Fast Contraband Detection In Large Capacity Disk Drives

By

Phil Penrose, William Buchanan and Richard Macfarlane

From the proceedings of

The Digital Forensic Research Conference

DFRWS 2015 EU

Dublin, Ireland (Mar 23rd- 26th)

Digital Investigation 12 (2015) S22–S29

Contents lists available at ScienceDirect

Digital Investigation

journal homepage: www.elsevier.com/locate/diin

Fast contraband detection in large capacity disk drives

Philip Penrose*, William J. Buchanan, Richard Macfarlane

Edinburgh Napier University, Edinburgh, United Kingdom

Keywords:
Disk sampling
Contraband detection
Digital forensics
Triage
Bloom filter
Sampling
Sample size

A B S T R A C T

In recent years the capacity of digital storage devices has been increasing at a rate that has left digital forensic services struggling to cope. There is an acknowledgement that current forensic tools have failed to keep up. The workload is such that a form of 'administrative triage' takes place in many labs where perceived low priority jobs are delayed or dropped without reference to the data itself. In this paper we investigate the feasibility of first responders performing a fast initial scan of a device by sampling on the device itself. A Bloom filter is used to store the block hashes of large collections of contraband data. We show that by sampling disk clusters, we can achieve 99.9% accuracy scanning for contraband data in minutes. Even under the constraints imposed by low specification legacy equipment, it is possible to scan a device for contraband with a known and controllable margin of error in a reasonable time. We conclude that in this type of case it is feasible to boot the device into a forensically sound environment and do a pre-imaging scan to prioritise the device for further detailed investigation.

Introduction

Kryder (2009) shows that the areal density (the number of bits stored per unit area of disk) has been increasing at 40% per year and this is projected to continue for the foreseeable future since the technology is, as yet, nowhere near fundamental limits. This is referred to as 'Kryder's Law' (Walter, 2005) – analogous to Moore's Law for semiconductors. By 2020 it is estimated that a 2.5 inch disk will have a capacity of 14 TB and cost $40. Garfinkel (2010) claims that, because of this, much of the progress made in digital forensic tools over the last decade is becoming irrelevant. These tools were designed to help forensic examiners to find evidence, usually from a relatively low capacity hard disk drive, and do not scale to the capacity of digital storage devices commonly available today. To put this in perspective Roussev et al. (2013) benchmarked the acquisition of a fast 3 TB hard disk drive using a standard acquisition utility at over 11 h. In the UK the Association of Chief Police Officers (ACPO) has acknowledged this situation. Many digital forensics units already have large backlogs and the rate of technological change is likely to accelerate and so exacerbate the situation. Where there is insufficient time or resources to cope with the volume of digital devices being presented a system of forensic triage should be introduced. We will use the term triage to mean a fast initial scan by sampling a digital device, conducted perhaps under severe time and resource constraints, to prioritise the device for possible further detailed investigation. Young et al. (2012) argue that it is critical for forensic investigators to have such a triage process so that they can quickly detect bad or illegal files on a large disk. While consulting with digital forensic analysts from Police Scotland it was queried whether it was feasible to implement such a forensic triage tool directly on a suspect machine so that a responding officer could quickly assess whether it held any contraband. The requirements would be that the system should:

* Corresponding author.
E-mail address: p.penrose@napier.ac.uk (P. Penrose).

http://dx.doi.org/10.1016/j.diin.2015.01.007

- Be 99.9% accurate
- Give results in a reasonable time
- Execute on low specification legacy equipment.

Restrictions on our methodology are imposed by these requirements. To achieve results in a reasonable time we must sample the device rather than inspect every sector. If we sample complete files on a disk then the file metadata is read from the file system and then the file is accessed. This involves considerable physical head movement in the disk drive. Fujitsu (Fujitsu Technology Solutions GmbH, 2011) benchmarked a fast hard disk drive which showed random access throughput at 3 MB/s and sequential access at 200 MB/s. Thus, random sampling of files would be considerably slower for triage purposes. In addition, it relies on the file system metadata and so would not cover unallocated space on the disk where illicit material may well be hidden. Statistically sampling disk sectors overcomes both problems. If the random selection of sector addresses is sorted before accessing the disk, then this incurs only a single sequential pass over the disk since the disk is treated as simply a sequence of blocks. Additionally, since the sample is chosen from the whole address range of the disk it would sample all areas including unallocated space. Also, using sector sampling bypasses the file system by sampling raw disk sectors and so the nature of the file system on the disk is irrelevant.

