On Secure Data Deletion

Joel Reardon, David Basin, Srdjan Capkun
Institute of Information Security, ETH Zurich
{reardonj,basin,capkuns}@inf.ethz.ch

Abstract

Being able to permanently erase data is a security requirement in many environments. But what that actually means for a concrete setting varies widely. This article explores different approaches to securely deleting data and identifies key ways to classify them. We describe adversaries that differ in their capabilities, we show how secure deletion approaches can be integrated into systems at different interface layers, and we identify the assumptions made about the interfaces. Finally, we examine the main properties of secure deletion approaches.

***Keywords*—Secure deletion, Flash memory, Magnetic memory, File systems**

1 Introduction

During New York City's 2012 Thanksgiving day parade, sensitive personal data rained from the sky. Makeshift confetti, formed out of shredded police case reports and personnel files, landed on spectators who observed something peculiar about it: having been shredded horizontally, entire stretches of text (names, social security numbers, arrest records, etc.) were completely legible [1]. It is likely that the documents were shredded to securely delete the *sensitive* data they contained (and not simply to make confetti).

Secure data deletion is the task of deleting data from a physical medium (anything that stores data, such as a hard drive, a phone, or a blackboard) so that the data is irrecoverable. This irrecoverability is what distinguishes secure deletion from *regular* file deletion, which deletes unneeded data only to reclaim resources. We *securely* delete data to prevent an adversary from gaining access to it.

The Need for Secure Deletion In the physical world, the importance of secure deletion is well understood: sensitive mail is shredded; published gov-

ernment information is selectively redacted; access to top secret documents is managed to ensure all copies can be destroyed when necessary. In the digital world, the importance of secure deletion is also well recognized. Legislative or corporate requirements can require secure deletion of data prior to disposing or selling hard drives; particularly when the data is considered to be sensitive, for example, health data, financial data, trade secrets, and privileged communications. Regulations may change or new ones enforced, causing data assets to become data liabilities. This can entail the sudden need to securely delete vast quantities of data. An example of this is the United Kingdom's demand that Google securely delete Wi-Fi data illegally collected by Google's Street View cars, wherever and however it was stored [2].

Secure deletion is not limited to one-off events. A network service operator may collect logs for intrusion detection or other administrative purposes. However, a privacy-focused network service (such as an anonymous message board, mix network, or Tor relay) may wish to securely delete any log data once it is no longer needed, requiring secure deletion on a continuous basis. Network services may also need secure deletion simply to comply with regulations regarding their users' private data. Two examples are the European Union's *right to be forgotten* [3] that would force companies to store personal data in a manner that supports the secure deletion of all data about a particular user upon request, and California's legislation that enforces similar requirements only for minors.

Secure deletion is also needed to achieve other security properties. An example is *forward secrecy*: the desirable property that ensures that the compromise of a user's long-term cryptographic key does not affect the confidentiality of past communications. This is often achieved by protecting the communications with session keys securely negotiated using the long-term key. Forward secrecy then requires secure deletion to ensure that both session keys and negotiation parameters are irrecoverable. Another example is the Ephemerizer [4], which provides users with ephemeral communication by associating each message with a time-based key; the eponymous trusted-third party uses secure deletion to ensure that these keys *expire* at the correct time, making the communications they encrypt irrecoverable. Secure deletion is required to implement the ephemerizer's key expiration functionality.

Find and Delete As is often the case in the digital world, a straightforward security problem is fraught with challenges and complications, and secure deletion is no exception. Digital data is effortlessly replicated, often without any record. Simply *finding* where data is stored over a vast number of computer systems and storage media may present a logistical nightmare, particularly when servers replicate data, go offline indefinitely, crash during copy operations, and have their hardware swapped around. Even a single copy on a single hard drive may be duplicated without notice, for instance, when the file system rearranges its storage during defragmentation.

Even when all the locations where data is stored can be found, it still may not be possible to securely delete the data. Overwriting magnetic data may still leave analog remnants available to adversaries with forensic equipment. Flash memory cannot be efficiently overwritten directly and so new versions of files are instead written to a new location with the old one left behind. High-capacity magnetic tapes must be written end-to-end; worse, they are then often shipped off to a vault for off-line archiving. Optical discs like DVDs are a kind of WORM medium—write once, read many—and such media only achieve secure deletion through physical destruction. The steps to achieve secure deletion vary depending on the actual storage medium being used.

