



# A statistical method for detecting on-disk wiped areas

Antonio Savoldi\*, Mario Piccinelli, Paolo Gubian

Department of Information Engineering, DEA, University of Brescia, Via Branze, 38, I-25123 Brescia, Italy

## ARTICLE INFO

### Article history:

Received 12 May 2010

Received in revised form 23 June 2011

Accepted 28 June 2011

### Keywords:

Wiping detection

Anti-forensics

Data wiping

Anti-anti-forensics

Linear classifier

## ABSTRACT

Data-wiping tools are meant to securely erase data. Malicious users may resort to such tools to eliminate traces of a crime they have committed. State-of-the-art wiping detection techniques rely on artifacts left by the use of such tools. However, in certain cases such artifacts can be obfuscated and the investigator is left with almost no clues that could point to a digital crime. Indeed, in this paper we would like to present a scenario involving an ideal data-wiping case (i.e. a method that does not leave any usual exploitable artifacts). In addition, we demonstrate an efficient statistical technique which allows the detection of on-disk wiped areas, both filled with random and periodic data. The performance and usability of the proposed techniques are discussed as well.

© 2011 Elsevier Ltd. All rights reserved.

## 1. Introduction

Digital forensics has initially emerged as an applied discipline and has eventually grown into a science to face the issues posed by cybercrime, by providing certified and reliable methodologies for answering basic questions such as who committed the digital crime, when it happened, and which techniques were used against the system. An example of a well-know illicit action is the broadcasting or downloading of contraband images by means of a computer system. The perpetrator can easily destroy the evidence of the illicit action by means of the plethora of wiping tools, available as commercial and free offerings. Usually, when such a tool is installed/uninstalled, or launched, it generates artifacts such as an installation folder, a prefetch file, various Registry keys, and signatures in the file entry on an NTFS file system. However, with the advancement of the anti-forensics field (Harris, 2006),

defined as the set of methodologies used against a computer system to conceal any evidence that an illicit action has been done, new challenges to the digital forensics science are posed. Particularly, in some cases the use of specific and crafted wiping tools might not leave any ordinary artifacts on a system that can be exploited in a digital investigation.

In fact, the aim of this paper is to discuss a possible ideal data-wiping scenario where usual artifacts left by data-wiping tools are obfuscated. In addition, a statistical procedure on how to detect on-disk wiped areas is outlined and analyzed.

The remainder of this paper is structured as follows. Initially, a brief overview of the data-wiping technology is given, by outlining the basic algorithms that are used on portable wiping tools. This is followed by a discussion on the methodology that highlights artifacts left by wiping tools. Furthermore, an example of artifacts generated by a well-known portable wiping tool is illustrated. Moreover, a perfect data-hiding scenario is presented, where usual artifacts left by the surveyed tool can be easily obfuscated. The article proceeds with a statistical analysis to detect on-disk wiping areas. This paper concludes with a brief overview of related works and some final general observations.

\* Corresponding author. Tel.: +39 303715436; fax: +39 30381014.

E-mail addresses: [antonio.savoldi@ing.unibs.it](mailto:antonio.savoldi@ing.unibs.it) (A. Savoldi), [mario.piccinelli@gmail.com](mailto:mario.piccinelli@gmail.com) (M. Piccinelli), [paolo.gubian@ing.unibs.it](mailto:paolo.gubian@ing.unibs.it) (P. Gubian).

## 2. Data-wiping technology: background and possible artifacts

In this section we discuss wiping technology, the most renowned algorithms, and the related artifacts that are usually left on non-volatile memory (e.g. in a hard disk). A wiping tool is conceived to irretrievably erase data stored on a storage device. Usually, the deletion process mediated by an ordinary operating system (e.g. Windows XP SP3) removes only the reference (pointer) of the file to the metadata area (Carrier, 2005). As a consequence, state-of-the-art wiping tools deploy different erasing strategies that affect both metadata and data areas. Wiping algorithms write different binary patterns on the disk areas occupied by files that should be erased. For instance, the US DoD 5220.22-M (8-306./E) is a three pass overwriting wipe algorithm. During the first pass the data on the disk is overwritten with zeroes, during the second pass with ones and, finally, during the third pass with random data. Other simpler strategies might involve the use of fixed patterns, such as '00' (e.g. by means of `/dev/zero` device in the Linux operating system) or patterns which are defined by the user (e.g. 'AA'). As discussed in Kim et al., submitted for publication according to the wiping tool we might expect different artifacts at the level of the file system such as the creation of an installation folder, the creation of a prefetch file, the generation of registry keys, and different signatures in the file entry of the deleted file according to the wiping tool. Possible wiping algorithms, for instance those that can be used in the portable tool Eraser (Trant, 2009), are the following:

- Gutmann (35 passes)
- US DoD 5220.22-M (8-306./E, C and E) (7 passes)
- US DoD 5220.22-M (8-306./E) (3 passes)
- Pseudorandom Data (1 pass)
- Only first and last 2 KB
- Schneier's 7 pass (7 passes)
- User defined (the wiping algorithm can be defined by the user and Gutmann's algorithm is set by default)

Every wiping algorithm uses different patterns which are written on the disk areas to erase. With respect to the previous tool we demonstrate that the majority of available algorithms generate random data, that is the wiped data on the storage device is randomly distributed. In Section 4.4 we will substantiate this claim in great detail, by justifying which of the mentioned wiping algorithms are capable of generating random data.

## 3. A methodology for highlighting wiping tool artifacts

In order to understand which is the full set of artifacts left by a tool in general, and by a data wiping one in particular, we can set up a general analysis framework. Normally, when a tool runs on a system, the related modifications, or artifacts, can be at the level of non-volatile and volatile information. A wiping tool produces tangible effects on the file system. Therefore, we would like

to monitor such modifications with a general analysis method that might also be used for any other counter-forensic tool.

When the interaction between a tool and the operating system is not well known, in terms of possible artifacts creation, we would like to have a method that logs all the changes non-volatile and volatile memory undergo. For this purpose, a process-monitoring tool, such as *Process Monitor* (Russovich and Cogswell, 2010), might be used. This tool is an advanced monitoring tool for Windows that shows real-time file system, Registry and process/thread activity. In addition, it includes filtering, comprehensive event properties such session IDs and user names, reliable process information, full thread stacks with integrated symbol support for each operation, and simultaneous logging to a file. Therefore, we might set up this logging system to monitor the file system changes as a consequence of installation, use, and removal of different wiping tools. First of all, we created a new virtual box, equipped with Windows XP SP3, 1 Gb RAM and 10 Gb hard drive (with NTFS file system), and without any additional software. After that, we set up the monitoring tool for logging the file system and Registry changes.

As Fig. 1 illustrates, we can start the analysis at  $T_0$ , by activating the monitoring tool. At that time, the installation of a wiping tool starts, and all the artifacts caused by the installation are logged. At a further instance of time, say  $T_1$ , the tool is launched, and a testing file of 1 Gbyte (e.g. `ex01.mkv`), containing a multimedia file, is deleted with a specific operational mode that is available for the wiping tool (e.g. US DoD 5220.22-M (8-306./E)). The removal of the tested wiping tool starts at  $T_2$  and ends at  $T_3$ . As the figure shows, the differences between the file system and Registry logs, namely  $\Delta S_{10}$ ,  $\Delta S_{21}$ , and  $\Delta S_{32}$ , captured by the process-monitoring tool, take into account all the artifacts that are actually involved in different phases, such as installation, runtime, and tool removal. Furthermore, this method can be applied whenever we would like to know all the artifacts caused by any counter-forensic tools on a system.

Following the previous methodology, the artifacts caused by any portable data-wiping tool can be detected. In fact, there are examples of data-wiping tools which do not require any installation and can be launched from any possible external storage device (e.g. a USB pen drive). As Fig. 2 shows, at  $T_0$  time an external storage device where the data-wiping tool is stored is plugged in the system. At this time, the monitoring tool is active to capture all the artifacts that the tool produces on the system. After that, the tool runs and wipes a test file (e.g. `ex01.mkv`). Ultimately, at  $T_1$  both the tool and the monitoring process are terminated. As a result, the log  $\Delta S_{10}$ , obtained with the process-monitoring tool, is able to quantify all the artifacts that the tool caused on the system.

### 3.1. An example of portable wiping artifacts

Let us consider a practical example involving the use of a well-known portable data-wiping tool, namely *Eraser Portable* (Trant, 2009). The bundle has a few files, as illustrated in Table 1. The tool can be used both from

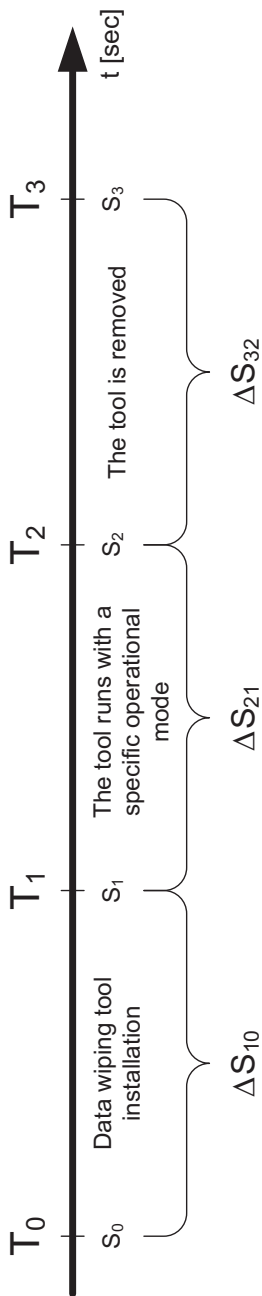


Fig. 1. Timeline of the experiment that quantifies the artifacts caused by an installed data-wiping tool.

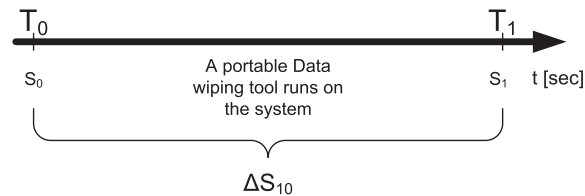


Fig. 2. Timeline of the experiment that quantifies the artifacts caused by a portable data-wiping tool.

command line (`Eraser1.exe`) and from a graphical user interface (`Eraser.exe`), which is a wrapper of the former tool. Listing 1 illustrates an example of possible use of such tool from command line. The target file is `ex01.pdf`, the erasing method is US DoD 5220.22-M (8-306./E) (3 passes), and there is an explicit indication of no graphical user interaction (`-silent` option). In addition, the test was done on the virtual machine, as described in Section 3.

