Adding Secure Deletion to an Encrypted File System on Android Smartphones

Alexandre Melo Braga, Alfredo H. Gallinucci Colito
Centro de Pesquisa e Desenvolvimento em Telecomunicações (Fundação CPqD)
Campinas, São Paulo, Brazil
{ambraga,acolito}@cpqd.com.br

***Abstract*—Nowadays, mobile devices are powerful enough to accomplish most of the tasks previously accomplished only by personal computers; that includes file management. However, on many devices the file deletion operation misleads the user into thinking that the file has been permanently removed, when that is usually not the case. Also, with the increasing use of encryption, attackers have been directed to weaker targets. One of them is the recovery of supposedly deleted data from flash memories. This paper describes a way to integrate secure deletion technologies in an encrypted file system in Android smartphones.**

Keywords-secure delete; secure storage; encrypted file system; flash memory; mobile devices; Android.

I. INTRODUCTION

Nowadays, many users keep their sensitive data on mobile devices. However, mobile devices are vulnerable to data leakage. As the amount of digital data grows, so does the theft of sensitive data through loss of device, exploitation of vulnerabilities or misplaced security controls. Sensitive data may also be leaked accidentally due to improper disposal or resale of devices.

With the increasing use of encryption systems, an attacker wishing to gain access to sensitive data is directed to weaker targets. One possible attack is the recovery of supposedly erased data from internal storage, possibly a flash memory card. To protect the secrecy of data during its entire lifetime, encrypted file systems must provide not only ways to securely store, but also reliably delete data, in such a way that recovering them from physical medium is almost impossible.

The new generations of mobile devices are powerful enough to accomplish most of the tasks previously accomplished only by personal computers. That includes file management operations (e.g., create, read, update, and delete). Also, today's devices possess operating systems that are hardware-agnostic by design and abstract from ordinary users all hardware details, such as writing procedures for flash memory cards.

Additionally, it is a real threat the misuse by intelligence agencies of data destruction standards as well as embedded technologies, which can suffer from backdoors or inaccurate implementations, in an attempt to facilitate unauthorized access to supposedly deleted data. In fact, there is a need for practical security technologies that work at the operating system level, under the control of the user. This technology has to be easy to use in everyday activities and easily integrated into mobile devices with minimal maintenance and installation costs.

This paper describes a way to integrate secure deletion technologies to an encrypted file system in Android smartphones. This work is part of an effort to build security technologies into an integrated framework for mobile device security [1][2].

The remaining parts of the text are organized as follows. Section II offers background information. Section III discusses related work. Section IV details the proposed integration of encrypted file systems and secure deletion functions. Section V presents a performance evaluation for the secure deletion function. Section VI discusses improvements on the proposed approach. Section VII concludes this text.

II. BACKGROUND

Traditionally, the importance of secure deletion is well understood by almost everyone and several real-world examples can be given on the subject: sensitive mail is shredded; published government information is selectively redacted; access to top secret documents ensures all copies can be destroyed; and blackboards at meeting rooms are erased after sensitive appointments.

In mobile devices, that metaphor is not easily implemented. All modern file systems allow users to "delete" their files. However, on many devices the remove-file command misleads the user into thinking that her file has been permanently removed, when that is not the case. File deletion is usually implemented by unlinking files, which only changes file system metadata to indicate that the file is "deleted"; while the file's full contents remain available in physical medium. This simple procedure is called logical or ordinary deletion.

Unfortunately, despite the fact that deleted data are not actually destroyed in the device, logical deletion has the additional drawback that ordinary users are generally unable to completely remove her files. On the other hand, advanced users or adversaries can easily recover logically deleted files.

Deleting a file from a storage medium serves two purposes: (i) it reclaims storage to operating system and (ii) ensures that any sensitive information contained in the file becomes inaccessible. The second purpose requires that files are securely deleted.

Secure data deletion can be defined as the task of deleting data from a physical medium so that the data is

 ISBN: 978-1-61208-376-6

irrecoverable. That means its content does not persist on the storage medium after the secure deletion operation.