The Deletion Confusion Another challenge in secure deletion is that many users are unaware that additional steps are needed to sanitize their storage media. All modern file systems offer users the ability to "delete" their files. However, they all implement this feature by just *unlinking* the file. Abstractly, unlinking a file only changes file system metadata to state that the file is now "deleted"; the file's full contents remain available. This is done for efficiency reasons—deleting a file would require changing all its data, while unlinking a file requires only changing one bit. File system designers have consistently made the assumption that the only reason a user deletes a file is to recover storage resources to allocate to new files. The resources are assumed to be free, but it is only when they are needed that they are reclaimed by another entity.

Even for those who know the best practices for secure deletion, the nature of digital information makes it hard to verify that the data is indeed irrecoverable. The user interfaces for deleting digital data simply do not provide the same rapid assurance of secure deletion as does a pile of (vertically) shredded mail. Forensic investigators of Chelsea Manning's[1] laptop, for instance, discovered that he had tried to securely delete the contents of his laptop by overwriting its contents 35 times—an aggressive approach—but, unknown to him, the operation had stopped midway and left most of the data intact [5].

2 Adversarial Capabilities

Having now established secure deletion as an important security problem, the next step is to consider exactly from whom we are deleting the data, i.e., our adversarial model. Different adversaries have different strengths and so a secure deletion approach must be designed to thwart the appropriate adversary. Here we describe three dimensions in the space of adversaries.

[1] Known as Bradley Manning at the time of the investigation.

The Unanticipated Adversary In the waning days of East Germany—after the Berlin wall had fallen—the secret police were kept busy frantically destroying their vast collection of paper documents to avoid their own prosecution. Being an organization bent on collecting as much data as possible—literally kilometres of filing cabinets—their own high-power shredders were too limited and broke under the strain. The agents worked around the clock for three months, manually ripping up documents that now form the pieces of the world's largest jigsaw puzzle [6].

The lesson for us is that the adversary can arrive unanticipatedly. Much existing research focuses on the case where users hand over their storage media to an adversary, but can first perform elaborate sanitization procedures and only yield control after completion. In this case, factors such as efficiency and wear are less relevant in the design of an approach. In the real world, however, adversaries can arrive without warning: your mobile phone can be stolen, your computer systems can be broken into, and police can seize your storage media when executing a warrant or subpoena. In these cases, no elaborate, extraordinary sanitization can be performed; the only assurance of secure deletion available is that which comes from the precautions taken as a matter of routine. Consequently, issues such as efficiency, device wear, and other inconveniences become relevant. Typically, this manifests itself as the classic security versus usability trade-off: prompter secure deletion (more security) at the cost of convenience (less usability). Approaches that destroy the storage medium or securely delete all data thereon are no longer suitable against an adversary that can strike without warning.

The Forensic Adversary Peter Gutmann famously observed that, for magnetic media, the precise analog voltage of a stored bit offers insight into the medium's previously held values [7]. This is because analog-to-digital conversion operates analogously to an error-correcting code: a range of analog values are mapped to a single binary digit. The larger space of analog values can therefore accommodate more data, independent of the "official" bit stored in that location. Precision can be improved by using more advanced equipment; storage manufacturers, generally developing next generation hardware with double the precision as the hardware available on the consumer market, are better able to view these analog remnants on older drives that clumsily wrote on twice as many molecules as are now needed.

Gutmann's solution to this problem is to overwrite data multiple times; the most aggressive of his proposed solutions involves 35 passes over the data, each time writing with different patterns. While such time-consuming methods may no longer be needed on modern magnetic hard drives, it remains safe to say that each additional overwrite does not make the data easier to recover—in the worst case it simply provides no additional benefit. More importantly, Gutmann's results highlight that analog-to-digital conversion can leave remnants on any storage medium. When securely deleting data, it is important to con-

sider the "secrecy" of the data with respect to the adversary's sophistication. An adversary who steals your phone for your passwords uses less sophisticated methods than an extremely well-funded one determined to exfiltrate as much data as possible.

The Coercive Adversary Some may not worry about sanitizing their storage media before disposal because they always used full-disk encryption, thus ensuring that no data is ever written to their storage medium in plain-text. Without the secret key or password, the adversary is helpless to recover this data. There is no need to overwrite it 35 times!