As already pointed out in a previous research work (Kim et al., submitted for publication), a portable wiping tool might leave artifacts in the prefetch folder, in the Registry, in the metadata area, and on the data areas of the storage device where the tool has been run. When using `eraser1.exe` command line tool, one of the artifacts left on the storage device where the tool acts is the prefetch file (e.g. `ERASERL.EXE-35096D64.pf`). However, when launching the GUI wrapper, namely `eraser.exe`, the `ERASERL.EXE-215B4C49.pf` prefetch file is created. In addition, several registry keys are created as well, as can be seen in Listing 2.

So far, we have verified that when using the command line tool `eraser1.exe`, there are no artifacts left at the level of the registry. The only artifact left is the prefetch file. Also, by looking at the file entry of the erased file in the Master File Table before and after the wiping operation (Figs. 3 and 4), we can notice that the original name within the entry, namely `ex01.pdf`, is substituted by another name, which in this case is `imjp81k.dll`. This file also appears as a deleted one on the volume where the wiping test was done (e.g. `F:\imjp81k.dll`). We repeated this procedure several times and obtained different names such as `amstream.dll`, `MsCtfMonitor.dll`, and `mapi32.dll`.

Apparently, these file names are not predictable and change every time a file is deleted on a volume. Thus, it is not clear whether this can represent an exploitable fact to

Table 1  
Eraser portable bundle anatomy.

File Name	Function
<code>COPYING.txt</code>	Tool license
<code>Eraser.dll</code>	Dynamic library
<code>Eraser.exe</code>	Graphic User Interface (GUI) of <code>Eraser1.exe</code> command tool
<code>Eraser1.exe</code>	Basic tool which can be launched from shell
<code>ErsChk.exe</code>	GUI wrapper to set the basic erasing algorithm
<code>shedlog.txt</code>	Log file which records the tool starting and ending
<code>eraser.ini</code>	Configuration file
<code>default.ers</code>	Last erased file

**Listing 1**

An example of how to use Eraser.exe tool.

```
Eraser1.exe -file ex01.pdf -method 'DoD_E' -silent
```

infer that the *Eraser Portable* tool was used. Also, when using the mentioned tool to erase a multitude of files (e.g. 1000 files within a directory), all the original names in the master file table entries are overwritten by randomly generated names which do not have a predictable pattern. In addition, the same files can be found as deleted on the file system by means of a forensic tool such as *EnCase*. However, being the file name not predictable, it is quite difficult to associate those artifacts with the wiping tool.

### 3.2. A perfect data-wiping scenario

When the *Eraser1.exe* tool is used, only a prefetch file is generated. We further present a plausible scenario where the tool is renamed in a way that the resulting artifact is changed. First of all, we need to understand how a prefetch file is generated.

It is well known that prefetch files are used to proactively load the data in the memory in order to improve software performance (Bunting et al., 2007) (Bosschert, 2006). If the operating system is Windows XP or a later version, a prefetch file is created in the %SYSTEMROOT%\Prefetch\ folder. A prefetch file has the general form of filename.exe\_[HASH].pf (e.g. ERASERL.EXE-35096D64.pf), where [HASH] is the hash value which changes according to the location of the file itself. More details about the hash function are discussed by (Khatri, 2009). In addition, if the prefetch file of an application (e.g. *Eraser1.exe*) is deleted, another prefetch file with a different hash value is created. Prefetch file analysis is important since it can give the investigator clues about whether a certain tool was used at a certain time,

and the number of times it was launched. As shown in Fig. 5, there are different fields in a prefetch file, among which are the three grayed items that are particularly important as they show whether the tool was used. The *File Name* field contains the name of the executable (e.g. notepad.exe). *Last Execution Time* refers to the last time when the tool was used. More precisely, it uses Windows FILETIME function and stores the UTC (Coordinated Universal Time) time (Carvey, 2007). Ultimately, *Counter* is the number of times the tool was run.

Having discussed the basic structure of a prefetch file, we demonstrate a basic scenario where the wiping tool is renamed in a way that the resulting artifact, that is the prefetch file, is changed accordingly. In Listing 3 we can observe the result of analysis performed on three prefetch files. The analysis was done by means of Windows File Analyzer (Windows File Analyzer, 2009) and Windows Prefetch Folder (Milo09, 2009). There are similar available tools which parse the binary structure of the prefetch file. The wiping tool was used to erase a file, *ex01.pdf*, which was located on an external USB pen drive. In the first case the *eraser1.exe* tool was used, which was the original one without any modifications. The analysis shows the name of the file, MD5 hash value, last accessed time, last modified, creation time, and number of runs. In the second case the tool was slightly modified by adding the surname of the author in the slack space of the binary. The prefetch file of the first case was deleted and a new one was therefore generated by the Windows operating system. In the third case, we renamed the wiping tool of the second case from *eraser1.exe* to *savo.exe*. In addition, the previous prefetch file was deleted and the new renamed tool was launched. As a result, the prefetch file *savo.exe-30dbbc47.pf* was generated. Meanwhile, the system (Registry and file system) was monitored by the Process Monitor tool. For this purpose we set up simple filters which allowed to trace any system modification related to

**Listing 2**

Registry artifacts caused by Eraser1 tool.

```
HKEY CURRENT USER\Software\Microsoft\Windows\CurrentVersion\Explorer\FileExts
\ .pdf\OpenWithList\
key name:C Data: Eraser.exe

hive:
HKEY CURRENT USER\Software\Microsoft\Windows\ShellNoRoam\MUICache
key name: E:\Eraser Standalone\Eraser.exe Data: Eraser

hive:
HKEY USERS\S-1-5-21-117609710-1035525444-682003330-500\Software\Microsoft\
Windows\CurrentVersion\Explorer\FileExts\ .pdf\OpenWithList
key name:C Data: Eraser.exe

hive:
HKEY USERS\S-1-5-21-117609710-1035525444-682003330-500\Software\Microsoft\
Windows\ShellNoRoam\MUICache
key name: E:\Eraser Standalone\Eraser.exe Data: Eraser
```

```

035607 .....
035750 ..... FILEO.....1F0.....8.....X.....#.....
035893 .....H.....U4iU8.....iO8.....iO8.n6e8iU8.....0.....p.....R.....D0 U4iU8.D0 U4
036036 iU8.D0 U4iU8.D0 U4iU8.....e.x.0.1.....p.d.f.....H.....«.....@.....J.....I4J.....I4J.....yyy
036179 y0 yG.....
036322 .....
036465 .....
036608 .....
036751 ..... FILEO.....i.....8.....@.....
036894 .....$.....yyy.....
037037 .....

```

Fig. 3. NFTS entry related to file ex01.pdf.

```

035607 .....
035750 ..... FILEO.....2[i.....8.....X.....#.....
035893 .....H.....C.s\q".48P*iU8.48P*iU8.48P*iU8.....0.....p.....C.s\q".C.s
036036 \q".w0±iU8.C.s\q".....i.m.j.p.8.1.k..d.1.1.0.....H.....E.....@.....p.....8....."C.U.....yyy
036179 y0 yG.....
036322 .....
036465 .....
036608 .....
036751 ..... FILEO.....i.....8.....@.....
036894 .....$.....yyy.....
037037 .....

```

Fig. 4. NFTS entry after the file ex01.pdf has been wiped.

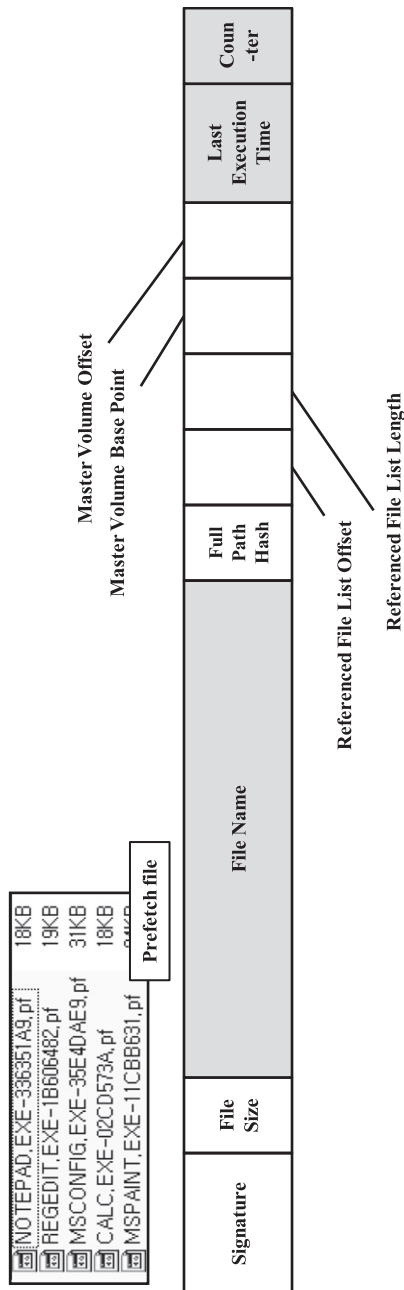


Fig. 5. A detailed view of a prefetch file.

### Listing 3

Analysis of prefetch files related to eraserl tool.

```

1)
Filename: eraserl.exe-00e5f760.pf
MD5: 257904afd3d551a976a3a6b249e2b2e4
Last accessed time: Wed Apr 14 15:11:38 2010
Last modified date: Thu Apr 15 00:11:46 2010
File creation date: Thu Apr 15 00:11:46 2010
Total number of runs: 1

2)
Filename: eraserl.exe-35096d64.pf
MD5: 5cfee18b32f37e992ef63e22ff656097
Last accessed time: Wed Apr 14 15:07:45 2010
Last modified date: Thu Apr 15 00:07:53 2010
File creation date: Thu Apr 15 00:14:56 2010
Total number of runs: 1

3)
Filename: savo.exe-30dbbc47.pf
MD5: f26e9dd869337f3bb70387742886c740
Last accessed time: Wed Apr 14 15:19:49 2010
Last modified date: Thu Apr 15 00:19:57 2010
File creation date: Thu Apr 15 00:19:57 2010
Total number of runs: 1

```

the mentioned processes (*eraserl.exe* and *savo.exe*). The result was that the only tangible artifact on the system (Registry and file system) was the creation of the mentioned prefetch file *savo.exe-30dbbc47.pf*. The analysis of this prefetch file apparently does not provide any clues about the nature of the tool (i.e. whether the tool is legal or illicit such as a malware).

So far, we have illustrated a simple, albeit seemingly effective, solution to conceal artifacts created by a portable wiping tool, namely *eraserl.exe*. The renaming of the tool creates other artifacts, which cannot be easily related to the wiping activity.

