recovery method consisting of going through a raw data stream (a filesystem dump in this case) looking for possible 'headers' and 'footers' (beginnings and ends) of known, chosen file types, such as JPEG pictures, MPEG video files, PDF documents, etcetera. The more strict a file type specification is, the easier it is for the carving tool to identify and recover that file type. In addition, some tools perform sanity checks such as establishing a file size limit or checking each recovered item against its format specification, to determine whether it may be corrupt.

One of the benefits of the carving technique is that it can be used, with more or less success, over any kind of data, be it a known or unknown filesystem, a portion of it, or something completely different, such as network captures or RAM memory dumps. Note, however, that many tools implement specific strategies for common filesystems in order to improve overall success rate and extract items only from the unallocated space, skipping the disk space used by normal existing files. This is something very useful when the user just wants to focus on recovering deleted files.

This technique was applied by means of the open source, widely used `photorec` tool [19], which has a long track of usefulness in this kind of scenarios [20] [18], even on mobile devices [21].

Every dump obtained during the tests was carved for PDF files using the default `photorec` parameters. Fine-tuning these parameters would have probably improved the recovery success rate, however, after the tests were completed, it was found unnecessary given that the results achieved were indeed very positive.

The results of trace extraction using the photorec tool were analyzed from two different standpoints:

1) **Recoverability**. Tenths of the temporary PDF files generated by AirPrint were successfully recovered from each filesystem dump, which confirms that those files are indeed flushed to disk.
2) **Persistence**. The artifacts created during our first tests were still recoverable after the last tests. This suggests that, probably due to iOS file allocation strategies, the temporary files generated by AirPrint are not very prone to occupy the disk space previously assigned to other of those files.

Figure 3 summarizes the results. The horizontal axis enumerates each of the test steps previously explained in section IV-A, whereas the blue and orange lines indicate, respectively, how many AirPrint temporary files had been introduced at each step, and how many of them were successfully recovered by `photorec`.

From the results of the experiments, it can be concluded that, immediately after printing one or more documents, there is a very high probability that the file will actually be written to physical disk and, thus, recovering the temporary files generated by this process. For instance: 100% of the temporary artifacts created during $test1$ were successfully recovered from $dump1$. All of them were also successfully recovered from $dump2$ (altogether with the new artifacts introduced during $test2$).

Nevertheless, as expected, the recoverability rate decreases as time goes by and the device continues working. In the tests, this can be seen after $test4$ and $test7$. Considering that the artifacts are stored in unallocated space, it is unavoidable that using the device for long periods of time reduces the chance of recovering such artifacts, as new data stored in the device

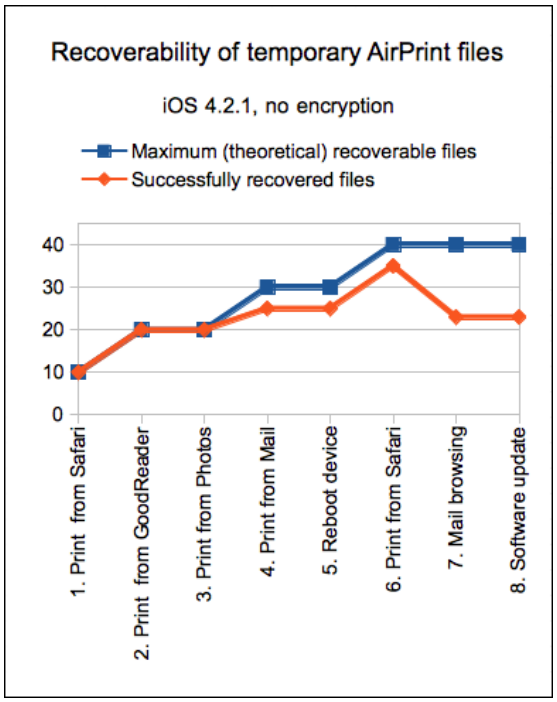

Fig. 3. Recoverability test results

may overwrite that disk space. However, the tests show that the probability does not decrease very quickly, and there is at least a good chance to recover most of the traces.

V. Conclusions and Future Work

This paper analyzes the forensic traces left by usage of the AirPrint functionality in iOS based devices. As a result, it was found that the use of this feature to wirelessly print documents leaves a trace in the device in the form of temporary files, containing a copy of the printed information and indicating precisely when it was printed. The recovery of these AirPrint artifacts can be very valuable from a forensic standpoint in scenarios such as information leaks or inadequate content distribution.

