TrustZone Enhanced Plausibly Deniable Encryption System for Mobile Devices

Jinghui Liao
jinghui@wayne.edu
Wayne State University
Detroit, MI, USA

Bo Chen
bchen@mtu.edu
Michigan Technological University
Houghton, MI, USA

Weisong Shi
weisong@wayne.edu
Wayne State University
Detroit, MI, USA

ABSTRACT

Modern mobile devices are increasingly used to store and process sensitive data. In order to prevent the sensitive data from being leaked, one of the best ways of protecting them and their owner is to hide the data with plausible deniability. Plausibly Deniable Encryption (PDE) has been designed for such purpose. The existing PDE systems for mobile devices however, have suffered from significant drawbacks as they either ignore the deniability compromises present in the special underlying storage media of mobile devices or are vulnerable to various new attacks such as side-channel attacks.

In this work, we propose a new PDE system design for mobile devices which takes advantage of the hardware features equipped in the mainstream mobile devices. Our preliminary design has two major component: First, we strictly isolate the hidden and the public data in the flash layer, so that a multi-snapshot adversary is not able to identify the existence of the hidden sensitive data when having access to the low layer storage medium of the device. Second, we incorporate software and operating system level deniability into ARM TrustZone. With this TrustZone-enhanced isolation, our PDE system is immune to side-channel attacks at the operating system layer.

KEYWORDS

Mobile Devices, Plausibly Deniable Encryption, TrustZone, TEE, FTL

ACM Reference Format:
Jinghui Liao, Bo Chen, and Weisong Shi. 2022. TrustZone Enhanced Plausibly Deniable Encryption System for Mobile Devices. In *Proceedings of ACM Conference (Conference'17).* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Mobile computing devices have played an essential role in our daily life, as we increasingly rely on them for daily communication, web browsing, online shopping, mobile banking, etc. Especially, quite a few people also use mobile devices to deal with professional work or even political tasks. This, however, creates a large amount of personally private or even mission critical data in those devices. To protect sensitive information, major mobile operating systems have incorporated various encryption tools which can allow users to encrypt their critical data [6, 30]. Full Disk Encryption (FDE) has been deployed broadly in mobile devices to defend against passive attackers, preventing them from retrieving sensitive information from the data storage. However, active attackers, who can physically approach the device owner, may coerce the owner to decrypt the sensitive information. Therefore, we need a new technique which can protect sensitive data even if the data owner is forced to unveil the secrecy.

Plausibly Deniable Encryption (PDE) is such a technique designed to address the aforementioned issue. The PDE can convincingly deny the very existence of sensitive data, by which even the active attackers cannot prove that sensitive data exists. The PDE typically works as follows: Two keys, a true key and a decoy key will be generated; the original sensitive data will be encrypted into cipherthext in such a way that, upon decryption, if the true key is used, the ciphertext will be decrypted into the original sensitive plaintext but, if the decoy key is used, the ciphertext will be decrypted into some different reasonable and innocuous plaintext; therefore, upon being coerced, the victim can simply disclose the decoy key to avoid being tortured, while protecting the original sensitive data. Leveraging PDE, a variety of deniable storage systems have been designed for personal computers, which include TrueCrypt [35], Rubberhose [25], HIVE [9], Gracewipe [37], Steganographic File Systems [5, 29, 31], etc. Despite the success of PDE for PC platforms, achieving deniability in mobile platforms is much more challenging, for two reasons: 1) Mobile devices are equipped with unique hardware, creating extra attack vectors leading to compromise of deniability [24]; 2) Mobile devices are sensitive to electricity consumption and only have limited computational resources, i.e., the PDE designed for mobile platforms has much higher requirements in energy efficiency.

Recently, PDE systems for mobile devices have been proposed [12–14, 19, 24, 32–34, 36]; nevertheless, the existing mobile PDE systems still suffer from some critical drawbacks:

Deniability Compromise in Lower Storage Layer. Most existing PDE systems [12–14, 33, 34, 36] for mobile devices are built on the block layer and unfortunately suffer from deniability compromise in the underlying raw flash memory. A fundamental reason is that raw flash memory requires special internal management to handle its unique nature [27]. Even if sensitive data have been isolated carefully on the block layer, they will not be carefully isolated by the internal management of flash memory which works independently and is transparent to the upper layer. Therefore, traces of sensitive data may be present in the raw flash memory and observed by the snapshot adversaries[1], compromising deniability.

