The Block-based Mobile PDE Systems Are Not Secure - Experimental Attacks

Niusen Chen[1], Bo Chen*[1], and Weisong Shi[2]

[1]Department of Computer Science, Michigan Technological University, Michigan, United States

[2]Department of Computer Science, Wayne State University, Michigan, United States

Abstract

Nowadays, mobile devices have been used broadly to store and process sensitive data. To ensure confidentiality of the sensitive data, Full Disk Encryption (FDE) is often integrated in mainstream mobile operating systems like Android and iOS. FDE however cannot defend against coercive attacks in which the adversary can force the device owner to disclose the decryption key. To combat the coercive attacks, Plausibly Deniable Encryption (PDE) is leveraged to plausibly deny the very existence of sensitive data. However, most of the existing PDE systems for mobile devices are deployed at the block layer and suffer from deniability compromises.

Having observed that none of existing works in the literature have experimentally demonstrated the aforementioned compromises, our work bridges this gap by experimentally confirming the deniability compromises of the block-layer mobile PDE systems. We have built a mobile device testbed, which consists of a host computing device and a flash storage device. Additionally, we have deployed both the hidden volume-based PDE and the steganographic file system-based PDE at the block layer of our testbed and performed disk forensics to assess potential compromises on the raw NAND flash. Our experimental results confirm it is indeed possible for the adversary to compromise the block-layer PDE systems when the adversary can have access to the raw NAND flash in real world. We also discuss practical issues when performing such attacks in practice.

1 Introduction

Mobile computing devices are widely used in our daily life nowadays and, with their increased use, more and more sensitive data are stored and processed in the mobile devices.

*Corresponding author. bchen@mtu.edu

Therefore, it turns to become an urgent need of protecting those sensitive data, and one of the most critical data security issues is confidentiality. A straightforward approach to protect data confidentiality is to use encryption. Currently, Full Disk Encryption (FDE) has been deployed to the mainstream mobile operating systems including Android [1] and iOS [8]. In FDE, encryption and decryption are completely transparent to users. Without the key, the attacker cannot obtain any knowledge about the original sensitive data. However, FDE cannot defend against a novel coercive attack in which the attacker can force the device owner to disclose the key, and decrypt the ciphertext to obtain the original sensitive data. For example, a journalist or a human rights worker [34, 15] who is working in a country of conflict or oppression, has captured some sensitive evidence of atrocities and tries to cross the border; to protect the evidence, he/she encrypts the evidence; the border inspector however, may be aware of the ciphertext and force the journalist to disclose the decryption key.

Plausibly Deniable Encryption (PDE) can be used to combat coercive attacks. In PDE, the plaintext is encrypted with a decoy key and a true key. When decrypting the cipher using the decoy key, we will obtain a decoy message and when decrypting the cipher using the true key, we will obtain the true message. Upon being coerced by the attacker, the device owner can only disclose the decoy key and keep the true key secret. In this way, the sensitive data can be protected against the coercive attackers as the attackers cannot notice the existence of the hidden sensitive data. Following the concept of PDE, a large number of PDE systems [31, 32, 34, 15, 30, 26, 14, 16, 23, 18, 19, 20, 21] have been designed for mobile devices. In general, the existing mobile PDE systems can be divided into three categories: C1) block-layer PDE systems [31, 32, 34, 15, 14, 16, 23]; C2) flash translation layer (FTL) PDE systems [26, 20]; and C3) deniability aware flash file systems [30, 19]. A majority of the existing mobile PDE systems belong to the category C1 which deploys PDE on the block layer. The reason is that deploying the PDE on the block layer could be achieved much more easily, resulting in a much better usability. However, the block-layer PDE systems are insecure, because: the hidden sensitive data will leave special traces in the underlying flash memory and such traces cannot be removed by the block-layer PDEs; by having access to the raw flash memory, the adversary may compromise the deniability [26]. The compromises have been analyzed theoretically by DEFTL [26], but none of the existing works have confirmed such compromises experimentally. This work thus aims to bridge this gap by conducting the first experimental study on understanding the deniability compromises of the existing block-layer PDE systems.

Comparison with DEFTL. Our work is different from that of the DEFTL [26] in a few aspects: First, DEFTL theoretically analyzes the potential deniability compromises when deploying the PDE on the block device layer. However, our work experimentally validates the deniability compromises in real-world devices. Especially, we have created a mobile device testbed which includes a host computing device (ARM architecture) and a self-made flash-based block device (using an open-source flash controller and a cheap USB development prototype board). This self-built mobile device follows the architecture of mainstream mobile devices in real world. We then deploy a few representative block-based PDE systems in our testbed, and perform forensic analysis over the raw NAND flash to study

the deniability compromises. Second, DEFTL only focuses on the deniability compromises on the PDE systems which use hidden volume technique, but our work assesses both the hidden volume-based and the steganographic file system-based PDE. Third, we have identified extra deniability compromises which have not been discovered in DEFTL.