Another solution which might avoid the generation of any artifacts (e.g. Registry keys or prefetch files) is represented by the U3 platform (Bosschert, 2006), which is a combination of a hardware platform, a launchpad, and U3-compliant tools:

- Hardware platform: a U3 flash drive presents itself to the host system as a USB hub with a CD drive and a standard USB mass storage device attached to it;
- Launchpad: it is a Windows program manager that is preinstalled on every U3 smart drive;
- U3 tools: to be fully U3 compliant, an application has to be programmed to clean up its own data from the local machine. It must also be packaged in U3's special program format. U3 applications will only run from a U3 device.

In this way, a U3-compliant wiping tool (e.g. a re-engineered version of *Eraser Portable*) acts on a live system without generating any tangible system-level artifacts (i.e. any modifications at the level of the Registry or file system).

### 3.3. Other elements that might be used in the analysis

Below we outline a scenario where a malicious user can wipe out relevant traces related to the use of data-wiping



tools by acting on a pagefile, RAM contents, and a hibernation file.

- **pagefile:** the pagefile is a fundamental component of virtual memory in every modern computer-based system. On the Windows platform, a user can wipe (zero) the pagefile at the shut-down of the system.<sup>1</sup> This solution is valid for Windows XP, Vista, and Seven OSs). In this way, any trace of illicit actions (e.g. contraband images that might have been downloaded and seen) can be wiped out.
- **RAM contents:** it is well known that RAM contents persist, even though the system is rebooted and thus they can be recovered (Halderman et al., 2008). However, the RAM contents can be wiped out by forcing the system to allocate memory pages with known contents (e.g. allocation of memory blocks of 1 Mb with zeros). Also, BIOS can perform a wiping of the RAM when the fast POST option is deselected.
- **hibernation file:** traces of a malicious user's activity might be found on a hibernation file since the RAM is copied in this file as soon as the system goes in hibernation mode. By avoiding hibernation, the related analysis cannot be performed.

The following sections introduce a statistical methodology to figure out whether a wiping tool might have been used against any storage device.

#### 4. On-disk wiping area detection

One quite trivial question, albeit fundamental, is whether there is a reliable method to detect on-disk wiped data areas in the unallocated space. First of all, we need to set up some basic terminology and definitions.

- **Unallocated on-disk area:** also referred to as free space. This term is used to describe disk sectors or clusters of disk sectors that are on the file system's "free list" and can be allocated to newly created files (Garfinkel and Malan, 2006).
- **On-disk wiped data area:** is defined as an unallocated area which has been overwritten by the wiping tool, with a specific wiping algorithm.

The basic idea that we wanted to verify is that all wiped areas created by means of wiping algorithms that generate random data indeed have random contents.<sup>2</sup> This statement has been proved by using two techniques which detect data area filled with random data.

In the remainder of the section we survey two methods to assess whether a data fragment is wiped or not. The first method, which is used as a reference approach, involves an entropy test. The second method, which is

**Table 2**

Test set used in the experiments.

File type	# of chunks
directory	000
csv	608
doc	8462
gif	14
gz	1022
html	3905
jpg	305
log	1795
pdf	41280
png	4
pps	569
ppt	16472
ps	7179
text	32255
xls	11561
xml	1485
total	126916

based on multiple statistical tests, provides very promising results.

##### 4.1. Training set

In order to test our classifiers, we looked on the Internet for publicly available repositories which offer files that can be used for such a case. Our purpose was to find samples of standard files (e.g. pdf, jpg, ppt) and possibly wiped ones as well. Unfortunately, it turned out that only standard files were available. Interestingly, we found a recently established repository, namely Digital Forensic Corpora (Garfinkel and Malan, 2006), which was set in order to have a common database that can be used for research and educational purposes<sup>3</sup>. From that repository we downloaded directory 000 of 609 Mb. From the directory we cataloged fifteen different types of files, as illustrated in Table 2. In addition, 4 Kb-sized chunks were created from every file. The resulting set, namely 000, is called *standard set* because it refers to ordinary file types. The basic idea was to be able to analyze data units of the size of an average cluster to make the analysis more realistic. Moreover, from the generated chunks we removed all the clusters whose size was less than 4 Kb. This was done in order to perform the statistical analysis as is detailed further below.

Since we did not find any public repository with wiped generated files, we created such file by using the following procedure. Initially, an SD card was wiped out (zeroed) by means of dd tool (`dd if=/dev/zero of=/dev/sdb1 bs=1M`). After that, the card was formatted NTFS. Next, a file of 1119.57 Mb (`ex01.mkv`) was copied onto the card. Then, the physical location of the file on the storage device was recorded by means of EnCase v.4 (Start cluster: 313,376 and Number of Clusters: 286,612). In this way we verified that the file was allocated by using consecutive clusters since the disk was not fragmented and no files had previously been allocated. At the same time, the file entry of the `ex01.mkv` in the MFT (Master File Table) was verified, as

<sup>1</sup> In HKEY\_LOCAL\_MACHINE\SYSTEM\CurrentControlSet\Control\Session Manager\Memory Management Registry key set `ClearPageFileAtShutdown` key to one.

<sup>2</sup> For the sake of simplicity, we assume that wiping algorithms produce random data.

<sup>3</sup> <http://domex.nps.edu/corp/files/govdocs1>.

shown in Fig. 3. Eventually, the file was wiped by means of the US DoD 5220.22-M (8-306/E) algorithm, which uses 3 passes by default. Finally, the on-disk wiped area was recovered by means of (`dd if=\\.\F: of=.\mkv_wiped skip=313,376 count=286,612 bs=4096`). In this way, an example of a wiped file was obtained and used to generate chunks for the statistical analysis (286,592 chunks were obtained). From this set, referred to as *wiped set*, 94,378 wiped fragments were considered.

#### 4.2. Entropy-based classifier

A very basic approach to evaluating whether a fragment of wiped data (e.g. 4 Kb sized) looks like a random one might be to consider its entropy value, where the entropy function can be defined as  $E = \sum_{i=1}^{255} P_i \times \log_2 P_i$  (i.e.  $E$  is expressed in bits per symbol, and  $P_i$  represents the frequency of each symbol within the data fragment). It is a well-known fact that every file category (e.g. jpg, txt) has its own average entropy value. For instance, data fragments of text and html typically have entropy values between 4 and 6, but data fragments of compressed (gzip) and jpeg have entropy values between 7 and 8, as can be noticed in [Table 3](#). Since we were interested in discriminating wiped data chunks from the other types (e.g. pdf, jpg, txt, gz, etc.), we implemented a linear classifier based on the following algorithm:

- Foreach directory (i)
- Calculate the sample average (equation (1)) and the sample variance (equation (2)) ( $\hat{\mu}_{E_i}, \hat{\sigma}_{E_i}^2$ ).
- Calculate the confidence interval defined as  $C_i = [\hat{\mu}_{E_i} - 3\hat{\sigma}_{E_i}, \hat{\mu}_{E_i} + 3\hat{\sigma}_{E_i}]$ , that is  $C = [lwb, upb]$ , where  $lwb$  is the lower bound of the interval, whereas  $upb$  is the upper bound. The confidence interval is set to 3 times  $\sigma$  (equation (3)) to take into account 99.7% of the values under the assumption that entropy values follow a normal distribution.

$$\hat{\mu}_E = \frac{1}{n} \sum_{i=1}^n E_i \quad (1)$$

$$\hat{\sigma}_E^2 = \frac{1}{n-1} \sum_{i=1}^n (E_i - \hat{\mu}_E)^2 \quad (2)$$

$$\hat{\sigma}_E = \sqrt{\frac{1}{n-1} \sum_{i=1}^n (E_i - \hat{\mu}_E)^2} \quad (3)$$

Having calculated the confidence interval for the wiped directory, the classification algorithm is the following. If the entropy value of data fragment  $j$  ( $E_j$ ) lies in  $C_{WIP}$  (i.e. if  $E_j$  is above the lower bound of  $C_{WIP}$ ), then the fragment is classified as wiped. Conversely, the fragment is classified as non-wiped (standard).

- Foreach directory ( $i$ th)
- Foreach file (chunk) ( $j$ th)
- if  $E_j \geq \text{lwb}_{WIP}$  then the  $j$ -th fragment is wiped
- else the  $j$ -th fragment is not wiped

In order to test the entropic classifier we implemented a Perl script (`statEntropy.pl`) which scans a directory provided as an input parameter, to find out whether fragments are wiped or standard. This type of classifier is a discrete one since it provides a binary answer (i.e. the fragment can be predicted as wiped or standard). In addition, we determined the confidence interval of the wiped class to set a threshold that is used by the classifier (i.e. fragments whose entropy value is below the threshold are classified as standard, whereas the ones over the threshold are classified as wiped). The threshold was calculated by doing an entropy analysis on the wiped set and the lower

**Table 3**  
Entropic classifier confusion matrix (threshold = 6.9647).

[illegible]



bound ( $lwb_{wip} = \hat{\mu}_{E_{wip}} - 3\hat{\sigma}_{E_{wip}}$ ) has been determined ( $lwb = 6.9647$ ). Table 3 illustrates the experimental results performed against the standard and wiped sets (wip and 000 directories). We can notice that the true positive rate (i.e. classification of a fragment as wiped) is 98.74%, which is a significant result. However, the true negative rate (i.e. classification of a fragment as standard) is remarkably low, being only 46.20%. This result is not surprising since we have many file types (e.g. gif, gz, jpeg, pdf, png and ppt) whose sample average entropy lies in the range 7–8, which is too close to the one of the wiped set. In other words, there are too many file types whose confidence interval overlaps with the wiped ones and this fact, as a result, causes a rate of false positive (i.e. standard fragment classified as wiped) of 46.20%. The net consequence is that this type of classifier, which was considered only as a reference test method, is not good for our purposes. The test was done on a virtual box equipped with 1 Gb RAM, 20 Gb hard disk, and Linux Suse 11.2 operating system. The computation time, shown in Table 3, is quite good, with an average of 60 analyzed fragments per second.

#### 4.3. NIST-based classifier

Being aware of the inefficiency of the entropic classifier, we propose a better system to discriminate between wiped and standard sets. The National Institute of Standards and Technology (NIST) developed a statistical package (Rukhin et al., 2008) consisting of fifteen tests that evaluate the randomness of arbitrary long binary sequences produced by either hardware or software based cryptographic random or pseudorandom number generators. These tests focus on a variety of different types of non-randomness that could exist in a sequence. The entire set of tests is illustrated in Table 4 and briefly discussed in the Appendix of the paper.