Side-Channel Attacks. Most existing PDE systems [9, 32] suffer from side-channel attacks [21]. The fundamental reason is that existing designs do not strictly isolate the public mode (in which the public

Conference'17, July 2017, Washington, DC, USA
2022. ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

[1]The adversary may obtain a snapshot of the storage device at different points of time and, integrating the hidden volume mechanism [24] into the flash memory cannot defend against such a multi-snapshot adversary.

non-sensitive data is processed and stored) and the hidden mode (in which the hidden sensitive data is processed and stored) and, the information of the hidden mode may be leaked to the public mode, and therefore the adversary may learn the existence of the PDE by analyzing shared resources such as memory.

The above deficiencies limit the applications of the existing PDE systems in mainstream mobile devices and, there is an urgent need of developing a more secure mobile PDE system. Therefore, our primary goal is to develop a new mobile PDE system which can defend against snapshot attackers that can compromise deniability by: 1) exploring leaks in the lower storage layer; 2) performing side-channel attacks over the shared resources between the two modes.

To realize this goal, we design a novel mobile PDE system that achieves deniability by leveraging existing hardware features of mobile devices such as flash memory and ARM TrustZone. Our preliminary design consists of two major components:

Data Hiding in FTL. To prevent deniability compromise, the public and the hidden data should be isolated in the external flash storage, such that by having access to the lower storage layer, the adversary is not able to identify the existence of the hidden sensitive data. Prior research isolates them in the upper block layer, which cannot prevent deniability compromise from the raw flash memory snapshot attacks. Therefore, we will implement data hiding techniques in the flash translation layer (FTL), such that the public and the hidden data can be strictly isolated in the flash memory. Our design will eliminate any sources of deniability compromise in the flash memory and defend against adversaries which can take a snapshot of the flash memory at different points of time.

Strong Isolation in OS. The public and the hidden mode must be strictly isolated when running in the device, eliminating the possibility of information leakage. We will study novel approaches to defend against side-channel attacks due to the shared system software and applications. We will leverage ARM TrustZone technology to execute a secure OS and sensitive applications (i.e., apps used in the hidden mode) in an isolated execution environment, so that we can mitigate information leakage of the hidden mode from flash, memory, cache, and I/O devices. Note that TrustZone is an existing hardware feature that is compatible with mainstream mobile devices.

2 BACKGROUND

2.1 Plausibly Deniable Encryption

Plausibly deniable encryption (PDE) is explored to maintain the privacy of sensitive data against a coercive attacker, who can force the data owner to reveal the decryption key. PDE allows data owners to decrypt ciphertext to plausible and benign decoy plaintext with a different key, such that the data owner is able to deny the existence of the original sensitive data. In order to provide reasonableness and plausibility, the PDE system usually requires the decoy plaintext can be normally found on a computer and all the ciphertexts are accounted for. Existing PDE systems rely on either steganography or hidden volumes to achieve deniability.

With PDE, users can convincingly deny that a given data is encrypted, or that they can decrypt the given data, or that some specific encrypted data exists. Such denials may or may not be genuine, and PDE can eradicate the attacker's confidence that the data is encrypted or that the person who owns the data can decrypt and provide the relevant plaintext.

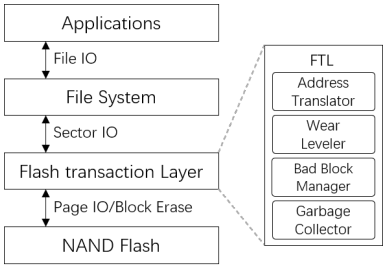

Figure 1: The architecture of an FTL-based flash storage system. Applications read and write files via the file system, while the file system do sector operations with the FTL. FTL interacts with the *NAND* flash with I/Os on pages and erasures on blocks. FTL contains address translator, wear leveler, bad block manager, and garbage collector.