There are still cases, however, where encryption alone is insufficient: keys can be compromised, for example, weak passwords can be guessed. Moreover, there are *coercive* adversaries that can force users to reveal their secret keys and passphrases. Two real-world examples are crossing (particular) national borders and legal subpoenas. In both cases, users may not only be forced to give access to their storage media but also to provide any keys or passphrases required to access the data—under threat of obstruction of justice, or worse. A coercive adversary is equivalent to *the user* trying to recover the data: data is only securely deleted if the user's own best efforts are unable to recover the data.

How does a coercive border crossing adversary differ from an adversary with a subpoena? In the same way that selling a hard drive differs from its theft: whether the user or the adversary chooses the physical medium's access time. Before crossing a border, the user is free to execute any costly, elaborate, extraordinary, secure-deletion proceedure, whereas in the latter case the user must rely only on established routine practices for secure deletion.

3 Deletion by Layers

If given access to a storage medium and told to securely delete some data, how would you do it? Presumably, you would first check for a secure deletion feature in the medium's interface, and failing that, use the available interface functions in a creative way to achieve the goal. In the physical world there are many options at your disposal: you can scribble on the paper, shred it, or set it ablaze. In the digital world, however, your interface to the storage medium is often constrained.

There are many abstraction layers between the user and the physical storage medium. A secure deletion approach can be integrated into any of these layers. The further away from the actual storage medium you are, however, the less able you are to directly manipulate stored data. Storage medium access becomes more abstracted as additional layers are added, such as virtualization. The only recourse to compensate for this is to make stronger assumptions

about that interface's actual behaviour. Overwriting files with zeros, for example, makes the assumption that this actually replaces the unique copy on the storage medium. Degaussing a hard drive, in contrast, makes only simple electromagnetic assumptions.

This problem of granularity is pervasive to low-layer secure deletion approaches. While the access to the physical medium is less abstracted, the information about what actually should be deleted—a file, an SMS, a row in a database—is simply not available.

The choice of layer for a secure deletion approach is a trade-off between these factors. At the physical layer, we can *ensure* that a data object is truly irrecoverable, and at the user layer, we can easily *identify* the data object that should be made irrecoverable. Indirect information is given to the file system, e.g., by unlinking a file or hole-punching a sparse file. However, no information goes further down; the file system knows the space is free and may reallocate it for another file at a later time. The storage medium, however, assumes the data should be retained until it is replaced with a new version.

Physical Interfaces and Digital Controllers The lowest layer is always the physical medium itself. Its interface is also physical: depending on the medium it can be degaussed, incinerated, or shredded, and NIST provides an extensive description of how to faithfully destroy all data on a variety of storage media [8]; in many cases the physical destruction of the storage medium is a consequence of the operation. For example, floppy disks must be shredded or incinerated; compact discs must be incinerated or subjected to an optical disk grinding device. Not all approaches work for all media types—you can put any medium in an NSA/CSS-approved degausser; whether or not this results in any secure deletion depends on whether the data is stored with magnetic alignment.

A physical medium is often operated by a controller that translates between the physical medium's analog format (e.g., magnetic voltages) and the data format (e.g., binary) used at higher layers. Several standardized interfaces exist for controllers that permit reading and writing of fixed-sized blocks (e.g., ATA and SCSI). Given the controller interface, there are different actions one can take to securely delete data. Either a single block can be overwritten with a new value thereby displacing the old one, or all blocks can be overwritten. However, unless you know exactly which device blocks store the data you want deleted (i.e., the file system's organization of data into blocks) then it is not possible to securely delete with precision. Instead, the controller must sanitize every block to achieve secure deletion. Indeed, both ATA and SCSI offer such a sanitization command, called either *secure erase* or *security initialize*. They work like a button that erases all data on the device by exhaustively overwriting every block. NIST recommends using these commands to securely delete magnetic hard drives in a non-destructive way.

User-Level Approaches At the other extreme, user-level approaches are simple utility programs that users can run on their computers. The program's interface to the storage medium is limited to what is offered by the file system; typically a POSIX-compliant file system interface. Achieving the secure deletion of files must be done with this limited interface, which provides only file manipulation such as reading, writing, creating and unlinking. Little else is guaranteed about the behaviour of the file system, the underlying device driver, and the further underlying hardware controller; any of which may complicate secure deletion.