Another consequence of the restrictions is that any reference data set should be held in RAM so that lookups can be done to match disk read speed. Lookup of a disk based database would be slower than the sequential sampling. Hence a compact representation of any reference data set is needed if we are to legacy equipment with limited RAM.

Previous work

Triage

Pollitt (2013) argues that the process of digital forensic triage is an admission of failure. The backlog of cases is due to the systemic failure of the digital forensic process and of digital forensic software. These have not adapted to the vast increase in digital data that is involved in a modern case. Triage has become necessary because investigators often prefer some useful evidence quickly rather than wait, perhaps some considerable time, for all detectible evidence to be found. He argues that by focussing on a particular outcome such as the existence of specific types of data, we miss important information, such as logs or e-mail that might reveal a wider group of suspects. However, we argue that triage must not be the only tool used in an investigation. If any incriminating evidence is found during the process of triage then the device should be subject to a full digital forensic analysis.

Shaw and Browne (2013) note that '*administrative*' triage already takes place in many organisations and criteria are used to either prioritise or exclude a device from examination. Horsman et al. (2014) maintain that organisations may also be cautious because there is a perception that there is a risk of missing evidence where triage only samples a device. We argue that in our scenario of detecting contraband, there is less risk in a system of triage allowing the timely analysis of a device using forensically sound boot media, and with a controllable probability of missed evidence, than a system of administrative triage which operates without any reference to the physical media.

Existing triage solutions

There are a number of triage packages available, both open source and proprietary, such as Strike, EnCase Portable, AD Triage, Triage IR, Kludge. These packages typically perform data collection, often with no intervention by the operator, of such things as internet history, the registry, file metadata, recently used files, image files, hash lookup of user home directory files and comparing to a selected file hash database, indexing, keyword matching and so on. Some even do full disk imaging as part of triage. File carving from unallocated disk space is also an option on some. They uniformly behave as versions of full forensic analysis packages, collecting appropriate data for later examination. Most are designed so that they can be used by an untrained operative and give on-screen display of images or analysis results as they are produced. Casey et al. (2013) view this type of product as freeing forensic analysts from the routine task of acquiring forensic evidence and empowering them to concentrate on the more interesting aspects of their work. However these tools should be regarded as automating the acquisition stage of a forensic investigation rather than as triage. All the operations performed tend to be I/O and processor intensive and defeat the purpose of our definition of triage – a fast initial scan to ascertain if a device contains images or documents of interest. Gillam and Rogers (2005) developed File Hound, a forensic tool for first responders. This application was file based and would not be suitable for direct sector sampling. It reported all images found and did not use a reference data set. Roussev et al. (2013) treat triage as an intrinsic part of the digital forensic process. They advocate that target acquisition and forensic processing should be done in parallel, with results being reported as soon as they are available. Their model requires that data is analysed as it is being acquired so that analysis should finish at the same time as the data acquisition. This requires that analysis, including cryptographic hashing and lookup, similarity hashing, decompression, file content extraction and indexing all be done in parallel at the speed of data acquisition. To do this needs considerably more computing power than is available in the field. We would argue that there is still a demonstrated need for a separate initial triage stage prior to the computing power for this acquisition and forensic processing becoming available to the investigator. Garfinkel (2013) developed *bulk_extractor* to scan an entire disk image. A disk image is scanned without reference to partitions or the file system metadata. Since this method does not have to find, extract identify and process *files*, it is shown to be at least ten times faster than traditional file based methods. It used a number of filters running in parallel, each optimised to detect patterns

indicative of one of the common artefacts required for a digital investigation such as telephone numbers, e-mail addresses or credit card numbers. However, the process did not use a database of known content.