2.2 NAND Flash and Flash Translation Layer

Main-stream mobile computing devices use *NAND* flash as external storage. *NAND* stores information in an array of memory cells, which are grouped into blocks. Each block contains a few pages and every page usually has a small spare out-of-band (OOB) area. *NAND* flash usually exhibits a few special characteristics: 1) *NAND* flash has an erase before-write design, i.e., a flash cell needs to be erased before it can be overwritten. 2) The unit for a read/program operation in *NAND* is a page, but the unit for an erase operation is a block. 3) Flash memory is update unfriendly since updating a page requires erasing the entire encompassing block. Therefore, it usually uses out-of-place update. 4)A flash block usually has a finite number of program-erase cycles, and will be worn out if the number of programs/erasures performed over it exceeds a certain threshold.

Flash Translation Layer (FTL). To remain compatible with traditional block-based file systems (e.g., EXT4 and FAT32), flash memory is usually emulated as a block device. Therefore, most existing flash storage media (e.g., eMMC cards, MiniSD cards) used in mobile devices are used as block devices. The Flash Translation Layer (FTL) [23, 26–28], a piece of special firmware, is introduced between the file system and the raw NAND flash to transparently handle the unique nature of flash memory (as shown in Figure 1). FTL usually implements address translator, garbage collector, wear leveler, and bad block manager. Since flash storage is emulated as a block device, FTL needs to translate the address between block and flash memory addresses. The address translator has an internal mapping table to convert the page from logical block address (LBA) to physical block address (PBA). Figure 2 is an example of how address translator works. In the figure, the address translator accepts a write request from file system to write data `m` to address `b` (which is an LBA), converting `b` to its corresponding PBA `b'`.

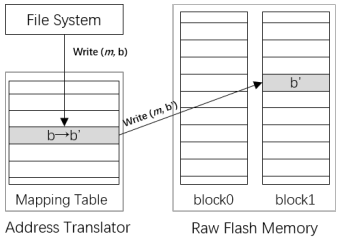

Figure 2: An example of the process of address translation in FTL. The file system sends a write request (*m*, b) to the *NAND*, where *m* is the data and b is the target page. The address translator of FTL convert the page from LBA b to PBA b' in the mapping table. Then write *m* to b'

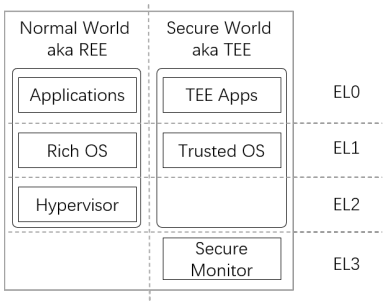

Figure 3: TrustZone virtualizes the CPU into two states, namely a Normal World and a Secure World. The Normal World is a Rich Execution Environment (REE) and the Secure World is a Trusted Execution Environment (TEE).

2.3 TrustZone

ARM TrustZone technology [8] is a hardware feature that creates an isolated execution environment since ARMv6 around 2002 [1]. Similar to other hardware isolation technologies, it provides two environments or worlds. The Trust Execution Environment (TEE), the secure world, and the Rich Execution Environment (REE), the normal world. The CPU on a TrustZone-enabled ARM platform has two security modes: secure mode and normal mode. Figure 3 shows the processor modes in a TrustZone-enabled ARM platform. Each processor mode has its own memory access region and privilege. The code running in the normal mode cannot access the memory in the secure mode, while the program executed in the secure world can access the memory in the normal mode. As shown in Figure 3, ARM involves different Exception Levels (EL) to indicate different privileges in ARMv8 architecture, and lower EL owns lower privilege. The EL3, which is the highest EL, serves as a gatekeeper managing the switches between the normal mode and the secure mode.

2.4 Hidden Volumes

The hidden volumes mechanism can be used to implement PDE, which works as follows: Two volumes, a public volume and a hidden volume, are created on a given disk. The public volume is encrypted using a decoy key and is placed across the entire disk, while the hidden volume is encrypted using a true secret key and is placed towards the end of the disk from a secret offset. The sensitive data being protected will be stored in the hidden volume. Note that the entire disk should be filled with random data initially. Upon being coerced by the adversary, to protect the true key, the victim can simply disclose the decoy key. Using the decoy key, the adversary can decrypt the public volume but cannot detect the existence of the hidden volume, as he/she cannot differentiate the encrypted hidden volume from the randomness filled initially. Correspondingly, a public mode and a hidden mode are introduced to manage the two volumes. When working in the public mode, the user can manage (e.g., read, write, process) public non-sensitive data stored in the public volume; when working in the hidden mode, the user can manage (e.g., read, write, process) the sensitive data.

