Remote Wiping and Secure Deletion on Mobile Devices: A Review

Ming Di Leom,[1] B.Sc.; Kim-Kwang Raymond Choo,[2,1] Ph.D.; and Ray Hunt,[3] Ph.D.

[1]University of South Australia, GPO Box 2471, Adelaide, SA 5001, Australia.

[2]Department of Information Systems and Cyber Security, University of Texas at San Antonio, San Antonio, TX 78249-0631, USA.

[3]Department of Computer Science and Software Engineering, University of Canterbury, Christchurch, New Zealand.

Abstract

Mobile devices have become ubiquitous in almost every sector of both private and commercial endeavour. As a result of such wide-spread use in everyday life, many users knowingly and unknowingly save significant amounts of personal and/or commercial data on these mobile devices. Thus loss of mobile devices through accident or theft can expose users – and their businesses – to significant personal and corporate cost. To mitigate this data leakage issue, remote wiping features have been introduced to modern mobile devices. Given the destructive nature of such a feature, however, it may be subject to criminal exploitation (e.g. a criminal exploiting one or more vulnerabilities to issue a remote wiping command to the victim's device). To obtain a better understanding of remote wiping, we survey the literature, focusing on existing approaches to secure flash storage deletion and provide a critical analysis and comparison of a variety of published research in this area. In support of our analysis, we further provide prototype experimental results for three Android devices; thus, providing both a theoretical and applied focus to this paper as well as providing directions for further research.

KEYWORDS: forensic science, remote wiping, secure deletion, flash storage, NAND non-volatile flash memory, SSD (Solid State Drive/Disk)

Mobile devices have become increasingly ubiquitous and no longer used only for making and receiving phone calls. Examples of such usage include receiving and sending email and instant messages, making VoIP calls, taking and uploading of photos and video clips, and finding one's way around using mapping apps; all of which results in an increasing amount of data (and metadata such as geolocation) stored and transmitted from such devices. Due to the size of these devices, they can be easily lost or stolen. For example, the number of mobile devices reported lost or stolen every year include an estimated 150,000 in Australia in 2011 (1), and 30,000 in London alone in 2013 (2).

With the advent of cloud storage services (e.g. Dropbox, Apple's iCloud, and Google Drive), mobile device users are able to synchronise the data stored on their devices to their cloud storage accounts. There is, however, potential for information leakage should the account be compromised (e.g. the compromising of several celebrities' online accounts (3, 4, 5) or when the device is lost, stolen or compromised (e.g. via malware). Physical theft and loss of devices are among the most common cause of data leakage in organisations according to the 2014 study by (6). The cost of the hardware and software due to lost or stolen devices is generally less than the direct and indirect cost resulting from the leakage of the information. In a study by (7), 50 mobile devices were intentionally "lost" and then monitored for any access attempt, resulting in 96% of these devices being reportedly accessed by the finders of these devices (perhaps due to the inherent curiosity of human nature). The study also highlighted the difficulty for device owners to regain possession of the devices as only 50% of the "lost" devices in this study were actually recovered even though the owner's contact information was clearly shown on the device.

To mitigate the issue of data leakage due to lost or stolen devices, the remote wiping feature has been introduced to modern mobile devices such as iOS, Android, Windows Phone and BlackBerry devices. This feature allows the device owner to send a remote command to wipe the contents on the lost or stolen device. Such a command has been referred to as "kill pill" (8, p. 8) or "poison pill" (9, p. 57; 10, p. 3) in the literature.

The earliest implementations of this remote wiping feature were on Blackberry devices (11, 12) and the now defunct Microsoft Windows Mobile (succeeded by Windows Phone) (13, 14, 15) in 2005. It is not surprising as BlackBerry devices are known for their security features and one of the first devices to be approved by government use (16, 17, 18). Remote wiping was introduced to Apple's iPhones in 2009 through a service known as "Find My iPhone"

(19). “Find My iPhone” service was initially only available to the now defunct MobileMe (replaced by iCloud and discontinued from June 2012 (20) subscription. It was not until the release of iOS 4.2 released in November 2010 (21) that “Find My iPhone” service became a free service (22). In August 2013, Google introduced the remote wiping feature by way of the Android Device Manager (ADM) to devices running Android 2.2 or above (23). This has become an official feature of Android devices, which was previously only available to Google Apps customers (24) or via a third-party app (e.g. Cerberus and SeekDroid).

Remote wipe functionality can be a very useful feature in helping to prevent information leakage when a mobile device is no longer in the owner’s possession due to theft, robbery, or simply misplaced. Given the destructive nature of this function, however, an attacker might misuse this function by sending such commands to the normal users to cause serious disruption. Thus, it is very important to ensure that the wipe command can only be triggered by the owner or authorised person.

The purpose of this survey and the prototype experiment is to provide an in-depth understanding of the current state of play. Once a mobile device has been remotely wiped, the deleted data should be irrecoverable. Since the majority of the mobile devices found today use flash storage, also known as NAND flash memory (25), we discuss how secure deletion is addressed on this particular type of storage.

Remote wiping is one of several anti-theft methods for mobile device. Other anti-theft methods include remote tracking, data loss prevention (DLP) systems deployed in enterprise environments, and tools that activate self-destruction upon predetermined conditions. In this survey, we limit our discussion to systems that wipes itself only when the user triggers it.

One common approach to mitigating data leakage is through storage encryption (i.e. encrypting of data-at-rest on the device). In mobile device, storage encryption has been available to BlackBerry version 4.0 or above (26), Apple iOS since introduction of iPhone 3GS (27, 28), Windows Mobile 6.5 (29) and reintroduced back in Windows Phone 8 (30, 31). Storage encryption has also been available to Android device users version 3.0 (Honeycomb) (32). Google initially announced that all devices shipped with version 5.0 (Lollipop) would have storage encryption enabled by default (33). Due to performance issue (34), it is not enabled on every new Lollipop device (35), despite the optimisations implemented later (36, 37, 38). Despite the initial announcement, storage encryption was never made mandatory for existing device upgrading to Lollipop. In our prototype experiments to be described in “Case

Study on Android Approach" section, we will, therefore, not use encryption on our Lollipop device – the Motorola Moto G, as it is not enabled by default.

Prior to version 4.4 (KitKat), the encryption key was stored in flash memory and encrypted with a weak key derivation function (KDF), PBKDF2 with 2,000 iterations only. This made extraction and brute forcing the encrypted key fairly trivial. KitKat replaced the KDF with scrypt – a password-based key derivation function, which was specifically designed to make it costly to perform large-scale custom hardware attacks by requiring large amounts of memory and thus making brute forcing more expensive (39). On Lollipop, the key is encrypted with a master key that is not stored in flash memory and cannot be extracted even with root access (40). This has made unauthorised access to encrypted flash memory much more difficult.

However, no matter how secure the key is, existing law can compel a person to surrender the key (e.g. 41, 42). There is also a misconception that secure deletion is unnecessary when storage encryption is available since all the data is secured. Secure deletion is a condition where an "adversary is given access to the system but not able to recover the deleted data from the system" (43, p. 301). However, cryptography is only effective due to computational cost (44) given the best cryptanalysis tools currently available. Given advances in cryptanalysis and quantum cryptography, there is no guarantee that the data resided even in encrypted form, could not be deciphered by an adversary in the near future. Any future vulnerability(ies) in the device hardware or software may be exploited by an adversary to gain unauthorised access to the decryption key. Thus, secure deletion methods that can ensure deleted data removed from storage might still be relevant is also discussed in this paper.

In this paper, we located 15 patents and 18 academic publications published in English between November 1999 and June 2014. On the remote wiping topic, materials were located using Google Scholar (including Google Patents) and academic databases such as ScienceDirect, ACM Digital Library, IEEE, and Springer using search terms such as remote (wipe OR wiping), (sanitize OR sanitization) mobile phone, secure (erase OR delete) (flash OR NAND).

Past surveys on secure deletion (43, 45) investigated two common types of non-volatile memory, namely; magnetic hard drive and flash memory. Our discussion in "**Secure Flash Storage Deletion**" section, however, focuses on flash memory only. Similar to (43), our discussion on secure deletion does not include information deletion – which encompasses

searching and removing all traces of some information. There are two approaches to secure deletion, namely; data overwriting or encryption, and physical destruction (e.g. disk crusher, degaussing, incineration) (46). The main difference is that the second approach will render the drive unusable; thus, irrecoverable (47) while the first approach does not. Our discussion is, therefore, limited to the first approach and neither does it focus on any specific data storage mechanism such as database or cloud storage.

This work is organised as follows. "Reviews of Remote Wiping and Secure Flash Storage Deletion" section reviews existing approaches to remote wiping, and in particular, secure deletion techniques. This section also includes a prototype experiment on remote wiping using three Android devices. "Discussion" section discusses the limitation of existing approaches and finally, "Concluding Remarks" section concludes the paper and outlines potential future work.

Reviews of Remote Wiping and Secure Flash Storage Deletion

General Review of Remote Wiping

In this section, we discuss relevant publications on remote wiping which include eight patents and five academic publications.

The remote wiping process (Fig. 1) can generally be described as follow:

1. A user enrols with the remote wiping system maintained by an organisation. When the mobile device is lost, the user reports this to the same organisation and requests for the mobile device to be wiped.
2. The organisation sends the wiping command to the intended mobile device. Depending on the transmission channel, the command may be sent through a wireless access point (AP) using an Internet connection, or via a traditional 3G/4G mobile communication channel.
3. Upon receiving the wipe command, the mobile device erases the data.

Table 1 provides a summary of the security properties for the relevant references and these are further illustrated in Fig. 1.

Authenticate Reporter

A system should verify the identity of the reporter and the device's owner when a mobile device is reported lost or stolen. This can be done through information known only to the user (e.g. account number, address, and last bill number). Existing authentication methods considered by the literature are summarised in Table 2.

Brown et al. (49) considered only the authorisation level to determine the data type that the reporter or any person who initiate the remote wipe, is permitted to wipe. The proposed scheme did not indicate any means of authenticating the identity of the reporter.

Several high-profile hacks (3, 4, 61, 62) have demonstrated that using a secret question or personal identifiable information alone is not sufficient to authenticate a person. The victim's account may be compromised because an adversary is able to answer a secret question by supplying publicly available information deduced from the Internet. Those incidents could have been prevented with two-factor authentication.

Nearly half of the proposals specified more than one mechanism for authenticating the reporter but they did not consider multi-factor authentication. Of these proposals, only (56) proposed inclusion of two-factor authentication even though it was not explicitly indicated in the original proposal. In the proposal of (56), the authentication process involves two steps; firstly the reporter submits personal identifiable information to identify themselves to the service provider, then they provide a PIN code to be verified by the mobile device. Thus, it can be considered as two-factor authentication, even though the authentication is performed by two entities; service provider and mobile device.

Authenticate Origin of Wipe Command

When receiving the wipe command from any channel (e.g. 3G/4G channel or web server via the Internet), the mobile device should check whether the source is authorised to send the command or instruction.

Existing literature generally prefers the use of public key cryptography. Pretty Good Privacy (PGP) is one such popular encryption system - a sender first creates a digital signature by hashing the message and encrypts the hash with its private key to *sign* the message. The sender then encrypts the signed message with the recipient's public key. Only the recipient

can decrypt the message with its private key and verify the message using the sender's public key.

Onyon (54) suggested storing the sender's public key when enrolling into the remote wiping system. Brown et al. (49) and Gajdos & Kretz (55) assumed the sender's public key has been stored on the mobile device during manufacturing. Angelo et al. (48) suggested a signature-based approach whereby the sender encrypts the message with the recipient's private key. The signed message can then be decrypted by the recipient using its own public key.

Authentication can also be established using a shared code between the sender and the recipient. In SSL/TLS, for example, a symmetric key is exchanged using public key cryptography. However, Brown et al. (49), Onyon et al. (54) and Gajdos & Kretz (55) did not consider the need to protect the shared code. Without any protection, the shared code could be exposed allowing an adversary to spoof as a valid sender or recipient.

Park et al. (57) and Yu et al. (56) proposed a mechanism to request a password from the reporter, which will be sent with the wipe command. The mobile device authenticates the password to determine the authenticity of the wipe command. But this mechanism could only authenticate the reporter, since if a correct password is provided, then *any* server can send the command, and the mobile device will simply accept it. Thus, this mechanism is considered to be more suitable to authenticate the reporter. Additionally, it is important to consider the response mechanism when dealing with an incorrect password. If there is no limit to the number of failed attempts, this will allow an online brute-force dictionary attack. Therefore, it is recommended that the system lock the account should the number of failed attempted exceed a predetermined limit.

Kuppusamy et al. (58) proposed a system that checks the incoming telephone number of the received SMS against trusted numbers. This is a failback mechanism when the message is not secured through the authentication method proposed by Park et al. (57). Instead of using this as a failback mechanism, the system should always check the incoming number as origin authentication, with the trusted number set beforehand. This mechanism can reduce the possibility of a replay attack (see "Replay Attack Mitigation" section) unless the number is spoofed. Guo et al. (63) assuming the telephone number is checked separately and not embedded inside the encrypted or encoded message.

Secure Wipe Command

The “wipe” command sent as a message or packet should be secured against sniffing to avoid tampering. Surprisingly, only Brown et al. (49) considered encrypting the wipe command despite the potential for the wipe command to be hijacked and modified.

Secure Delete

The wiping process should result in the wiped data being irrecoverable, and secure deletion is discussed further in “**Secure Flash Storage Deletion**” section.

Brown et al. (49) and Onyon et al. (54) proposed overwriting the wiped data with zeroes, ones, or random combination of them. Sennett and Daly (53) proposed permanent physical self-destruction. Gajdos and Kretz (55) proposed self-destruction by overwriting the firmware. Yu et al. (56) only mentioned the future possibility of incorporating a secure deletion solution. Although the patent by Kenney (51) does not provide secure deletion, it proposed a method to render data stored on the device inaccessible. The latter is similar to the approach undertaken by Apple where data is ‘wiped’ by rendering all files cryptographically inaccessible when the file system key used to encrypt the files is deleted (64).

The secure deletion method described in the patent by Brown et al. (49), and owned by BlackBerry, Inc (formerly known as Research In Motion, RIM), is deployed on BlackBerry devices. The pattern or process differs between the OS versions, type of storage, and type of data (65, 66). The difference between “flash storage” and “user files” is that user files refer to a portion of storage that allows user file-level access. Typical, “user files” is used to store media files such as pictures and videos. “Flash storage” in this case includes the rest of user’s data such as contacts, SMS, e-mail, calendar entries, etc (66, p. 45), but does not include the operating system (OS). This level of wiping is also commonly known as a factory reset.