2 Background

2.1 Flash Memory

Flash memory especially NAND flash has been used broadly as the external storage of mobile computing devices nowadays. Flash memory usually consists of blocks, and each block consists of pages. Typically, each flash block is a few hundreds of kilobytes in size and each page is a few kilobytes in size. Compared to conventional hard disk drives (HDD), flash memory has a few different features: 1) The unit of a read/write operation is a page, but the unit of an erase operation is a block. 2) A flash page needs to be erased before it can be programmed. 3) Due to the unique features of 1) and 2), the in-place update in flash memory would be expensive. Therefore, the flash storage typically uses an out-of-place instead of in-place update strategy [24]. 4) Each block in the flash memory can only be programmed/erased for a limited number of times and, therefore, programmings and erasures should be distributed evenly across the entire flash to prolong the service life.

2.2 Flash Translation Layer

To manage flash memory, we can use a flash-specific file system like YAFFS or JFFS. However, the flash-specific file systems are rarely used in mobile computing devices today. Instead, a flash translation layer (FTL) is incorporated into the flash storage media (e.g., SD cards, UFS cards, MMC cards) to transparently handle the unique nature of NAND flash hardware, so that the flash storage media can expose a block access interface externally and the traditional block-based file systems can be deployed. The core functions implemented in the FTL include garbage collection, wear leveling, and bad block management.

Garbage collection. As the flash storage media adopt the out-of-place update strategy, the flash pages storing old data may be invalidated. Garbage collection is typically used to reclaim those invalid pages. The garbage collection usually works as follows: The FTL selects a victim block which has the largest number of invalid pages. It then copies data stored in valid pages in the victim block to an empty block, and erases the victim block.

Wear leveling. Each flash block only supports a limited number of program/erase (P/E) cycles. The main purpose of wear leveling is to distribute P/E cycles evenly across the entire flash. There are a lot wear leveling strategies including static wear leveling and dynamic wear leveling. A fundamental idea is to swap hot and cold data, so that the hot data will be relocated to those blocks with least P/E cycles and the cold data will be relocated to those blocks with most P/E cycles.

Bad block management. Over time, a flash block may turn "bad" and cannot be used to

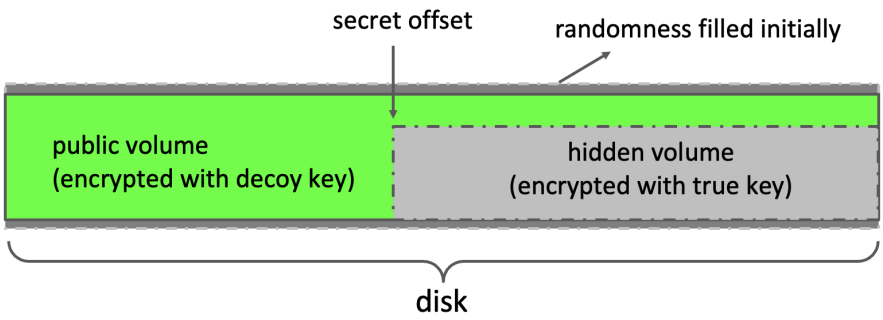

Figure 1: The hidden volume-based PDE technique.

reliably store data, as there were too many P/E cycles performed on this block in the past. Therefore, the FTL needs to keep track of those bad blocks and prevents them from being used to store data. Typically, a bad block table can be used to keep track of bad blocks. If a block turns bad, the FTL will copy data from this block to an empty block and add this bad block to the bad block table.

2.3 Plausibly Deniable Encryption

Plausibly deniable encryption can be leveraged to combat coercive attacks. Typically, there are two techniques which can be used to implement the PDE system, namely, the hidden volume technique [6, 7] and the steganographic file system [9, 28].

For the hidden volume technique (see Figure 1), the entire disk is filled with random data initially. Two volumes — a public volume and a hidden volume — will be introduced. Correspondingly, two keys — a decoy key and a true key— are selected. The public volume is encrypted via the decoy key and placed across the entire disk, and the hidden volume is encrypted with the true key and placed to the end of the disk starting from a secret offset (derived from the true key). Upon being coerced, the victim will simply disclose the decoy key. Via the decoy key, the attacker can decrypt the public volume, but will not notice the existence of the hidden volume stored stealthily among the random data.