3 SYSTEM AND ADVERSARIAL MODEL

System Model. We consider a mobile computing device which: 1) is equipped with a flash-based block device (e.g., a microSD card or an eMMC card which manages *NAND* via FTL); and 2) has built-in TrustZone support and the ATF is the most recent version. Note that the use of TrustZone and the secure OS itself does not compromise deniability as it is common in real world. We assume the TrustZone itself is secure, a reasonable assumption in the domain of TrustZone technologies [20, 22].

Adversarial Model. We consider a computationally bounded adversary which can capture a victim mobile device, having access to the disk and the memory of the device; in addition, the adversary can capture the device's owner, forcing him/her to decrypt the system.

The adversary is able to take a snapshot of the target device upon capturing it. Each snapshot can include information from both the storage medium and the memory. The snapshot on the storage medium can be the physical image of raw NAND flash, obtainable by forensic data recovery tools [10]. Unlike prior works [13, 24, 34, 36], we set no limitation on the number of times the adversary can observe the victim mobile device [9, 32]. Henceforth, the adversary can either perform a single-snapshot attack or multi-snapshot attack, such as the Evil Maid attack [3] where the adversary periodically obtains multiple copies of the hard disk or memory of the targeted device.

The users might switch between public and hidden mode often and, henceforth, sensitive information might reside in device memory and cache. However, we assume the device owner will not work in the secure mode upon being captured by the adversary.

4 DENIABILITY COMPROMISES

Deniability Compromises in The Flash Layer. Most existing PDE systems are specifically built for the block layer [5, 9, 11–14, 29, 33–36]. But they suffer from deniability compromise if the adversary can have access to the underlying flash memory. The unique nature of flash memory requires a special internal management (usually implemented in FTL), which leads to a different view of data on the flash memory than that on the block layer. By having access to the flash memory [10], the adversary may be able to observe unexpected footprints of hidden sensitive data, compromising deniability. Moving the hidden volumes mechanism to the

flash layer [10] may mitigate this compromise. However, the design is still vulnerable to a multi-snapshot adversary which can access the flash memory at different points of time. The multi-snapshot adversary can have access to the flash memory multiple times over time and, by comparing different snapshots captured, it can detect unaccountable changes on the random data caused by writes performed by the hidden mode, compromising deniability.

Deniability Compromises in The OS Layer. Existing PDE systems that defend against multi-snapshot adversaries like HIVE [9] and DEFY [32] suffer from side-channel attacks [21, 34] including leakage from the OS or applications. The fundamental reason for the leakage is because public and hidden modes share the system software (i.e., OS) and applications. Suppose a user opened a secret file in the hidden mode, and edited the file in her device; then she switched the device from the hidden mode to the public mode before crossing the border for inspection. If a customs officer notices that a secret file shortcut is stored, this shows significant evidence that hidden data exists. Additionally, besides the leakage caused by OSes like Windows, applications could generate traces that lead to deniability compromise as well. For example, Microsoft Word has an auto backup function that prevents data loss. Supposing a user opened MS word to edit a secret file in the hidden mode, the data might be saved to folders that are in the public volume due to the auto backup function. Though the saved backup files normally are deleted after the editing finishes, researchers have shown that these files can be recovered from the Word auto-recovery folder [2]. Additionally, shared applications used in hidden mode might leave footprints in memory since recent PDE systems [12–14] switch modes without a power interrupt, and sensitive data may remain in memory. We argue that, due to the shared software (deniability-unaware), the operations on the hidden data leave footprints/artifacts in the public mode [21], leading to a compromise of deniability.

5 OUR DESIGN

Design Overview. As shown in Figure 4, our PDE system contains two levels of deniability, the flash layer (the *NAND* flash), protected by the FTL, and the OS layer (the application and the OS), protected by the TrustZone.

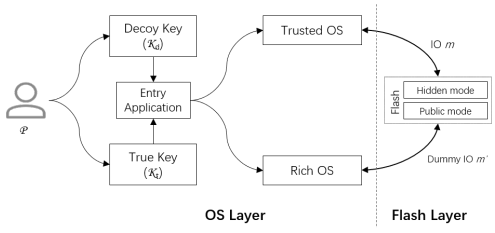

Figure 4: The system structure of our PDE system. The user $\mathcal{P}$ has two keys, the true key $\mathcal{K}_t$ to encrypt and decrypt the sensitive data m in the Trusted OS, and the decoy key $\mathcal{K}_d$ to deal with the decoy data m' in the Rich OS. The flash has two mode, the public mode and the hidden mode.