Ensure Wiping Operation is Completed

This feature if implemented ensures that the wiping process is completed successfully, even when it is interrupted by switching off the mobile device as the process will resume once the mobile device is switched back on. When a device is switched off, it would be challenging

for an adversary to gain access to the data stored on the device even though the wiping process might be interrupted. Only Brown et al. (49) and Yu et al. (56) considered this aspect in their approaches.

Acknowledge Source that Wipe is Completed

The lost mobile device informs the owner or the system operator that the wiping operation has been completed successfully.

Kenney (51) suggested using a handshake, ACK or NACK (Negative ACK, also known as NAK), or ping. Sennett and Daly (2013) suggested broadcasting a signal to indicate that the mobile device has received a valid command and will proceed to execute the command. The latter, however, does not specifically check that the mobile device has successfully completed the task. Adusumalli (60) and Joe and Lee (59) suggested using SMS and "service complete code" to confirm successful execution of command respectively.

Replay Attack Mitigation

If an adversary manages to capture the command, the adversary can send the same command to another device to wipe it. In other words, the adversary can conduct a "Replay attack" by replaying a previous request made by a reporter to wipe the new (replacement) device.

Angelo et al. (48) proposed using timestamp, non-repeating sequence number, and a randomly generated number to mitigate replay attack. Similarly, Park et al. (57) also suggest using a concatenation of the wipe command and the timestamp using Base64 encoding to mitigate replay attack (referred to as reply attack). This measure is, however, ineffective as an adversary is able to decode the message, modify the timestamp so that the timestamp fulfils the requirement, and re-encode the modified concatenated message. It is recommended that the suggested approaches of Angelo et al. (48) and Park et al. (57) be deployed using encryption to avoid modification of the wipe command.

Vendor Implementation

Android

The remote wiping feature was officially introduced into Android via ADM (Android Device Manager) (23), which was previously only available via a third-party app. ADM is remotely controlled via Google Cloud Messaging (GCM). GCM is a push messaging service used in the Android platform that enables developers of mobile applications (apps) to deliver data to their apps running on their customers' mobile devices. With push messaging, a developer can push notifications, messages and even commands to the apps, without continuous polling from the apps which consumes resources. A developer operates a server (referred to as *app server*) to send data to the app installed in a particular user's device. In push messaging, the app server does not directly initiate the connection to the mobile device. Instead, the data is relayed through the provider of the push messaging service (referred to as *connection server*). In addition, any message received from the connection server is processed by the service client (akin to OS service) before passing to the app (Fig. 2). The connection server usually restricts the data size, and in this case, the mobile device can be instructed to download the data from an app server. Before an app can receive data through the GCM service, it needs to register itself to the service with a registration ID and also the app server's sender ID. This allows the cloud server to associate an app server with an app installed on a particular mobile device. Thus, only an app server with an authorised sender ID can push a message to the app with related registration ID (67).

ADM utilises Android Device Administration API to execute wiping operation (68, 69). This API is also available to a third-party app provider for mobile device management (MDM) features at the system level. An IT administrator can write application that a user installs on the mobile device to enrol into the MDM system of the company. The API only provides a factory reset functionality, and a developer could choose to use GCM or any other method to trigger the wipe remotely.

BlackBerry

Remote wiping can be triggered via BlackBerry Enterprise Server (BES) for corporate environments or BlackBerry Protect for personal customers. The device is preloaded with a

root certificate during manufacture which authenticates using the BES (71). BES also has the ability to track wiping status of the device (72), a property discussed earlier ("Acknowledge Source that Wipe is Completed" section).

Devices enrolled in BlackBerry Protect cannot be associated with BES. In BlackBerry Protect, the user has the ability to instruct the lost device to perform back-up (to BlackBerry Protect service) before full wiping when running BlackBerry OS version 7.1 or earlier (73). Later version does not have this feature.

iOS

Remote wiping can be triggered via iCloud (for personal consumers) or third-party mobile device management (MDM) system (for corporate environments). Third-party MDM utilises either Apple's MDM API or Exchange ActiveSync (EAS) (64). In the third-party MDM system, an employee's mobile device is managed from a MDM server installed with software provided by the MDM provider. The server creates a *configuration profile* (75), which is uploaded to Apple's server. A user then downloads and installs the *configuration profile* to enrol into the system. This configuration profile allows the system administrator to have control over the employee's iOS device including the ability to remotely wipe. Subsequently, whenever the MDM server wants to communicate with an iOS device, it does so via Apple's Push Notification Service (APNS), a push messaging protocol (70) secured with SSL/TLS (74), instructing the device to check in. The iOS device then initiates SSL/TLS connection with the server to check in. The server uses the connection to perform administrative tasks including remote wiping. According to official Apple documentation (64), when performing remote wipe, the mobile device will reply with an acknowledgement upon receiving the wipe command. The device only checks in when using EAS; thus, it appears that the wipe command via a single "push" message is sent by the MDM server through APNS without the device checking in.

Windows Phone

Remote wipe can be initiated via the Exchange Management Console (EMC), Microsoft Outlook Web Access (OWA), or a third-party MDM system depending upon how the mobile

device was enrolled initially. The wipe command is sent via the ActiveSync protocol. ActiveSync is a push messaging protocol used for exchanging messages in the Microsoft Exchange environment (76).

The mobile device can either be partially wiped or fully wiped. Partial wipe applies whenever the device "un-enrols" or "retires" from the corporate MDM system. All the corporate information, email accounts, VPN connections, Wi-Fi connections, policy settings, apps, and data that the apps deployed are removed, except for personal apps or data on the device that the user installed. Full wipe removes all the apps and information on a device and returns the device to factory settings. (77)

Vulnerabilities

Implementations of the ActiveSync protocol in Android and iOS were discovered to be flawed (78). The implementations failed to warn the user when presented with untrusted SSL certificates and in some cases they will accept any certificate presented. This vulnerability can be exploited to spoof an Exchange server to initiate unauthorised policy enforcement such as performing remote factory reset on a mobile device. Similar flaw was also discovered in some Android apps due to the incorrect use of the Android API when implementing SSL. Consequently, these apps are insecure against man-in-the-middle (MitM) attacks (79, 80). SSL certificate validation was also found to be broken in Amazon's EC2 Java library (81) and Apple's SecureTransport library (82). These flaws highlighted the potential risk posed by SSL implementation in non-browser applications, which have not evolved to the degree that web browsers have (81). Such risk could extend to other remote wiping applications that utilises SSL.

There was a flaw previously found in Google Cloud Messaging (GCM) that allowed an adversary to control the ADM installed on the victim's device (67). As mentioned in ("Android" subsection of "*Vendor Implementation*" section), before an app can receive a message through GCM, it needs to register itself to the service with a registration ID and also app server's sender ID. This allows a connection server to identify to which mobile device and app to push the message. With a malicious app installed on the victim's device which acts as a man-in-the-middle (MitM), an adversary can intercept the registration request and steal the registration ID, in this case registration ID of ADM. The adversary can then proceed

to control the ADM with the stolen registration ID. In addition, the connection server is supposed to only allow the authorised app server to push a message by checking the sender's ID. However, this policy is not enforced and can allow an adversary to push a message from their "unauthorised" app server (83).

Apple's iCloud allegedly allowed unlimited password attempts, and thus is vulnerable to brute-force attacks (84). This vulnerability was apparently responsible for compromising several celebrities' iCloud accounts that ultimately lead to their photo leaks (85). Proof-of-concept code to launch a brute-force attack on an iCloud service was published on August 30, 2014 at (https://github.com/hackappcom/ibrute). The vulnerability was apparently fixed two days later (85). Apple denied that the incident was due to the vulnerability of iCloud services, and instead claimed that it was a result of "very targeted attack on user names, passwords and security questions," (4). Although the photo leaks incident was about exposure of private data, it could have been abused to erase victim's data stored on their mobile devices.

Another recent high profile incident involves the unauthorised reactivation of iOS devices which were locked using the "Activation Lock" feature (86). This feature was introduced to deter theft by rendering stolen devices unusable. Unauthorised reactivation is performed by redirecting the iCloud connection to a spoofed iCloud server. The server can then send an unlock command to the locked device. Normally, reactivation requires sign-in to the iCloud account associated with the mobile device. The existence of a spoofed iCloud server suggests that the "Activation Lock" command might have been successfully reverse-engineered. Such a feat could theoretically also extend to replicating the remote wipe command.

The cross-site request forgery (CSRF) (87) vulnerability was discovered in Samsung's "Find My Mobile". This vulnerability is identified as CVE-2014-8346 (88). The vulnerability allows unauthorised remote locking of a mobile device by sending a specially crafted link with an embedded remote lock command to the victim. Assuming that the victim is logged on to "Find My Mobile" service, when a victim clicks on the malicious link, the web browser would send a remote lock request, in which the service proceeds to lock victim's mobile device. The attacker can customise the link to lock and change the unlock PIN, resulting in the victim not able to unlock their mobile device.

The implications of unauthorised remote wipe is potentially damaging with increasing reliance on digital devices in modern society. For example, a journalist described that his "entire digital life was destroyed" when his online accounts were compromised (61). He

reportedly lost “more than a year’s worth of photos, emails, documents, and more.” after an attacker remotely wiped all his devices, and could not “send or receive text messages or phone calls” after his Google Voice account was deleted (89). His accounts were compromised due to a weakness in the password reset policy adopted by a customer service representative. Such incidents highlighted the heterogeneity of threats in online services, and the fact that attack vectors are not restricted to web application flaws (90) or technical vulnerabilities.

Summary

This concludes our survey of existing proposals on remote wiping. Existing proposals generally do not consider securing the wipe command nor provide any mechanism to automatically resume interrupted wiping process. Secure deletion is seldom used on mobile devices. “Factory reset” serves as a simple method to remove all user data from mobile device. However, studies have shown that factory reset does not sufficiently remove personal data from mobile devices (91, 92, 93, 94, 95, 96, 97). Factory reset typically just logically deletes data, leaving data residue that could be forensically recovered. Secure deletion seems to be a clear solution to this issue, but it is actually not that straightforward due to the use of flash storage in mobile device. Various challenges of secure flash storage deletion and how existing proposals attempt to overcome them will be examined in the next section.

Secure Flash Storage Deletion

Secure deletion is sometimes referred to as *forgotten*, *erased*, *deleted*, *completely removed*, *reliably removed*, *purged*, *self-destructed*, *sanitized*, *revoked*, *assuredly deleted*, and *destroyed* in literature (43, p. 301). Before we discuss existing secure flash storage deletion approaches, we present an overview of flash storage.

Flash Storage: An Overview

Flash Storage Layers and Structures

The process of storing new or deleting existing data by an app usually takes place over different layers of the OSI model as outlined in Fig. 3. For example, an app usually modifies data by *calling* the Application Programming Interface (API) function provided by the OS. The data modification is then processed by the file system, and in the Android environment, there are actually different ways for the file system to handle the data.

Before Android version 3.0 (Honeycomb), file system accesses the flash storage through a memory technology device (MTD) (99). There is a built-in software flash translation layer (FTL) responsible for remapping logical block address to physical location (100). FTL emulates a normal block device such as a magnetic hard drive to enable the use of a common block file system (e.g. ext4 and FAT32) (101, 102). Unsorted block image (UBI) is an alternative interface to access flash memory, functioning as a layer on top of MTD. UBI implements a FTL separate from the FTL in the OS (103). In Android version 3.0, the MTD interface is replaced by an embedded multi-media card (eMMC) and secure digital (SD) device. Both eMMC and SD have integrated FTL implemented in the hardware controller, and therefore, software FTL is no longer necessary (102).

Like a magnetic hard drive, the flash storage is connected via the host interface connection (e.g. SATA, PCI-Express, SCSI, Fibre Channel and USB). All data input/output (I/O) is processed by the processor regardless of the type of FTL. The processor translates binary data to electrical voltage to store data in the flash package.

The structure of the flash package is as follows; each *package* is made up of multiple dies, each *die* consists of multiple planes, each *plane* comprises multiple blocks, and finally, each *block* is made up of multiple pages (101). A *Page* is the smallest unit of I/O operation, and the flash storage is accessed through the page unit (104). Each page has a spare area, known as out-of-band (OOB), which is used to store the error correcting code (ECC) (105).

Data Overwriting in Flash Storage

Secure deletion usually involves overwriting the original data to make it unrecoverable (106, 107, 108, 109), and data overwriting can be performed through software running on the OS (110) or firmware-based such as ATA's Secure Erase (109).

Government standards such as in the US (111) and Australia (112) recommend using ATA's Secure Erase command or overwrite the media at least once in its entirety. Researchers such

as Garfinkel & Shelat (108), Joukov et al. (113) agreed that overwriting once is probably adequately secure, although this view is not necessarily shared by others (114). Gutmann (106), for example, suggested a 35-pass overwriting pattern and subsequently he clarified that a few passes (rather than 35 passes) should be adequate in most situations (115). Garfinkel and Shelat (108) also explained that Gutmann's (106) demonstration that it is possible to recover data using a one-pass wipe is due to older hard drives having gaps between "tracks" and such gaps are not found in modern high-density magnetic hard drives.

However, flash storage could not rely on simple data overwriting for secure deletion. FTL not just remaps logical block address, but also distributes write access across flash storage (116). In flash storage, newer data is written to another valid *block* while the original block is simply marked as "invalid" (117). This is known as an *out-of-place* update; whilst an *in-place* update writes new data on top of the previous. Therefore, in an *out-of-place* update, the original content is preserved even with an overwrite request. Wei et al. (107) demonstrated that existing single-file secure deletion tools are ineffective in securely deleting file content in flash storage. To truly overwrite the data, flash storage must erase the block prior to overwriting with new data. However, an erase operation is significantly slower than a write operation (118) and the number of erase operations allowed on a single block is limited, ranging from 10,000 to 100,000 before it becomes a *bad block* - a block that cannot store data anymore (103, 119).