Secure deletion enables users to protect the confidentiality of their data if their device is logically compromised (e.g., hacked) or stolen. Until recently, the only user-level deletion solution available for mobile devices was the factory reset, which deletes all user data on the device by returning it to its initial state. However, the assurance or security of such a deletion cannot be taken for granted, as it is highly dependent on device's manufacturer. Also, it is inappropriate for users who wish to selectively delete data, such as some files, but still retain their address books, emails and installed applications.

Older technologies [14] claim to securely delete files by overwriting them with random data. However, due the nature of log-structured file systems used by most flash cards, this solution is no more effective than logically deleting the file, since the new copy invalidates the old one but does not physically overwrite it. Old secure deletion approaches that work at the granularity of a file are inadequate for mobile devices with flash memory cards.

Today, secure deletion is not only useful before discarding a device. On modern mobile devices, sensitive data can be compromised at unexpected times by adversaries capable of obtaining unauthorized access to it. Therefore, sensitive data should be securely deleted in a timely fashion.

Secure deletion approaches that target sensitive files, in the few cases where it is appropriate, must also address usability concerns. A user should be able to reliably mark their data as sensitive and subject to secure deletion. That is exactly the case when a file is securely removed from an encrypted file system.

On the other hand, approaches that securely delete all logically deleted data, while less efficient, suffer no false negatives. That is the case for purging techniques.

III. RELATED WORK

This section briefly describes related work on the subjects of secure deletion and encrypted file systems on mobile devices, particularly Android.

The use of cryptography as a mechanism to securely delete files was first discussed by Boneh and Lipton [6]. Their paper presented a system which enables a user to remove a file from both file system and backup tapes on which the file is stored, just by forgetting the key used to encrypt the file.

Gutman [14] covered methods available to recover erased data and presented actual solutions to make the recovery from magnetic media significantly more difficult by an adversary. In fact, the paper covered only magnetic media and, to a lesser extent, RAM. Flash memory barely existed at the time it was written, so it was not considered by him.

Kyoungmoon et al. [12] proposed an efficient secure deletion scheme for flash memory storage. This solution resides inside the operating system and close to the memory card controller.

Diesburg and Wang [16] presented a survey summarizing and comparing existing methods of providing confidential storage and deletion of data in personal computing environments, including flash memory issues.

Wang et al. [19] present a FUSE (File-system in USErspace) encryption file system to protect both removable and persistent storage on devices running the Android platform. They concluded that the encryption engine was easily portable to any Android device and the overhead due to encryption is an acceptable trade-off for achieving the confidentiality requirement.

Reardon et al. [7]-[10] have shown plenty of results concerning both encrypted file system and secure deletion. First, Reardon et al. [11] proposes the Data Node Encrypted File System (DNEFS), which uses on-the-fly encryption and decryption of file system data nodes to efficiently and securely delete data on flash memory systems. DNEFS is a generic modification of existing flash file systems or controllers that enables secure data deletion. Their implementation extended a Linux implementation and was integrated in Android operating system, running on a Google Nexus One smartphone.

Reardon et al. [7] also propose user-level solutions for secure deletion in log-structured file systems: purging, which provides guaranteed time-bounded deletion of all data previously marked to be deleted, and ballooning, which continuously reduces the expected time that any piece of deleted data remains on the medium. The solutions empower users to ensure the secure deletion of their data without relying on the manufacturer to provide this functionality. These solutions were implemented on an Android smartphone (Nexus One) and experiments have shown that they neither prohibitively reduce the longevity of flash memory nor noticeably reduce device's battery lifetime.

In two recent papers, Reardon et al. [8][9] study the issue of secure deletion in details. First [9], they identify ways to classify different approaches to securely deleting data. They also describe adversaries that differ in their capabilities, show how secure deletion approaches can be integrated into systems at different interface layers. Second [8], they survey the related work in detail and organize existing approaches in terms of their interfaces to physical media. They further present taxonomy of adversaries differing in their capabilities as well as systematization for the characteristics of secure deletion approaches.

More recently, Reardon et al. [10] presented a general approach to the design and analysis of secure deletion for persistent storage that relies on encryption and key wrapping.