The flash layer is divided into two modes, the public mode and the hidden mode. When the flash layer is in the public mode, it only processes ordinary data request, denoted as `m`, from the rich OS. Once the flash layer is in hidden mode, it handles data operations, denoted as `m'` from TA. Furthermore, to isolate the operation on `m'`, we design two mapping table into FTL, the mapping table for `m` is *MAP-PUBLIC*, and the other one to handle `m'` is *MAP-HIDDEN*. two tables work separately and are transparent to each other. Figure 5 shows an example of how those two mapping tables in the FTL.

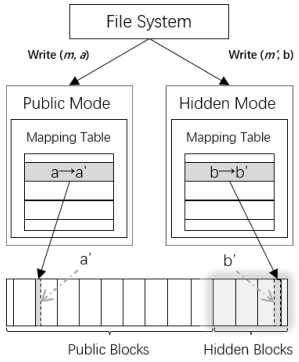

Figure 5: The raw flash blocks are divided into two parts, the public blocks and the reserved hidden blocks. Correspondingly, FTL has two modes, the public mode and the hidden mode. While in the public mode, the mapping table only maps the pages of the Public Blocks. The hidden blocks can only be accessed in the hidden mode.

The OS layer contains two OSes, a rich OS, Android for example, to run the ordinary applications, and a trusted OS in the TrustZone to run the trusted application (TA) of the PDE (see more detail in Figure 3). The device owner creates two keys to switch between these two OSes, one true key $\mathcal{K}_t$ used to encrypt and decrypt the sensitive data, and one decoy key $\mathcal{K}_d$ to deceive the adversary. The entry point is the place where the user types in the keys.

In the following, we elaborate our design details for the flash layer and the OS layer, respectively. Note that a portion of the design of the flash layer appeared as a preprint in arXiv [16].

5.1 Deniability in Flash Layer

To combat the multi-snapshot attacks using raw flash memory, we need to carefully modify the FTL to support PDE as follows: 1) Two modes, a public and a hidden mode, are incorporated into the FTL. When the system enters the public mode, the user can write public non-sensitive data and, when the system enters the hidden mode, the user can write hidden sensitive data. For each mode, the system should maintain a table which keeps mappings between LBAs and PBAs. Therefore, there should be two independent mapping tables in the FTL, *MAP-PUBLIC* for the public mode and *MAP-HIDDEN* for the hidden mode. 2) Upon performing public data writes, the FTL will perform additional dummy writes of random data. Deniability can be achieved since the encrypted sensitive data cannot be differentiated from randomness created by dummy writes and, even though the multi-snapshot adversary can notice change of randomness across flash memory, it can be denied as created by new dummy writes. One security issue here is that, rarely, there are

no public data writes between the two snapshots captured by the adversary. In this case, if any hidden writes are performed between the two snapshots, the adversary will be clear that uncountable changes are caused by the hidden writes, compromising deniability. To avoid this compromise, the FTL should actively perform a few dummy writes if there are no public data writes for a long time. 3) The FTL places all the data (resulted from public, dummy and hidden writes) randomly to flash memory. The random placement technique is feasible for flash memory, because random seeks on flash memory are as efficient as sequential seeks. In addition, random placements inherently distribute data evenly among flash, naturally achieving good wear leveling [17]. Finally, public and hidden data can be easily located by the user using *MAP-PUBLIC* and *MAP-HIDDEN*. A few additional issues are addressed as follows:

Hiding The Hidden Mode Mapping Table. *MAP-HIDDEN* can be stored encrypted (using the true key) and stored in a few random locations. The existence of flash pages storing the encrypted *MAP-HIDDEN* can be denied as storing dummy data. These random locations can be derived from the true key only known to the hidden mode. Once the user enters the hidden mode, the FTL will use the true key to compute and locate flash pages storing *MAP-HIDDEN*, decrypt *MAP-HIDDEN*, and identify all flash pages storing hidden data.