Classification of Secure Flash Storage Deletion

In this survey, we adapt the categorisation approach proposed by Diesburg's (45) and Reardon et al. (43). We broadly categorised secure deletion approaches into the user-space layer, the file system layer and the physical layer (Fig. 3). An alternative approach is to organise them into complete overwrite, random overwrite, delete sensitive, and block deletion (120). However, the latter approach is more appropriate in describing features provided by a secure deletion method. Thus, it is not suitable to classify each secure deletion method distinctly because each method described in this work most likely incorporates more than one feature.

User-space Layer

Spreitzenbarth and Holz (116) developed a secure deletion tool for the Symbian OS, which overwrites personal data (e.g. contacts, calendar entries, and SMS messages) using the OS API. Since the tool utilised an OS API, it can be ported to other platforms. Wear levelling techniques to prolong the service life of the storage media (e.g. flash memory) that have a limited number of erase cycles before being rendered unreliable was not considered in this work, and therefore, limiting its widespread adoption.

Reardon et al. (121) developed a secure deletion tool for the Android OS that will monitor the amount of free space and fill it with random data. This ensures unwanted data marked as invalid is filled with random data to achieve random deletion.

Albano et al. (122) proposed using standard Linux commands (e.g. cp, rm and dd) to delete data of interest in Android devices, without using any cryptographic primitives or kernel modules that will raise suspicion during a forensics analysis. The deletion process is summarised as follows:

1. Copy */data* partition to an external SD card.
2. Zero the partition while deleting the data of interest on external SD.
3. Move the remaining data on external SD back to the */data* partition.
4. Zero the external SD.

The proposed method requires BusyBox (www.busybox.net) to be installed (which provides the standard Linux commands). Installing BusyBox requires the user to have root privilege, but such an approach will void the warranty and result in the device being vulnerable to other malicious threats (123).

Kang et al. (124) proposed another method of data wiping for mobile phones, which overwrites only part of the data that will render it unidentifiable instead of overwriting the entire data. Their approach is designed only for data of the following file formats, namely; JPEG, BMP, FLV, DOC and XLS.

Steele et al. (125) proposed a system to wipe several USB flash drives simultaneously. The system checks the pre-set status of the drive upon insertion to determine whether to wipe the drive. Data overwriting defaults to zero-overwriting but it is able to accommodate user-defined patterns. The proposed system did not take into consideration wear-levelling or FTL

since the motivation behind the publication is simply because “there is literature on the Internet that suggests that such recovery is possible for the dedicated hacker up to at least 10 layers of previously written data in some versions of solid state memory”. However, we were unable to locate such literature.

The secure deletion approach proposed by Jevans et al. (126) uses either overwriting with zeroes and ones, or cryptographic secure bit patterns. The proposed approach is designed with a mechanism to resume interrupted wiping after the device is subsequently powered on, and to ensure wear levelling. The approach described in this patent was implemented in IronKey’s product (www.ironkey.com/en-US).

File System Layer

Weng and Wu (127) proposed using data encryption for secure flash storage deletion. Each data block is encrypted with a key and whenever the data needs to be deleted, the key will be removed. This work did not mention any mechanism to securely erase the key. The proposed method of Lee et al. (128) has a similar concept but their method ensures that keys are stored in the same block using an “unbalanced binary hash tree” algorithm. Thus, a file can be securely deleted by erasing the file header block which deletes the key. This work has been patented (129). Lee et al. (130) extended their previous work (128) to include US government standards on data sanitisation. It will overwrite the data before erase operation whereas previous work only used the erase operation. This is without additional operations required in their previous work; and thus, the claim that the newer scheme is more secure and efficient than the previous scheme. The proposal by Guyot et al. (131) is also based on data encryption. The proposed method not only deletes the keys but includes garbage collection to remove duplicate keys due to wear-levelling effect.

Reardon et al. (103) criticised the proposal by Lee et al. (128) saying that it is only conceptual and that if implemented it would cause too much wear on flash memory. They then proposed a scheme that is similar to Lee et al. (128) where each data block is encrypted with a key and the key is purged through an *erase* operation when the data is no longer needed. The proposed scheme encrypts each block of data with a distinct 128-bit AES key in counter mode. An IV (initialisation vector) is not used due to the use of a distinct key. The scheme is implemented in UBIFS (Unsorted Block Image File System), a log-structured file

system that builds upon UBI and has been tested on Android. The authors conducted various tests including wear analysis, power consumption, and I/O performance. However, Skillen and Mannan (102) argued that the proposed method by Reardon et al. (103) could only work with memory technology device (MTD) due to dependency on the UBI.

Sun et al. (132) proposed a hybrid secure deletion scheme that utilises two techniques; block cleaning and zero overwriting. Block cleaning is basically an "*erase*" operation ("Data Overwriting in Flash Storage" section). The proposed scheme calculates the cost of each technique before performing the secure delete and choosing the technique with a lower cost. There are different scenarios which can affect the cost of each technique. In block cleaning, any valid pages (pages that are storing valid data) residing inside blocks have to be copied somewhere else before erasing the block. Thus, the cost of block cleaning increases as the number of valid pages in a block increases. In contrast, zero overwriting can selectively overwrite affected pages (pages that store the data which the user wants to delete) only. Thus, block cleaning tends to be slower whenever large numbers of valid pages are involved since the operation has to copy them first. Lee et at. (130) argued that block erasure does not involve overwriting and therefore does not meet US government standard. Subha (133) further pointed out that the calculation introduces latency.

Choi et al. (118) proposed a scheme involving password-based per-file encryption and secure data deletion. The proposed encryption scheme encrypts file name and file data separately. When the user wants to delete a particular file, the process is as follow:

1. Write zeroes to all spaces occupied by file name, file address, and file data.
2. Read the spaces and verify that it is "0x00".
3. Execute the TRIM function that allows an operating system to inform the flash storage that the spaces are no longer used and can be erased.
4. Finally, erase the space(s) so that new data can be stored.

Choi et al. criticised Truecrypt, a software-based full disk encryption, for storing secret information (e.g. encryption/decryption key, user's password, and master key) while their proposed solution does not. The key problem with this criticism is that Truecrypt securely stores the *master* key encrypted using the *header* key. The header key is not stored but derived either from the user's password or keyfile or both (134). This is similar to the proposed file encryption scheme of Choi et al., but simpler (Fig. 4).

Choi et al. chose to use single file encryption because only important data is targeted and they noted that full disk encryption is significantly slower. The advantage is protection against the *cold-boot attack* (135, 136) because a user's password and key created temporarily in RAM can be safely disposed after the encryption or decryption process (118). Compared to full disk encryption where data is encrypted and decrypted on-the-fly, the key has to be stored in RAM or cached somewhere else to encrypt and decrypt data. However, important data does not constitute just a single file but may encompass multiple files. In the proposed method, if a user wants to access multiple files, the user has to provide individual passwords for decryption of each file, which is inefficient.

Physical Layer

Shin (117) explored the feasibility of implementing secure deletion in different FTL schemes. Shin claimed that current approaches are effective but suffer from low performance due to the design limitation in existing FTL schemes, and suggested the need for a new FTL scheme that is both effective and achieves good performance. Wei et al. (107) developed a new FTL scheme which involves zero-overwrite of unused copies of data. The scheme works by re-programming an unused cell to flip remaining ones to zeroes. Wei et al. cautioned that this approach could trigger a 'Program Disturb' error (i.e. memory cells that are not being programmed receiving elevated voltage stress). Reardon et al (121) criticised such an approach as reprogramming operates outside existing specification, but Wei et al. claimed that the 'Program Disturb' issue will not affect all drives.

Qin et al. (119) also adopted a similar approach to that of Wei et al. (107) in their proposal to ensure traditional data overwriting is still effective. In order to mitigate the negative effect of reprogramming, the use of RAID-5 is suggested. Although RAID-5 is generally found in corporate environments, it is widely supported in consumer product without using dedicated RAID hardware (137, 138). Despite RAID-5 being able to provide fault tolerance, there are several disadvantages. For example, a typical RAID-5 setup requires at least three disks (139), and part of the storage capacity is allocated to store parity to achieve fault tolerance. Therefore, RAID-5 is not suitable for the general consumer (due to the expenses involved).

Rather than overwriting the data to be deleted, Subha (133) proposed to render the data inaccessible. The error correcting code (ECC) is stored in a reserved area known as out-of-

band (OOB), and Subha proposed to overwrite certain parts of ECC to introduce a read error, and therefore, rendering the data that resides in the block inaccessible. This results in a faster secure deletion time since the size of the ECC is typical very small. However, it is yet unknown whether data overwritten using ECC can be subsequently accessible by the OS.

Diesburg et al. (140) proposed *TrueErase*, an approach to correctly propagate secure deletion information across layers. The approach considered the full data-paths from user-space down to physical layer. Reardon et al. (43) likened this approach to a TRIM command but argued that *TrueErase* is more efficient as it can only target sensitive parts of the file instead of the entire file. Due to consideration of all layers involved, it is the most comprehensive approach but such an approach is complex to implement (141).

Linnell (142) proposed a block erasure method whereby a block is erased by reprogramming all pages into zeroes, similar to Wei et al. (107). Before the block is erased, any pages still holding valid data (i.e. valid page) are copied to another block. However, such an approach was pointed out in earlier work (132) to be slower than zero-overwriting when there are a large number of valid pages to be copied.

Rather than using a series of write commands from the OS or host interface, Koren et al. (143) proposed wiping flash storage using a single command from the host device, which will trigger the flash storage to wipe itself. The command can also be sent wirelessly via Bluetooth or infrared. Logical conditions such as higher than usual read operations can be used to trigger the self-wiping to mitigate unauthorised disk duplication. The proposed method includes a mechanism to automatically resume an interrupted wiping process (described in "Ensure Wiping Operation is Completed" section). After the entire flash storage has been wiped, the flash storage has a built-in status to indicate the process completion.

Android Implementation

According to Simon and Anderson (97), flash memory can be erased through ioctl() system call provided by Linux kernel. On the MTD interface, *ioctl(MEMERASE)* method is used to erase flash blocks so that they are available to store new data (144, 145). When eMMC was introduced to Android devices, they used *ioctl(BLKDISCARD)* to send "TRIM" commands to the flash controller. It is not considered as secure deletion because it simply marks the block as available for new data. *ioctl(BLKSEDISCARD)* was later implemented in version 4.0 (Ice Cream Sandwich) for secure deletion. This system call will pass "SECURE ERASE" OR

"SECURE TRIM" to the flash controller. The flash controller would execute the "ERASE" or "TRIM" operation followed by "SANITIZE" (146). This is akin to ATA/ATAPI Command Set-2 (ACS-2) "SANITIZE BLOCK ERASE" used in the desktop drive (107).

Table 4 summarises the deletion methods implemented throughout the history of Android as identified by Simon and Anderson (97).

Case Study on Android Approach

As noted earlier, previous studies (notably by Wei et al. (107) and Reardon et al. (43)) have shown that secure deletion tools operating at the user-space layer are ineffective in sanitising data when running on flash memory. Wei et al. claimed it was due to FTL of flash memory while Reardon et al. highlighted the file system design that affects the effectiveness of those tools.

The secure deletion method implemented by Android in its Linux kernel is likely to be file system-agnostic since the OS directly instructs the flash controller to securely delete specific blocks. Since the flash controller is directly involved, we also assume it is a physical layer approach. In this section, we analyse the effectiveness of this approach in Android. Sets of applications and data are loaded onto the mobile devices. The mobile devices are then factory reset which would also securely delete user data in accordance with the appropriate Android version. Finally, the mobile devices are physically acquired to be analysed by forensic tools. The forensic tools would attempt to recover user data, and the amount of data recovered indicates how effective the secure deletion is.

We performed our experiments on three mobile devices. All three devices were installed with different Android versions (see : Tables 9-12). Moto G was shipped with version 4.3 Jellybean (JB) and has been upgraded to version 4.4.2 KitKat (KK). Nexus S was shipped with 2.3 Gingerbread (GB) and upgraded to 4.1.2 Jellybean (JB). Nexus 4 was shipped with 4.2 JB and upgraded to 5.1 Lollipop (L).

The mobile devices were restored to their factory image. Sets of applications and data were loaded onto the mobile devices. "Pre-wipe" dd physical image was then taken. Factory reset was then performed and "Post-wipe" image is taken. This was repeated for the other two devices under test in this prototype experiment. Following are the steps taken to prepare the data for "Pre-wipe":

1. Sign into Google account and connect to “unisa” wireless network.
2. Save 30 contacts and call 10 of them.
3. Sync email. (Emails were generated on the Google account beforehand)
4. Install Google Drive, Dropbox, Box, & OneDrive.
5. Download documents:
    a. 30 DOCX through Google Drive.
    b. 30 PPTX through Dropbox.
    c. 30 XLSX through Box.
    d. 30 PDF through OneDrive.
6. Transfer following files to the mobile device:
    a. 120 JPEG pictures.
    b. 35 MP4 videos.
    c. 30 MP3 audios.
7. Browse to 50 websites and bookmark them.
8. Sign into Reddit (www.reddit.com) and save login.
9. Install and sign into Facebook app.
10. Install and sign into Skype app (except for Nexus S).

We only acquire the */data* partition (for Moto G and Nexus 4) and the */media* partition (for Nexus S) for our experiments because the partition is used to store user data and there are different partition layouts depending on the Android version. Prior to Android 3.0 (Honeycomb), */data* and */media* are two separate partitions. On Honeycomb or later, /media is no longer a dedicated partition; instead, it becomes a folder */data/media* in */data* partition. Even though our Nexus S is equipped with JB, because it was originally shipped with GB, the previous partition layout is still retained.

Four forensic tools are used to perform data recovery on all physical images acquired, namely: UFED Physical Analyzer (v4.1 Trial), Internet Evidence Finder (IEF v6.5 Trial), PhotoRec (v6.14), and Scalpel (v2.0). Scalpel is conFig.d to recover JPEG thumbnail only (see (147) for detailed information on the recovery technique). “Pre-wipe” physical images serve as a baseline to verify capability of these forensic tools.

Results are shown in Appendix: Tables 9-12 indicating the number of files recovered. Our results are consistent with Table 4. Since the Android version running on Nexus S (JB) is using an insecure deletion method on media partition (*ioctl(BLKDISCARD)*), significant user

data can be recovered. This result is consistent with previous studies that have highlighted the issue of factory reset in Android being ineffective in sanitising user data. In contrast, the result suggests that if a secure deletion method (*ioctl(BLKSEDISCARD)*) is supported (in the case of Moto G and Nexus 4), user data can be sufficiently sanitised. Even when some results for Moto G and Nexus 4 are more than zero, which implies some data could be recovered, upon more inspection, we determined that the recovered data is false positive. Also note that although the deletion method of Lollipop used by Nexus 4 is not known, we assume it is either similar or perhaps more secure to KK.