One implementation of the steganographic file system is to fill the disk with random data initially, and to encrypt and to hide the sensitive data at a secret location which can be derived from a secret key. To prevent loss of sensitive data, multiple copies of sensitive data are stored in multiple locations across the disk.

3 Model and Assumptions

System model. We consider a mobile computing device which is equipped with flash memory (e.g., UFS cards, eMMC cards, microSD cards, etc) as the external storage. The storage architecture of main-stream mobile devices is shown in Figure 2. A mobile user directly communicates with apps (e.g., a PDF viewer app) running at the application layer. The OS/file system will manage storage hardware and provide system calls for the applications

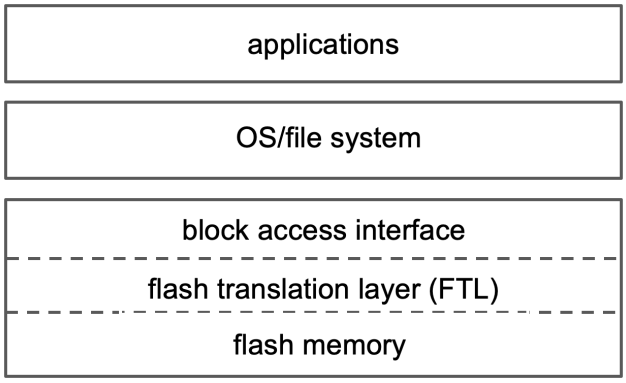

Figure 2: The storage architecture of main-stream mobile computing devices

to access the data stored at the storage hardware. The underlying flash memory storage is typically used in the form of a block device. The FTL will handle special nature of flash memory, exposing a block access interface externally.

Adversarial model. We assume the adversary can capture both the victim and his/her mobile device, and coerce the owner to disclose the decryption key. The adversary is rationale and will stop coercing the user once he/she believes that the decryption key is disclosed [31, 15, 16, 26]. Using the disclosed key, the adversary will play with the mobile devices to compromise the PDE. In addition, the adversary can extract the raw image from the flash storage equipped with the victim device and obtain the hardware parameters (e.g., page size and block size) of the underlying flash memory chips. The adversary can then perform forensic analysis on the raw image — with the help of the disclosed key — to identify the existence of PDE.

4 Experimentally Attacking The Block-layer PDE Systems

The hidden volume technique and the steganographic file system (Sec. 2.3) are two major techniques which have been leveraged to implement the PDE system at the block layer. We therefore focus on attacking those two types of PDE systems. For each type of PDE systems, we first deploy a representative PDE implementation on a self-built mobile device testbed, and then perform forensic analysis to identify any potential deniability compromises. We mainly concentrate on the deniability compromises in the underlying storage medium, which is typically NAND flash for mobile devices.

4.1 Experimental Setup

A challenge faced in our experiment was that, almost every commercially available mobile device (smartphones, tablets, smart watches, or the recent IoT devices like smart home assistants) uses a well encapsulated flash-based block device, e.g., UFS cards, eMMC, mi-

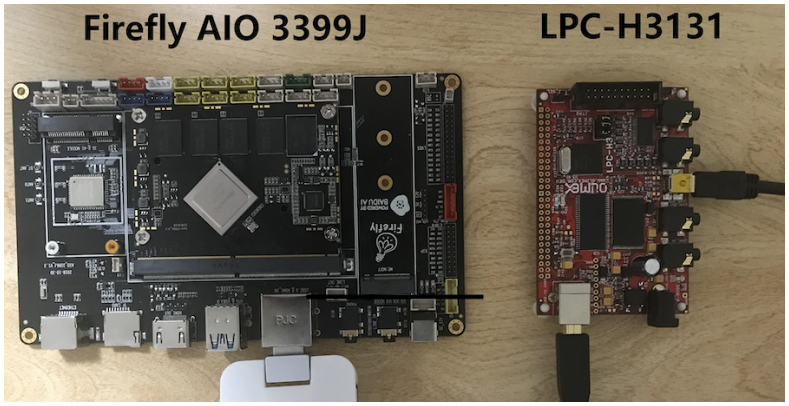

Figure 3: A self-made mobile device testbed for our experiment. Firefly AIO-3399J is the host computing device and LPC-H3131 (with flash controller) is the flash-based block device.

croSD cards. To facilitate our attacks, we have built a mobile device testbed, which consists of a flash-based block device and a host computing device (Figure 3). The flash-based block device was built by porting [33] an open-sourced flash controller OpenNFM [22] to a USB header development prototype board LPC-H3131 [3] (Major hardware: ARM9 32-bit ARM926EJ-S, 180Mhz, 32MB RAM, and 512MB NAND flash. The flash memory consists of 128KB blocks, and each block consists of 2KB pages). The host computing device was an embedded development board, Firefly AIO-3399J (Major hardware: Six-Core ARM 64-bit processor, 4GB RAM). The Firefly AIO-3399J was managed by Linux kernel 4.4.194. This mobile device testbed shares a common architecture with mainstream mobile devices in real world.