Hashing and Bloom filters – identifying known content

Identification of known files attempts to ascertain whether the device contains material that has originally come from some known '*good*' corpus (whitelists) such as the National Software Reference Library hash database (NSRL) issued by The National Institute of Standards and Technology (NIST), or from a corpus of illegal material. Examples of this include the database of child pornography image hashes held by Police Scotland or the Team Cymru Malware Hash Registry (blacklists). These databases hold fixed length cryptographic hashes of each file, for example the 128 bit MD5 hash or the 160 bit SHA-1, rather than the files themselves. An unknown file may be identified by calculating its hash and comparing this hash against the database. Such file based systems cannot be used when sectors/clusters are being sampled since we are accessing the disk without reference to any file system.

In Roussev et al. (2006) it was noted that the hash database for block level MD5 hashes of a 512 GB hard disk would require 32 GiB of RAM which was too big to use in a common workstation but that the use of a Bloom filter could reduce this by an order of magnitude for the cost of a small false positive rate. Bloom filters were therefore an efficient way to store large sets of hashes. Kornblum (2006) developed **ssdeep**, a system of fuzzy hashing which created a similarity hash of a file that could be used to detect similar files. Roussev (Roussev et al., 2010) developed the application **sdhash** which improved on the performance of **ssdeep** by using similarity digests created from statistically improbable features. Roussev et al. (2013) used this tool in streaming mode to hash data blocks and query a reference database at disk read speed. They showed that using a 48-core server the maximum size of reference database that could be queried at a read speed of 100 MB/s was 15 GB. Similarity digests produced by **sdhash** are up to 2.6% of the original data size thus the Police Scotland database could just be accommodated but the system requirements are beyond what can be expected in the field. They suggested that similarity digests should only be used in the field with reference databases up to 1 GB which would not be useable in our scenario.

Farrell et al. (2008) investigated the use of Bloom filters for the distribution of the NSRL hash database. Although they rejected the idea of distributing the database in this format since it is relatively easy to engineer any malicious file to give a false positive if the content of the Bloom filter is known, they found that Bloom filters were good for providing high speed matches against hash sets.

Garfinkel et al. (2010) look mainly at file fragment type discrimination but mention that statistical sector sampling could be used for detecting contraband data. EnCase (Guidance Software) includes a script – File Block Hash Map Analysis – which block hashes known files and then searches selected areas of the disk. However, they state that due to performance issues it is only suitable for a small number of target files. Young et al. (2012) came closest to meeting our requirements. They created a database of 1 billion sector hashes with the intention of deploying the system on a laptop. They used a hybrid approach by using a Bloom filter to screen out negative results so that only queries for hashes that may be in the corpus were passed to their customised hash database. This use of pre-filtering had been suggested by Farrell et al. (2008) and Garfinkel et al. (2010) and makes use of the Bloom filters very fast lookup for items not in the set. Using an SQL database on a Dual Xeon processor setup, each with 16 cores and 128 GiB RAM they could not perform hash lookups fast enough to keep up with the sector reads. However, they found that using the customised database that they developed on a well specified laptop with 8GiB of RAM and an SSD, they could perform lookups faster than sectors can be read from the drive being triaged.

Background

This section gives a background and justification to some of the ideas, methods and mathematical tools that will be used.

Sampling-choice of block size

A cluster is the minimum amount of data that can be accessed (either read or written) by the file system. Many file systems use a cluster size of 4 KiB by default. The default cluster size in all Microsoft operating systems since NT3.51 is 4 KiB in all disk sizes up to 16 TB (Microsoft, 2013). Since 2011, all disk drive manufacturers have standardised on a sector size of 4096 bytes, using an emulation mode (AF 512e) to present 512 byte sector size to a legacy file system where needed (IDEMA, 2013). Most files of interest will be considerably larger than 4 KiB and so the decreased granularity of working with 4096 byte blocks will be less significant (Garfinkel et al., 2010). Garfinkel (2010) also showed that the random sampling of disk drives using 512 byte sectors or 4096 byte blocks gave very similar results. Using 4 KiB block size will reduce the hash database size by a factor of 8 compared to that needed to store hashes of 512 byte sectors. For these reasons we intend to use a block size of 4096 bytes. There is a potential problem in that if the file system with cluster size of 4 KiB is used to write to an old disk with 512 byte sectors then the cluster might not be aligned on an 8-sector boundary and thus not be detected by our system (Young et al., 2012). We are taking a pragmatic approach to this and will address this issue if it appears significant in field-testing.