Our preliminary results suggest that the secure deletion method implemented by Android can sufficiently sanitise user data. However, it might not be applicable to other mobile device as proper hardware implementation is also required, as evidenced by Simon and Anderson (97). In addition, the data acquisition and recovery aspects could be limitations of our experiments. With respect to data recovery, our experiments could be limited by the capability of our forensic tools. This is despite the UFED Physical Analyzer and the Internet Evidence Finder being commercial forensic tools commonly used by forensic investigators. Manual scrutinisation could be topics for future work. With respect to data acquisition, even though we use *dd* - a very popular physical data acquisition tool, it operates at the user-space layer which might be hindered by FTL. Future study could use a data acquisition tool that is able to bypass FTL (e.g. custom hardware used by Wei et al. (107)). We could also review how viable such an approach is to an adversary.

Discussion

We now discuss the various limitations that we found in existing approaches before summarising this section.

Limitations of Existing Literature

Lack of Data Recovery Evaluation

It is clear that most existing approaches focus on data recoverability testing, but there is a lack of data recovery evaluation particularly at the file system and physical layers.

Almost all existing approaches had limited evaluations to determine their suitability (i.e. one experiment per device in most proposals) which brings into question whether the approach

can be widely deployed over the wide range of mobile devices. For example, Wei et al. (107) argued that different flash storage media could exhibit different behaviours and, therefore, conducted tests on a wide range of commercially available consumer hardware. In addition, the findings are dated, many of the devices tested such as HTC Nexus One are either discontinued or no longer available. Therefore, findings from Albano et al. (122), Reardon et al. (121), Reardon et al. (103) and others may no longer be applicable to newer devices and OS.

Limitation of Using Simulation in Evaluations

A number of proposals were evaluated in the simulated environment (119, 133). While there are advantages in using a simulated environment such as a mobile emulator (e.g. ease of use, without the need for a custom-build hardware platform, and the ability to evaluate an expensive or yet to be available commercial flash technology as outlined by Grupp et al. (148)), simulation results are likely to be less reliable than actual hardware-based evaluations; hence, limiting our understanding of the resulting real-world implications (149).

Performance

Modification at the hardware-level does not necessarily mean modifying the flash memory cell or the processor of the SSD controller. Rather, in this paper, it refers to the modification of the software or the firmware running at the physical level. Thus, the *software-based* approach referred to in Table 7 is implemented above the physical level, namely the file system and user-space layers.

Software-based implementation generally has a lower throughput due to the number of layers involved; whilst a hardware-based implementation operates on the disk's firmware which allows it to run at the disk's full bandwidth (150). For instance, software-based full disk encryption (FDE) generally impacts on the disk's performance even in flash storage (151).

Possible Attacks

A hardware-based secure deletion approach can perform erasure on all memory blocks (107), but a software-based solution may not be able to access some of the blocks. ATA's *Secure Erase* function is the most commonly available hardware-based secure deletion method,

which can be found in most drives manufactured on or after year 2001. However, researchers such as Wei et al. (107) and Swanson & Wei (152) found that some drives either fail to complete the deletion process required in the Secure Erase function or do not erase the data at all after executing that function.

Data can also be protected using encryption, such as hardware-based drive encryption (also known as self-encrypting drive – SED). Müller et al. (153) proposed a *hot plug attack*, where an adversary is able to gain unauthorised access to the data residing in the SED. In short, this is due to the fact that when using the SED, a user unlocks the drive when powering on the machine. After the disk is unlocked, and while the disk is still running ("*hot*"), an adversary simply re-plugs the SATA cable from the original machine to the adversary's machine to access the SED without knowing the password. An adversary can also access the drive directly by attaching a USB drive into the original machine if it is not screen locked.

To ensure high security compliance, there are several industry standards for SED, such as Opal Security Subsystem Class (Opal SSC) by Trusted Computing Group (TCG) (154) and "Encrypted Hard Drive" (eDrive) by Microsoft (155). The latter, for example, is partly based on Opal SSC and IEEE 1667 (156). Major SED manufacturers offer OPAL-compliant products (153, 157, 158, 159). Müller et al. (153) did not evaluate Opal-compliant SEDs, but claimed that the *hot plug attack* affects such drives too. Since then, there had been no major revision to the Opal SSC standard, and it is unknown whether such an attack claimed by Müller et al. is valid.

On the other hand, software-based disk encryption may be vulnerable to a *cold boot attack* (135; 136) because the encryption key is cached in RAM. In such an attack, an adversary removes the RAM, re-plugs into another machine, and extracts the key from the RAM. Such an attack is, however, difficult to carry out and can be mitigated by keeping the key outside of RAM (160). In addition, software-based disk encryption does not encrypt the boot sector. Therefore, an adversary is able to launch an *evil maid attack* (161) by installing a *bootkit* (**boot** sector root**kit**) into the victim's machine to capture the password entry. Another form of evil maid attack can be launched against hardware-based disk encryption. In this case, an adversary removes the victim's disk and replaces it with another disk loaded with the adversary's modified OS designed to capture the password entry (153, 162). In this case, it can be thwarted using ATA's password. The evil maid attack is possible in either software-

based or hardware-based systems due to a lack of a trusted boot environment (163) to authenticate the boot sector or the disk *to* the user (162).

Summary

Table 7 summarises the key differences between hardware- and software-based implementations of secure flash storage deletion.

The advantage in using a software-based approach is that it allows for easy verifiability. For example, if the source code is available, then a public security audit can be conducted using forensic techniques as demonstrated on TrueCrypt (134). Hardware-based verification may require not only building a customised platform to access the memory directly but also dismantling the device (152).

Modification on the hardware level can be very challenging, as one would generally require having access to the source code of the firmware or the specification of the disk controller. It may be possible to replace the firmware, but there is the risk of *bricking* the device. Software-based solutions have a lower risk of damaging the hardware. Software-based modification, especially at the file system layer, usually builds on an existing open source file system. The improvement can be implemented simply by installing a new patch. Even in the case of a new software component, existing data can be migrated with relative ease.

In addition, software-based modification is generally hardware independent. Implementation at the hardware-level, however, may require compatibility at the higher layers. The user would need to install new software, for example to support Opal SSC (164), and acquire new hardware. For example, in Intel SED, the drive is encrypted by default using the unique key generated during manufacture (165). It is activated simply by using the drive. This mechanism could only protect in a situation where the NAND chip has been removed, assuming that the key is not stored in the NAND chip or the location is only known to the controller. Thus, SED behaves more like a "self-decrypting disk" (153) in the default configuration. To protect the drive, the user would need to set the ATA password which controls access to the drive, and consequently the data. This mechanism requires ATA specification compatibility. Although the majority of consumer hard drives use a Serial ATA (SATA) interface, there are drives that utilise SCSI/SAS, Fibre Channel, or PCIe host interfaces.

Some new hardware-based features require installation of new hardware which is a more expensive option. In the software-based approach, a user can take advantage of new features via software updates.

Concluding Remarks

We have examined (the limited) literature on remote wiping, particularly secure deletion on flash storage. Despite the prevalence of remote wiping, most existing literature provided a high-level approach to remote wiping and secure data deletion. There are relatively few technical papers evaluating the implementation of such approaches or techniques on a wide range of popular mobile devices. One reason that this may not have been thoroughly explored is due to the cost and efforts associated with such evaluations.

In addition to conducting a comparative summary of existing approaches, we identified existing limitations and the research trends over the years (see Table 8).

As shown in Table 8, the majority of remote wiping patents were filed prior to 2010, although academic interest on the topic appears to have increased since then. A similar trend was observed with secure flash storage deletion, where there are at least three publications annually since 2010.

This review highlights a number of literature gaps which are as follow:

1. *The need to provide message confidentiality using encryption and ensure that the wiping process cannot be interrupted.* From our survey described in "Reviews of Remote Wiping and Secure Flash Storage Deletion" section, existing proposals generally do not consider securing the wipe command ("Secure Wipe Command" section) nor provide any mechanism to automatically resume an interrupted wiping process ("Ensure Wiping Operation is Completed" section).
2. *The need for comprehensive evaluations on the security and effectiveness based on real-world implementations of remote wiping*. It is essential to ensure that the remote wiping command cannot be hijacked by attackers (e.g. to prevent the wiping of lost or stolen devices) or initiated by attackers (e.g. to remotely wipe contents from a victim's device) and wiped data cannot be recovered using contemporary forensic techniques.

3. *Real world implementations that mitigate identified shortcomings.*
4. *The need for evaluation of physical layer implementation on actual hardware.* As discussed in "Limitation of Using Simulation in Evaluations" section, evaluations of existing hardware-level research on flash storage are generally conducted with a simulator. The findings may not take into consideration internal workings of the flash storage (as manufacturers may be hesitant to provide such information to protect their intellectual property) (149, 152). To overcome this limitation, Diesburg et al. (140) suggested using *OpenSSD* (166), a research platform designed for flash storage research. There are also a few alternative platforms, such as *FRP* (167), *BlueSSD* (168), and *Ming II* (169). These open platforms allow researchers to have unfettered access to the hardware especially the FTL, which is not possible on commercial flash storage.
5. *The need for stronger collaboration between manufacturer and academic researcher*. In this review, we highlighted the importance of FTL as a vital factor in secure flash storage deletion. For instance, Diesburg et al. (140) acknowledged their work is only possible with access to software FTL and pointed out the trend of hardware FTL in recent times. For example, newer Android versions have started utilising hardware FTL ("Flash Storage Layers and Structures" section). Since hardware FTL is not accessible in commercial hardware or the open research platforms mentioned earlier, any research or evaluation on FTL could not be conducted without the involvement and collaboration of a manufacturer. Therefore, it is argued that a stronger collaboration between manufacturer and academic researcher will result in a more secure product. Researchers can work with Open NAND Flash Interface (ONFi), a consortium of flash memory manufacturers, to incorporate a secure deletion method into the ONFi specification to facilitate wider adoption.

Does stronger security hinder law enforcement? Stronger security of remote wiping mechanisms can help protect the privacy of the consumer. However, such a benefit could also be abused by criminals to remove incriminating evidence (170, 171). In addition, law enforcement have explained that they could not extract useful evidence from mobile devices due to storage encryption (172, 173, 174), particularly in the post NSA revelations as mobile device vendors and other technology companies enforce encryption by default in their products (175, 176). Whether such a trend really does hinder law enforcement is the subject of controversy and, perhaps, worthy of discussion. For example, how do we balance the need

for user privacy with the legitimate needs of government and law enforcement agencies to access data to facilitate their investigations?

References

1. AMTA 2011, Lost and stolen phones, Australian Mobile Telecommunications Association; http://www.amta.org.au/pages/Lost.and.stolen.phones.
2. Lynn G, Davey E. Black market' for stolen smartphones exposed. BBC, 2014; http://www.bbc.com/news/uk-england-london-26979061 (accessed October 7, 2015).
3. Anderson N. Hacking Scarlett Johansson – and 50 other celebs – using google and gumption. Ars Technica, 2012; http://arstechnica.com/tech-policy/2012/03/hacking-scarlett-johanssonand-50-other-celebsusing-google-and-gumption/ (accessed October 7, 2015).
4. Gallagher S. What Jennifer Lawrence can teach you about cloud security. Ars Technica, 2014; http://arstechnica.com/security/2014/09/what-jennifer-lawrence-can-teach-you-about-cloud-security/ (accessed October 7, 2015).
5. Greenberg A. The police tool that pervs use to steal nude pics from apple's iCloud. Wired, 2014; http://www.wired.com/2014/09/eppb-icloud/ (accessed October 7, 2015).
6. Verizon. 2014 data breach investigation report; http://www.verizonenterprise.com/DBIR/2014/ (accessed October 7, 2015).
7. Symantec. Symantec smartphone honey stick project. Media Release, 2012; https://www.symantec.com/about/news/resources/press_kits/detail.jsp?pkid=symantec-smartphone-honey-stick-project (accessed October 7, 2015).
8. Caldwell T. The mobile ‘kill pill’ – poison or panacea. Computer Fraud & Security 2011;10:8–12.
9. Burnett RD, Friedman M, Rodriguez RP. Managing laptop security. Journal of Corporate Accounting & Finance 2011;22(5):53–61.

10. Hansen CK. Technology trends in mobile communications how mobile are your data? IEEE Reliability Society 2009 Annual Technology Report, 2010; http://paris.utdallas.edu/IEEE-RS-ATR/document/2009/2009-07.pdf.
11. Evers J, Johnston CJ. System architecture. In: Professional blackberry. Indianapolis, IN: Wiley Publication, 2005;3–18.
12. Punja SG, Mislan RP. Mobile device analysis. Small Scale Digital Device Forensics Journal 2008;2(1):1–16.
13. Microsoft. Deploying windows mobile 5.0 with windows small business server 2003. TechNet Library, 2009; http://technet.microsoft.com/en-us/library/cc747512(v=WS.10).aspx (accessed October 7, 2015).
14. Microsoft. Microsoft releases windows mobile 5.0, 2005; http://www.microsoft.com/en-us/news/press/2005/may05/05-10windowsmobile5pr.aspx (accessed October 7, 2015).
15. Munro K. Ghost in the machine. Itnow 2008;50(2):10.
16. Erlichman J, Miller H. BlackBerry CEO briefs white house to keep Obama loyal. Bloomberg Technology, 2014; http://www.bloomberg.com/news/2014-03-06/blackberry-ceo-briefs-white-house-to-cultivate-vip-obama.html (accessed October 7, 2015).
17. Harauz J, Kaufman LM. A new era of presidential security: the president and his BlackBerry. IEEE Security & Privacy 2009;7(2):67–70.
18. Morrison G. Implementation guide for email protective markings for Australian government agencies, Australian Government Information Management Office, Department of Finance and Administration, Australia, 2005; http://www.finance.gov.au/publications/protective-markings/docs/Protective_Markings.pdf (accessed October 7, 2015).