Preventing Overwriting on Hidden Data. Note that for deniability purposes, the public mode should not be aware of the existence of the hidden mode; otherwise, the adversary who can have access to the public mode will trivially know the existence of hidden sensitive data. To address this issue, a global bitmap should be introduced to the FTL to keep track of the usage of flash pages. Specifically, once a flash page is used (by either public, dummy or hidden data), the page will be marked as used in the global bitmap. This would not be vulnerable to the multi-snapshot adversary, since the flash pages used by the hidden data can be denied as used by the dummy data. Another issue is how to store and maintain the global bitmap. Storing the bitmap in the flash memory may be problematic since flash memory is updated unfriendly, but the bitmap needs to be updated frequently in order to keep track of page usage. Storing the bitmap in memory can solve the problem. However, the information storing in memory will suffer from loss upon power failures. A solution could be: when the mobile device is powered off normally, the bitmap in memory will be committed to flash memory; when the mobile device encounters a sudden power failure, the FTL will perform a full disk scan to reconstruct the bitmap. The user should be involved and provide both the decoy key and the true key in order to localize the public and hidden data, reconstructing the portion of the bitmap for them. There is no need to reconstruct the portion of the bitmap for the dummy data and the corresponding flash pages can be reused. Since the power failure is a rare event, the user involvement will not create usability issues.

Preventing Garbage Collection on Hidden Data. Garbage collection reclaims the space occupied by dummy data periodically to prevent disk from filled by dummy data. Note that during each garbage collection, the FTL should only reclaim a random portion of flash pages occupied by dummy data to avoid deniability compromise. The challenge is, the public mode cannot differentiate randomness created by the hidden writes from that created by the dummy writes. Having observed that the hidden mode knows where are the actual dummy data, a promising strategy could be using a "Lazy" garbage collection in the public mode while using an active garbage collection in the hidden mode. Specifically, the public mode will not reclaim space occupied by dummy data if flash memory load factor (i.e., percentage of the entire space being used) does not exceed a large threshold and, when the user enters the hidden mode, the FTL will actively reclaim flash blocks occupied by dummy data and invalid data. To avoid significantly impact the performance of the hidden mode, the garbage collection could be performed during idle time of the hidden mode.

Due to special physical designs of flash memory, other unique sources for flash memory that causes deniability compromise could be present. One compromise comes from the program-erase (P/E) cycle-based design. Since each flash block has a limited number of P/E cycles, various flash functions like wear leveling, garbage collection, bad block management are performed based on measuring the programs/erasures on the blocks. For example, wear leveling [15] usually requires keeping track of P/E cycles on each flash block, and swaps blocks with large accumulative P/E cycles with those with small accumulative P/E cycles. A PDE system has an additional hidden mode for managing hidden sensitive data, which may significantly increase P/E cycles on flash blocks compared to a regular system without PDE. By comparing P/E cycles among different snapshots, the adversary may observe this abnormality, suspecting the existence of PDE. One mitigation could be leveraging [18], which does not keep track of P/E cycles during the device's lifetime, but only during a short period (i.e., an epoch).

Another potential compromise comes from OOB, a special area in each flag page used to store error correction code (ECC), bad block information, and file system-dependent data [4]. Since dummy data created in the public mode are useless, computing error-correcting code for them could be avoided to remove unnecessary computational overhead. However, without computing error code for dummy data, the adversary may differentiate flash pages storing dummy data from that storing hidden sensitive data by checking whether the ECC relationship holds in each page, compromising deniability. A defense for this compromise is: For hidden sensitive data, the ECC is computed over the plaintext, rather than the ciphertext; for dummy data, the ECC could simply be randomness (i.e., no need to compute the actual ECC). In this way, the ECC relationship will not hold for both the dummy data and the encrypted hidden data, and the aforementioned attack cannot be conducted.

5.2 Deniability in OS Layer

To achieve high-level deniability, the system and applications running in the normal world should be unaware of the existence of the PDE system. However, a PDE system that runs in the normal world will inevitably leave footprints in the register, cache, memory, and system log. Therefore, we need to isolate the operation of the PDE system from the normal world. We leverage ARM TrustZone technology as an isolated execution environment for the hidden mode. Specifically, we install two OSs on a target mobile device. One OS, the rich OS, will be installed for normal use in the public mode; the other OS, the trusted OS, will be installed in TrustZone for executing sensitive workloads of the hidden mode.