There are two classes of user-level approaches to secure deletion, which we call *overwriting* and *filling*. Overwriting approaches work by opening the file to be deleted and *overwriting* its contents with new, insensitive data, e.g., all zeros. When the file is later unlinked, only the contents of the most recent version are stored on the physical medium. Overwriting assumes that each file block is stored at known locations and when the file block is updated, then all old versions are replaced with the new version—we call this in-place updates. Note, however, if the file system does not perform in-place updates, then user-level overwriting tools may silently fail. The majority of sophisticated file systems do not, in fact, use in-place updates because journalling is an out-of-place update technique. Usability concerns also exist, because users are expected to use these tools whenever deleting sensitive files—they must change their routine behaviour. Care must be taken to avoid applications that create and delete their own files: a word processor that creates temporary swap files (possibly a near-exact replica of the file) probably does not securely delete these files with any non-default tool.

The other class is filling. Filling approaches work by *filling* the entire storage medium's empty capacity with insensitive data, e.g., zeros. Users do not need to take any special actions when using their applications and deleting their files; at some later point the file system's empty space is filled, hopefully securely deleting anything that has been previously deleted. Filling approaches rely on the assumption that the file system reports itself as unwritable only when there are no longer any unused blocks on the storage medium. It turns that the filling assumption holds true for many more file systems than the overwriting assumption. After all, using all the available space on a storage medium is a fundamental *feature* of any file system, while in-place updates are intentionally *avoided* for crash recovery, copy-on-write versioning, and rapid (seek-free) writes. Filling comes with a cost, however, as its running time is proportional to the empty space on the file system. Moreover, if the storage medium is susceptible to *wear*, such as flash memory, then the frequency of filling must also be controlled. Since filling runs only periodically, data may be deleted at one time and only securely deleted when filling is subsequently run, resulting in an increased *deletion latency*—the time the user must wait until data is securely deleted.

Block-Based File System Approaches A variety of approaches integrate secure deletion features into the file system itself. This is sensible because file systems are designed to know exactly when data is no longer needed. File systems can also compensate for the secure-deletion complications introduced by additional features the file system adds, such as journalling, versioning, and replication. Users do not need to remember to use special tools to securely delete their files; the file system automatically securely deletes the data when files are unlinked, truncated, or sparsified.

Secure deletion approaches have been developed for a variety of block-based file systems; block-based file systems are the predominant file system design paradigm, where data is stored and retrieved by accessing fixed-sized indexed blocks on the storage medium. These approaches generally work by having a sanitization daemon running in the background that overwrites discarded blocks before putting them on the free-block list. Of course, this makes the assumption that the device driver actually performs these updates in-place. These approaches often support a *sensitive* file attribute, which allows the user to mark files as sensitive at any point in the files' lifetime; the file system then provides secure deletion only for data from sensitive files.

Media with Erase/Write Asymmetry We mentioned earlier that flash memory does not support in-place updates of data. Before writing, flash memory must be first "erased"—only then can it write new data. The catch is that the granularity of erasures is orders of magnitude larger than the granularity of reading and writing. By way of analogy, think of the storage media consisting of a stack of punched cards. Once a hole is punched in a position (i.e., writing a zero), it cannot be unpunched (writing a one). Instead, a new blank card must replace it: all other data colocated on that card must be repunched (copied) on the fresh card with the changed location unpunched; the old card can then be destroyed. Flash memory consists of lines of floating-gate transistors: the charge from each can be easily drained to write a zero, or not drained to write a one. However, the charge can only be reset with high voltages at a large granularity (e.g., 128 KiB); this operation even physically damages the medium, eventually wearing it out.

An asymmetry between the write and erase granularities is not limited to flash memory (and punch cards). For example, a tape archive consists of many magnetic tapes, each storing, say, half a terabyte of data. Each tape must be written end-to-end in one operation; data intended to be archived on tape is heuristically bundled and written together. Later, to securely delete a single backup on the tape, the entire tape is re-written to a new tape and the expired backup is removed or replaced. The old tape is then erased and reused for new data in the tape archive. This operation incurs cost: tapes have a limited erasure lifetime and tape-drive time is an expensive resource for highly-utilized archives. Another example is an array of magnetic hard drives whose only accepted secure deletion method is the controller's secure erase command. The resulting erasure

granularity is an entire hard drive, where colocated data must be first copied to another array constituent. It also manifests itself in physical media comprised of many write-once read-many units—units that are unerasable but replaceable—such as a library of optical discs. In this case, each erasure requires destroying one constituent of the archive, which can be expensive if done frequently.