19. Ogg E. Updated: IPhone OS 3.0 now available, CNET, 2009; http://www.cnet.com/news/updated-iphone-os-3-0-now-available/ (accessed October 7, 2015).
20. Mayers S, Lee M. From MobileMe to iCloud. In: Learn OS X lion. New York, NY: Apress, 2011;245–53.
21. Apple. iOS 4.2 software update. Apple, Inc, 2010; http://support.apple.com/kb/DL1061 (accessed October 7, 2015).
22. Aomoth D. App of the week: find my iPhone. TIME, 2010; http://techland.time.com/2010/11/23/app-of-the-week-find-my-iphone/ (accessed October 7, 2015).
23. Poiesz B. Find your lost phone with android device manager. Android Official Blog, 2013; http://officialandroid.blogspot.com/2013/08/find-your-lost-phone-with-android.html (accessed October 7, 2015).
24. Zhang H. Make mobile more manageable. Official Google for Work Blog, 2012; http://googleforwork.blogspot.com/2012/08/make-mobile-more-manageable.html (accessed October 7, 2015).
25. Shin Y. Non-volatile memory technologies for beyond 2010. In: Proceedings of Symposium on VLSI Circuits Digest of Technical Papers; 2005 Jun 16-18; Kyoto, Japan. Piscataway, NJ: IEEE, 2005;156–9.
26. BlackBerry. Enforcing encryption of internal and external file systems on BlackBerry devices, 2010; http://www.manualslib.com/products/Blackberry-Enterprise-Solution-Security-Enforcing-Encryption-Of-Internal-And-External-File-Systems-On-Devices-2590666.html (accessed October 7, 2015).
27. Götzfried J, Müller T. ARMORED: CPU-bound encryption for android-driven ARM devices. In: Proceedings of Eight International Conference on Availability, Reliability

and Security; 2013 Sep 2-6; Regensburg, Germany. Piscataway, NJ: IEEE, 2013;161–8.

28. Teufl P, Zefferer T, Stromberger C. Mobile device encryption systems. In: Janczewski LJ, Wolfe HB, Shenoi S, editors. IFIP Advances in Information and Communication Technology. Heidelberg, Germany: Springer, 2013;405:203–16.
29. Microsoft. Device encryption, Security for Windows Mobile Devices, 2010; http://msdn.microsoft.com/en-us/library/bb964600.aspx (accessed October 7, 2015).
30. Belfiore J. Announcing Windows Phone 8. Windows Blog, 2012; http://blogs.windows.com/windowsexperience/2012/06/20/announcing-windows-phone-8/ (accessed October 7, 2015).
31. Godfrey R. Announcing Windows Phone 8. Microsoft UK Schools Blog, 2012; http://blogs.msdn.com/b/ukschools/archive/2012/06/27/announcing-windows-phone-8.aspx (accessed October 7, 2015).
32. Götzfried J, Müller T. Analysing Android's full disk encryption feature. J Wirel Mob Netw Ubiquitous Com Dependable Appl 2014;5(1):84–100.
33. Timberg C. Newest androids will join iPhones in offering default encryption, blocking police. The Washington Post, 2014; https://www.washingtonpost.com/news/the-switch/wp/2014/09/18/newest-androids-will-join-iphones-in-offering-default-encryption-blocking-police/ (accessed October 7, 2015).
34. Chester B, Ho J. Encryption and storage performance in android 5.0 lollipop. AnandTech, 2014; http://www.anandtech.com/show/8725/encryption-and-storage-performance-in-android-50-lollipop (accessed October 7, 2015).

35. Google. A sweet lollipop, with a kevlar wrapping: new security features in Android 5.0. Official Android Blog, 2014; http://officialandroid.blogspot.com/2014/10/a-sweet-lollipop-with-kevlar-wrapping.html (accessed October 7, 2015).
36. Malchev I. Enable hardware crypto for userdata encryption. Git at Google, 2014; https://android.googlesource.com/platform/system/vold/+/bb7d9afea9479eabbc98133d3d968225a1e1019e%5E%21/#F0 (accessed October 7, 2015).
37. Franco F. Continuing the tale of "Nexus 6 is way smoother on Android 5.1", 2015; https://plus.google.com/+FranciscoFranco1990/posts/3RKjDGjjPQ7 (accessed October 7, 2015).
38. Elenkov N. Hardware-accelerated disk encryption in Android 5.1, 2015; http://nelenkov.blogspot.com/2015/05/hardware-accelerated-disk-encryption-in.html (accessed October 7, 2015).
39. Percival C. Stronger key derivation via sequential memory-hard functions, 2009; https://www.tarsnap.com/scrypt/scrypt.pdf (accessed October 7, 2015).
40. Elenkov N. Revisiting Android disk encryption, 2014; http://nelenkov.blogspot.com/2014/10/revisiting-android-disk-encryption.html (accessed October 7, 2015).
41. Crimes Act s 3LA. Commonwealth of Australia, Australia (1914).
42. Ockenden W, Sveen B. Abdilo, infamous Australian teen hacker, raided by police and ordered to surrender passwords. ABC News, 2015; http://www.abc.net.au/news/2015-04-02/infamous-australian-teenager-hacker-abdilo-raided-by-police/6368612 (accessed October 7, 2015).
43. Reardon J, Basin D, Capkun S. SoK: Secure data deletion. In: Proceedings of Symposium on Security and Privacy; 2013 May 19-22; San Francisco, CA. Piscataway, NJ: IEEE, 2013;301–15.

44. Storer MW, Greenan K, Miller EL. Long-term threats to secure archives. In: Proceedings of the Second Workshop on Storage Security and Survivability; 2006 Oct 31-Nov 2; Alexandria, VA. New York, NY: ACM, 2006;9–16.
45. Diesburg SM. Per-file full-data-path secure deletion for electronic storage (PhD dissertation). Tallahassee, FL: Florida State University, 2012.
46. DON CIO Privacy Team. Methods for hard drive/disk destruction. Department of Navy Chief Information Officer, 2010; http://www.doncio.navy.mil/ContentView.aspx?ID=1867 (accessed October 7, 2015).
47. Hughes GF, Coughlin TM. Tutorial on disk drive data sanitization. San Diego, CA: Center for Magnetic Recording Research, University of California, 2006.
48. Angelo M, Novoa M, Olarig S, inventors. After the fact protection of data in remote personal and wireless devices. US patent 2003/0065934. 2003 Apr 3.
49. Brown MK, Brown MS, Little HA, Totzke SW, inventors. BlackBerry Ltd, assignee. Selectively wiping a remote device. US patent 8,056,143. 2011 Nov 8.
50. Walker DR, Fyke SH, inventors. BlackBerry Ltd, assignee. System and method for remote wipe through voice mail. EU patent EP2575384A1. 2013 Apr 3.
51. Kenney T, inventor. Systems and methods that provide user and/or network personal data disabling commands for mobile devices. US patent 2005/0186954. 2005 Aug 25.
52. Hasebe M, inventor. Toshiba Corp, assignee. System for remotely securing/locking a stolen wireless device via an email message. US patent 5,987,609. 1999 Nov 16.
53. Sennett DWA, Daly BK, inventors. AT&T Mobility II LLC, assignee. Remote disablement of a communication device. US patent 8,375,422. 2013 Feb 12.
54. Onyon R, Stannard L, Ridgard L, inventors. Remote cell phone auto destruct. US patent 2007/0056043A1. 2007 Mar 8.

55. Gajdos T, Kretz M, inventors. Sony Ericsson Mobile Communications AB, assignee. Method for disabling a mobile device. EU patent EP1725056A1. 2006 Nov 22.
56. Yu X, Wang Z, Sun K, Zhu WT, Gao N, Jing J. Remotely wiping sensitive data on stolen smartphones. In: Proceedings of the Ninth ACM Symposium on Information, Computer and Communications Security; 2014 Jun 4-6; Kyoto, Japan. New York, NY: ACM, 2014;537–42.
57. Park K, Ma GI, Yi JH, Cho Y, Cho S, Park S. Smartphone remote lock and wipe system with integrity checking of SMS notification. In: Proceedings of International Conference on Consumer Electronics; 2011 Jan 9-12; Las Vegas, NV. Piscataway, NJ: IEEE, 2014;263–4.
58. Kuppusamy KS, Senthilraja R, Aghila G. A model for remote access and protection of smartphones using short message service. Int J Comput Sci En Inf Technol 2012;2(1):91–100.
59. Joe I, Lee Y. Design of remote control system for data protection and backup in mobile devices. In: Proceedings of International Conference on Interaction Sciences; 2011 Aug 16-18; Busan, South Korea. Piscataway, NJ: IEEE, 2011;189–93.
60. Adusumalli P. Implementation of an android application to retrieve information from a lost android device [Graduate project report]. Corpus Christi, TX: Texas A&M University-Corpus Christi, 2014.
61. Honan M. How Apple and Amazon security flaws led to my epic hacking. Wired, 2012a; http://www.wired.com/2012/08/apple-amazon-mat-honan-hacking/all/ (accessed October 7, 2015).
62. Zetter K. Palin E-mail hacker says it was easy. Wired, 2008; http://www.wired.com/2008/09/palin-e-mail-ha/ (accessed October 7, 2015).

63. Guo C, Wang HJ, Zhu W. Smart-phone attacks and defences. In: Proceedings of Third Workshop on Hot Topics in Networks; 2004 Nov 15-16; San Diego, CA. New York, NY: ACM, 2004;235–45.
64. Apple. iOS security, 2015; https://www.apple.com/business/docs/iOS_Security_Guide.pdf (accessed October 7, 2015).
65. BlackBerry. Security technical overview of BES10 cloud solution, 2014a; https://www.blackberry.com/blackberrytraining/web/KryptonDocs/resources/BES10_Cloud_Market_Preview_Security_Technical_Overview_en.pdf (accessed October 7, 2015).
66. BlackBerry. Blackberry enterprise solution 5.0.2: Security technical overview, 2011; http://docs.blackberry.com/en/admin/deliverables/38816/BlackBerry_Enterprise_Server_5.0_SP3_and_BlackBerry_7.1-Security_Technical_Overview--1936256-0117012254-001-5.0.3-US.pdf (accessed October 7, 2015).
67. Li T, Zhou X, Xing L, Lee Y, Naveed M, Wang X, Han X. Mayhem in the push clouds: understanding and mitigating security hazards in mobile push-messaging services. In: Proceedings of Conference on Computer and Communications Security; 2014 Nov 3-7; Scottsdale, AZ. New York, NY: ACM, 2014a;978–89.
68. Android Developers n.d.a, Android security overview; http://source.android.com/devices/tech/security/index.html (accessed October 7, 2015).
69. Android Developers n.d.b. Device administration; https://developer.android.com/guide/topics/admin/device-admin.html (accessed October 7, 2015).

70. ManageEngine n.d. MDM architecture; http://www.manageengine.com/products/desktop-central/mobile-device-management-mdm-architecture.html (accessed October 7, 2015).
71. BlackBerry. BlackBerry device service solution security technical overview (v10.2), 2014b; http://docs.blackberry.com/en/admin/deliverables/60198/BES10_v10.2_BDS_Security_Technical_Overview_en.pdf (accessed October 7, 2015).
72. BlackBerry. BlackBerry device service advanced administration guide (v10.2), 2014c; http://docs.blackberry.com/en/admin/deliverables/59623/BES10_v10.2_BDS_Advanced_Admin_Guide_en.pdf (accessed October 7, 2015).
73. BlackBerry. BlackBerry protect user guide (v1.2.1), 2015; http://docs.blackberry.com/en/smartphone_users/deliverables/49400/BlackBerry_Protect-User_Guide-1343391475156-1.2.1-en.pdf (accessed October 7, 2015).
74. Apple n.d.a. iPhone in business: manage devices; https://www.apple.com/iphone/business/it/management.html (accessed October 7, 2015).
75. Apple n.d.b. About configuration profile; https://help.apple.com/configurator/mac/1.6/#/cadbf9e668 (accessed October 7, 2015).
76. Microsoft. Exchange ActiveSync, 2013; http://technet.microsoft.com/en-US/library/aa998357%28v=exchg.150%29.aspx (accessed October 7, 2015).
77. Microsoft. Windows Phone 8.1 security overview, 2014; http://www.microsoft.com/en-us/download/details.aspx?id=42509&751be11f-ede8-5a0c-058c-2ee190a24fa6=True (accessed October 7, 2015).

78. Hannay P, Carpene C, Valli C, Woodward A, Johnstone M. Exchanging demands: weaknesses in SSL implementations for mobile platforms. In: Proceedings of the Eleventh Australian Information Security Management Conference; 2013 Dec 2-4; Perth, Australia. Perth, Australia: Edith Cowan University, 2013;42–48.
79. Hubbard J, Weimer K, Chen Y. A study of SSL proxy attacks on android and iOS mobile applications. In: Proceedings of the Eleventh Consumer Communications and Networking Conference; 2014 Jan 10-13; Las Vegas, NV. Piscataway, NJ: IEEE, 2014;86–91.
80. Fahl S, Harbach M, Muders T, Baumgärtner L, Freisleben B, Smith M. Why eve and mallory love android: An analysis of android SSL (in)security. In: Proceedings of the Conference on Computer and Communications Security; 2012 Oct 16-18; Raleigh, NC. New York, NY: ACM, 2012;50–61.
81. Georgiev M, Iyengar S, Jana S, Anubhai R, Boneh D, Shmatikov V. The most dangerous code in the world: validating SSL certificates in non-browser software. In: Proceedings of the Conference on Computer and Communications Security; 2012 Oct 16-18; Raleigh, NC. New York, NY: ACM, 2012;38–49.
82. Ducklin P. Anatomy of a "goto fail" - Apple's SSL bug explained, plus an unofficial patch for OS X!, Naked Security, 2014; https://nakedsecurity.sophos.com/2014/02/24/anatomy-of-a-goto-fail-apples-ssl-bug-explained-plus-an-unofficial-patch/ (accessed October 7, 2015).
83. Li T, Zhou X, Xing L, Lee Y, Naveed M, Wang X, Han X. Supplement materials for mayhem in the push clouds paper, 2014b; https://sites.google.com/site/cloudmsging/ (accessed October 7, 2015).