Isolation on Execution. ARM TrustZone provides an isolated execution environment, so system software and applications running in TrustZone will be isolated from the normal OS and applications. Additionally, the trusted OS and applications running in TrustZone are PDE-aware, not leaving any footprints in the public mode.

Isolation on Memory and Cache. The memory regions of the normal and secure domains are also hardware-enforced isolated, such that the trusted OS and PDE applications for hidden mode will only be executed within the secure memory, which is inaccessible from the OS of the public mode. The NS attribute is a TrustZone extension to the MMU. It defines if the targeted memory region corresponding to the page is Secure or Non-secure, that is, if this memory region is accessed with Secure or with Non-secure rights. This bit is ignored in the Non-secure world. While in the secure world, if the NS Attribute is set to 1, the access is performed with Non-secure rights. Trustzone guarantees that applications running in the REE cannot access cache that is tagged in a secure state. With this mechanism, only the Secure world can perform Secure accesses, and consequently is the only one permitted to access Secure memory. The Secure world can also access Non-secure memory by setting the NS Attribute appropriately in the corresponding descriptor. The Non-secure world can only access Non-secure memory. [7]

The trusted OS and PDE applications are only allowed to save the data into the hidden volume, which is achieved by our modified FTL. In this case, all the saved data in the hidden mode is not aware by the rich OS and software running in the public mode.

Isolation on Mode Switching. Switching from the public mode to hidden mode does not have deniability concerns because the execution the environment becomes the hidden mode, and as we assume in Section 3 that the device owner will not work in the hidden mode under risk. Since all the required secure system software are pre-installed in the TrustZone, the time for mode switching will be the TrustZone domain switching time plus the operation that unmounts public volume and initializes the hidden volume.

6 CONCLUSION

In this work, we have proposed a preliminary design of FTL and Arm TrustZone enhanced plausibly deniable encryption system for mobile computing devices. The proposed PDE system contains two layers of deniability, the flash layer and the OS layer. In the flash layer, we strictly isolate the data stored in the public and the hidden volume. In the OS layer, we install two OSes, one rich OS installed in the normal world and another trusted OS installed in the TrustZone. To the best of our knowledge, ours is the first mobile PDE system which can simultaneously ensure deniability in both the lower storage layer and the OS layer.

REFERENCES

[1] [n. d.]. ARM White Paper, The ARM Architecture Version 6 (ARMv6)A. http://lars.nocrew.org/computers/processors/ARM/ARMv6.pdf. Accessed: 2021-08-23.

[2] [n. d.]. DataRecovery: freeware and made by TOKIWA. http://tokiwa.qee.jp/EN/dr.html/. Accessed: 2021-08-23.

[3] [n. d.]. Evil Maid goes after TrueCrypt. http://theinvisiblethings.blogspot.com/2009/10/evil-maid-goes-after-truecrypt.html. Accessed: 2021-09-15.

[4] [n. d.]. Memory technology device (mtd) subsystem for linuxA. http://www.linux-mtd.infradead.org/doc/nand.html. Accessed: 2021-08-23.

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

[6] Apple. 2013. Apple FileVault. https://support.apple.com/en-us/HT204837. Accessed: 2021-08-23.

[7] ARM. [n. d.]. NS attribute. https://developer.arm.com/documentation/ddi0301/h/memory-management-unit/memory-region-attributes/ns-attribute. Accessed: 2020-04-15.

[8] ARM-software. [n. d.]. Arm Trusted Firmware. https://github.com/ARM-software/arm-trusted-firmware. Accessed: 2021-08-23.

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

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

[11] Anrin Chakraborti, Chen Chen, and Radu Sion. 2017. Datalair: Efficient block storage with plausible deniability against multi-snapshot adversaries. *arXiv preprint arXiv:1706.10276* (2017).

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

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

[14] Bing Chang, Fengwei Zhang, Bo Chen, Yingjiu Li, Wen-Tao Zhu, Yangguang Tian, Zhan Wang, and Albert Ching. 2018. 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)*. IEEE, 454–465.

[15] Li-Pin Chang. 2007. On efficient wear leveling for large-scale flash-memory storage systems. In *Proceedings of the 2007 ACM symposium on Applied computing*. 1126–1130.