Choice of sample size

Choosing a sample of k items from a population of n items without replacement (no sector will be read twice) is represented by the Hypergeometric distribution (Zhang, 2008). Suppose a disk drive has n clusters. Further suppose that it has t clusters of interest – the target – from some illicit file or files. When we sample k clusters randomly from the total of n on the disk then the chance of **not** obtaining any target clusters is the same as drawing our sample entirely from the none-target clusters. Using the

Table 1
Target size 4 MiB. Probability of a sample containing a block of the target.

	Sample size									
		100,000	200,000	300,000	400,000	500,000	600,000	700,000	800,000	900,000	1,000,000
Disk size (GB)	120	0.96	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
	250	0.79	0.96	0.99	1.00	1.00	1.00	1.00	1.00	1.00	1.00
	320	0.71	0.91	0.97	0.99	1.00	1.00	1.00	1.00	1.00	1.00
	500	0.54	0.79	0.90	0.96	0.98	0.99	1.00	1.00	1.00	1.00
	1000	0.32	0.54	0.69	0.79	0.86	0.90	0.94	0.96	0.97	0.98

standard representation for combinations (Weisstein, 2014), the probability of drawing our sample from the non-target clusters is

$$\frac{\binom{n-t}{k}}{\binom{n}{k}} \tag{1}$$

The standard definition for these binomial coefficients is[1]

$$\begin{aligned}\binom{n}{k} &= \frac{n!}{k!(n-k)!} \\ &= \frac{\Gamma(n+1)}{\Gamma(k+1)\Gamma(n-k+1)}\end{aligned} \tag{2}$$

We have re-written the factorials in terms of the Gamma function (Γ) since, when we are using sample sizes of 1 million in populations of hundreds of millions, such as the number of 4096 blocks in a 1 TB disk drive, then the factorials quickly become large and so we need to work with logarithms to avoid numerical overflow. The Gamma function for positive integers is defined as $\Gamma(n) = (n-1)!$ (Weisstein, 2014) and many numerical applications have a highly accurate log Gamma function. Using the basic laws of logs in equation (2) we get

$$\log\binom{n}{k} = \log\Gamma(n+1) - \log\Gamma(k+1) - \log\Gamma(n-k+1)$$

Therefore

$$\begin{aligned}\log\frac{\binom{n-t}{k}}{\binom{n}{k}} &= [\log\Gamma(n-t+1) - \log\Gamma(k+1) - \log\Gamma(n-t-k+1)] - [\log\Gamma(n+1) - \log\Gamma(k+1) - \log\Gamma(n-k+1)] \\ &= \log\Gamma(n-t+1) - \log\Gamma(n-t-k+1) + \log\Gamma(n-k+1) - \log\Gamma(n+1)\end{aligned}$$

If we represent this calculated log value by x, then our probability of no hits in the sample is e^x, the inverse log of x. Hence, the probability of 1 or more hits is $1 - e^x$. Using this formula, Tables 1 and 2 show the probabilities of '*hitting*' a small 4 MiB and a larger 20 MiB target for a variety of sample sizes for common disk capacities.

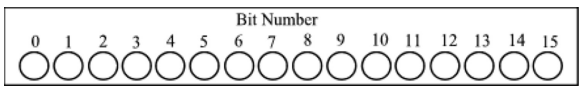
Bloom filters

The Scottish Police database of child pornography holds 5.1 million images in Category 1 (the most serious) and a further one million in categories 2–5. If we take an average jpeg file size of 100 KiB, then each of these 6 million images will take up 25 disk clusters of 4 KiB. A database of these cluster hashes would therefore contain 150 million hashes. If MD5 is used as the hash algorithm, then each hash is 128 bits or 16 bytes. Thus our hash database would take up 150 million $\times$ 16 bytes or 2.4 GB. We are constrained for the feasibility study to use '*legacy*' equipment which may well have a 32 bit operating system or limited RAM and so such a block hash database would be too large to hold in RAM. If the database were disk based then hash lookups would be far slower than the sequential read being used for our disk sampling thus would prove a severe bottleneck. One solution to provide a space efficient means of testing whether or not an element is a member of a set, at the controllable risk of false positives, is to use a Bloom filter (Bloom, 1970).