One of the prerequisites for applying those tests is to check the length of the binary sequence ( $n$ ). From the NIST manual (Rukhin et al., 2008), we verified that the minimum length varies according to the test itself. This is a very important detail that we must take into account in order to select which tests to apply to a data fragment according to its size. In the next paragraph we provide an overview of

the NIST statistical package, which was used to implement a better classifier.

#### 4.4. NIST package

The NIST statistical test suite (Banks et al., 2008) is written in ANSI C and is portable, flexible, and can be easily adapted according to the context. Once the package has been compiled, a tool named *assess* is created. It accepts as a parameter the bit stream length  $n$ , which defines the length of a run (i.e. binary sequence) that will be used in the analysis. In addition, the tool displays a series of menu prompts in order to select the data to be analyzed and the statistical tests to be applied. More precisely, the user must specify the generator option (e.g. input file to analyze), the subset of statistical tests to use, the number  $m$  of sequences that should be considered in the analysis (i.e. the number of sequences generated during the analysis starting from the input file), and the input file format (e.g. ASCII or binary). In order to automate the statistical analysis process as much as possible, we fixed the source code of the package by removing the user interaction part. The resulting tool can accept any file of a specified length  $L$  (i.e.  $\{L|n \times m \leq L\}$ ), that is the product between the bit stream length  $n$  and the number of sequences  $m$  must be lower or equal to  $L$  expressed in bits. By focusing the analysis on cluster-sized fragments (e.g. 4 Kb) we have  $L = 4 \times 1024 \times 8$ , that is  $L = 32,768$  bits. Accordingly, we fixed the bit stream length  $\hat{n}$  to 320 bits and the number of bit streams  $\hat{m}$  to 100 ( $\hat{L} = 32,000$ , which is less than the calculated limit). In this way we respect the previous constraint.

An important step to follow is to select which tests to apply on a generic input sequence (e.g. data fragment) in order to evaluate its randomness. From Table 4, we can notice that different tests require different bit stream length. By focusing on 4 Kb sized fragments, we have set  $\hat{n}$  to 320 bits. As a consequence, we selected a subset of statistical tests, as illustrated in Table 5.

##### 4.4.1. Background

In order to understand the motivation behind the use of the NIST statistical package, we provide a brief theoretical overview of the topic. The implicit assumption that we made is that a wiped fragment contains random generated data, which is a direct consequence of algorithms used in the wiping process. As a consequence, it appears reasonable to use a method (e.g. the NIST statistical package) which is able to predict whether a fragment contains random data.

**Table 4**  
NIST statistic tests.

#	Test name	Minimum length $n$ [bits]
1	Frequency (Monobit)	100
2	Frequency test within a block	100
3	Runs test	100
4	Longest-run-of-ones in a block	128
5	Binary matrix rank	38,912
6	Discrete fourier transform (DFT)	1000
7	Non-overlapping template matching	1,000,000
8	Overlapping template matching	1,000,000
9	Maurer's universal statistical	387,840
10	Linear complexity	1,000,000
11	Serial test	1,000,000
12	Approximate entropy	100
13	Cumulative sums	100
14	Random excursions	1,000,000
15	Random excursions variant	1,000,000

**Table 5**  
Statistical tests which have been used in the classifier.

#	Test name	Minimum length $n$ [bits]
1	Frequency (monobit)	100
2	Frequency test within a block	100
3	Runs test	100
4	Longest-run-of-ones in a block	128
5	Approximate entropy	100
6	Cumulative sums	100

**Table 6**  
Hypothesis testing.

True situation	Accept $H_0$	Accept $H_a$ (Reject $H_0$ )
Fragment is random (wiped) ( $H_0$ is true)	No error	Type I error
Fragment is non-random (standard) ( $H_a$ is true)	Type II error	No error

In fact, there are different statistical tests which can be applied to a sequence to verify its randomness. These tests are based on detecting the presence or absence of a particular pattern (dependent on the particular test) which, if detected, would indicate that the sequence is non-random. It is important to point out that there is no specific finite set of tests to verify with certainty that a sequence is truly random. To accomplish this task, a statistical test is formulated to test a specific *null hypothesis* ( $H_0$ ) that the sequence being tested is wiped. Along with the null hypothesis, there is the *alternative hypothesis* ( $H_a$ ) that the sequence is not wiped (e.g. standard). In addition, the statistical hypothesis testing is a conclusion-generation procedure that has two possible outcomes: either accept  $H_0$ , that is the data is wiped, or accept  $H_a$ , that is the data is standard (Table 6).

The probability of a Type I error is often called the level of significance of the test and is denoted with  $\alpha$  and common values of  $\alpha$ . For instance, in cryptography  $\alpha$  values are about 0.01. The probability of a Type II error is denoted with  $\beta$ . This parameter can take on many different values because there is an infinite number of ways that a data stream can be non-random, and each way yields a different

$\beta$ . The calculation of this parameter is more difficult than the calculation of  $\alpha$  because of the many possible types of non-randomness.

The test statistics is used to calculate a  $P$ -value that summarizes the strength of the evidence against the null hypothesis. A  $P$ -value of 1 indicates that the sequence appears to have perfect randomness. A  $P$ -value of 0 indicates that the sequence appears to be completely non-random. Having set a level of significance, say  $\alpha = 0.01$ , if  $P\text{-value} > \alpha$ , then the null hypothesis  $H_0$  is accepted (i.e. the fragment is considered as random).

#### 4.4.2. An example

An example of report generated by the NIST statistical test tool (assess) is provided in Listing 4. The report contains a summary of empirical results. The results are presented in a table with  $p$  rows and  $q$  columns. The number of rows,  $p$ , corresponds to the number of statistical tests applied (e.g. 6 in our case). By looking at Listing 4 the number of columns,  $q = 13$ , are distributed as follows: columns 1–10 correspond to the frequency of  $P$ -values (i.e. the probability interval is divided into 10 equal bins), column 11 shows  $P$ -values that come from the application of a chi-square test (i.e. to establish the uniformity of  $P$ -values in the  $i$ -th statistical test), column 12 shows the proportion of binary sequences that passed the test (e.g. 1.0000 = 100% of sequences passed the test), and the 13th column demonstrates the corresponding statistical test name. We can notice that `CumulativeSums` is reported twice in the report: this is the standard behavior of the tool (Table 6).

### Listing 4

A report generated by assess tool for a standard (CSV) fragment difficult than the calculation of  $\alpha$  because of the many possible types of non-randomness.

RESULTS FOR THE UNIFORMITY OF P-VALUES AND THE PROPORTION OF PASSING SEQUENCES														
generator is <data/split/000/csv/001055.csv_000007>														
C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	P-VALUE	PROPORTION			
STATISTICAL TEST														
100	0	0	0	0	0	0	0	0	0	0.000000	*	0.0200	*	Frequency
97	1	1	1	0	0	0	0	0	0	0.000000	*	0.0900	*	BlockFrequency
100	0	0	0	0	0	0	0	0	0	0.000000	*	0.0200	*	CumulativeSums
100	0	0	0	0	0	0	0	0	0	0.000000	*	0.0300	*	CumulativeSums
78	3	2	2	4	1	1	2	2	5	0.000000	*	0.2400	*	Runs
91	8	1	0	0	0	0	0	0	0	0.000000	*	0.4300	*	LongestRun
0	0	0	0	0	0	0	0	0	100	0.000000	*	1.0000		
ApproximateEntropy														
-----														
The minimum pass rate for each statistical test with the exception of the random excursion (variant) test is approximately = 0.960150 for a sample size = 100 binary sequences.														
For further guidelines construct a probability table using the MAPLE program provided in the addendum section of the documentation.														
-----														



**Listing 6**

Classification algorithm.

```

For each Report{
  Read P-value and Proportion from the report of the i-th analyzed chunk
  if ( (P-value) >= alpha) && (Proportion >= LowerBoundConfidenceInterval) ){
    %Testpassed
    counterTestPassed++;
  }
  else {
    %TestNotPassed
    counterTestNotPassed++;
  }
}

if (counterTestPassed >= Threshold){
  fragment i-th is wiped
}
else {
  fragment i-th is standard
}

```

considering that 12.2 fragments per second are analyzed. The parsing time (i.e. computational time required to detect different fragments) is negligible if compared with the analysis one.

**4.4.5. Classifiers performance**

One approach to verify a binary (or discrete) classifier performance is by means of an ROC (Receiving Operating Curve) graph, which depicts relative tradeoffs between benefits (true positives - tp) and costs (false positives - fp) (Fawcett, 2006). A discrete classifier, like the one presented in this paper, produces a class label, namely a (fp rate, tp

rate) pair corresponding to a single point in the ROC space. One way, albeit not the optimal one, to produce such graph is by varying the threshold of a classifier from  $-\infty$  and  $+\infty$ . For instance, Table 8 illustrates 8 points in the ROC space (fp rate, tp rate) related to the NIST-based classifier. Those points were generated by modifying the threshold (e.g. number of passed statistical tests) between 0 and 8. Ideally, the best classifier should operate near the point (0,1), that is with an fp rate of 0% and a tp rate of 100%. The present classifier performs well and is quite close to the ideal point when a threshold between 4 and 6 is used. We also calculated different parameters which help to understand

**Table 7**

Nist-based classifier results with a threshold of 4.

Dir	Files #	Standard #	Standard [%]	Wiped #	Wiped [%]
csv	608	608	100.00	0	0.00
doc	8462	7595	89.75	867	10.25
gif	14	14	100.00	0	0.00
gz	1022	632	61.84	390	38.16
html	3905	3888	99.56	17	0.44
jpg	305	263	86.23	42	13.77
log	1795	1795	100.00	0	0.00
pdf	41280	33840	81.98	7440	18.02
png	4	4	100.00	0	0.00
pps	569	460	80.84	109	19.16
ppt	16472	14318	86.92	2154	13.08
ps	7179	7178	99.99	1	0.01
text	32255	29662	91.96	2593	8.04
xls	11561	11554	99.94	7	0.06
xml	1485	1485	100.00	0	0.00
standard	126916	116674	91.93	10242	8.07
time				Analysis	Parsing
wip				10419	294
time	94378	1595	1.69	92783	98.31
				7801	230

**Table 8**

Performances of NIST-based classifier.