Finally, Skillen and Mannan [4] designed and implemented a system called Mobiflage that enables plausibly deniable encryption (PDE) on mobile devices by hiding encrypted volumes within random data on a device's external storage. They also provide [3] two different implementations for the Android OS to assess the feasibility

 ISBN: 978-1-61208-376-6

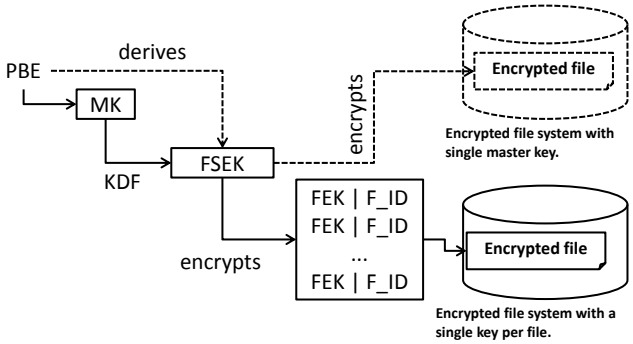

Figure 1. Extending an encrypted file system for secure deletion.

and performance of Mobiflage: One for removable SD cards and other for internal partition for both apps and user accessible data.

IV. DESCRIPTION OF PROPOSED SOLUTION

The rationale behind the proposed solution is the actual possibility of performing secure deletion of files from ordinary Android applications, in user mode, without administrative privileges or operating system customization. The solution handles two cases according to the place where the file already deleted or about to be deleted is stored:

1) The file is already kept by encrypted file system;
2) A file or bunch of files was logically deleted by the operating system and their locations are unknown.

A. Secure Deletion of Encrypted Files

The simplest way to fulfill the task of securely delete a file from an encrypted file system is to simply lose the encryption key of that file and then logically remove the file. This method does not need memory cleaning (purging) and is very fast. A prototype was built upon an Android port for the EncFS encrypted file system [18][19]. To accomplish this task, the way EncFS manages cryptographic keys had to be modified. EncFS encrypts all files with a single master key derived from a password based encryption (PBE) function. It is seams quite obvious that it is not feasible to change a master key and encrypt the whole file system every time a single file is deleted. On the other hand, if each file were encrypted with its own key, then that key could be easily thrown away, turning the file irrecoverable. The modification to EncFS consists in the following steps:

a) Use PBE to derive a master key MK;
b) Use a key derivation function (KDF) to derive a file system encryption key FSEK from MK;
c) Use an ordinary key generation function (e.g., PRNG) to generate a file encryption key FEK;
d) Encrypt files along with their names using FEK and encrypts FEK with FSEK and random IV.
e) Keep a mapping mechanism from FEK and IV to encrypted file (FEK||IV → file).

A simple way to keep that mapping is to have a table file stored in user space as application's data. Care must be taken to avoid accidentally or purposely remove that file when cleaning device's user space. In Android devices, this can be done by rewriting the default activity responsible for deleting application's data. An application-specific delete activity would provide a selective deletion of application's data or deny any deletion at all. The removal from table of the FEK and IV makes a file irrecoverable. The ordinary delete operation then return storage space of that file to operating system. Figure 1 depicts the solution.

Another way to keep track of keys and files is to store the pair {FEK,IV} inside the encrypted name of the encrypted file. In this situation, a file has to be renamed before removed from the encrypted file system. The rename operation destroys the FEK and makes file irrecoverable. The ordinary delete operation then return storage space to operating system.

It is interesting to note that the proposed solution contributes to solve some known security issues of EncFS [13][17]. By using distinct keys for every file, a Chosen Ciphertext Attack (CCA) against the master key is inhibited. Also, it reduces the impact of IV reuse across encrypted files. Finally, it eliminates the watermarking vulnerability, because a single file imported twice to EncFS will be encrypted with two distinct keys and IVs.

Finally, the key derivation function is based upon PBKDF2 standard [5], keys and IVs are both 256 bits, and the table for mapping the pair {key,IVs} to files is kept by an SQLite scheme accessible only by the application.