A Bloom filter consists of a set of hash functions and a bit array of a fixed length. All bits in the bit array are initialised to zero. Let us look at a very simple example to illustrate the principle which will ease the understanding of the theory.

[1] The formula $p = 1 - \prod_{i=1}^{n}((N-(i-1))-m)/(N-(i-1))$ can also be used (Young et al., 2012) but does not lead to such an easily computable result.

Suppose we have a bit array of 16 bits.

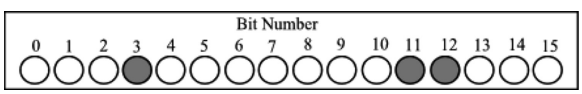


We now need a number of independent hash functions which hash input of arbitrary length and produce a value between 0 and 15, the address range of our bit array. Let's call them $h_1()$, $h_2()$ and $h_3()$.

Now we add the string 'Jupiter' to our filter. We hash 'Jupiter' with each of our hash functions. Suppose they generate

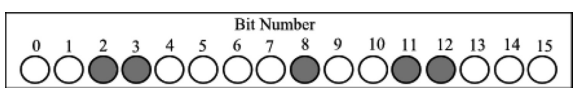
$h_1(\text{'Jupiter'}) = 3 \quad h_2(\text{'Jupiter'}) = 12 \quad h_3(\text{'Jupiter'}) = 11$

We now set the corresponding bits 3, 12 and 11 in our bit array.



Now adding 'Venus' in the same manner gives us

$h_1(\text{'Venus'}) = 11 \quad h_2(\text{'Venus'}) = 2 \quad h_3(\text{'Venus'}) = 8$



Notice that Bit 11 was already turned on. It is now storing information about both 'Jupiter' and 'Venus'. As we add more items to the filter it may store information about some of those as well. It is this overlap which gives Bloom filters their compactness. Any one bit may be storing information about multiple items simultaneously.

To check if an item already exists we just follow the same process of hashing it and check if the bits at all those positions are turned on. If they are then it is probably in the filter. It is only '*probably*' in the filter because there is the possibility of a false positive. Suppose we test if 'Mars' is in the filter.

$h_1(\text{'Mars'}) = 11 \quad h_2(\text{'Mars'}) = 3 \quad h_3(\text{'Mars'}) = 8$

All these bits are set so the filter reports that 'Mars' belongs to the set. Thus we see that a false positive is possible in a Bloom filter. However a false negative is not. If any bit is 0 for an examined hash, then that element cannot possibly be in the set otherwise the bit positions at its hash values would have been set to 1. This illustrates why searching elements not in the set is considerably faster. We can terminate the search as soon as an unset bit is encountered.

In this example we had a number of variables:

m = the number of bits in the array which represents the Bloom filter. Initially all bits are set to 0
n = the number of elements added to the filter
k = the number of independent hash functions $h_1, h_2, \ldots, h_k$ used.

These variables are important building blocks for the theory. Notice that our Bloom filter had $m = 2^4$ bits and our hashes needed to generate a hash with a bit length of 4 bits. In general a Bloom filter of size 2^p needs a hash of length p bits.

Each hash function generates a hash value in the range $(0, \ldots, m-1)$ for any input. To insert an element e_i from the set $E = \{e_1, e_2, \ldots, e_n\}$ of elements to be added to the filter, the hash values $h_1(e_i), h_2(e_i), \ldots, h_k(e_i)$ are calculated and the corresponding locations in the bit array are set to 1. This is repeated for each element to be added to the filter. To check if an element belongs to the set, we calculate its k hashes. If all these bits are set to one then the item may be in the set. If not all bits are 1, we know that the element is not in the set.

Design

Designing the filter

The sample sizes required for a variety of target and disk sizes are shown in Table 3. Initial contact with forensic investigators at Police Scotland has indicated that cases have many files. The most recently reported had over 46,000 category 1–5 images with an average size of 270 KB. On disk this gives a target size of approximately 12 GB. If we take this to be excessive and for the moment assume that an average case would involve a target size of 500 MB, then we see from Table 3 that on a 1 TB disk we would need a sample size of only 13,300 to give a probability of 99.9% of detecting the target and this could be done in minutes.