We then deployed a block-based PDE system in the host computing device. For the hidden volume-based PDEs, we deployed VeraCrypt [7], a fork of the discontinued TrueCrypt project. Note that a large number of PDE systems deployed on the block layer (including PDE systems [6, 11] designed for PCs as well as PDE systems [31, 32, 34, 15, 14, 16] designed for mobile devices) have utilized the hidden volume technique, and our attack can be applied to most of them. For the steganographic file systems, we deployed stegfs [5], a recent open-source implementation of steganographic file systems [9, 28, 29] in user space[1]. For each deployed PDE system, we analyzed the raw NAND flash to identify the potential PDE compromises.

4.2 Experimental Attacks

Experimentally Attacking the Hidden Volume-based PDEs. We deployed VeraCrypt [7] in the host computing device, and manually created both a public and a hidden volume via VeraCrypt. The public volume occupies the entire disk (i.e., the flash-based block device built by porting OpenNFM to LPC-H3131) and the hidden volume is 200MB in size. The

[1]Note that the original implementation of the steganographic file system [2, 9, 28] was done in 1999 for Ext2, and has not been updated since then.

file system deployed in the public volume was exFAT, which writes data sequentially from the beginning of the disk to avoid overwriting the hidden volume stored stealthily in the second half of the disk. We also deployed exFAT in the hidden volume. We performed three tests to simulate behaviors of a device owner as follows:

Test #1: We entered the public mode, and wrote non-sensitive data to the public volume. The size of the non-sensitive data being written is small (i.e., the size is in the magnitude of a few kilobytes, and should be always smaller than the size of a flash block). We then quit the public mode, entered the hidden mode, and wrote a small amount of sensitive data to the hidden volume. The size of the sensitive data being written is similar to the size of the non-sensitive data being written to the public volume. We also repeated the aforementioned operations a few times. This behavior is reasonable. For instance, the user may write a short article to the public volume and then store a small secret audio record to the hidden volume.

Test #2: We entered the public mode and wrote non-sensitive data to the public volume. The size of the non-sensitive data being written should be large, e.g., always larger than the size of one flash block. Then, we quit the public mode, entered the hidden mode, and wrote a small amount of sensitive data to the hidden volume. The size of the sensitive data being written should be small, e.g., in the magnitude of a few kilobytes which is always smaller than the size of a flash block. This behavior is reasonable. For instance, the user may store a large video to the public volume and then store a small secret audio record to the hidden volume.

Test #3: We entered the hidden mode and wrote a small file (i.e., file 1) to the hidden volume. The size of file 1 is a few kilobytes (e.g., 3 KBs). We then modified a few randomly selected locations in file 1 and saved it. Next, we wrote a large file (i.e., file 2) to the hidden volume. The size of file 2 is more than 128 kilobytes. This behavior is reasonable. For instance, the user may create a small secret document in the hidden mode and modify it later; the user may then create another secret document which is large in size.

After each test, we analyzed the corresponding flash memory image. Note that the coercive adversary should have access to the decoy key.

From the image obtained after running test #1, we have identified the first type of special flash blocks, i.e., "special block 1" in Figure 4. Such a block is completely filled with random data, but a portion of pages among this block cannot be decrypted successfully. *Without the PDE deployed*, there are only two possibilities for a block completely filled with random data: 1) All data stored in it can be decrypted successfully, i.e., the block is filled with public data. 2) All data stored in it cannot be decrypted successfully, i.e., the block is completely occupied by random data filled initially. However, *with the PDE deployed*, some of the pages in the block are occupied by the hidden data and cannot be decrypted, as we wrote a small amount of data to the public volume and the hidden volume in turn repeatedly during the test #1. The existence of "special block 1" indicates the device owner has entered the hidden mode and committed hidden sensitive data to the external storage before.