B. Secure Deletion of Ordinary Files

In this context, a bunch of files were logically deleted by the operating system for the benefit of the user, but they left sensitive garbage in the memory. Traditional solutions of purging memory cells occupied by those files are innocuous, because there is no way to know, from user's point of view, where purging data will be written.

An instance of this situation occurs when a temporary file is left behind by an application and manually deleted. This temporary file may be a decrypted copy of an encrypted file kept by the encrypted file system. Temporary unencrypted copies of files are necessary in order to allow other applications handle specific file types, e.g., images, documents, and spreadsheets.

Whether temporary files will or will not be imported back to the encrypted file system, they have to be securely removed anyway. A premise is that the files to be removed are not in use by any application. The secure deletion occurs in three steps:

1) Logically remove targeted files with ordinary deletion;
2) Write a temporary file of randomized content that occupies all memory's free space;
3) When there is no free space anymore, logically deletes that random file. That action purges all free memory in a way that no sensitive data is left behind.

The final result of this procedure is a flash storage free of sensitive garbage. Steps two and three can be encapsulated as a single function, called memory purging, and performed by an autonomous application. That application would be activated by the user whenever she needs to clean memory from sensitive garbage. The proposed solution adopted this implementation.

 ISBN: 978-1-61208-376-6

Unfortunately, this procedure has two drawbacks. First, it takes time proportional to the size of the free space to be cleaned and the speed of memory writes. Second, this procedure, in the long term, if used with high frequency, have the potential to shorten the lifetime of flash memories.

In order to minimize the negative impact over memory life and avoid excessive delays during operation, steps two and three from above should not be carried out for every single file deleted from the system.

C. Limitations of the solution

The protection of cryptographic keys is of major importance. In spite of being stored encrypted, decrypted just before being used, and then released, the protection of cryptographic keys relies on Android security and the application confinement provided by that operating system.

The proposed solution for memory purging is supposed to work in user-mode, as an ordinary mobile app, without administrative access, with no need for operating system modification, and using COTS devices. These decisions have consequences for security.

First of all, the solution is highly dependent on the way flash-based file systems and controllers behave. Briefly speaking, when the flash storage is updated, the file system writes a new copy of the changed data to a fresh memory block, remaps file pointers, and then erases the old memory blocks, if possible, but not certainly. This constrained design actually enables the alternatives discussed in Section VI.

A second issue is that the solution is not specifically concerned about the type of physical memory (e.g., internal, external SD, NAND, and NOR) as long as it behaves like a flash-based file system. The consequence is that only software-based attacks are considered and physical attacks are out of scope.

Finally, the use of random files is not supposed to have any effect on the purging assurance, but provides a kind of low-cost camouflage for cryptographic material (e.g., keys or parameters) accidentally stored on persistent media. An entropy analysis would not be able to easily distinguish specific random data as potential security material, because huge amounts of space would look random. Of course, this software-based camouflage cannot be the only way to prevent such attacks, but it adds to a defense in depth approach to security at almost no cost.

V. Performance Evaluation of Secure Deletion

Table I shows performance measurements for the secure deletion of ordinary files by purging. The measurements were taken on two smartphones: (i) LG Prada p940h, with 4 GB of internal storage available and Android 2.3.7; and (ii) Motorola Atrix with 16GB (only 11 GB available to final user) of internal storage and Android 2.3.6. File recovery was performed by PhotoRec recovery tool [15]. Random files created for purging had size of at most 2 GB.

Tests were performed over internal memory in three conditions: memory almost free (few files), memory half occupied (many files), and memory free (no files at all). The test procedure consisted of the following steps: (a) creation of ordinary content; (b) logical deletion of that content; (c) execution of secure deletion procedure; and (d) attempting of content recovery. Tests have shown that secure deletion time is proportional to memory size and quite similar to recovery time, as was expected. LG Prada was cleaned at a rate of one Gigabyte per minute (1 GB/min). Motorola Atrix was cleaned at a rate of half Gigabyte per minute (0.5 GB/min). Additionally, a test over a class C SD card of 4 GB was carried out at 0.25 GB/min. In all cases, PhotoRec was unable to recover secure deleted files.