We can use the sample sizes to choose design parameters for our Bloom filter.

Table 2
Target size 20 MiB. Probability of a sample containing a block of the target.

	Sample size									
	100,000	200,000	300,000	400,000	500,000	600,000	700,000	800,000	900,000	1,000,000
Disk size (GB) 120	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
250	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
320	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
500	0.98	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
1000	0.86	0.98	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00

Table 3
Sample size required for 99.9% probability of detecting a target for a variety of target and disk sizes.

Disk size (GB)	Target size (MB)							
	4	20	50	100	200	300	400	500
250	416,700	79,700	32,600	16,300	8200	5600	4100	3400
320	520,100	106,700	40,700	20,400	10,200	7100	5100	4200
500	833,300	159,400	65,200	32,600	16,300	11,200	8100	6700
1000	1,666,600	318,900	130,400	65,200	32,600	22,500	16,300	13,300

Mitzenmacher and Vadhan (2008) show that the false positive rate for a Bloom filter in terms of our values m, n and k is

$$P(\textit{false positive}) \approx \left(1 - e^{-\frac{kn}{m}}\right)^k \quad (3)$$

and that k is optimal when
$k = \frac{m \log(2)}{n}$

Using these formulae, we can engineer our Bloom filter for our application. We calculated that our envisaged database would contain 150 million 4 KiB blocks and so if we take $n = 200{,}000{,}000$ we allow for growth in the database. We look at the false positive rate for a variety of combinations of k and m. Table 4 is generated using formula (3).

We need to know what false positive rate is acceptable. If possible, we would like no false positives in our sample. In the worst case scenario, a 1 TB drive with a small 4 MiB target we will be taking over one million samples (see Table 1). We would hence like a false positive rate of less than 1 in a million. From Table 3 we can see that to meet this requirement the Bloom filter is going to be approximately 800 MiB in size and use 10 hashes, or 1 GiB in size and use 8 hashes. If the device being examined has a 1 TB drive then it is very likely to have 2 GiB RAM or more and a 1 GiB array would be acceptable. The larger Bloom filter size is a trade-off giving us fewer hashes. This is important since the speed of our algorithm is inversely proportional to the number of hashes. The more hashes that need calculated the slower the process. In addition we noted earlier that a bloom filter of size 2^b bits needed a hash of length b bits. Our 1 GiB array has 2^{33} bits and so we need a hash of length 33 bits. This can be generated efficiently by using SHA-384 as our hash algorithm. This generates a 384 bit hash. We will use the first 264 bits of the hash as 8×33 bit hashes. These hashes meet the requirement for independence since the SHA-384 hash can be assumed to be collision resistant and therefore any single bit or subset of bits may be taken to be independent random variables (Roussev et al., 2006). Since we are using eight hashes, each of which is used to set one bit in the Bloom filter, each entry in the filter requires a maximum of eight bits or the equivalent of one byte (as noted before, some of the bits might already be set) and so adding 200 million bytes created from the Police Scotland cluster hash database to a 1 GiB array leaves the Bloom filter lightly loaded.

Having designed our filter to cope with the worst case scenario of a small 4 MiB target on a large disk then, given the target sizes envisaged by Police Scotland, our sample sizes will be much smaller. Our false positive rate should then be much better than required.

Testing and results

A system was set up using the design created above.

Bloom filter creation

We used data from random.org (Random.org) to generate entries for 200 million blocks in our Bloom filter and then added the block hashes for 100 MB of real images. The Bloom filter was tested first by checking the hashes of one million of the block hashes originally included in the filter and all were reported to be present. Then one million block hashes were created with further data from random.org and tested for inclusion in the filter. There was only one positive match which agreed with theory since the false positive rate for the Bloom filter is 0.000000696 or 1 in 1,437,184 although this could be caused by one of the generated blocks matching one previously used. In either case the filter is operating as designed.