84. Greenberg A. The police tool that pervs use to steal nude pics from Apple's iCloud, Wired, 2014; http://www.wired.com/2014/09/eppb-icloud/ (accessed October 7, 2015).
85. Kingsley-Hughes A. Apple patches 'find my iPhone' exploit. ZDNet, 2014; http://www.zdnet.com/apple-patches-find-my-iphone-exploit-7000033171/ (accessed October 7, 2015).
86. Pagliery J. Hackers can 'un-brick' stolen iPhones, CNN, 2014; http://money.cnn.com/2014/05/21/technology/security/icloud-hack/ (accessed October 7, 2015).
87. OWASP. Cross-site request forgery (CSRF), 2014; https://www.owasp.org/index.php/Cross-Site_Request_Forgery_%28CSRF%29 (accessed October 7, 2015).
88. US-CERT. CVE-2014-8346, National Vulnerability Database, 2014; http://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2014-8346 (accessed October 7, 2015).
89. Honan M. Yes, I was hacked. Hard, 2012b; http://www.emptyage.com/post/28679875595/yes-i-was-hacked-hard (accessed October 7, 2015).
90. OWASP. OWASP top 10, 2013; https://www.owasp.org/index.php/Top10 (accessed October 7, 2015).
91. Siciliano R. I found your data on that used device you sold, McAfee, 2012; https://blogs.mcafee.com/consumer/i-found-your-data-on-that-used-device-you-sold (accessed October 7, 2015).

92. Honan M. Break out a hammer: you'll never believe the data 'wiped' smartphones store, Wired, 2013; http://www.wired.com/2013/04/smartphone-data-trail/ (accessed October 7, 2015).
93. The Guardian. Recycled mobile phones retain previous owner data, 2013; http://web.archive.org/web/20140529182505/http://www.theguardian.com/media-network/partner-zone-infosecurity/mobile-phones-previous-owner-data (accessed October 12, 2015).
94. Schwamm R. Effectiveness of the factory reset on a mobile device [Master's thesis]. Monterey, CA: Naval Postgraduate School, 2014.
95. Schwamm R, Rowe NC. Effects of the factory reset on mobile devices. J Digit Forensics, Security and Law 2014;9(2):205–20.
96. McColgan J. Tens of thousands of Americans sell themselves online every day, AVAST Software, 2014; https://blog.avast.com/2014/07/08/tens-of-thousands-of-americans-sell-themselves-online-every-day/ (accessed October 12, 2015).
97. Simon L, Anderson R. Security analysis of Android factory resets. In: Proceedings of the Mobile Security Technologies Workshop; 2015 May 21; San Jose, CA. Piscataway, NJ: IEEE, 2015a;1-10; http://ieee-security.org/TC/SPW2015/MoST/papers/s1p3.pdf (accessed October 12, 2015).
98. Agrawal N, Prabhakaran V, Wobber T, Davis JD, Manasse M, Panigrahy R. Design tradeoffs for SSD performance. In: Proceedings of the USENIX Annual Technical Conference; 2008 Jun 22-27; Boston, MA: USENIX Association, 2008;57–70.
99. Woodhouse D. General MTD documentation, Memory Technology Device (MTD) Subsystem for Linux, 2008; http://www.linux-mtd.infradead.org/doc/general.html (accessed October 7, 2015).

100. Intel. Understanding the flash translation layer (FTL) specification, 1998; http://www.eetasia.com/ARTICLES/2002MAY/2002MAY07_MEM_AN.PDF (accessed October 12, 2015).

101. Ma D, Feng J, Li G. A survey of address translation technologies for flash memories. ACM Comput Surv 2014;46(3):1–39.

102. Skillen A, Mannan M. On implementing deniable storage encryption for mobile devices. In: Proceedings of the Twentieth Annual Network and Distributed System Security Symposium; 2013 Feb 24-27; San Diego, CA. Reston, VA: Internet Society, 2013;1–17.

103. Reardon J, Capkun S, Basin D. Data node encrypted file system: efficient secure deletion for flash memory. In: Proceedings of the Twenty Second USENIX Security Symposium; 2013 Aug 14-16; Washington, D.C. Berkeley, CA: USENIX Association, 2013b;333–48.

104. Kim H, Kim J, Choi S, Jung H, Jung J. A page padding method for fragmented flash storage. In: Gervasi O, Gavrilova ML, editors. Lecture Notes in Computer Science. Heidelberg, Germany: Springer, 2007;4705:164–77.

105. Huang P, Zhou K, Wu C. ShiftFlash: make flash-based storage more resilient and robust. Performance Evaluation 2011;68(11)1193–206.

106. Gutmann P. Secure deletion of data from magnetic and solid-state memory. In: Proceedings of the Sixth USENIX Security Symposium; 1996 Jul 22-25; San Jose, CA. Berkeley, CA: USENIX Association, 1996;1–18.

107. Wei MYC, Grupp LM, Spada FE, Swanson S. Reliably erasing data from flash-based solid state drives. In: Proceedings of the Ninth USENIX Conference on File and Storage Technologies; 2011 Feb 15-17, San Jose, CA. Berkeley, CA: USENIX Association, 2011;8–20.

108. Garfinkel SL, Shelat A. Remembrance of data passed: a study of disk sanitization practices. IEEE Security & Privacy 2003;1(1):17–27.

109. Hughes GF, Coughlin TM. Secure erase of disk drive data. IDEMA Insight Magazine 2002;25.

110. Cornell University. Disk and file erasure, best practices for media destruction, 2012; http://www.it.cornell.edu/security/depth/practices/media_destruct.cfm#erasure (accessed October 12, 2015).

111. Kissel R, Regenscheid A, Scholl M, Stine K. Guidelines for media sanitization, NIST Special Publication 800-88, 2014; http://dx.doi.org/10.6028/NIST.SP.800-88r1 (accessed October 12, 2015).

112. Australian Signals Directorate. 2015 Australian Government Information Security Manual, 2015; http://www.asd.gov.au/publications/Information_Security_Manual_2015_Controls.pdf (accessed October 12, 2015).

113. Joukov N, Papaxenopoulos H, Zadok E. Secure deletion myths, issues, and solutions. In: Proceedings of the Second Workshop on Storage Security and Survivability; 2006 Oct 31-Nov 2; Alexandria, VA. New York, NY: ACM, 2006;61–6.

114. Wright C, Kleiman D, Sundhar S. Overwriting hard drive data: the great wiping controversy. In: Sekar R, Pujari AK, editors. Lecture notes in computer science. Heidelberg, Germany: Springer, 2008;5352:243–57.

115. Gutmann P. Secure deletion of data from magnetic and solid-state memory, Department of Computer Science, University of Auckland, New Zealand, 2003; https://www.cs.auckland.ac.nz/~pgut001/pubs/secure_del.html (accessed October 12, 2015).

116. Spreitzenbarth M, Holz T. Towards secure deletion on smartphones. In: Proceedings of the Fifth Conference of the GI Special Interest Group “Sicherheit, schutz und zuverlässigkeit”; 2010 Oct 5-7; Berlin, Germany. Bonn, Germany: Gesellschaft für Informatik e.V. (GI), 2010;165–76.

117. Shin I. Secure file delete in NAND-based storage. International Journal of Security and its Applications 2012;6(2):257–60.

118. Choi Y, Lee D, Jeon W, Won D. Password-based single-file encryption and secure data deletion for solid-state drive. In: Proceedings of the Eighth International Conference on Ubiquitous Information Management and Communication; 2014 Jan 9-11; Siem Reap, Cambodia. New York, NY: ACM, 2014;1–7.

119. Qin Y, Tong W, Liu J, Zhu Z. SmSD: a smart secure deletion scheme for SSDs. J Converg 2013;4(4);30–5.

120. Gupta A, Singhal M, Gangil A, Mishra A. SDC: secure deletion classification. In: Proceedings of the International Conference on Recent Trends in Information Technology; 2011 Jun 3-5; Chennai, India. Piscataway, NJ: IEEE, 2011;1303–7.

121. Reardon J, Marforio C, Capkun S, Basin D. User-level secure deletion on log-structured file systems. In: Proceedings of the Seventh Symposium on Information, Computer and Communications Security; 2012 May 1-3; Seoul, South Korea. New York, NY: ACM, 2012;63–73.

122. Albano P, Castiglione A, Cattaneo G, De Santis A. A novel anti-forensics technique for the android OS. In: Proceedings of the Sixth International Conference on Broadband and Wireless Computing, Communication and Applications; 2011 Oct 26-28; Barcelona, Spain. Piscataway, NJ: IEEE, 2011;380–5.

123. Pieterse H, Olivier MS. Security steps for smartphone users. In: Proceedings of the Twelfth International Information Security for South Africa Conference; 2013 Aug 14-16; Johannesburg, South Africa. Piscataway, NJ: IEEE, 2013;1–6.

124. Kang S, Park K, Kim J. Cost effective data wiping methods for mobile phone. Multimed Tools Appl 2013;71(2):643–55.

125. Steele RK, Key DS, Abbasi MA, Lutz, BC, inventors. Method and apparatus for sanitizing or modifying flash memory chip data. US patent application 2009/0113113. 2009 Apr 30.

126. Jevans D, Ficcaglia R, Spencer G, Ryan S, inventors. IronKey Inc, assignee. Memory data shredder. US patent application 2007/0300031. 2007 Dec 27.

127. Weng WK, Wu HH, inventors. Skymedi Corp, assignee. Secure erase system for a solid state non-volatile memory device. US patent application 2012/0079289. 2012 Mar 29.

128. Lee J, Yi S, Heo J, Park H, Shin SY, Cho Y. An efficient secure deletion scheme for flash file systems. J Inf Sci Eng 2010;26(1):27–38.

129. Park S, Kim J, Jung Y, Lim D, Seo Y, Cho Y, Yi S, Lee J, Kim S, Oh J, inventors. Electronics and Telecommunications Research Institute, assignee. Flash memory device having secure file deletion function and method for securely deleting flash file. US patent 8,117,377. 2012 Feb 14.

130. Lee B, Son K, Won D, Kim S. Secure data deletion for USB flash memory. Journal of Information Science and Engineering 2011;27(3):933–52.

131. Guyot C, Bandic ZZ, Cassuto Y, Espeseth AM, Sanvido M, inventors. HGST Netherlands BV, assignee. Implementing secure erase for solid state drives. US patent 8,250,380. 2012 Aug 21.

132. Sun K, Choi J, Lee D, Noh SH. Models and design of an adaptive hybrid scheme for secure deletion of data in consumer electronics. IEEE Trans Consumer Electron 2008;54(1):100–4.

133. Subha S. An algorithm for secure deletion in flash memories. In: Proceedings of the Second International Conference on Computer Science and Information Technology; 2009 Aug 8-11; Beijing, China. Piscataway, NJ: IEEE, 2009;260–2.

134. Brož M, Matyáš V. The TrueCrypt on-disk format-an independent view. IEEE Security & Privacy 2014;12(3):74–7.

135. 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. Commun ACM 2009;52(5):91–8.

136. Müller T, Spreitzenbarth M. FROST: Forensic Recovery of Scrambled Telephones. In: Proceedings of the Eleventh International Conference on Applied Cryptography and Network Security; 2013 Jun 25-28; Alberta, Canada. Piscataway, NJ: IEEE, 2013;373–88.

137. AMD. RAIDXpert user manual, 2010; http://www2.ati.com/relnotes/AMD_RAIDXpert_User_v2.1.pdf (accessed October 12, 2015).

138. Intel. RAID 0, 1, 5, 10, matrix RAID, RAID-ready, Intel® Rapid Storage Technology (Intel® RST), 2014; http://www.intel.com/support/chipsets/imsm/sb/CS-009337.htm (accessed October 7, 2015).

139. Vantage Technologies n.d., RAID 5 data recovery FAQ; http://www.vantagetech.com/faq/raid-5-recovery-faq.html (accessed October 7, 2015).

140. Diesburg SM, Meyers C, Stanovich M, Mitchell M, Marshall J, Gould J, Wang AA, Kuenning G. TrueErase: per-file secure deletion for the storage data path. In: Proceedings of the Twenty Eighth Annual Computer Security Applications Conference; 2012 Dec 3-7; Orlando, FL. New York, NY: ACM, 2012;439–48.

141. Bonetti G, Viglione M, Frossi A, Maggi F, Zanero S. Black-box forensic and antiforensic characteristics of solid-state drives. Journal of Computer Virology and Hacking Techniques 2014;10(4):255–71.

142. Linnell TE, inventor. EMC Corp, assignee. Securely erasing flash-based memory. US patent 8,130,554. 2012 Mar 6.

143. Koren R, Leibinger E, Wiesz N, Zilberman E, Tzur O, Aharonoff S, Teicher M, inventors. SanDisk IL Ltd, assignee. Methods of sanitizing a flash-based data storage device. US patent 7,089,350. 2006 Aug 8.

144. Sang W. [RFC 04/10] devfs & mtd: Add MEMERASE ioctl support, 2012; http://lists.infradead.org/pipermail/barebox/2012-October/010713.html (accessed October 7, 2015).

145. Jangir ML. Working with MTD devices, Open Source For U, 2012; http://www.opensourceforu.com/2012/01/working-with-mtd-devices/ (accessed October 7, 2015).

146. Google n.d. Implementing security; https://source.android.com/devices/tech/security/implement.html (accessed October 7, 2015).

147. Leom MD, D'Orazio CJ, Deegan G, Choo KKR. Forensic collection and analysis of thumbnails in Android. In: Proceedings of the Fifth International Symposium on Trust and Security in Cloud Computing; 2015 Aug 20-22; Helsinki, Finland. Piscataway, NJ: IEEE, 2015;1059–66.

148. Grupp LM, Caulfield, AM, Coburn, J, Davis, JD, Swanson S. Beyond the datasheet: using test beds to probe non-volatile memories' dark secrets. In: Proceedings of the IEEE Globecom 2010 Workshop on Application of Communication Theory to Emerging Memory Technologies; 2010 Dec 6-10 Miami, FL. Piscataway, NJ: IEEE, 2010;1930–5.

149. Saxena M, Zhang, Y, Swift, MM, Arpaci-Dusseau, AC, Arpaci-Dusseau RH. Getting real: lessons in transitioning research simulations into hardware systems. In: Proceedings of the Eleventh USENIX Conference on File and Storage Technologies; 2013 Feb 12-15; San Jose, CA. Berkeley, CA: USENIX Association, 2013;215-228.

150. Reddy VK, Rao JE. Survey on security in cloud using homographic and disk encryption methods. International Journal of Computer Sciences and Engineering 2014;2(4):107–12.

151. Larabel M. The performance impact of Linux disk encryption on Ubuntu 14.04 LTS, Phoronix, 2014; http://www.phoronix.com/scan.php?page=article&item=ubuntu_1404_encryption (accessed October 7, 2015).