TABLE I. TESTING SECURE DELETION.

LG Prada p940h	Few files	Many files	No files
Free before purging	~3.9 GB	2.21 GB	3.98 GB
Purging time	4min19s	2min37s	4min24s

Motorola Atrix	Few files	Many files	No files
Free before purging	~10 GB	5,2 GB	10,59 GB
Purging time	18min51s	10min53s	19min22s

VI. Improvements under Development

The solution for memory purging is the simplest implementation of a general policy for purging flash memories. In fact, a general solution has to offer different trade-offs among security requirements, memory life, and system responsiveness. The authors have identified three points for customization:

1. The period of execution for the purging procedure;
2. The size and quantity of random files;
3. The frequency of files creation/deletion.

By the time of writing, different trade-offs among the three customization points previously identified were being implemented and evaluated. In all of them, the random file created in order to clean memory space is called bubble, after the metaphor of soap cleaning bubbles over a dirty surface. These alternatives are discussed in next paragraphs.

A. Static single bubble

The solution described in this text implements the idea of a single static bubble that increases in size until it reaches the limit of free space, and then bursts. This solution is adequate for the cases when memory has to be cleaned in the shortest period of time, with no interruption. A disadvantage is that other concurrent application can starve out of memory. This solution is adequate when nothing else is happening, but the purging.

B. Moving or sliding (single) bubble

In this alternative, a single bubble periodically moves itself or slides from one place to another. The moving bubble has size of a fraction of free space. For example, if bubble size is 1/n of free space, the moving bubble covers all free memory after n moves, considering the amount of free space does not change. A move is simply the rewriting of the bubble file, since flash memories will perform a rewrite in a different place.

 ISBN: 978-1-61208-376-6

In a period of time equals to T*(n/2), where T is the time between moves, the chance of finding sensitive garbage in memory is 50%. This solution is adequate when memory has a low to moderate usage by concurrent applications. This solution preserves system responsiveness (usability) but diminishes security.

C. Moving or sliding (multiple) bubbles

This alternative uses more than one bubble instead of a single one. The size and amount of bubbles are fixed. For instance, if bubble size is 1/n of free space, two moving bubble covers all free memory after n/2 moves each. The advantage of this method is to potentially accelerate memory coverage, reducing opportunity for memory compromising.

In the example, two bubbles of size 1/n each can move at every T/2 period, and then concluding in T*n. Alternatively, they can move at period T and terminate in 2*T*n, and so on. This solution is adequate when memory has a moderate usage by concurrent applications. This solution is probabilistic in the sense that as smaller the duration of T and greater the size of bubbles, greater the chance of successfully clean all memory.

D. Sparkling bubbles

This solution varies the size and amount of bubbles. The idea is to create a bunch of mini bubbles that are sparkled over free memory. Bubbles are created and instantly removed at period T, which can be constant or random between zero and T. The sparking of bubbles stops when the sum of sizes for all created bubbles surpasses free space. Bubble size can be small enough to not affect other applications.

This solution is adequate when memory has a moderate to high usage by concurrent applications. This solution is probabilistic in the sense that as smaller the duration of T, greater the chance of successfully clean the whole memory.

VII. CONCLUDING REMARKS

This paper discussed the implementation of two user-level approaches to perform secure deletion of files. One works on secure deletion of encrypted files and the other handles de deletion assurance of ordinary (unencrypted) files. Secure deletion of encrypted files was fully integrated to an encrypted file system and is transparent to the user. Secure deletion of ordinary files was fulfilled by an autonomous application activated under the discretion of the user. Preliminary performance measurements have shown that the approach is feasible and offers a trade-off between time and deletion assurance. Further tests have to be performed to fine-tune the solution in order to preserve system responsiveness. Also, a deep security assessment has to be performed in order to give the actual extend of the security provided by the proposed solution.