From the image obtained after test #2, we have identified the second type of special blocks, i.e., "special block 2" in Figure 4. Such a block has a few pages in the beginning

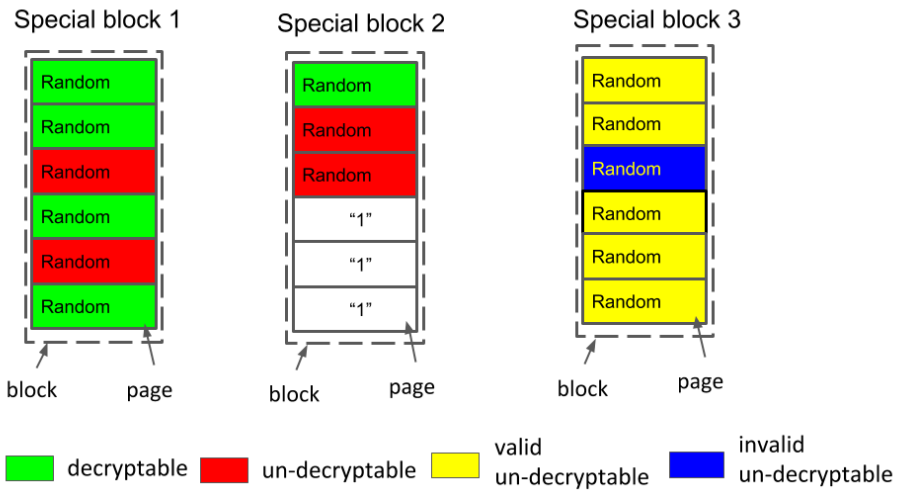

Figure 4: Special blocks observed in raw NAND flash.

storing random data and the remaining pages filling with all '1' bits; among the random data, those located in the end cannot be decrypted. *Without the PDE deployed*, a block is erased and then partially used by the public data which are all decryptable. However, *with the PDE deployed*, some of pages in this block may be used by the hidden data and hence cannot be decrypted. Especially in our test #2, the hidden data will occupy those pages before the empty pages (i.e., a page with all '1's) of the block. Therefore, the existence of "special block 2" also indicates the device owner has committed hidden sensitive data to the external storage before.

From the image obtained after running test #3, we have identified the third type of special blocks, i.e., "special block 3" in Figure 4. Such a block is completely filled with undecryptable random data, but some of them (i.e., in arbitrary locations across the block) are marked as invalid. A snapshot of a portion of special block 3 is also provided in Figure 5. This is because: *With the PDE deployed*, a flash block may have been used by the hidden volume, and arbitrary pages across the block may have been updated by the user and hence invalided by the FTL; in addition, the hidden data are encrypted by the true key and cannot be decrypted via the decoy key. However, *Without the PDE deployed*, the data being updated by the user and invalidated by the FTL will be the public data which are decryptable via the decoy key. Therefore, the existence of "special block 3" indicates the existence of the hidden volume. Note that the "special block 3" has not been discovered in the literature.

Experimentally Attacking The Steganographic File System-based PDEs. We deployed stegfs [5] in the host computing device. Note that the steganographic file system works differently from the hidden volume technique that: the file system is initially filled with randomness and the sensitive data are encrypted via a secret key and stored at random locations of the entire disk; it also needs to maintain a few copies of the hidden data across the disk to mitigate loss of hidden sensitive data as the public data may overwrite them over time. We performed one test to simulate the behavior of a device owner as follows:

We first mounted the FAT file system on the flash device, and wrote a certain amount of

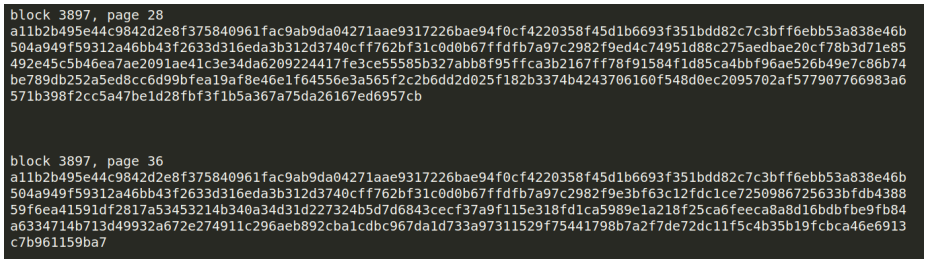

Figure 5: A snapshot (portion) of special block 3. Every 4 bits have been converted to the corresponding hexadecimal digit. In this snapshot, the data stored on page 28 has been updated and invalidated by the FTL, and the newly updated data are written to page 36. The data stored in the aforementioned pages are undecryptable.

public non-sensitive data. Then, we manually mounted the steganographic file system, and wrote a certain amount of sensitive data. This behavior is reasonable. For instance, the user may first store a few non-sensitive images to the disk via the public file system and then store some secret documents to the disk using the steganographic file system.