The naive secure deletion approach for physical media with asymmetric write and erase granularities is to immediately compact the erase unit that contains the deleted data: copy the valid colocated data elsewhere and execute the erasure operation. There is no other immediate secure deletion approach based on erasures that can do better than one erasure per deletion; *something* must be actually erased from the storage media to achieve secure deletion. An obvious alternative is to intermittently perform this compaction-based secure deletion. This approach is no worse than the naive one in terms of execution time and physical wear, although the deletion latency increases.

Batching-based compaction can be made more efficient by using encryption as a compression technique. In the data-node-encrypted file system, each block of data is encrypted with its own unique key [9]. Encryption keys are stored tightly packed on a special area of the flash memory. To securely delete all the data on the file system, it suffices to perform batching-based compaction on the much smaller area storing the encryption keys, resulting in far fewer erasure operations.

Another technique is a mixed-media approach. The medium with a large erasure granularity is treated as persistent storage; data is stored on it encrypted with an appropriate granularity. The encryption keys are then managed using key wrapping and a medium that supports secure deletion. In Boneh and Lipton's approach [10], for example, a single master key is used to encrypt many data backup keys that are stored alongside the encrypted data. To securely delete data, a new master key is generated and new wrapped keys are provided for all the backups that are not deleted.

TRIM Commands A TRIM command is a command issued from the file system to a lower-layer device stating that a particular continuous range of data is no longer needed by the file system [11]. TRIM commands are not a secure deletion approach, but rather a widely-supported storage medium interface feature that we can leverage for secure deletion. Data (contained in some file) is logically removed from a file system in three ways: by overwriting the old data, by unlinking the file, and by truncating or sparsifying (hole-punching) that part of the file. Now, when we explicitly overwrite part of a file, the information that the old version can be deleted is implicitly passed to the lower-layer: even if writing is not done in-place, the fact that there is a "new" version of the data is known to the lower layer. For file unlinking, truncating, and sparsifying, however, no such indication is given. This is why so many secure deletion approaches resort to the tedious writing of zeros; even if this is not *sufficient*

to achieve secure deletion, it is at minimum *necessary* that this new-version information is known. Similarly, the SQLite database offers a secure deletion feature that overwrites deleted records with zeros: necessary, but not sufficient, to achieve secure deletion. TRIM commands offer the file system a more efficient way of passing this information to lower layers: when deleting a file, a TRIM command simply tells the lower-level the start address and the length of the trimmed range.

TRIM commands were actually invented as an efficiency measure for flash memory to prevent a thrashing effect that occurs once the device is full: unless the flash controller knows which blocks of the file system are no longer needed, it must assume that all blocks are necessary and therefore copies large amounts of unnecessary data around when trying to free space for new data. Despite TRIM commands original purpose, there is no reason that a device driver or hardware controller cannot use information from TRIM commands to perform secure deletion. TRIM commands are already widely supported and indicate every time a file system block is discarded—there are no false negatives. It is not possible, however, to restrict TRIM commands only to sensitive blocks without the loss of the TRIM commands' intended purpose. Therefore, the underlying mechanism that securely deletes the data should be efficient.

4 Other Properties of Approaches

In the last two sections, we saw that a key aspect of the different secure deletion approaches is the specific assumptions they make—assumptions on the adversary and the interface to the storage medium that is available to use to achieve secure deletion. When these assumptions are met, then the approach provides secure deletion along with other properties that we now present.

Deletion Granularity The granularity of an approach is the approach's deletion unit. Secure deletion can have a *per-physical-medium*, *per-file*, or *per-data-block* granularity. A per-physical-medium approach deletes *all* data on a physical medium. As such, we consider it an extraordinary measure—something we may do once before crossing a border but not after deleting each email. At the other extreme, we can securely delete data at the smallest granularity offered by the physical medium: the data block size (also called sector size or page size). Per-data-block approaches securely delete any deleted data from the file system.

Between these extremes lies per-file secure deletion, which targets files as the deletion unit: a file remains available until it is securely deleted. While per-file secure deletion approaches are widespread—and it is natural to reason about data deletion in the context of file deletion—we caution that the file is not the natural unit of deletion; it often provides similar utility as per-physical-medium deletion. Long-lived files such as databases frequently store user data; the An-

droid phone uses them to store text messages, emails, etc. A virtual machine may store an entire file system within a file: per-file secure deletion means that anything deleted from this virtual file system remains recoverable until the user deletes the entire virtual machine's storage medium. Consequently, in such settings, per-file secure deletion requires the deletion of all stored data in the DB or VM, which is an extraordinary sanitization proceedure similar to a per-physical-medium approach.