Table 4
False positive probabilities for varying values of m and k.

k	m = Bloom filter size (MiB)					
	512	600	700	800	900	1024
4	0.000834149	0.000466407	0.000263157	0.000159487	0.000102189	0.000062537
6	0.000209786	0.000091112	0.000039869	0.000019271	0.000010073	0.000004912
8	0.000087538	0.000030242	0.000010473	0.000004100	0.000001769	0.000000696
10	0.000051133	0.000014373	0.000004016	0.000001292	0.000000466	0.000000149
12	0.000037893	0.000008855	0.000002034	0.000000546	0.000000166	0.000000044
14	0.000033433	0.000006629	0.000001274	0.000000289	0.000000075	0.000000017
16	0.000033607	0.000005763	0.000000942	0.000000183	0.000000041	0.000000008

Table 5
Core i3 Desktop PC sampling accuracy and speed.

Disk and size	Target size	Samples	Hits	False positives	Time min:sec
250 GB SSD	4 MB	416,700	31	0	00:42
250 GB SSD	20 MB	79,700	7	0	00:07
250 GB USB HDD	4 MB	416,700	7	0	35:02
250 GB USB HDD	20 MB	79,700	4	0	08:31
1 TB HDD	4 MB	1,666,600	6	1	108:54
1 TB HDD	20 MB	318,900	8	2	27:42

Testing

Testing was done on two different computers – a desktop PC with a Core i3 processor and 4 GB RAM, HDD, SSD and external USB HDD drives and a netbook with an Intel Atom CPU, 2 GB RAM, a 250 GB SSD and a USB attached HDD.

A selection of 4 MB of the real images was added to each of the disk drives being tested. The sampling, hashing and Bloom filter lookup was done. A further 16 MB of contraband was then added to each disk and the process repeated.

The Microsoft NTFS File Sector Information Utility (Microsoft, 2530) was used to look up each hit to ascertain if it was a true hit or a false positive. This utility allows the user to enter the drive and sector number and reports from the MFT the full path of the file, if any, which the sector belongs to.

Results

The results are shown in Tables 5 and 6.

Analysis

Although some false positives were encountered, the methodology can still be considered 'fit for purpose' since these results would all have been positive for the existence of contraband data. The contraband data introduced consisted entirely of JPEG images and all the false positives resulted from a Windows Media Player recorded TV programme (.wtv format).

For running on the Netbook we had to modify the system to allow for the shortage of RAM. We created the same data to fill a 512 MiB filter with 10 hashes instead of a 1 GiB filter with eight hashes. The extra hashing during lookups would account for some of the lack of speed on the Netbook but it is clear that since the SSDs on the PC and Netbook were similar, the performance of the Intel Atom processor is the bottleneck rather than the SSD. This effect is less marked with the slower I/O of the HDD. Although the theoretical false positive rate for the Bloom filter in this configuration is 0.00005 (1 in 20,000) we experienced no false positives.

Table 6
Intel Atom Netbook sampling accuracy and speed.

Disk and size	Target size	Samples	Hits	False positives	Time min:sec
250 GB SSD	4 MB	416,700	8	0	09:33
250 GB SSD	20 MB	79,700	12	0	01:54
250 GB USB HDD	4 MB	416,700	5	0	44:34
250 GB USB HDD	20 MB	79,700	7	0	11:04

One USB HDD that we used did not give any hits. The drives partition was looked at and found to start at LBA 63 as older disks did. We used a partition manager to quickly re-align the disk partition to LBA 2048 and we could scan it as normal. This block alignment is related to the problem described by Young et al. (Young et al., 2012).

Conclusions

We have shown that a fast initial scan of a device using legacy equipment is feasible. Although typical cases may contain many files, we tested with '*worst case*' target sizes and found that we could produce a result with 99.9% certainty within minutes. If larger target sizes are selected then the sample size required reduces and the scan will be faster. This could both ease the backlog of devices needing further investigation and be ethically more acceptable. While an investigation is backlogged an implied or imagined accusation can result in harm to social and professional relationships, or a guilty party, possibly a danger to society, could be left at large.

We have presented a theoretical justification to our approach and tested it on legacy computing equipment.

Future work

Initial testing with SD cards and images of virtual machines is positive. An investigation of the applicability of the methodology for scanning a variety of digital devices needs to be done. The technique of sampling and checking of devices against any large reference data set is widely applicable, for example to intellectual property theft and possibly to IP streams.