After the test, we extracted the corresponding flash memory image and analyzed it. We have identified a few special traces due to the existence of the PDE: 1) Trace #1: public data and random data are interleaving across the entire flash. However, *without the existence of the hidden sensitive data*, the distribution of the data across the flash should be public data followed by random data. This is because: The steganographic file system fills random data across the entire flash initially and, since the FTL uses log-structured writing, regardless how the file system writes public data at the upper layer, the FTL will always program flash blocks from the beginning. Therefore, the observed trace #1 indicates the existence of the hidden sensitive data. 2) Trace #2: public data and random data share the same flash block. Figure 6 shows a snapshot we obtained from one flash block after the test, in which we can observe some of the pages in a flash block store public data which are semantically meaningful, while some of the pages of the block store random undecryptable data. However, *without the existence of the hidden sensitive data*, the distribution of data in a flash block should be either i) public data, followed by all '1' bits, or ii) all public data. This is because: Without the existence of hidden data, each time when the FTL writes public data but cannot find empty pages, it will erase a flash block, and write public data sequentially from the beginning of the block due to the use of log-structured writing; if any pages in this block have not been filled, they remain empty and contain all '1's. Therefore, the observed trace #2 indicates the existence of the hidden sensitive data.

5 Discussion

Assessing the difficulty of performing our attacks. To compromise the deniability by having access to the raw flash memory, the adversary needs to tackle two issues: 1) how to extract an image from the NAND flash memory given a victim mobile device, and 2) how to perform forensic analysis over the raw flash memory data. For the first issue, Breeuwsma

Figure 6: A snapshot (portion) from a flash block when attacking the steganographic file system

et al. [12] introduced a few low-level data acquisition methods for flash memory, including flasher tools, using an access port commonly used for testing and debugging, etc. Chen et al. [20] mentioned a method of obtaining raw data from SSDs "by opening the covers and directly reading the memory chips with cheap off the shelf readers". For the second issue, the adversary can use the existing digital forensic tools available on the market (e.g., Photorec [4], etc.) or develop new special tools to analyze the captured image.

Implications of our experimental attacks. Our attacks performed in this work confirm that it is indeed feasible to compromise the block-based PDE systems in practice. Our results further justify that the deniability compromise in the lower storage medium is indeed a significant issue and should be considered seriously when designing any future PDE systems for mobile computing devices. An immediate remediation would be moving the entire PDE system design to the flash translation layer (FTL) [25, 20] which however, would not be a good solution as it will impose a large burden on the FTL firmware. In addition, as the PDE integrated in the FTL firmware is far away from the user applications, making it user unfriendly. It is unclear how to design a PDE system which is 1) secure (i.e., eliminating deniability compromises in the flash memory), and 2) keeping the FTL lightweight, and 3) user-friendly. This is still an open problem in the literature.

Other attacks on the PDE systems. This work only focuses on the single-snapshot attack in which the adversary is only allowed to have access to the victim device once. A stronger adversary may conduct the multiple-snapshot attack by periodically accessing to the victim device [17, 18, 20]. By capturing different snapshots of the external storage over time and comparing the different snapshots, the adversary will detect changes of the hidden sensitive data, compromising the deniability. For example, if the hidden volume technique is used, by comparing different snapshots, the adversary may observe data changes performed in the space which is claimed empty but actually hides the hidden volume. Some of mitigation strategies can be accompanying public writes with dummy writes and hiding the sensitive data into the dummy writes [17, 18], or using the WOM (write-once memory) code to encode the hidden data in a public cover [20]. In addition, this work only focuses on the

deniability compromises in the external storage, but hidden sensitive data may leave traces in the internal memory, and such traces may be extracted by the adversary by performing memory forensics [13]. One potential solution is to power-off the device each time after quitting the hidden mode in which the user can manage the hidden sensitive data. Another solution could be leveraging trusted execution environments (TEE) like Arm TrustZone [27] so that the memory used to process the hidden sensitive data can be protected, avoiding being accessed by the adversary.

6 Related Work

In the following, we summarize the major PDE systems utilizing the hidden volume technique or the steganographic file systems. A thorough literature review of PDE system can be found in [35].

6.1 The Hidden Volume-based PDE systems

Skillen et al. proposed Mobiflage [31, 32], which adapts the hidden volume technique to Android devices. There are a few variants of Mobiflage. One variant assumes the existence of an FAT32 SD card, and deploys the public volume/hidden volume to this SD card. Another variant releases the aforementioned assumption by using a modified Ext4 file system. Yu et al. proposed [34] MobiHydra to mitigate a booting-time attack faced by Mobiflage. In addition, MobiHydra allows the user to switch from the public to the hidden mode without rebooting the device and supports multi-level deniability. Chang et al. designed Mobipluto [15, 14], the first file system friendly PDE system which allows any block-based file systems to be deployed on the public volume, by smartly integrating the hidden volume technique with thin provisioning. Chang et al. further extended the hidden volume technique to combat the multi-snapshot adversary by introducing dummy writes on the block layer [16]. Jia et al. proposed DEFTL [26], the first hidden volume-based PDE system integrated with the flash translation layer. Barker et al. [10] proposed Artifice, which can meet a few additional security requirements including: 1) information leakage resistance, and 2) deniable changes, and 3) deniable software.