The performed tests show that these traces may be recoverable even after the device continues undergoing activity. Even when the experiments were designed to skip the iOS encryption (optional in iOS 4, enforced in all devices in iOS 5, released in October 2011), these findings could be successfully applied to encrypted devices when combined with the recent undeletion-and-unencryption process implemented by Jerome and Sigwald [12], and hence could be integrated into iOS forensic tools.

In fact, further work includes a more detailed analysis of file recovery in encrypted devices in order to test AirPrint artifact recovery under such scenario. In addition, it would be interesting to assess whether additional information related to the AirPrint capability could be obtained, apart from that directly extracted from the temporary files, such as the specific printer the job was submitted to.

Acknowledgments

This work was partially supported by the Spanish MCYT and the FEDER funds under grant TSI2007-65406-C03-03 E-AEGIS and CONSOLIDER CSD2007-00004 "ARES", funded by the Spanish Ministry of Science and Education.

References

[1] Apple Computer Inc., "Apples AirPrint Wireless Printing for iPad, iPhone & iPod touch Coming to Users in November", 2010, http://www.apple.com/uk/pr/library/2010/09/15airprint.html.

[2] InformationWeek, "iPad is top selling tech gadget ever", 2010, http://www.informationweek.com/showArticle.jhtml?articleID=227700347.

[3] A. Levinson, B. Stackpole, and D. Johnson, "Third party application forensics on apple mobile devices", in *System Sciences (HICSS), 2011 44th Hawaii International Conference on*, 2011, pp. 1 –9.

[4] Andrew Hay, Dennis Krill, Benjamin Kuhar, and Gilbert Peterson, "Evaluating digital forensic options for the apple ipad", in *Advances in Digital Forensics VII*, vol. 361 of *IFIP Advances in Information and Communication Technology*, pp. 257–273. Springer Boston, 2011.

[5] Google Inc., "Google Cloud Print", 2010, http://www.google.com/cloudprint/learn/.

[6] Apple Computer Inc., "iOS: AirPrint 101", 2011, http://support.apple.com/kb/ht4356.

[7] Netputing, "AirPrint Activator", 2011, http://netputing.com/airprintactivator.

[8] EuroSmartz Ltd., "PrintCentral for iPad, iPhone or iPod Touch", 2012, http://mobile.eurosmartz.com/products/printcentral.html.

[9] Avatron Software Inc., "Print Sharing", 2011, http://avatron.com/apps/print-sharing.

[10] D. Steinberg and S. Cheshire, *Zero Configuration Networking: The Definitive Guide*, O'Reilly, 2005.

[11] Apple Inc., "How Printing Works in iOS", 2011, https://developer.apple.com/library/ios.

[12] J. Sigwald and J.B. Bedrune, "iPhone data protection tools", 2011, http://code.google.com/p/iphone-dataprotection/.

[13] U.S. Copyright ONce, "Rulemaking on Exemptions from Prohibition on Circumvention of Technological Measures that Control Access to Copyrighted Works", 2010, http://www.copyright.gov/1201/2010.

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

[15] J.R. Rabaiotti and C.J. Hargreaves, "Using a software exploit to image RAM on an embedded system ", *Digital Investigation*, vol. 6, pp. 95–103, 2010.

[16] iPhone Dev Team, "redsn0w", 2011, http://blog.iphone-dev.org/post/5239805497/tic-tac-toe/.

[17] L. Gomez-Miralles and J. Arnedo-Moreno, "Versatile iPad forensic acquisition using the Apple Camera Connection Kit", *Computers And Mathematics With Applications*, vol. 63, no. 2, pp. 544 – 553, 2012.

[18] M.I. Cohen, "Advanced carving techniques", *Digital Investigation*, vol. 426, no. 3-4, pp. 119–128, 2007.

[19] CGSecurity, "PhotoRec, digital picture recovery", 2009, http://www.cgsecurity.org/wiki/PhotoRec.

[20] S. Garfinkel, "Forensic feature extraction and cross-drive analysis", *Digital Investigation*, vol. 3, no. 1, pp. 71–81, 2006.

[21] I. Pooters, "Full user data acquisition from symbian smart phones", *Digital Investigation*, vol. 6, no. 3-4, pp. 125–135, 2010.