FSPDE: A Full Stack Plausibly Deniable Encryption System for Mobile Devices

Jinghui Liao
Wayne State University
United States
jinghui@wayne.edu

Niusen Chen
Michigan Technological University
United States
niusenc@mtu.edu

Lichen Xia
University of Delaware
United States
lxia@udel.edu

Bo Chen
Michigan Technological University
United States
bchen@mtu.edu

Weisong Shi
University of Delaware
United States
weisong@udel.edu

ABSTRACT

In today's digital landscape, the ubiquity of mobile devices underscores the urgent need for stringent security protocols in both data transmission and storage. Plausibly deniable encryption (PDE) stands out as a pivotal solution, particularly in jurisdictions marked by rigorous regulations or increased vulnerabilities of personal data. However, the existing PDE systems for mobile platforms have evident limitations. These include vulnerabilities to multi-snapshot attacks over RAM and flash memory, an undue dependence on non-secure operating systems, traceable PDE entry point, and a conspicuous PDE application prone to reverse engineering.

To address these limitations, we have introduced FSPDE, the first Full-Stack mobile PDE system design which can mitigate PDE compromises present at both the execution and the storage layers of mobile stack as well as the cross-layer communication. Utilizing the resilient security features of ARM TrustZone and collaborating multiple storage sub-layers (block device, flash translation layer, etc.), FSPDE offers a suite of improvements. At the heart of our design, the `MUTE` and `MIST` protocols serve both as fortifications against emerging threats and as tools to mask sensitive data, including the PDE access point. A real-world prototype of FSPDE was developed using OP-TEE, a leading open-source Trusted Execution Environment, in tandem with an open-sourced NAND flash controller. Security analysis and experimental evaluations justify both the security and the practicality of our design.

CCS CONCEPTS

• **Security and privacy → Mobile platform security**.

KEYWORDS

Plausibly Deniable Encryption, Mobile Devices, TrustZone, Flash Translation Layer, Full Stack

ACM Reference Format:
Jinghui Liao, Niusen Chen, Lichen Xia, Bo Chen, and Weisong Shi. 2024. FSPDE: A Full Stack Plausibly Deniable Encryption System for Mobile Devices. In *Proceedings of the Fourteenth ACM Conference on Data and Application Security and Privacy (CODASPY '24), June 19–21, 2024, Porto, Portugal*. ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3626232.3653262

CODASPY '24, June 19–21, 2024, Porto, Portugal

ACM ISBN 979-8-4007-0421-5/24/06
https://doi.org/10.1145/3626232.3653262

1 INTRODUCTION

Mobile computing devices, such as smartphones and tablets, have become deeply integrated into modern life for communication, work, healthcare, finance, and other sensitive tasks [34]. Because of their mobility and accessibility, mobile devices today emerge as the optimal tools for individuals to record crucial information, such as criminal evidence or even life-threatening data. For example, a human right worker may use his/her smartphone to capture the brutal activities in a country of conflict or oppression and cross the border while the evidence is stored encrypted in the phone; however, a border inspector may notice the encrypted ciphertext and coerce the victim to disclose the decryption key. Traditional encryption methods, such as full disk encryption, are insufficient when faced with coercive attackers. Once encryption keys are revealed, either voluntarily or involuntarily, traditional encryption schemes do not provide any residual protections for the confidentiality and privacy of user data [32].

To address this significant vulnerability, plausibly deniable encryption (PDE) has emerged as a promising approach to improve security in scenarios that involve coercion attacks, involuntary key disclosure, or unauthorized access [38]. PDE provides an additional layer of defense by empowering users to credibly deny the existence of encrypted sensitive data even when adversaries actively compromise encryption keys or passwords using coercion, threats, or involuntary extraction techniques [9]. This is achieved by enabling the decryption of ciphertext into plausible, benign decoys using different keys, preventing adversaries from definitively proving the presence of any secret encrypted content [36].

PDE systems typically rely on hidden volumes or advanced steganographic techniques to effectively conceal sensitive encrypted data (accessible in the *hidden mode*) seamlessly among benign random decoy content (accessible in the *public mode*). For example, a 10 GB disk may have a public volume of 10 GB encrypted with a decoy key, while stealthily embedding a hidden 5 GB volume starting from a secret offset encrypted with a hidden key [4]. The entire storage is initially filled with random data, while the hidden volume resides silently among the random data. If asked coercively

to reveal the key, the user provides the decoy key, allowing access to only the public volume. With no means to definitively prove the presence of a hidden volume, the user can plausibly deny its existence.