6.2 The Steganographic File Systems

Anderson et al. [9] proposed the first steganographic file system. One of their constructions is to hide the secret data among the randomness. The system maintains several copies of secret data to reduce the possibility of losing them. Inspired by Anderson et al's construction, McDonald et al. [28] designed a more practical as well as efficient steganographic file system, in which secret files are hidden in unused blocks of a partition which also contains normal files. Pang et al. [29] proposed StegFS, a new steganographic file system which allows the user to hide his/her files or directories in a selective manner. A salient advantage of StegFS is that it can ensure integrity of files while maintaining effective disk utilization. Zhou et

al [36] further mitigate the attacks which may compromise the steganographic file system by analyzing data access of use applications.

7 Conclusion

In this work, we have experimentally confirmed the deniability compromises of the block-layer PDE systems deployed on the mobile computing devices. Our work conducts the first experimental attacks by 1) deploying both the hidden volume-based PDE and the steganographic file system on the block layer of a mobile device testbed, and 2) allowing the adversary to have access to the flash memory and to perform forensic analysis over the raw flash memory data. Our results strengthen the necessity of taking care of the deniability compromises in the lower storage layer when designing any future PDE systems for mobile devices.

Acknowledgments.

This work was supported by US National Science Foundation under grant number 1928349-CNS, 1928331-CNS, 1938130-CNS, and 2043022-DGE.

References

[1] Android full disk encryption. Retrieved on April 21, 2022, from `https://source.android.com/security/encryption/`.

[2] Index of `/~mgk25/stegfs`. Retrieved on April 21, 2022, from `https://www.cl.cam.ac.uk/~mgk25/stegfs/`.

[3] Lpc-h3131. Retrieved on April 21, 2022, from `https://www.olimex.com/Products/ARM/NXP/LPC-H3131/`.

[4] Photorec. Retrieved on March 28, 2022, from `https://www.cgsecurity.org/wiki/PhotoRec`.

[5] stegfs. Retrieved on April 21, 2022, from `https://sourceforge.net/projects/stegfs/`.

[6] Truecrypt. Retrieved on April 21, 2022, from `http://truecrypt.sourceforge.net/`.

[7] Veracrypt. Retrieved on April 21, 2022 from `https://www.veracrypt.fr/code/VeraCrypt/`.

[8] How to encrypt your devices, 2017. Retrieved on April 21, 2022, from `https://spreadprivacy.com/how-to-encrypt-devices/`.

[9] Ross Anderson, Roger Needham, and Adi Shamir. The steganographic file system. In *International Workshop on Information Hiding*, pages 73–82. Springer, 1998.

[10] Austen Barker, Yash Gupta, Sabrina Au, Eugene Chou, Ethan L Miller, and Darrell D Long. Artifice: Data in disguise. In *Proceedings of the 36th International Conference on Massive Storage Systems and Technology (MSST 2020)*, 2020.

[11] Erik-Oliver Blass, Travis Mayberry, Guevara Noubir, and Kaan Onarlioglu. Toward robust hidden volumes using write-only oblivious ram. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, pages 203–214. ACM, 2014.

[12] Marcel Breeuwsma, Martien De Jongh, Coert Klaver, Ronald Van Der Knijff, and Mark Roeloffs. Forensic data recovery from flash memory. *Small Scale Digital Device Forensics Journal*, 1(1):1–17, 2007.

[13] Mariusz Burdach. Physical memory forensics. *USA: Black Hat*, 2006.

[14] Bing Chang, Yao Cheng, Bo Chen, Fengwei Zhang, Wen-Tao Zhu, Yingjiu Li, and Zhan Wang. User-friendly deniable storage for mobile devices. *computers & security*, 72:163–174, 2018.

[15] Bing Chang, Zhan Wang, Bo Chen, and Fengwei Zhang. Mobipluto: File system friendly deniable storage for mobile devices. In *Proceedings of the 31st annual computer security applications conference*, pages 381–390, 2015.

[16] Bing Chang, Fengwei Zhang, Bo Chen, Yingjiu Li, Wen-Tao Zhu, Yangguang Tian, Zhan Wang, and Albert Ching. Mobiceal: Towards secure and practical plausibly deniable encryption on mobile devices. In *2018 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pages 454–465. IEEE, 2018.