Scope Many secure deletion approaches use the notion of a sensitive file. Instead of securely deleting all deleted data from the file system in an untargeted way, they only securely delete known sensitive files, requiring the user to mark sensitive files as such. We divide the approach's scope into *untargeted* and *targeted*. A targeted approach only securely deletes sensitive files; it can be turned into an untargeted one by marking all files as sensitive.

While targeted approaches are more efficient than untargeted ones, there are limits to their usefulness. First, as with granularity, the file is not necessarily the correct unit to classify data's sensitivity; an email database is an example of a large file whose content has varying sensitivity. The benefits of targeting therefore depend on the deployment environment. Second, some approaches do not permit files to be marked as sensitive after their initial creation, such as approaches that must encrypt data objects *before* writing them onto a physical medium. Finally, targeted approaches introduce usability concerns and consequently false classifications due to user error. Users must change their habits to deliberately mark files as sensitive, or use different tools when deleting or working with the files. A false positive costs some efficiency but, much worse, a false negative may disclose confidential data.

A useful middle ground is to broadly partition the storage medium into a securely-deleting user-data partition and a normal operating system partition. Untargeted secure deletion is used on the user-data partition to ensure that there are no false negatives and this requires no change in user behaviour or applications. No secure deletion is used for the OS partition to gain efficiency for files trivially identified as insensitive. Of course, one size does not fit all. Some users may want to securely delete an application and ensure that there is no evidence that it was ever used on their system, including metadata.

Metadata Recall that we securely delete data with the intention of ensuring that it is unavailable to an adversary. Some users may want to securely delete data that the adversary already has, to prevent the adversary from knowing that the user has it too. In this case, the user should also delete file metadata: file names, sizes, and so forth. Some file systems store file checksums in metadata for integrity: an adversary could use this to confirm the exact copy of a file they suspect the user of storing. Many high-level secure deletion approaches do not specifically address file metadata, and user-level tools that do attempt to

overwrite metadata are unsuccessful on most file systems. This is because most file systems handle metadata in a log-structured manner, that is, by adding a new version that supersedes the previous.

Deletion Latency Many secure deletion approaches offer *immediate* assurances: use this tool to overwrite the file with zeros and the data is gone. As we saw for flash memory, however, the only way we can achieve any kind of efficiency is to batch deletions and perform periodic deletion. Hence, a *delayed* approach executes intermittently and provides a larger deletion latency. If it is run periodically, then it provides a fixed worst-case bound on the deletion latency. The actual deletion latency for data is then a useful metric by which to evaluate secure deletion approaches.

Example Comparison: Overwriting versus Filling We return to the two classes of user-level approaches and compare their assumptions and properties. Recall that both classes operate at the user-level. One works by overwriting a file with zeros (and assumes that writes are in-place), while the other works by filling the entire storage medium with a new, insensitive file (and assumes that all unused blocks are allocated in the process). Overwriting has a *file granularity*: files are used until they are securely deleted. Filling has a *per-block granularity*: any unused block of data is allocated to the filling file. Overwriting has a *targeted scope*: only the selected file is securely deleted. Filling has an *untargeted scope*: all deleted data on the file system is securely deleted. Neither delete *metadata*, however some file systems store metadata alongside data, in which case filling also deletes this. Finally, overwriting is an *immediate approach*: after running the program, the data is securely deleted. Filling can also securely delete data immediately, but since the cost of executing it is quite high, it is an extraordinary measure that is suitable only when the disclosure time is known. Since filling is non-destructive, however, it can be run *periodically* (e.g., overnight) thereby having a configurable upper bound on deletion latency.

5 Summary and Future Directions

Here we would like to summarize a few key points touched on in this article.

- Secure deletion is not only useful before selling or discarding a hard drive. Sensitive data can be compromised at unexpected times by adversaries capable of obtaining any secret keys required to access it. Sensitive data should be securely deleted in a timely fashion.
- Overwriting a file with zeros probably does not securely delete it. It may still be necessary to write zeros just to send a signal to lower-layers, even if

these stored zeros are never later read. File systems should better support hole-punching sparse files and pass this information immediately down as TRIM commands.