Prior PDE systems [12, 13, 15, 16, 20, 26, 29, 36, 38, 43] designed for mobile platforms have made notable advances, but still suffer from significant limitations that severely hinder their effectiveness and mainstream adoption on today's mobile systems. Such limitations include: **(L1)** All of them remain vulnerable to side-channel attacks arising from insufficient isolation between non-sensitive public data and sensitive hidden data in concealed operation modes, resulting in inadvertent information leakage that betrays the presence of hidden data protected by PDE. For example, shared caches or memory pages accessed during hidden-mode operations can retain artifacts detectable in the public mode through cache timing or memory inspection attacks. **(L2)** Naively designed communication mechanisms between the PDE system and the block device involve passing through the unsecured OS, which can unintentionally leak sensitive timing or protocol patterns that betray the existence of hidden modes or data [32]. **(L3)** Attackers can often access the executable binary files that hide the PDE system. This access allows them to reverse-engineer the executable files, potentially uncovering evidence of the PDE's existence. **(L4)** They either 1) simply deploy hidden volumes on the block device [12, 13, 26, 38, 43], remaining intrinsically susceptible to catastrophic deniability compromise from advanced forensic analysis at the raw NAND flash [19, 29], or 2) require significant modifications into the firmware staying inside the flash memory storage [15, 16, 20, 29, 36], rendering it hard to deploy the proposed systems in real-world devices.

To holistically address the aforementioned limitations, we introduce FSPDE, the first Full-Stack mobile PDE system designed from the ground up for modern mobile devices and architectures. FSPDE aims for a *full-stack* PDE system design, covering not only the execution and storage layers of the mobile stack, but also the subsequent cross-layer communication.

At the execution layer, FSPDE uses the hardware isolation provided by ARM TrustZone to robustly isolate the PDE system in a small trusted execution environment (TEE), effectively eliminating side channels and forensic footprints compared to traditional OS-based isolation techniques [8] (addressing **L1**). TrustZone provides isolated secure worlds at the hardware level, perfect for concealing PDE operations from the untrusted public-mode OSes. However, simply using TrustZone does not immediately address all the PDE compromises at the execution layer and extra design considerations are needed. Firstly, traditionally, a PDE system must maintain an entry point in the non-secure world, which could potentially leak the existence of the PDE system during execution. To address this, our design adopts a multi-layered approach within the TrustZone, where the entry point is hidden within the TA (Trusted Application that runs inside the TrustZone) part of a standard application. To log into the PDE system, users must first log into the standard application, which does not contain any PDE logic. Then, they enter the standard application's TA to activate the PDE's entry point and log into the PDE system. This ensures that the entire PDE login process occurs within the TrustZone, avoiding exposure to the PDE entrance. Secondly, the TrustZone (e.g., OP-TEE [35]) typically relies on the OS's filesystem for I/O operations. This dependence means that reading and writing data within the PDE system could leave traces in the non-secure system, potentially revealing the PDE system's existence. To mitigate this, we have implemented a flash driver within the TrustZone for direct interaction with the storage device, allowing data to be read from and written to storage media without going through the system's filesystem (addressing **L2**). Thirdly, the PDE application binary exists in the non-secure world, which means that an adversary may reverse-engineer the binary to identify the existence of PDE. To prevent this attack, the TA binary for PDE will be stored encrypted, with a secret key generated and managed solely by the TrustZone (rather than the user). The encrypted binary cannot be decrypted until it is loaded into the TrustZone upon launching the hidden mode. In this manner, the reverse-engineering attack vector over the PDE binary can be disabled reasonably (addressing **L3**).

At the storage layer, we need to eliminate any deniability compromises when the adversary can have access to the external storage at different points of time (addressing **L4**). The access to the storage mainly results in the adversarial peeking over the file data, the block device data and the flash memory raw data. Due to the hardware-level isolation of the hidden mode by the TrustZone, the adversary will not notice the existence of this mode, and hence will not gain access to any file data belonging to this sensitive mode. To prevent deniability compromises from peeking the block device in the public mode, a random placement together with a "dummy-write" mechanism [13] is adopted. The sensitive data are encrypted and stored hidden among the dummy data. Even if the adversary can obtain multiple snapshots of the block device over time, it cannot notice the existence of the sensitive data due to the "dynamic obfucation" brought by the random placements of the public data and the associated dummy data. Due to the introduction of the dummy writes on the block device, the encrypted sensitive data are inherently hidden among the dummy data in the raw flash memory. However, upon peeking the underlying raw flash memory, the adversary may identify deniability compromises as the log-structured writing [36] implemented in most flash storage devices essentially converts the random placements on the block device to sequential placements on the raw flash. The remediation is to simply introduce a random placement strategy into the flash translation layer (FTL), a firmware layer staying between the block device and the raw flash memory in mainstream flash storage devices. This only requires a minimal modification into the FTL to achieve the desired deniability as now the random placement and the "dummy-write" mechanism are both present in the flash memory.