TH #	TPR	FPR	P #	N #	TP #	FP #	FN #	TN #	ACC	SPC	PPV	NPV	FDR	MCC	F1
0	1.0000	1.0000	94378	126916	94378	126916	0	0	0.4265	0.0000	0.4265	NaN	0.5735	NaN	0.5979
1	0.9907	0.1976	94378	126916	93500	25079	878	101837	0.8827	0.8024	0.7885	0.9915	0.2115	0.7865	0.8781
2	0.9873	0.1287	94378	126916	93179	16334	1199	110582	0.9208	0.8713	0.8508	0.9893	0.1492	0.8493	0.9140
3	0.9853	0.1004	94378	126916	92991	12742	1387	114174	0.9361	0.8996	0.8795	0.9880	0.1205	0.8761	0.9294
4	0.9831	0.0807	94378	126916	92783	10242	1595	116674	0.9465	0.9193	0.9006	0.9865	0.0994	0.8947	0.9400
5	0.9768	0.0568	94378	126916	92188	7209	2190	119707	0.9575	0.9432	0.9275	0.9820	0.0725	0.9147	0.9515
6	0.9420	0.0260	94378	126916	88904	3300	5474	123616	0.9604	0.9740	0.9642	0.9576	0.0358	0.9189	0.9530
7	0.8242	0	94378	126916	77786	0	16592	126916	0.9250	1.0000	1.0000	0.8844	0.0000	0.8538	0.9036
8	0	0	94378	126916	0	0	94378	126916	0.5735	1.0000	NaN	0.5735	NaN	NaN	0

how a classifier performs. These parameters can be seen in Table 10. For instance, the best accuracy (ACC) of this classifier can be obtained with a threshold of 6, whereas the maximum specificity (SPC = 1), a condition where we do not have any false positive, implies a false negative of about 18%. In addition, the ROC curve of nist-based classifier is illustrated in Fig. 7 with the related area under the curve (AUC), which is 0.99.

Similarly, Table 9 refers to points in the ROC space related to the entropic classifier. We can notice that in this case the results are not satisfactory as in the previous case: while the tp rate is almost constant when varying the threshold (i.e. the threshold can vary between 0 and 8), the fp rate remains considerably high. This classifier has been discussed only as a reference case to prove that the NIST-based classifier performs much better. Fig. 6 illustrates the ROC curve of this classifier with its AUC value (0.823).

#### 4.5. Randomness of wiping generated data

Among the surveyed wiping algorithms in Section 2 we verified that all algorithms but one (Only first and last 2 Kb) produce random data. Particularly, we wiped out 7 different files (12 Mb sized), which were generated with a script (i.e. the file contents was set to 'A') by means of the `eraser1.exe` tool and using all the available wiping algorithms (Table 12). Every wiped file was recovered by means of the methodology presented in Section 4.1. After that, 3072 chunks (4 Kb sized) were generated from each

wiped file. Therefore, all data chunks were analyzed and subsequently parsed. The result of the classification is shown in Table 11. We can notice that the true positive rate for all algorithms but one is above 99%, which means that almost all data fragments were correctly classified as wiped. Interestingly, the algorithm `only first and last 2 Kb` failed the test almost completely (only one fragment recognized as wiped). The user-defined algorithm by default was the Gutmann ones. Indeed, the results of the test supports this fact.

As a result, it appears reasonable to use a statistical methodology which looks for data area filled with random data. In fact, we have demonstrated that 6 out of 7 wiping algorithms which are used by the `eraser1` tool generate random data.

#### 4.6. Detection of disk areas filled with periodic data

The existing wiping algorithms can securely erase data on a digital media in two ways:

- by means of random data
- by repeating a fixed data pattern

While disk areas filled with random data generated by wiping algorithms can be detected by the methodology illustrated in the previous sections of this paper, disk areas filled with periodic sequences of characters (i.e. the pattern

**Table 9**

Performances of Entropic classifier.

TH #	TPR	FPR	P #	N #	TP #	FP #	FN #	TN #	ACC	SPC	PPV	NPV	FDR	MCC	F1
0	1.0000	1.0000	94378	126916	94378	126916	0	0	0.4265	0.0000	0.4265	NaN	0.5735	NaN	0.5979
1.00	0.9996	0.9936	94378	126916	94340	126104	38	812	0.4300	0.0064	0.4280	0.9556	0.5720	0.0480	0.5993
2.00	0.9995	0.9836	94378	126916	94331	124835	47	2081	0.4357	0.0164	0.4304	0.9778	0.5696	0.0806	0.6017
3.00	0.9993	0.9631	94378	126916	94312	122233	66	4683	0.4473	0.0369	0.4355	0.9861	0.5645	0.1235	0.6067
4.00	0.9981	0.8181	94378	126916	94199	103830	179	23086	0.5300	0.1819	0.4757	0.9923	0.5243	0.2902	0.6443
5.00	0.9971	0.5166	94378	126916	94104	65565	274	61351	0.7025	0.4834	0.5894	0.9956	0.4106	0.5301	0.7408
6.00	0.9948	0.4508	94378	126916	93887	57214	491	69702	0.7392	0.5492	0.6214	0.9930	0.3786	0.5781	0.7649
7.00	0.9876	0.4376	94378	126916	93208	55538	1170	71378	0.7437	0.5624	0.6266	0.9839	0.3734	0.5795	0.7667
7.20	0.9874	0.4330	94378	126916	93189	54955	1189	71961	0.7463	0.5670	0.6290	0.9837	0.3710	0.5829	0.7685
7.40	0.9872	0.4180	94378	126916	93170	53051	1208	73865	0.7548	0.5820	0.6372	0.9839	0.3628	0.5946	0.7745
7.60	0.9868	0.4015	94378	126916	93132	50957	1246	75959	0.7641	0.5985	0.6464	0.9839	0.3536	0.6073	0.7811
7.80	0.9864	0.3143	94378	126916	93094	39890	1284	87026	0.8139	0.6857	0.7000	0.9855	0.3000	0.6788	0.8189
7.85	0.9860	0.2919	94378	126916	93057	37047	1321	89869	0.8266	0.7081	0.7153	0.9855	0.2847	0.6974	0.8291
7.90	0.9841	0.2296	94378	126916	92877	29140	1501	97776	0.8615	0.7704	0.7612	0.9849	0.2388	0.7503	0.8584
7.95	0.8524	0.0262	94378	126916	80448	3325	13930	123591	0.9220	0.9738	0.9603	0.8987	0.0397	0.8424	0.9031
8.00	0.0000	0.0000	94378	126916	0	0	94378	126916	0.5735	1.0000	NaN	0.5735	NaN	NaN	0.0000

**Table 10**  
Parameters for evaluating binary classifier performances.

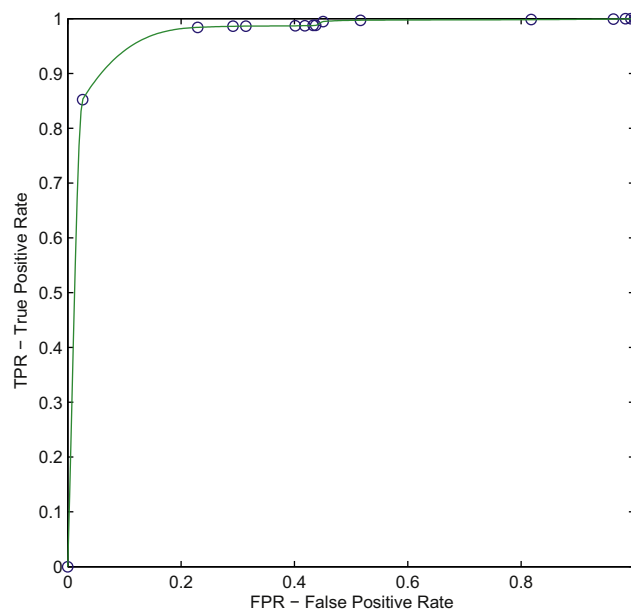
TH (threshold)	It defines the value which is used by the classifier to determine one of the two possible outcomes
P (positive), N (negative)	Number of actual samples tested by the classifier (e.g. wiped (P), and standard (N))
P', N'	Number of samples predicted as P or N
TP (true positive)	Number of samples P predicted correctly as P'
FP (false positive)	Number of samples N predicted incorrectly as P' (Type I error, $\alpha$ )
FN (false negative)	Number of samples P predicted incorrectly as N' (Type II error, $\beta$ )
TN (true negative)	Number of sample N correctly predicted as N'
TPR (true positive rate)	Also defined as sensitivity of the classifier. $TPR = TP/(TP + FN)$
FPR (false positive rate)	$FPR = FP/(FP + TN)$
ACC (accuracy)	$ACC = (TP + TN) / (P + N)$
SPC (specificity)	$SPC = TN/P = 1 - FPR$
PPV (positive predictive value)	$PPV = TP/(TP + FP)$
NPV (negative predictive value)	$NPV = TN/(TN + FN)$
FDR (false discovery rate)	$FDR = FP/(FP + TP)$
MCC (Matthews correlation coefficient)	$MCC = \frac{TP \times TN - FP \times FN}{\sqrt{P \times N \times P' \times N'}}$
F1 score	$F1 = 2 * TP / (P + P')$

AAAABBBB is used to fill a disk area of 100 Mb) could be more difficult to detect. Therefore, the discussed methodology cannot be used. Nonetheless, an area wiped in this way has spectral properties which are very different from a not wiped one. In the following section we present a methodology which can be used to detect this type of wiped data areas by means of a modified version of the

Discrete Fourier Transform (DFT) test of the NIST statistical suite.

The original DFT NIST test is used to investigate whether a run might be classified as random or not. Initially, the original binary sequence formed by zeros and ones is converted to values of  $-1$  and  $+1$  to create the sequence  $X = x_1, x_2, \dots, x_n$  where  $x_i = 2e_i - 1$ . After that, the Discrete Fourier Transform is calculated on  $X$  to produce  $S = DFT(X)$ , which is a sequence of complex variables representing periodic components of the sequence of bits at different frequencies. Therefore, the modulus function on the first  $n/2$  elements of  $S$  is calculated by producing a sequence of peak heights ( $M = |S_i|$ ). At this point the test looks for the number of peaks below the threshold  $T$ , which is indicated in equation (4). Under the assumption of randomness, 95% of the peaks of  $M$  should not exceed  $T$ . For instance, by considering a random sequence of 32,000 bits, we should expect to see  $0.05 \times 32,000 \times 0.5 = 800$  peaks above the threshold  $T$ . However, as illustrated in Fig. 8, in case of a periodic sequence, we should expect to see a few prominent peaks in the FFT spectrum. Accordingly, we might exploit this behavior to discriminate between periodic and not-periodic sequences.