ACKNOWLEDGMENT

The authors acknowledge the financial support given to this work, under the project "Security Technologies for Mobile Environments – TSAM", granted by the Fund for Technological Development of Telecommunications – FUNTTEL – of the Brazilian Ministry of Communications, through Agreement Nr. 01.11. 0028.00 with the Financier of Studies and Projects - FINEP / MCTI.

REFERENCES

[1] A. M. Braga, E. Nascimento, and L. Palma, “Presenting the Brazilian Project TSAM – Security Technologies for Mobile Environments”, in proceeding of the 4th International Conference in Security and Privacy in Mobile Information and Communication Systems (MobiSec 2012), LNICST volume 107, 2012, pp. 53-54.

[2] A. M. Braga, “Integrated Technologies for Communication Security on Mobile Devices”, The Third International Conference on Mobile Services, Resources, and Users (Mobility’13), 2013, pp. 47-51.

[3] A. Skillen and M. Mannan, “Mobiflage: Deniable Storage Encryption for Mobile Devices”, IEEE Transactions on Dependable and Secure Computing, vol.11, no.3, May-June 2014, pp.224,237.

[4] A. Skillen and M. Mannan, “On Implementing Deniable Storage Encryption for Mobile Devices”, in 20th Annual Network & Distributed System Security Symposium, February 2013, pp. 24-27.

[5] B. Kaliski, RFC 2898, PKCS #5: Password-Based Cryptography Specification Version 2.0. Retrieved [July 2014] from http://tools.ietf.org/html/rfc2898.

[6] D. Boneh and R. J. Lipton, “A Revocable Backup System”, in USENIX Security, 1996, pp. 91-96.

[7] J. Reardon, C. Marforio, S. Capkun, and D. Basin, “User-level secure deletion on log-structured file systems”, in Proceedings of the 7th ACM Symposium on Information, Computer and Communications Security, 2012, pp. 63-64.

[8] J. Reardon, D. Basin, and S. Capkun, “Sok: Secure data deletion”, in IEEE Symposium on Security and Privacy, 2013, pp. 301-315.

[9] J. Reardon, D. Basin, and S. Capkun, “On Secure Data Deletion,” Security & Privacy, IEEE , vol.12, no.3, May-June 2014, pp.37-44.

[10] J. Reardon, H. Ritzdorf, D. Basin, and S. Capkun, “Secure data deletion from persistent media”, in Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security (CCS '13). ACM, New York, NY, USA, 2013, pp. 271-284.

[11] J. Reardon, S. Capkun, and D. Basin, “Data node encrypted file system: Efficient secure deletion for flash memory”, in USENIX Security Symposium, 2012, pp. 333-348.

[12] K. Sun, J. Choi, D. Lee, and S.H. Noh, "Models and Design of an Adaptive Hybrid Scheme for Secure Deletion of Data in Consumer Electronics," IEEE Transactions on Consumer Electronics, vol.54, no.1, Feb. 2008, pp.100-104.

[13] M. Riser, “Multiple Vulnerabilities in EncFS”, 2010. Retrieve [july 2014] from: http://archives.neohapsis.com/archives/fulldisclosure/2010-08/0316.html.

[14] P. Gutmann, "Secure deletion of data from magnetic and solid-state memory," proceedings of the Sixth USENIX Security Symposium, San Jose, CA, vol. 14, 1996.

[15] PhotoRec, Digital Picture and File Recovery. Available [July 2014] from: http://www.cgsecurity.org/wiki/PhotoRec.

[16] S. M. Diesburg and A. I. A. Wang, “A survey of confidential data storage and deletion methods”, ACM Computing Surveys (CSUR), v. 43, n.1, p.2, 2010.

[17] T. Hornby, “EncFS Security Audit”. Retrived [July 2014] from: https://defuse.ca/audits/encfs.htm.

[18] V. Gough, “EncFS Encrypted Filesystem”, stable release 1.7.4 (2010). Available [July 2014] from: http://www.arg0.net/encfs.

[19] Z. Wang, R. Murmuria, and A. Stavrou, “Implementing and optimizing an encryption filesystem on android”. In IEEE 13th International Conference on Mobile Data Management (MDM), 2012, pp. 52-62.

 ISBN: 978-1-61208-376-6