152. Swanson S, Wei M. Safe: fast, verifiable sanitization for SSDs. San Diego, CA: University of California-San Diego, 2010.

153. Müller T, Latzo T, Freiling FC. Self-encrypting disks pose self-decrypting risks. In: Proceedings of the Twenty Ninth Chaos Communication Congress; 2012 Dec 27-30; Hamburg, Germany. Hamburg, Germany: Chaos Computer Club, 2012;1–10.

154. Trusted Computing Group. TCG storage security subsystem class (SSC): Opal, specification version 2.00, revision 1.0 ed, 2012;

http://www.trustedcomputinggroup.org/resources/storage_work_group_storage_security_subsystem_class_opal (accessed October 7, 2015).

155. Microsoft. Encrypted hard drive, TechNet Library, 2012; http://technet.microsoft.com/en-us/library/hh831627.aspx (accessed October 7, 2015).

156. Rich D. Authentication in transient storage device attachments. Comput 2007;40(4):102–4.

157. Intel. Intel® solid-state drive pro 1500 series (M.2), Product Specification, 2013; http://www.intel.com/content/dam/www/public/us/en/documents/product-specifications/ssd-pro-1500-series-m2-specification.pdf (accessed October 7, 2015).

158. Kingston. Kingston introduces optional TCG opal 1.0 compliant SSD, 2013 Flash Press Release, 2013; http://www.kingston.com/us/company/press/article/6979 (accessed October 7, 2015).

159. TCG n.d., Data production solution, Trusted Computing Group; http://www.trustedcomputinggroup.org/solutions/data_protection (accessed October 7, 2015).

160. Wetzels J. Hidden in snow, revealed in thaw: cold boot attacks revisited, 2014; http://arxiv.org/abs/1408.0725 (accessed October 12, 2015).

161. Rutkowska J. Evil maid goes after TrueCrypt, The Invisible Things Lab's blog, 2009; http://theinvisiblethings.blogspot.com/2009/10/evil-maid-goes-after-truecrypt.html (accessed October 7, 2015).

162. Rutkowska J. Anti evil maid, The Invisible Things Lab's blog, 2011; http://theinvisiblethings.blogspot.com/2011/09/anti-evil-maid.html (accessed October 7, 2015).

163. Tereshkin A. Evil maid goes after PGP whole disk encryption. In: Proceedings of the Third International Conference on Security of Information and Networks; 2010 Sep 7-11; Taganrog, Russia. New York, NY: ACM, 2010.

164. Sophos. SafeGuard device encryption: OPAL support, Sophos Knowledgebase Support, 2014; https://www.sophos.com/en-us/support/knowledgebase/113366.aspx (accessed October 7, 2015).

165. Intel. Data security features in the intel solid - state drive 520 series, Intel SSD 520 Series Technology Brief, 2012; http://www.intel.com/content/dam/www/public/us/en/documents/technology-briefs/ssd-520-aes-tech-brief.pdf (accessed October 7, 2015).

166. Lee S, Kim J. Understanding SSDs with the OpenSSD platform. Seoul, South Korea. Seoul, South Korea: Sungkyunkwan University, 2011.

167. Davis JD, Zhang L. FRP: a nonvolatile memory research platform targeting NAND flash. In: Proceedings of the First Workshop on Integrating Solid-state Memory into the Storage Hierarchy; 2009 Mar 7-11; Washington, D.C. New York, NY: ACM, 2009;1–8.

168. Lee S, Fleming K, Park J, Ha K, Caulfield A, Swanson S, Arvind, Kim J. BlueSSD: an open platform for cross-layer experiments for NAND flash-based SSDs. In: Proceedings of the Fifth Workshop on Architectural Research Prototyping; 2015 Jun 19-23; Saint-Malo, France. New York, NY: ACM, 2015;1–5; http://research.microsoft.com/pubs/198375/WARP2010_BlueSSD.pdf (accessed October 12, 2015).

169. Bunker T, Wei M, Swanson SJ. Ming II: a flexible platform for nand flash-based research. San Diego, CA: Department of Computer Science and Engineering, University of California-San Diego, 2012.

170. Mislan RP, Casey E, Kessler GC. The growing need for on-scene triage of mobile devices. Digital Investigation 2010:6(3-4):112–24.

171. Wakefield J. Devices being remotely wiped in police custody. BBC, 2014; http://www.bbc.com/news/technology-29464889 (accessed October 7, 2015).

172. Barrett D, Yadron D, Wakaba D. Apple and others encrypt phones, fueling government standoff. Wall Street Journal, 2014; http://www.wsj.com/articles/apple-and-others-encrypt-phones-fueling-government-standoff-1416367801 (accessed October 7, 2015).

173. Hattem J. Crypto wars' return to congress. The Hill, 2014; http://thehill.com/policy/cybersecurity/221147-crypto-wars-return-to-congress (accessed October 12, 2015).

174. Timberg C, Miller G. FBI blasts apple, google for locking police out of phones. The Washington Post, 2014; http://www.washingtonpost.com/business/technology/2014/09/25/68c4e08e-4344-11e4-9a15-137aa0153527_story.html (accessed October 12, 2015).

175. Frizell S. Yahoo is making it harder for the NSA to read your emails. TIME, 2014; http://time.com/3092881/email-encryption/ (accessed October 12, 2015).

176. Gustin S. NSA spying scandal could cost U.S. tech giants billions. TIME, 2013; http://business.time.com/2013/12/10/nsa-spying-scandal-could-cost-u-s-tech-giants-billions/ (accessed October 12, 2015).

Additional Information — Reprints Not Available From Author:

Kim-Kwang Raymond Choo, Ph.D.

University of Texas at San Antonio

Department of Information Systems and Cyber Security

One UTSA Circle

San Antonio, TX 78249-0631

E-mail: Raymond.Choo@fulbrightmail.org

Appendix

TABLE 1—*Security features in the existing remote wiping literature.*

Feature	Angelo et al. (48)	Brown et al. (49)	Walker & Fyke (50)	Kennedy (51)	Hasbe (52)	Sennett & Daly (53)	Onyon et al. (54)	Gajdos & Kretz (55)	Yu et al. (56)	Park et al. (57)	Kuppusamy et al. (58)	Joe & Lee (59)	Adusumalli (60)
Authenticate reporter	Y	Y	Y	Y	Y	Y	N	Y	Y	Y	Y	Y	N
Authenticate origin of wipe command	Y	Y	N	N	N	Y	Y	Y	Y	N	Y	N	N
Secure wipe command	N	Y	N	N	N	N	N	N	N	N	N	N	N
Transmission channel	Internet	Internet	Cellular	Cellular	Internet	Cellular	Internet	Internet	Cellular (Emergency call)	Cellular (SMS)	Cellular (SMS)	Internet	Cellular (SMS)
Secure delete	N	Y	N	N	N	Y	Y	Y	Y	N	N	N	N
Ensure wiping operation is completed	N	Y	N	N	N	N	N	N	Y	N	N	N	N
Acknowledge source that wipe is completed	N	N	N	Y	N	N	N	Y	N	N	N	Y	Y
Replay attack mitigation	Y	N	N	N	N	N	N	N	N	Y	N	Y	N

TABLE 2—*Authentication methods in the existing remote wiping literature.*

	Biometrics	Certificate	Password	Secret question	Username and password	Identification information
Angelo et al. (48)	●			●		
Brown et al. (49)						
Walker & Fyke (50)			●			
Kenney (51)				●	●	
Hasebe (52)			●			
Sennett & Daly (53)					●	
Onyon et al. (54)						
Gajdos & Kretz (55)					●	
Yu et al. (56)			●			●
Park et al. and Kuppusamy et al. (57, 58)			●			
Joe & Lee (59)		●			●	
Adusumalli (60)			●			

TABLE 3—*Summary of methods used to authenticate origin of wipe command.*

	Public key cryptography	Shared code / password	Incoming number
Angelo et al. (48)	●		
Brown et al. (49)	●	●	
Sennett & Daly (53)		●	
Onyon et al. (54)	●	●	
Gajdos & Kretz (55)	●	●	
Yu et al. (56)		●	
Kuppusamy et al. (58)			●

TABLE 4—*Deletion method of factory reset in AOSP (adapted from Simon and Anderson (97)).*

		Android version				
Code	Partition	Froyo	GB	4.0.x (ICS)	JB	KK
Android	media	*format()*			*ioctl(BLKDISCARD)*	
	External SD	*None*				
Recovery	data	*ioctl(MEMERASE)*	*ioctl(BLKDISCARD)*	*ioctl(BLKSEDISCARD)*		

TABLE 5—*Device specification.*

Device	**Motorola Moto G**	**Samsung Nexus S**	**LG Nexus 4**
Model	XT1033	GT-I9020T	LGE960
Android OS	4.4.2	4.1.2	5.1
Android Build	KXB20.25-1.31	JZO54K	LMY47O
Linux kernel	3.4.0	3.0.31	3.4.0
Storage	8 GB	16 GB	8GB
RAM	1 GB	512 MB	2 GB

TABLE 6—*Comparative summary of existing secure deletion techniques*.

Technique	**Advantages**	**Disadvantages**	**Evaluation criteria / claimed features (for approaches without an experiment)**	**Research set-up**
User-space				
Spreitzenbarth & Holz (116)	• Simple to implement. • No OS modification required. • Cross-platform.	• Does not provide wear-levelling. • Limited data type support.	• Data recoverability	• Experiment using Nokia E90 (Symbian 9.2)/S60 platform
Reardon et al. (121)	• Data type agnostic. • Provides wear-levelling.	• Excessive writes or wear on storage. • Slow	• Effects of different parameter on deletion latency and lifetime. • Battery consumption.	• HTC Nexus One
Albano et al. (122)	• Simple operation. • Data type agnostic, • No OS modification required.	• Does not provide wear-levelling. • Works on Android or any Linux-based OS only. • Requires root and Busybox installed on Android. • Slow.	• Data recoverability	• HTC Nexus One (MIUI ROM based on Android v2.3.4)
Kang et al. (124)	• Efficiency as only parts of data needs to be overwritten.	• Does not provide wear-levelling. • Limited data type support.	• Data recoverability • Deletion time	• Samsung Galaxy S3
Steele et al. (125)	• Wipe several USB flash drives simultaneously. • Questionable motivation behind the proposal.	• Does not address wear-levelling.	NA	NA
Jevans et al. (126)	• Resume interrupted wiping.	• Limited wear levelling.	NA	NA
File system				
Weng & Wu (127)	• Only small data (key) needs to be deleted.	• No mention of secure deletion for keys.	NA	NA
Lee et al. (128) and Park et al. (118)	• Encryption keys are arranged closely for faster delete operation.	• Excessive wear on flash storage. • Conceptual (yet to be implemented / evaluated).	• Amortized number of block erase	• No experiment conducted
Lee et al. (130)	• Incorporate US government standards • More efficient than (89) and introduce less wear on flash storage.	• Conceptual (yet to be implemented / evaluated).	• Amortized number of block erase	• No experiment conducted
Guyot et al. (131)	• Method to remove duplicates of deleted data provided.	• Latency of garbage collection operation.	NA	NA
Reardon et al. (103)	• Can be modified into full disk	• Designed for UBIFS, a file	• Execution time for various file	• HTC Nexus One (Linux

Technique	Advantages	Disadvantages	Evaluation criteria / claimed features (for approaches without an experiment)	Research set-up
	encryption for confidentiality.	system not found in Android but supported by the Linux kernel. • Depends on now defunct MTD	system functionality (e.g. mount/unmount, read/write) • Power consumption	v2.6.35.7)
Sun et al. (132)	• Hybrid scheme which chooses faster method by evaluating the nature of data location.	• Latency during cost calculation.	• Time taken to complete various workloads.	• Embedded board 400MHz Intel XScale CPU, 64MB SDRAM, 64MB Samsung NAND flash memory.
Choi et al. (118)	• Compatibility with TRIM • Verify the data has been overwritten	• Additional operations increase deletion time.	NA	NA
Physical				
Wei et al. (107)	• Almost native performance	• May result in writing error. • Possible violation of flash storage's specification.	• Write latency • Secure deletion latency on different flash storage and applications	• Custom-built FPGA-based flash testing hardware on 16 chips spanning 6 manufacturers, 5 technology nodes covering both MLC and SLC chips
Qin et al. (119)	• Use RAID-5 to mitigate negative effect of reprogram.	• Require multiple drives for RAID; thus not cost-effective / suitable for general consumer usage.	• Read/write response time	• Simulator SSDsim
Subha (133)	• Least data to be affected thus very fast deletion time.	• Access to ECC is questionable.	• Deletion time	• C program running on Linux file system to simulate ECC, with a Rich Text File (RTF) as input data.
Diesburg et al. (140)	• Most comprehensive approach	• Complex to implement	6. Data recoverability • Disk performance	SanDisk's DiskOnChip with Inverse NAND • File Translation Layer (INFTL) kernel module on Linux 2.6.25.6
Linnell (142)	• FTL compatibility.	• Slower than simple zeroes overwriting in some cases.	NA	NA
Koren et al. (143)	• Erase operation is independent from host device (OS and motherboard). • Resume interrupted wiping.	• Does not provide wear-levelling	NA	NA

TABLE 7—*Key differences between hardware- and software-based implementations (Adapted from Choi et al. (118)).*

	Hardware-based approach	**Software-based approach**
Performance	Generally higher	Slower
Security issue	‘Hot plug’ attack	‘Cold boot’ and related attacks
Verifiability	Difficult	Possible
Ease of implementation	Hard	Easy
Cost	High	Low

TABLE 8—*Remote wiping and secure flash storage deletion publications by research focus.*

Publications	Years	Research focus
Angelo et al. (48), Kenney (51), Hasebe (52)	1999-2005	Remote wiping
Onyon et al. (54), Gajdos & Kretz (55)	2006-2010	
Brown et al. (49), Park et al. (57), Joe & Lee (59)	2011	
Kuppusamy et al. (58)	2012	
Walker & Fyke (50)	2013	
Yu et al. (56), Adusumalli (60)	2014	
Sun et al. (132), Jevans et al. (126), Koren et al. (143)	2006-2008	Secure flash storage deletion
Spreitzenbarth & Holz (116), Steele et al. (125), Lee et al. (128), Subha (133)	2009-2010	
Wei et al. (107), Albano et al. (122), Lee et al. (130)	2011	
Linnell (142), Reardon et al. (121), Weng & Wu (127), Park et al. (129), Guyot et al. (131)	2012	
Reardon et al. (103), Qin et al. (119), Kang et al. (124)	2013	
Choi et al. (118)	2014	