In order to verify whether a sequence is periodic or not we can look at the fraction of  $M$  samples (the modulus of the FFT of the input sequence), which are above the threshold  $T$ . This value is represented by the  $N$ -value, defined in equation (5). For instance, for a periodic sequence, the  $N$ -value is expected to be very low, as illustrated in Fig. 8, where the FFT peaks above the 95% threshold are about 8 out of 9000 FFT elements. Conversely, for a random sequence of the same length, we expect to see  $0.05 \times 9,000 \times 0.5 = 225$  peaks above  $T$ .



**Fig. 6.** ROC entropy classifier (AUC = 0.823).



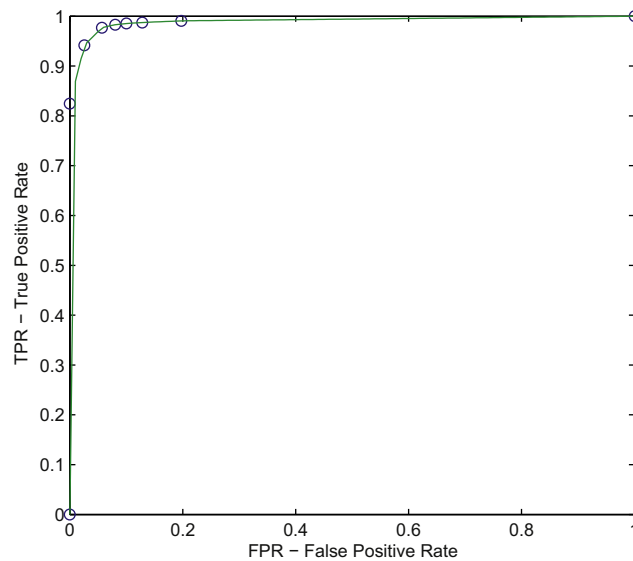


Fig. 7. ROC nist-based classifier (AUC = 0.986).

$$T = \sqrt{\log_{10}\left(\frac{1}{0.05}\right)n} \quad (4)$$

$$N = \frac{\text{elements of FFT} - M \text{ sequence} > T^*}{n} 100 \quad (5)$$

Each input sequence to be analyzed is divided in blocks of  $n$  bytes (e.g. 4 Kb) and the  $N$ -value is calculated for each block. If an  $N$ -value is below a certain threshold  $\alpha$ , which has been experimentally determined equal to 2%, the block is considered as periodic. A sequence is classified as periodic when at least 90% of the blocks are classified as periodic. The threshold,  $\beta$ , has been experimentally determined as a trade off between false positive and true positive rate. Equation (6) illustrates the basic idea behind the classifier.

For the sake of clarity, let us consider an example which illustrates the difference between the  $N$ -value distribution of two different sequences: a periodic file, which has been generated by repeating a certain pattern, and a doc file type, which contains classified information. The analysis starts by dividing the sequence in blocks of  $n$  Kb, say 4 Kb, and calculating the  $N$ -value for each of the block. After that, the range of  $N$ -values is divided in bins in order to represent

the histogram of values. For instance, Table 13 illustrates the distribution of  $N$ -values in the range of possible values of  $N$ -values: for each bin (e.g. from 0 to 0.99%) there is the percentage of runs that fall in that bin. In this case, we can notice that the totality of the runs lies in the first two bins of the distribution, which implies that the  $N$ -values are below the threshold  $\alpha$  of 2%.

Conversely, a non-periodic sequence, such as the one represented by the tested doc file, has a more uniform  $N$ -value histogram, as indicated in Table 14.

$$\left(\frac{\# \text{ of } N \text{ values} < \alpha}{\# \text{ of total } N \text{ values}}\right) \cdot 100 \geq \beta \quad (6)$$

The classifier for discriminating between periodic and non-periodic sequences, which is expressed by equation (6), has been verified against periodic and non-periodic sequences to draw its ROC (Receiver Operating Characteristic). For the negative pool we used the training set described in Section 4.4.4 (excluding files smaller than 4 Kb). While, for the positive pool we used a Python script which generates test files by repeating a principal pattern (e.g. A@#BK236) plus an additional block, of the same size of the basic pattern, which takes into account already overwritten disk areas in the unallocated disk space or disk reading errors. The additional block is added to the

Table 11

Test to state whether a wiping algorithm produces random data.

Algorithm	# of passes	Dir	Files #	Standard #	Standard [%]	Wiped #	Wiped [%]
Gutmann	35	f0	3072	15	7.13	3057	99.51
US DoD 5220.22-M (8-306/E,C and E)	7	f1	3072	21	0.68	3051	99.32
US DoD 5220.22-M (8-306/E)	3	f2	3072	15	0.49	3057	99.51
Only first and last 2 Kb	–	f3	3072	3071	99.97	1	0.03
Schneier	7	f4	3072	23	0.75	3049	99.25
Pseudorandom data	1	f5	3072	13	0.46	3059	99.54
User defined	7	f6	3072	15	0.49	3057	99.51

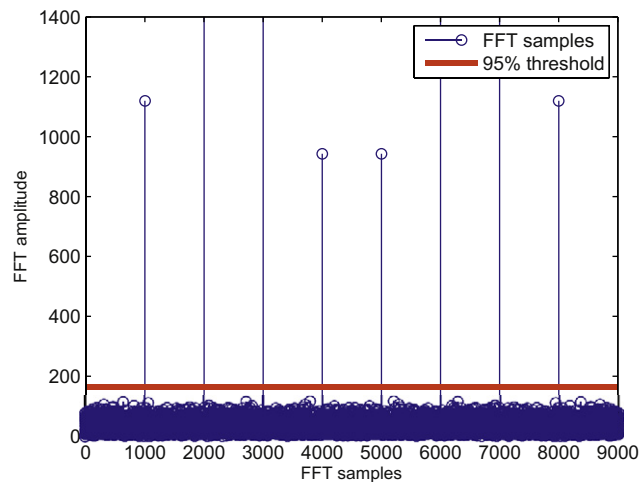


Fig. 8. FFT of a periodic sequence.

principal one with a probability ranging between 10% and 40%. In addition, the principal pattern has been generated by randomly selecting a variable sequence of characters (from 1 to 20) from the ASCII character set.

Each point of the ROC has been obtained by varying the number of bit of the sequence to be tested, ranging between 32,000 and 8,000,000 bits, and keeping  $\alpha = 2\%$  and  $\beta = 90\%$ . The first test has been conducted with the following parameters:

$$\alpha = 2\%, \beta = 90\%, n = 32,000$$

The result of the test is illustrated in Fig. 9. It can be noticed that the red dot marker, which is a point on the ROC of the classifier, is calculated by averaging different tests that have been done by randomly selecting negative and positive samples.

It is shown that the classifier (with parameters  $\alpha$ ,  $\beta$ ,  $n = 32,000$ ) has a low false positive rate, about 10%, and a true positive rate of about 70%. Of course, these values are based on the assumption that the DFT analysis can be realized with the available bits of the sample. It is worth saying that the number  $n$  has to be no less than an order of magnitude of the period length of the periodic sequence, otherwise the Fourier analysis will not provide reliable results.

Further tests with different values of  $n$  have shown that increasing the size of the analyzed sequence the *True Positive Rate* augments (i.e. longer almost-periodic

sequences have more defined spectral features over the noise component of the sequence). At the same time, this fact increases the *False Positive Rate* (periodic-like sequences, like ASCII files, are more likely to have their periodicity highlighted over long runs). Fig. 10 shows the results of the same test described before. This time however the test was done with blocks of  $n = 1,024,000$ .

$$\alpha = 2\%, \beta = 90\%, n = 1,024,000$$

With this value of  $n$  it can be noticed that the TPR is 100%, while the FPR is still about 10%.

Increasing too much the value of  $n$  highlights the spectral properties of non-periodic sequences (like ASCII files), diminishing the FPR while keeping the TPR about 100%. Fig. 11 shows the test results for  $n = 8,000,000$  (1 megabyte sequence).

By using the average results (TPR versus FPR) of the tests described above it is possible to draw the ROC for the classifier illustrated by equation (6). The final ROC is shown in Fig. 12 with the related area under the curve (AUC), which is 0.9349.

#### 4.7. Performances of the solution and limitations

The presented approach to detecting on-disk wiped fragments was tested against a public data set. The tests were done on a virtual machine equipped with Suse Linux 11.1, 1 Gb RAM and 20 Gb hard drive. This statistical method

Table 12

Wiping algorithms results.

#	Wiping algorithm	# of passes	Generated data
1	Gutmann	35	Random
2	US DoD 5220.22-M (8-306./E,C and E)	7	Random
3	US DoD 5220.22-M (8-306./E)	3	Random
4	Pseudorandom Data	1	Random
5	Only first and last 2 KB	1	Not random
6	Schneier's 7 pass	7	Random
7	User defined	7	Random

Table 13

Histogram of N-values for a periodic sequence.

Range [%]	Percentage of runs in the range	
0/0.99	85.71%	Values < $\alpha$
1/1.99	14.29%	100%
2/2.99	0.00%	Values > $\alpha$
3/3.99	0.00%	0%
4/4.99	0.00%	
5/5.99	0.00%	

**Table 14**Histogram of  $N$ -values for a non-periodic sequence from a.doc file.

Range [%]	Percentage of runs in the range	
0/0.99	7.14%	Values $< \alpha \sim 33\%$
1/1.99	26.19%	
2/2.99	19.05%	Values $> \alpha \sim 67\%$
3/3.99	14.29%	
4/4.99	14.29%	
5/5.99	19.05%	

relies on the NIST assess tool, which was modified to automate the analysis process, as seen in Listing 7.

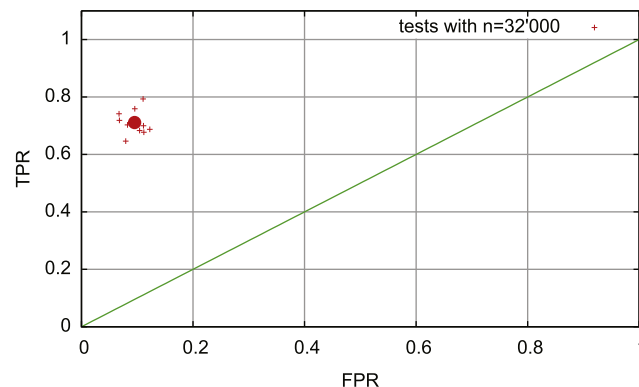
Presently, one of the limitations of the current approach (NIST-based classifier) is the slowness of the fragment analysis phase. The virtual machine was hosted by a Toshiba Satellite A300, equipped with an Intel Core 2 Duo T9400 2.53 GHz, Windows Seven Professional, 4 Gb RAM, and 500 Gb hard drive. On average, we were able to analyze 12.2 fragments per second, which is really not enough if we want to have a real-time analyzer. This bottleneck might be avoided by means of multi core architectures (e.g. GPUs) and parallelization (e.g. launching many instances of the tool which scan different not overlapped portions of the target file(s)).