[16] Bo Chen. 2020. Towards Designing A Secure Plausibly Deniable System for Mobile Devices against Multi-snapshot Adversaries–A Preliminary Design. *arXiv preprint arXiv:2002.02379* (2020).

[17] Bo Chen, Shijie Jia, Luning Xia, and Peng Liu. 2016. Sanitizing data is not enough! Towards sanitizing structural artifacts in flash media. In *Proceedings of the 32nd Annual Conference on Computer Security Applications*. 496–507.

[18] Bo Chen and Radu Sion. 2015. Hiflash: A history independent flash device. *arXiv preprint arXiv:1511.05180* (2015).

[19] Niusen Chen, Bo Chen, and Weisong Shi. 2021. MobiWear: A Plausibly Deniable Encryption System for Wearable Mobile Devices. In *EAI International Conference on Applied Cryptography in Computer and Communications*. Springer, 138–154.

[20] Niusen Chen, Wen Xie, and Bo Chen. 2021. Combating the OS-Level Malware in Mobile Devices by Leveraging Isolation and Steganography. In *International Conference on Applied Cryptography and Network Security*. Springer, 397–413.

[21] Alexei Czeskis, David J St Hilaire, Karl Koscher, Steven D Gribble, Tadayoshi Kohno, and Bruce Schneier. 2008. Defeating Encrypted and Deniable File Systems: TrueCrypt v5. 1a and the Case of the Tattling OS and Applications.. In *HotSec*.

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

[23] Aayush Gupta, Youngjae Kim, and Bhuvan Urgaonkar. 2009. DFTL: a flash translation layer employing demand-based selective caching of page-level address mappings. *Acm Sigplan Notices* 44, 3 (2009), 229–240.

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

[25] Ralf P. Weinmann Julian Assange and Suelette Dreyfus. [n. d.]. Rubberhose Filesystem. https://github.com/sporkexec/rubberhose. Accessed: 2021-08-23.

[26] Jeong-Uk Kang, Heeseung Jo, Jin-Soo Kim, and Joonwon Lee. 2006. A superblock-based flash translation layer for NAND flash memory. In *Proceedings of the 6th ACM & IEEE International conference on Embedded software*. 161–170.

[27] Jesung Kim, Jong Min Kim, Sam H Noh, Sang Lyul Min, and Yookun Cho. 2002. A space-efficient flash translation layer for compactflash systems. *IEEE Transactions on Consumer Electronics* 48, 2 (2002), 366–375.

[28] Sang-Won Lee, Dong-Joo Park, Tae-Sun Chung, Dong-Ho Lee, Sangwon Park, and Ha-Joo Song. 2007. A log buffer-based flash translation layer using fully-associative sector translation. *ACM Transactions on Embedded Computing Systems (TECS)* 6, 3 (2007), 18–es.

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

[30] Microsof. [n. d.]. BitLocker. https://docs.microsoft.com/en-us/previous-versions/windows/it-pro/windows-server-2012-R2-and-2012/hh831713(v=ws.11)?redirectedfrom=MSDN. Accessed: 2021-08-23.

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

[32] Timothy M Peters, Mark A Gondree, and Zachary NJ Peterson. 2015. DEFY: A deniable, encrypted file system for log-structured storage. (2015).
[33] Adam Skillen and Mohammad Mannan. 2013. Mobiflage: Deniable storage encryptionfor mobile devices. *IEEE Transactions on Dependable and Secure Computing* 11, 3 (2013), 224–237.
[34] Adam Skillen and Mohammad Mannan. 2013. On implementing deniable storage encryption for mobile devices. (2013).
[35] TrueCrypt. [n. d.]. Free open source on-the-fly disk encryption software.version 7.1a. http://www.truecrypt.org/. Accessed: 2021-08-23.
[36] Xingjie Yu, Bo Chen, Zhan Wang, Bing Chang, Wen Tao Zhu, and Jiwu Jing. 2014. Mobihydra: Pragmatic and multi-level plausibly deniable encryption storage for mobile devices. In *International conference on information security*. Springer, 555–567.
[37] Lianying Zhao and Mohammad Mannan. 2015. Gracewipe: Secure and Verifiable Deletion under Coercion.. In *NDSS*.