References

ACPO (Association of Chief Police Officers). ACPO managers guide good practice and advice guide for managers of e-crime investigation.
Bloom BH. Space/time trade-offs in hash coding with allowable errors. Commun ACM 1970;13(7):422–6.
Casey E, Katz G, Lewthwaite J. Honing digital forensic processes. Digit Investig Sep. 2013;10(2):138–47.
Farrell P, Garfinkel SL, White D. Practical applications of bloom filters to the NIST RDS and hard drive triage. In: 2008 Annual computer security applications conference (ACSAC); Dec. 2008. p. 13–22.
Fujitsu Technology Solutions GmbH. White paper FUJITSU PRIMERGY server. 2011.
Garfinkel S. Random sampling with sector identification. Naval Postgraduate School presentation. 2010.
Garfinkel S. Digital media triage with bulk data analysis and bulk_extractor. Comput Secur 2013;0029.
Garfinkel S. Digital forensics research: the next 10 years. Digit Investig Aug. 2010;7:S64–73.
Garfinkel S, Nelson A, White D, Roussev V. Using purpose-built functions and block hashes to enable small block and sub-file forensics. Digit Investig Aug. 2010;7:S13–23.
Gillam W, Rogers M. File hound: a forensics tool for first responders. In: DFRWS; 2005. p. 1–7.
Guidance Software. Encase forensic v7 [Online]. Available: https://www.guidancesoftware.com/products/Pages/encase-forensic/overview.aspx [accessed: 23.03.14].

Horsman G, Laing C, Vickers P. A case-based reasoning method for locating evidence during digital forensic device triage. Decis Support Syst May 2014;61:69–78.

IDEMA. The advent of advanced format. International Disk Drive Equipment and Materials Association; 2013 [Online]. Available: http://www.idema.org/?page_id=2369 [accessed: 27.08.13].

Kornblum J. Identifying almost identical files using context triggered piecewise hashing. Digit Investig 2006;3(Suppl.):91–7.

Kryder MH. After hard drives—what comes next? IEEE Trans Magn Oct. 2009;45(10):3406–13.

Microsoft. Microsoft default cluster size. Knowl Base 2013 [Online]. Available: http://support.microsoft.com/kb/140365 [accessed: 08.07.14].

Microsoft. Microsoft NTFS file sector information utility [Online]. Available: http://support.microsoft.com/kb/253066 [accessed: 07.07.14].

Mitzenmacher M, Vadhan S. Why simple hash functions work: exploiting the entropy in a data stream. In: Proceedings of the 19th annual ACM-SIAM symposium on discrete algorithms; 2008. p. 746–55.

Pollitt MM. Triage: a practical solution or admission of failure. Digit Investig Sep. 2013;10(2):87–8.

Random.org. Pregenerated random numbers [Online]. Available: http://www.random.org/files/.

Roussev V. Data fingerprinting with similarity digests. In: Chow K-P, Shenoi S, editors. Advances in digital forensics VI. Springer Berlin Heidelberg; 2010. p. 207–26.

Roussev V, Quates C, Martell R. Real-time digital forensics and triage. Digit Investig 2013;10(2):158–67.

Roussev V, Chen Y, Bourg T, Richard GG. md5bloom: forensic filesystem hashing revisited. Digit Investig Sep. 2006;3:82–90.

Shaw A, Browne A. A practical and robust approach to coping with large volumes of data submitted for digital forensic examination. Digit Investig Sep. 2013;10(2):116–28.

Walter C., Kryder's law, *Sci Am*, August 1, 2005. [Online] Available: http://www.scientificamerican.com/article/kryders-law/?page=1 [accessed: 14.09.2014]

Weisstein E. Combinations. MathWorld – A Wolfram Web Resource. [Online]. Available: http://mathworld.wolfram.com/Combination.html [accessed: 07.07.14].

Young J, Foster K, Garfinkel S, Fairbanks K. Distinct sector hashes for target file detection. Computer 2012;45(12):28–35.

Zhang L. Hypergeometric distribution. Appl Stat 2008;1 [Online]. Available: http://www.math.utah.edu/~lzhang/teaching/3070summer2008/DailyUpdates/jun23/sec3_5.pdf [accessed: 10.08.14].