Another issue of the discussed approach is that the statistics of wiped files is the same as the encrypted ones. In fact, we conducted another test by considering an additional fragment type, the encrypted one, in the test set. For this purpose we generated 6145 fragments taken from a file which was encrypted by means of aescrypt Linux tool (i.e. `aescrypt -e -p password -o enc file -eTest.txt`). After that, all the fragments were analyzed and the generated reports parsed. The result was that 99.24% of fragments was classified as wiped. This is far from surprising. Rather, it is just a confirmation that the statistical distribution of encrypted fragments (files) follows the one of the wiped fragments (i.e. the two mentioned file types have a distribution of data which can be considered as random). The only exploitable trace of an encrypted file is the header. In fact, from the previous file we found the following header (CREATED BY aescrypt 3.05). At least, adopting the presented method, encrypted fragments do not seem different from the wiped ones. This

fact can lead to the conclusion that on-disk wiped areas might be further overwritten by an encrypted file big enough to implement a plausible deniability case (i.e. the resulting scenario is plausible and cannot be denied). Similarly, deleted true-crypt volumes, whose data distribution is random, might be wrongly attributed to a wiping tool effect.

As a consequence, we can present an overview of possible scenarios which might involve the presence of wiped areas in the unallocated space:

- Wiped areas without any deleted encrypted files: on-disk wiped areas can be found within the unallocated space of a storage device and can be reliably detected with the presented methodology.
- Wiped areas on a disk with a few small-sized encrypted files which were deleted: on-disk wiped areas might be detected with difficulty. If we can recover the deleted encrypted files, by relying on their headers, we may guess that the remaining unallocated area can be attributed to a wiped one. Other elements might confirm this thesis (e.g. analysis of page file(s) when available, or volatile memory contents).
- Deleted encrypted volumes (e.g. true-crypt volume): being a true-crypt file volume header-less, as in the case of a wiped area, the analysis leads to a wrong conclusion. In fact, as we have demonstrated, all the encrypted files are classified as wiped.
- Wiped areas on a disk and creation of an encrypted file big enough to fill the unallocated space: this scenario leads to a plausible deniability one. In this case, we need to consider that the attacker might have wiped out the page file as well as the RAM traces (a dead process related to the wiping tool which was used), for instance by allocating all the available RAM with a script that generates memory blocks with a known content (e.g. 'AAAA').
- Disk encryption: the unallocated space is filled with random data. This is an alternative way to secure data on a disk avoiding the detection of on-disk wiping. Thus, the presented methodology does not apply to full encrypted disks (FEDs).

**Fig. 9.** Test results for for  $n = 32,000$ .

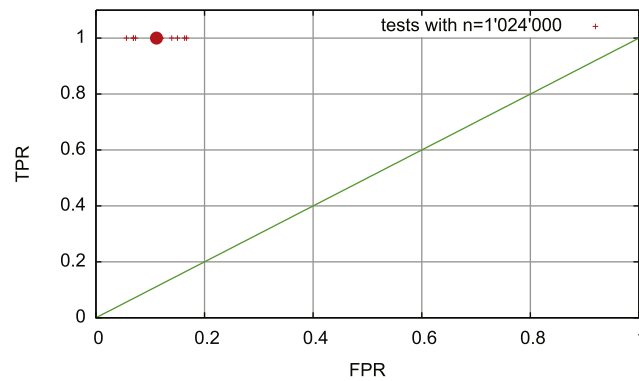


Fig. 10. Test results for  $n = 1,024,000$ .

## 5. Case study

In this section we provide a methodology based on our approach to dealing with practical cases. We outline the main points of our approach in the remainder of this section.

- The usual scenario is to analyze whether there are traces of on-disk wiped areas. The analysis might be performed on every type of storage device (hard disk, pen drive, etc.).
- The artifacts, in the worst case scenario, might be found only by analyzing the unallocated storage area (e.g. unallocated clusters). This area can be collected by means of well-known forensic offerings, such as EnCase, FTK, or SleuthKit.
- The slack space, which is an unstructured blob of binary data, might be analyzed with the proposed methodology.

Let us describe an experiment that we made to validate our procedure. An SD card (3.7 Gbyte) formatted NTFS contains 3 mkv files (ex01, ex02, and ex03) and a directory (CONTRABAND) with 403 Mb of jpeg files (all the images are legal and taken from personal repositories). We used the `eraser1.exe` tool to erase the entire folder (`eraser1 -folder ex01 -subfolders -method`

`'DoD_E' -silent`). After that, we analyzed the SD card with EnCase v.4 noticing that 257 deleted files were created (e.g. `xactengine2_3.dll`, `xactengine2_6.dll`, `XAPOFX1_1.dll`, etc.). As already mentioned, these traces might not be attributed to the eraser tool, since there is no fixed pattern in their name. As a consequence, this fact suggests that the mentioned folder was not necessarily deleted with a wiping tool. After that, we collected the unallocated space by obtaining a binary file of 407 Mb (`unallocatedSpaceSD`). This binary file was tested to evaluate whether wiping traces were present. In the first test, a chunk size of 4 Kb was considered ( $n = 320$  bits and  $m = 100$  sequences); 104192 fragments were analyzed in 142 min 97.6% of the fragments were correctly classified as wiped. In order to accelerate the analysis phase we considered a chunk size of 4 Mb, with parameters  $n = 32000$  bits and  $m = 100$  sequences. The analysis endured only 15 min and 94.14% of the chunks were detected as wiped. Interestingly, the discussed method performs well even with data chunks of different size. This method can considerably speed up the analysis phase.

## 6. Related works

Anti-forensic detection has been surveyed and considered in numerous research papers. Kim et al. (Kim et al., submitted for publication) survey wiping artifacts left by

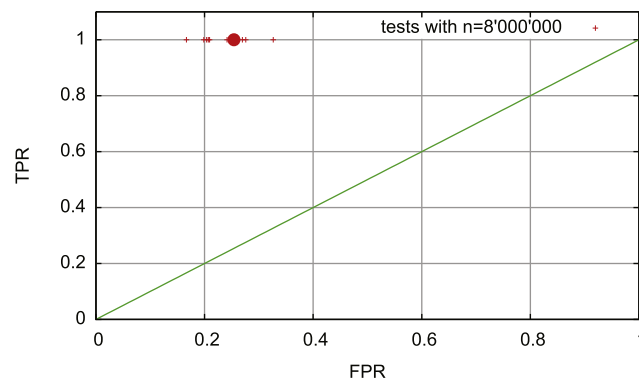


Fig. 11. Test results for  $n = 8,000,000$ .

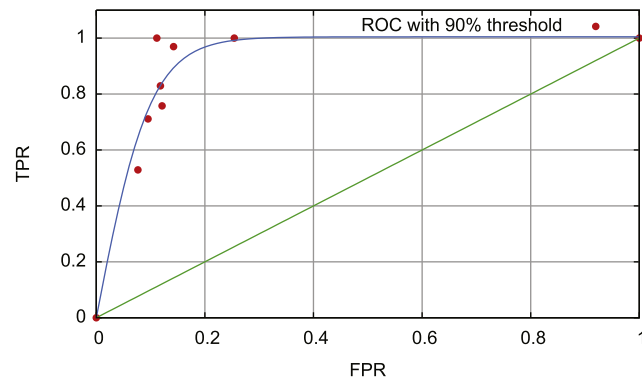


Fig. 12. Discriminator ROC for  $n$  between 32,000 and 8,000,000.

#### Listing 7

The customized NIST assess tool.

```
savoldi@linux-lvr8:/data/sts-2.0b> ./assess
Usage: assess <stream length> <file_in> <finalAnalysisReport> <numOfBitStreams>
<stream length> is the length of the individual bit stream(s) to be processed
<file> is the file to process
<finalAnalysisReport> is the file where the statistics is saved
<numOfBitStreams> is the number of maximum sequences that can be processed
according to the chunk size
StreamLength x numOfBitStreams <= sizeOfChunk
```

wiping tools on a Windows-based system. In their paper, they discuss tangible artifacts such as installation folder, prefetch files, Registry keys, file entry signatures which are caused by different wiping tools. In addition, artifacts left by two portable wiping tools are discussed. However, the analysis does not consider the detection of on-disk wiped areas, which can help in answering whether a wiping tool might have been used.

Geiger (Geiger, 2005) surveys different commercial offerings of wiping tools. The analysis evaluates their performance and ability to wipe sets of files. However, that study does not analyze wiped areas *per se*, thus missing an important step towards the detection of wiping tool usage.

Roussev et al. (Russev and Garfinkel, 2009) discuss fragment classification by means of file-dependent features, such as fixed patterns and intrinsic features. They therefore state that the statistical analysis is not reliable for any file classification. However, in their study they do not consider encrypted or wiped files detection. In our present study, not only do we demonstrate the ability of a statistical approach to detecting wiped areas, but we also state that this method is reliable, effective, successful, and might be the only feasible way to detect, albeit in certain circumstances, on-disk artifacts caused by a wiping tool.

The proposed anti-anti-forensic method, which is based on a statistical analysis, is efficient, scalable, and can be easily used on parallel architectures to further speed up the analysis of unallocated storage device areas.

## 7. Conclusion

Anti-forensics is evolving rapidly to subvert any possible forensic investigation which should shed light on cyber-crimes. In this paper, we presented a methodology that might be used, in certain circumstances, to detect on-disk artifacts left by portable wiping tools, which can be launched without leaving any tangible artifacts, at least at the system level (Registry, file system, metadata). After having surveyed diverse data-wiping scenarios where no traces are left on the system, we proposed a viable and promising approach which is based on a statistical method capable of detecting data fragments that contain random data. In addition, we illustrated how to build up a linear binary classifier which is capable of discriminating between wiped and standard data fragments. Furthermore, we discussed a viable solution which allows the detection of on-disk wiped areas filled with periodic patterns.

## Appendix

In this appendix we provide a background on the NIST statistical tests.

### Frequency (monobit) test

The purpose of this test is to determine whether the number of ones and zeros in a sequence are approximately the same as would be expected for a truly random

sequence. The test assesses the closeness of the fraction of ones to  $1/2$ , that is, the number of ones and zeroes in a sequence should be about the same. All subsequent tests depend on the passing of this test.