TABLE 9—*Recovered data by Cellebrite.*

Cellebrite	Moto G		Nexus S		Nexus 4	
Data types	Pre-wipe	Post-wipe	Pre-wipe	Post-wipe	Pre-wipe	Post-wipe
Call log	12	0	0	0	10	0
Chats	0	0	0	0	0	0
Google Talk	1	0	0	0	0	0
Hangouts	2	0	0	0	2	0
Kik	2	0	0	0	1	1
Contacts	31	0	0	0	31	0
Cookies	516	0	0	0	20	0
Emails	45	0	0	0	2	0
Installed applications	41	0	0	0	49	0
Passwords	6	0	0	0	6	0
Powering Events	1	0	0	0	3	0
Searched items	1	0	0	0		0
SMS	1	0	0	2	3	0
Timeline	718	0	0	2	172	0
User accounts	9	0	0	0	9	0
Web bookmarks	50	0	0	0	50	0
Web history	57	0	0	0	40	0
Wireless networks	1	0	0	0	1	0
Data Files	0	0	0	0		0
Applications	434	3	0	0	381	0
Audio	31	0	32	0	32	0
Databases	226	1	0	0	178	0
Documents	180	0	213	0	181	0
Images	1910	0	756	0	1363	0
Text	362	2	1	0	289	0
Videos	35	0	35	0	35	0

Carved Images	1375	2	6238	2759	966	0
Activity analytics	58	0	0	0	36	0
Analytics emails	0	0	0	0	0	0
user email	3	0	0	0	4	0
Uncategorized	19	0	0	0	0	0
Analytics phones	32	0	0	0	30	0
Skype	1	0	0	0	1	0

TABLE 10—*Recovered data by IEF.*

IEF	Moto G		Nexus S		Nexus 4	
Data type	Pre-wipe	Post-wipe	Pre-wipe	Post-wipe	Pre-wipe	Post-wipe
Google Analytics First Visit Cookies	23	0	0	0	0	0
Google Analytics Referral Cookies	23	0	0	0	0	0
Google Analytics Sessions Cookies	23	0	0	0	0	0
Google Analytics URLs	33	0	0	0	0	0
Google Searches	1	0	0	0	0	0
Social Media URLs	107	0	0	0	2	0
Chat	0	0	0	0	0	0
Skype accounts	1	0	0	0	1	0
Skype Contacts	2	0	0	0	2	0
Skype IP Addresses	0	0	0	0	2	0
Chat	0	0	0	0	0	0
Google Drive	2	0	0	0	4	0
Documents	0	0	0	0	0	0
Excel	36	0	36	7	36	0
PDF	38	0	30	30	36	0
PowerPoint	60	0	60	30	60	0
Text	77	0	0	0	22	0
Word	60	0	30	30	60	0
Media	0	0	0	0	0	0
Carved video	52	0	41	38	43	0
Pictures	17946	2	8202	7682	14321	0
Videos	35	0	35	0	35	0
Web related	0	0	0	0	0	0
Browser Activity	368	0	0	0	20	0
Chrome bookmarks	0	0	0	0	50	0
Chrome cookies	516	0	0	0	20	0
Chrome favicons	63	0	0	0	39	0

Chrome logins	2	0	0	0	2	0
Chrome top sites	1	0	0	0	40	0
Chrome web history	55	0	0	0	40	0
Chrome web visits	108	0	0	0	1	0
Chrome/360 Safe Browser/Opera Carved web history	171	0	0	0	0	0
Firefox web history	10	0	0	0	13	0
Google Analytics First Visit Cookies	37	0	0	0	0	0
Google Analytics Referral Cookies	25	0	0	0	0	0
Google Analytics Sessions Cookies	41	0	0	0	0	0
Google Analytics URLs	33	0	0	0	0	0
Google maps	18	0	0	0	10	0
Google maps tiles	22	0	0	0	0	0
Safari history	0	0	0	0	9	0

TABLE 11—*Recovered data by PhotoRec.*

PhotoRec	Moto G		Nexus S		Nexus 4	
Data type	Pre-wipe	Post-wipe	Pre-wipe	Post-wipe	Pre-wipe	Post-wipe
docx	47	0	0	0	53	0
http cache	5	0	0	0	31	0
jar	13	0	0	0	9	0
java	17	0	39	39	95	0
jpg	538	0	460	460	459	0
mp3	30	0	373	373	34	0
ogg	3	0	2	2	4	0
pdf	15	0	6	6	18	0
png	101	0	49	49	372	0
pptx	20	0	1	1	23	0
sqlite	9339	7	1	1	178	0
txt	500	0	811	811	2577	2
xlsx	7	0	1	1	4	0
zip	15	0	3	3	4	0

TABLE 12—*Recovered thumbnail by Scalpel.*

Scalpel	Moto G		Nexus S		Nexus 4	
	Pre-wipe	Post-wipe	Pre-wipe	Post-wipe	Pre-wipe	Post-wipe
No. of thumbnails recovered	885	0	564	564	664	0

FIG. 1—General overview of the remote wiping process

Legend: (a) Authenticate reporter

(b) Authenticate origin of wipe command

(c) Secure wipe command

(d) Transmission channel

(e) Secure delete

(f) Ensure wiping operation is completed

(g) Acknowledge source that wipe is completed

(h) Replay attack mitigation

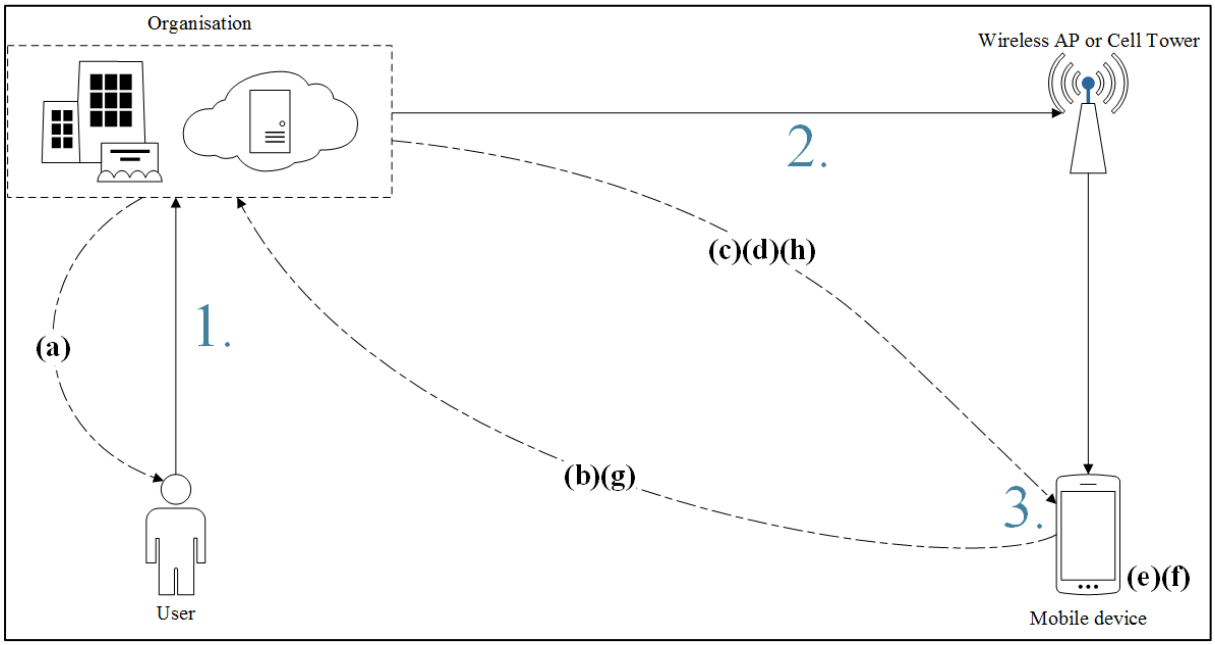

FIG. 2—Push messaging architecture. Adapted from (67, 70)

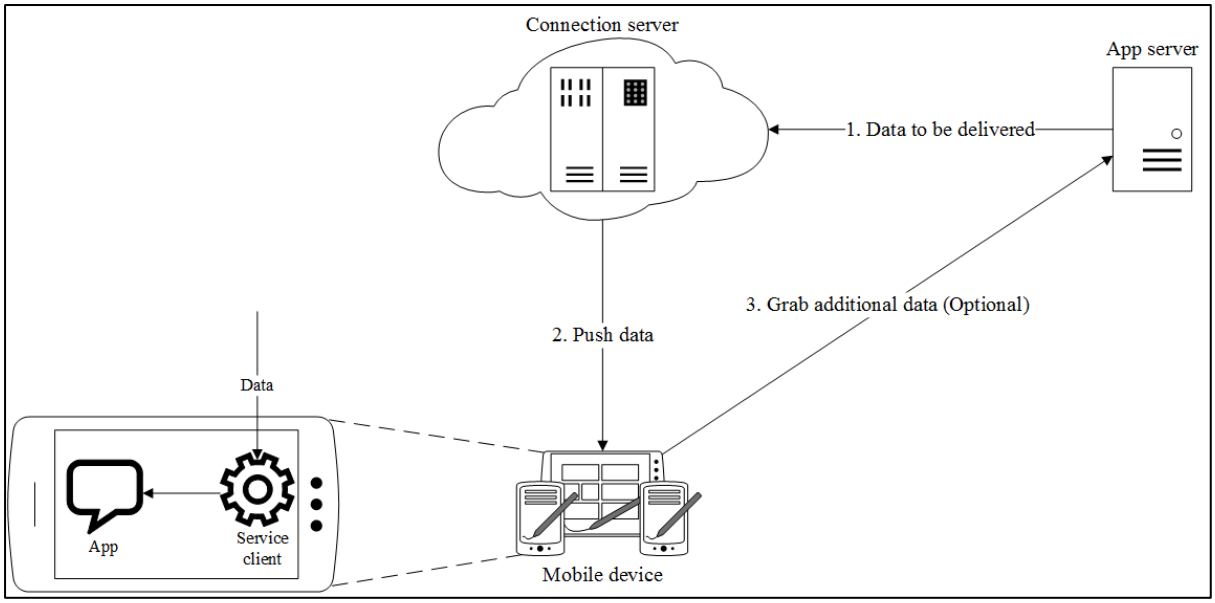

FIG. 3—Data access layers in flash storage. (Adapted from 43, 98)

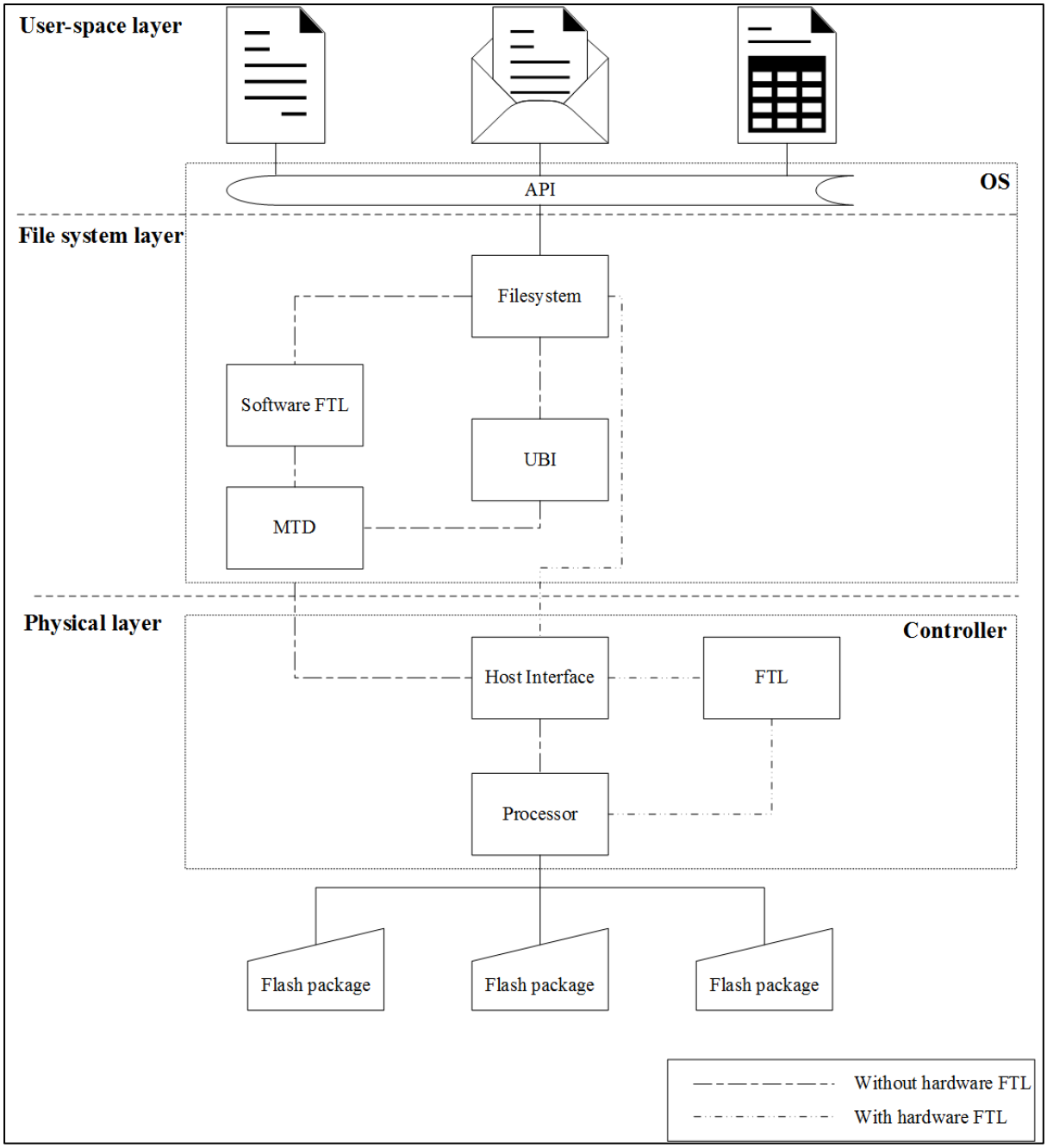

FIG. 4—Decryption process in Truecrypt and Choi et al. (118).