[17] Bo Chen. Towards designing a secure plausibly deniable system for mobile devices against multi-snapshot adversaries–a preliminary design. *arXiv preprint arXiv:2002.02379*, 2020.

[18] Bo Chen and Niusen Chen. Poster: a secure plausibly deniable system for mobile devices against multi-snapshot adversaries. In *2020 IEEE Symposium on Security and Privacy Poster Session*, 2020.

[19] Chen Chen, Anrin Chakraborti, and Radu Sion. Infuse: Invisible plausibly-deniable file system for nand flash. *Proceedings on Privacy Enhancing Technologies*, 4:239–254, 2020.

[20] Chen Chen, Anrin Chakraborti, and Radu Sion. Pearl: Plausibly deniable flash translation layer using wom coding. In *The 30th Usenix Security Symposium*, 2021.

[21] Niusen Chen, Bo Chen, and Weisong Shi. Mobiwear: A plausibly deniable encryption system for wearable mobile devices. In *EAI International Conference on Applied Cryptography in Computer and Communications*, pages 138–154. Springer, 2021.

[22] Google Code. Opennfm. Retrieved on April 21, 2022, from `https://code.google.com/p/opennfm/`.

[23] Wendi Feng, Chuanchang Liu, Zehua Guo, Thar Baker, Gang Wang, Meng Wang, Bo Cheng, and Junliang Chen. Mobigyges: A mobile hidden volume for preventing data loss, improving storage utilization, and avoiding device reboot. *Future Generation Computer Systems*, 2020.

[24] Le Guan, Shijie Jia, Bo Chen, Fengwei Zhang, Bo Luo, Jingqiang Lin, Peng Liu, Xinyu Xing, and Luning Xia. Supporting transparent snapshot for bare-metal malware analysis on mobile devices. In *Proceedings of the 33rd Annual Computer Security Applications Conference*, pages 339–349. ACM, 2017.

[25] Shijie Jia, Luning Xia, Bo Chen, and Peng Liu. Nfps: Adding undetectable secure deletion to flash translation layer. In *Proceedings of the 11th ACM on Asia Conference on Computer and Communications Security*, pages 305–315. ACM, 2016.

[26] Shijie Jia, Luning Xia, Bo Chen, and Peng Liu. Deftl: Implementing plausibly deniable encryption in flash translation layer. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 2217–2229, 2017.

[27] Jinghui Liao, Bo Chen, and Weisong Shi. Trustzone enhanced plausibly deniable encryption system for mobile devices. In *2021 IEEE/ACM Symposium on Edge Computing (SEC)*, pages 441–447. IEEE, 2021.

[28] Andrew D McDonald and Markus G Kuhn. Stegfs: A steganographic file system for linux. In *International Workshop on Information Hiding*, pages 463–477. Springer, 1999.

[29] HweeHwa Pang, K-L Tan, and Xuan Zhou. Stegfs: A steganographic file system. In *Proceedings 19th International Conference on Data Engineering (Cat. No. 03CH37405)*, pages 657–667. IEEE, 2003.

[30] Timothy M Peters, Mark A Gondree, and Zachary NJ Peterson. DEFY: A deniable, encrypted file system for log-structured storage. In *22th Annual Network and Distributed System Security Symposium, NDSS*, 2015.

[31] Adam Skillen and Mohammad Mannan. On implementing deniable storage encryption for mobile devices. In *20th Annual Network and Distributed System Security Symposium, NDSS 2013, San Diego, California, USA, February 24-27*, 2013.

[32] Adam Skillen and Mohammad Mannan. Mobiflage: Deniable storage encryptionfor mobile devices. *IEEE Transactions on Dependable and Secure Computing*, 11(3):224–237, 2014.

[33] Deepthi Tankasala, Niusen Chen, and Bo Chen. A step-by-step guideline for creating a testbed for flash memory research via lpc-h3131 and opennfm. 2020.

[34] Xingjie Yu, Bo Chen, Zhan Wang, Bing Chang, Wen Tao Zhu, and Jiwu Jing. Mobihydra: Pragmatic and multi-level plausibly deniable encryption storage for mobile devices. In *International conference on information security*, pages 555–567. Springer, 2014.

[35] Qionglu Zhang, Shijie Jia, Bing Chang, and Bo Chen. Ensuring data confidentiality via plausibly deniable encryption and secure deletion–a survey. *Cybersecurity*, 1(1):1, 2018.

[36] Xuan Zhou, HweeHwa Pang, and Kian-Lee Tan. Hiding data accesses in steganographic file system. In *Proceedings. 20th International Conference on Data Engineering*, pages 572–583. IEEE, 2004.