- Secure deletion approaches that work at the granularity of a file are insufficient. The file is not the only unit of deletion and often not the most natural one. User data is often stored in databases as a single file that remains on a system indefinitely.
- Secure deletion approaches that only target sensitive files must also address usability concerns. If a user cannot reliably mark their data as sensitive, then the approach provides little benefit. Approaches that securely delete all deleted data, while less efficient, suffer no false negatives.

There are numerous areas where further research in secure deletion is direly needed. New storage technologies invariably complicate secure deletion, often in completely new ways. Storage technology advances the state of the art in many ways: capacity, reliability, performance, and price. Secure deletion, however, is not a design requirement and creative approaches to achieve it are usually needed after new hardware is introduced.

In distributed cloud storage, secure deletion is particularly challenging because the abstracting interfaces accumulate. This setting requires reliability and availability in spite of frequent expected failures. Secure deletion should be possible, however, even when an entire computer or set of computers becomes unresponsive while maintaining the target level of reliability and availability before the data is deleted.

6 Acknowledgments

This work was partially supported by the Zurich Information Security Center. It represents the views of the authors. We would like to thank Ari Juels, Kari Kostiainen, Srdjan Marinovic, Alina Oprea, Christina Pöpper, Thomas Themel, and Nils Ole Tippenhauer for their many helpful comments.

References

[1] "Macy's parade: 'Shredded police papers in confetti'," BBC News. Retrieved from http://www.bbc.co.uk, November 25, 2012.

[2] M. Feldman, "UK Orders Google to Delete Last of Street View Wi-Fi Data," IEEE Spectrum. Retrieved from http://www.ieee.com, June 24, 2013.

[3] "EU proposes 'right to be forgotten' by internet firms," BBC News. Retrieved from http://www.bbc.co.uk, January 23, 2012.

[4] R. Perlman, "The Ephemerizer: Making Data Disappear," Sun Microsystems, Tech. Rep., 2005.

[5] A. Greenberg, *This Machine Kills Secrets*. Penguin Books, 2012.

[6] C. Bowlby, "Stasi files: The world's biggest jigsaw puzzle," BBC News. Retrieved from http://www.bbc.co.uk, September 13, 2012.

[7] P. Gutmann, "Secure Deletion of Data from Magnetic and Solid-State Memory," in *USENIX Security Symposium*, 1996, pp. 77–89.

[8] R. Kissel, M. Scholl, S. Skolochenko, and X. Li, "Guidelines for Media Sanitization," September 2006, National Institute of Standards and Technology.

[9] J. Reardon, S. Capkun, and D. Basin, "Data Node Encrypted File System: Efficient Secure Deletion for Flash Memory," in *USENIX Security Symposium*, 2012, pp. 333–348.

[10] D. Boneh and R. J. Lipton, "A Revocable Backup System," in *USENIX Security Symposium*, 1996, pp. 91–96.

[11] Intel Corporation, "Intel Solid-State Drive Optimizer," 2009. [Online]. Available: Retrieved from www.intel.com

Joel Reardon is a doctoral student at the ETH Zurich. He received his Bachelor's and Master's from the University of Waterloo in 2006 and 2008 respectively. His research interests are focused on privacy in the information age: how new information media will affect our lives and our identity in the new millennium.

David Basin is a full professor and has the chair for Information Security at the Department of Computer Science, ETH Zurich since 2003. From 2003–2011 he was founding director of the ZISC, the Zurich Information Security Center. He received his Ph.D. from Cornell University in 1989, and his Habilitation from the University of Saarbrücken in 1996. His research focuses on information security, in particular methods and tools for modeling, building, and validating secure and reliable systems.

Srdjan Capkun is an Associate Professor in the Department of Computer Science, ETH Zurich and Director of the Zurich Information Security and Privacy Center (ZISC). He was born in Split, Croatia. He received his Dipl.Ing. Degree in Electrical Engineering / Computer Science from the University of Split, Croatia (1998), and his Ph.D. degree in Communication Systems from EPFL (Swiss Federal Institute of Technology - Lausanne) (2004). Prior to joining ETH Zurich in 2006 he was a postdoctoral researcher in the Networked & Embedded Systems Laboratory (NESL), University of California Los Angeles and an Assistant Professor in the Informatics and Mathematical Modeling Department (IMM), Technical University of Denmark (DTU).