#### *Frequency test within a block*

The aim of this test is to determine whether the frequency of ones in an  $M$ -bit block is approximately  $M/2$ , as would be expected under an assumption of randomness. For block size  $M = 1$ , this test degenerates to test 1, the Frequency (Monobit) test.

#### *Runs test*

The focus of this test is the total number of runs in the sequence, where a run is an uninterrupted sequence of identical bits. A run of length  $k$  consists of exactly  $k$  identical bits and is bounded before and after with a bit of the opposite value. The purpose of the runs test is to determine whether the number of runs of ones and zeros of various lengths is as expected for a random sequence. In particular, this test determines whether the oscillation between such zeros and ones is too fast or too slow.

#### *Test for the longest run of ones in a block*

This test determines whether the length of the longest run of ones within the tested sequence is consistent with the length of the longest run of ones that would be expected in a random sequence. Note that an irregularity in the expected length of the longest run of ones implies that there is also an irregularity in the expected length of the longest run of zeros. Therefore, only a test for ones is necessary.

#### *Binary matrix rank test*

The focus of the test is the rank of disjoint sub-matrices of the entire sequence. The test checks for linear dependence among fixed length substrings of the original sequence.

#### *Discrete fourier transform (spectral) test*

The purpose of this test is to detect periodic features (i.e., repetitive patterns that are near each other) in the tested sequence that would indicate a deviation from the assumption of randomness. The intention is to detect whether the number of peaks exceeding the 95% threshold is significantly different than 5%.

#### *Non-overlapping template matching test*

The focus of this test is the number of occurrences of pre-specified target strings. This test detects generators that produce too many occurrences of a given non-periodic (aperiodic) pattern. For this test an  $m$ -bit window is used to search for a specific  $m$ -bit pattern. If the pattern is not found, the window slides one bit position. If the pattern is

found, the window is reset to the bit after the found pattern, and the search resumes.

#### *Overlapping template matching test*

The focus of the Overlapping Template Matching test is the number of occurrences of pre-specified target strings. Both this test and the Non-overlapping Template Matching test use an  $m$ -bit window to search for a specific  $m$ -bit pattern. As with the test in the previous section, if the pattern is not found, the window slides one bit position. The difference between this test and the previous test is that when the pattern is found, the window slides only one bit before resuming the search.

#### *Maurer's universal statistical test*

This test focuses on the number of bits between matching patterns (a measure that is related to the length of a compressed sequence). The aim of the test is to detect whether or not the sequence can be significantly compressed without loss of information. A significantly compressible sequence is considered to be non-random.

#### *Linear complexity test*

The focus of this test is the length of a linear feedback shift register (LFSR). This test determines whether or not the sequence is complex enough to be considered random. Random sequences are characterized by longer LFSRs. An LFSR that is too short implies non-randomness.

#### *Serial test*

This test deals with the frequency of all possible overlapping  $m$ -bit patterns across the entire sequence. The purpose of this test is to determine whether the number of occurrences of the  $2^m$   $m$ -bit overlapping patterns is approximately the same as would be expected for a random sequence. Random sequences have uniformity; that is, every  $m$ -bit pattern has the same chance of appearing as every other  $m$ -bit pattern.

#### *Approximate entropy test*

The focus of this test is the frequency of all possible overlapping  $m$ -bit patterns across the entire sequence. The goal of the test is to compare the frequency of overlapping blocks of two consecutive/adjacent lengths ( $m$  and  $m + 1$ ) against the expected result for a random sequence.

#### *Cumulative sums (cusum) test*

The focus of this test is the maximal excursion (from zero) of the random walk defined by the cumulative sum of adjusted  $(-1, +1)$  digits in the sequence. The test determines whether the cumulative sum of the partial sequences occurring in the tested sequence is too large or too small relative to the expected behavior of that cumulative sum for random sequences. This cumulative sum may be considered as a random walk. For a random sequence,



the excursions of the random walk should be near zero. For certain types of non-random sequences, the excursions of this random walk from zero will be large.

#### Random excursions test

The focus of this test is the number of cycles having exactly  $K$  visits in a cumulative sum random walk. The cumulative sum random walk is derived from partial sums after the (0,1) sequence is transferred to the appropriate  $(-1, +1)$  sequence. A cycle of a random walk consists of a sequence of steps of unit length taken at random that begin at and return to the origin. This test determines if the number of visits to a particular state within a cycle deviates from what one would expect for a random sequence. This test is actually a series of eight tests (and conclusions), one test and conclusion for each of the states:  $-4, -3, -2, -1$  and  $+1, +2, +3, +4$ .

#### Random excursions variant test

This test focuses on the total number of times that a particular state is visited (i.e., occurs) in a cumulative sum random walk. The test detects deviations from the expected number of visits to various states in the random walk. This test is actually a series of eighteen tests (and conclusions), one test and conclusion for each of the states:  $-9, -8, -1$  and  $+1, +2, +9$ .

## References

- Banks D, Barker E, Dray J, Heckert A, Leigh S, Levenson M, et al. NIST statistical test suite. Retrieved April, 2010, from, <http://csrc.nist.gov/groups/ST/toolkit/rng/documents/sts-2.1.zip>; 2008.
- Bosschert T. Battling anti-forensics: beating the U3 stick. *Journal of Digital Forensic Practice* 2006;1(4):265–73.
- Bunting SA. Mastering windows network forensics and investigation. Wiley Publishing; 2007.
- Carrier B. File system forensic analysis. 1st ed. Addison-Wesley Professional; 2005.
- Carvey H. Windows forensic analysis DVD toolkit. Syngress; 2007.
- Fawcett T. An introduction to roc analysis. *Pattern Recogn Lett* 2006; 27(8):861–74.
- Garfinkel SL, Malan DJ. One big file is not enough: a critical evaluation of the dominant free-space sanitization technique. Berlin Heidelberg: Springer-Verlag. Retrieved April, 2010, from, <http://www.cs.harvard.edu/malan/publications/pet06.pdf>; 2006.
- Geiger M. Evaluating commercial counter forensic tools. In: The proceedings of the DFRWS 2005. Digital Forensic Research Workshop. Retrieved April, 2010, from, <http://www.dfrws.org/2005/proceedings/geigercounterforensics.pdf>; 2005.
- Halderman JA, Schoen SD, Heninger N, Clarkson W, Paul W, Calandrino JA, Feldman AJ, Appelbaum J, Felten EW. Lest we remember: cold boot attacks on encryption keys. In: Proc. 2008 USENIX Security Symposium; 2008. Retrieved April, 2010, from <http://citp.princeton.edu/pub/coldboot.pdf>.
- Harris R. Arriving at an anti-forensics consensus: examining how to define and control the anti-forensics problem. *Digital Investigation* 2006;3S:S44–9. Retrieved April, 2010, from, <http://www.dfrws.org/2006/proceedings/6-Harris.pdf>.
- Khatri Y. Prefetch file algorithm. Retrieved January, 2010, from, <https://42llc.net/index.php?option=commyblog&blogger=Yogesh%20Khatri&Itemid=39>; 2009.
- Kim Y, Savoldi A, Bang J, Lee S. Forensic artifacts left by data wiping tools. *Digital Investigation*, submitted for publication.
- Milo09. Windows prefetch folder tool. Retrieved April, 2010, from, <http://code.google.com/p/prefetch-tool/>; 2009.
- Rukhin A, Soto J, Nechvatal J, Smid M, Barker E, Leigh S, et al. A statistical test suite for random and pseudorandom number generators for cryptographic applications. Retrieved April, 2010, from, [lcsrsc.nist.gov/publications/nistpubs/800-22-rev1/SP800-22rev1.pdf](http://lcsrsc.nist.gov/publications/nistpubs/800-22-rev1/SP800-22rev1.pdf); 2008.
- Russev V, Garfinkel S. File fragment classification – the case for specialized approaches. In: Proc. 2009 SADFE conference, 2009. Retrieved April, 2010, from <http://simson.net/clips/academic/2009.SADFE.Fragments.pdf>.
- Russinovich M, Cogswell B. Process monitor v2.9. Retrieved April, 2010, from, <http://technet.microsoft.com/en-us/sysinternals/bb896645.aspx>; 2010.
- Trant G. Eraser portable. Retrieved January, 2010, from, <http://eraser.heidi.ie/>; 2009.
- WFA. Windows file analyzer. Retrieved January, 2010, from, <http://www.mitec.cz/wfa.html>; 2009.

**Antonio Savoldi** received the PhD degree from University of Brescia, Brescia, Italy, in March 2009, where he is a research fellow at the moment. He has been involved in digital forensics science for 5 years giving contributions in the analysis of embedded systems, such as SIM cards and PDA devices, steganalysis of active media and security of embedded systems. He also developed methodologies for discovering hidden channels within the slack memory part of an embedded system, such as the non-standard part of a SIM/USIM card and the flash memory of a Windows CE based PDA. In addition, he actively collaborates with Korean forensic institutes in the development of tools and methodologies to be used in the realm of digital forensics. He has been a program committee member of different international conferences such as the SADFE (Systematic Approaches to Digital Forensic Engineering), IJHMSP (Intelligent Information Hiding and Multimedia Signal Processing), and FGCN (Future Generation Communication and Networking). Finally, he contributed for international conferences and journals such as IFIP WG 11.3, SAC (Symposium on Applied Computing), IJDE (International Journal of Digital Evidence) and IJDCF (International Journal of Digital Crime and Forensics).

**Mario Piccinelli** is a PhD candidate at University of Brescia, Brescia, Italy. His research interests include digital forensics and security of embedded systems.

**Paolo Gubian** received the Dr. Ing. degree “summa cum laude” from Politecnico di Milano, Italy, in 1980. After an initial period as a research associate at the Department of Electronics of the Politecnico di Milano, Italy, he started consulting for ST Microelectronics (then SGS-Microelectronics) in the areas of electronic circuit simulation and CAD system architectures. During this period he worked at the design and implementation of ST-SPICE, the company proprietary circuit simulator. Besides, he worked in European initiatives to define a standard framework for integrated circuit CAD systems. During 1984, 1985 and 1986 he was a visiting professor at the University of Bari, Italy, teaching a course on circuit simulation. He also was a visiting scientist at the University of California at Berkeley in 1984. In 1987 he joined the Department of Electronics at the University of Brescia, Italy as an Assistant Professor in Electrical Engineering. He is now an Associate Professor in Electrical Engineering. His research interests are in reliability and robustness of electronic systems architectures and in security and forensic applications to digital computer systems in general, and to embedded systems in particular.