To implement the aforementioned ideas, we have introduced two key protocols, `MIST` and `MUTE`. `MIST` operates fully within the TrustZone environment to avoid leaving footprints of hidden data in the memory of the untrusted system. It cleverly intersperses hidden data among dummy data blocks, recorded in a mapping table secured in TrustZone. All data are uniformly encrypted. This provides an efficient and low overhead mechanism to conceal hidden data from being learned by inspecting the external storage. `MUTE` (Section 5) protocol obscures the PDE entry point. It bifurcates operations between the rich execution environment (REE) and the trusted execution environment (TEE). The entry point in

the REE is designed to mimic normal software. After hiding-key validation in the TEE, the secure PDE core is loaded. The protocol also uses dummy applications and encrypted loading to prevent reverse engineering.

Contributions. Our major contributions are summarized below:

- We have identified the limitations of existing PDE systems, such as their vulnerability to deniability compromises at the lower storage layer, susceptibility to side-channel attacks, and challenges associated with cross-layer communication (i.e., the communication between the execution and the storage layer).
- We have proposed FSPDE, a novel multi-layered PDE system that addresses the limitations of current PDE solutions and provides robust protection against various threats, including coercion, side channel attacks, and compromises of deniability due to the cross-layer communication.
- We have designed a unique storage layer, which eliminates potential sources of deniability compromises at various storage sub-layers and provides robust protection against multi-snapshot adversaries.
- We have enhanced execution layer isolation and entry point concealment through innovative protocols like `MUTE` and `MIST` to prevent leakage between public and hidden modes. In addition to that, FSPDE leverages the TEE key to encrypt and decrypt PDE executable files to defend against reverse engineering attacks.
- We have implemented a prototype of FSPDE on real mobile hardware using ARM TrustZone and open-sourced flash memory controllers, demonstrating its practicality.

2 BACKGROUND

Plausibly Deniable Encryption. Plausibly deniable encryption (PDE) is a cryptographic technique designed to protect the confidentiality of sensitive data against a coercive attacker who can force the owner of the data to reveal the decryption key. PDE allows data owners to decrypt the ciphertext into plausible and benign decoy plaintext with a different key, enabling the data owner to deny the existence of the original sensitive data. To provide reasonableness and plausibility, the PDE system usually requires that the decoy plaintext appears as normal data commonly found on a computer and that the entire ciphertext can be taken into account. *Throughout the paper, "hidden data", "PDE data", and "sensitive data" are used interchangeably to represent the original sensitive data. Similarly, "public data", "decoy data", and "non-sensitive data" are also used interchangeably to represent the decoy plaintext.* PDE allows users to convincingly deny the existence of encrypted data, the ability to decrypt the given data, or the existence of specific encrypted data. Such denials may or may not be genuine. PDE can eradicate the attacker's confidence that the data are encrypted or that the owner can decrypt and provide the relevant plaintext.

TrustZone. ARM Holdings, a leading semiconductor manufacturer, introduced TrustZone technology as an integral part of the ARMv6K architecture in 2004 [7], marking a significant step toward enhancing security in microprocessor design. TrustZone signifies an architectural solution that provides a system-wide approach to security. The aim is to create a dedicated, isolated environment within the system-on-chip (SoC), segregated from the primary operating system. TrustZone operates on the principle of partitioning system resources into two distinct worlds, a "secure world" and a "non-secure world". Each world has its own set of resources, including processor modes, memory areas, and peripherals, to name a few. These worlds are designed in such a way that neither can directly access the other's resources, ensuring the integrity of the secure world even if the non-secure world becomes compromised. The secure world can provide a trusted execution environment (TEE) to run sensitive applications while the non-secure world allows running the non-sensitive applications in a rich execution environment (REE). *Throughout the paper, "secure world" and "TEE" are used interchangeably to denote an identical concept, similarly for the "non-secure world" and "REE".* Open portable trusted execution environment (OP-TEE) [35] is an open-source TEE for ARM TrustZone, in which the programs are divided into two privilege layers: the user level, where trusted applications (TA) operate, and the kernel level, which is the domain of pseudo-trusted applications (PTA).

Over the years, TrustZone's comprehensive system-wide security solution has been widely deployed across a variety of devices, such as smartphones, tablets, IoT devices, and smart televisions. Notable applications of TrustZone include securing mobile payment systems, device encryption keys, digital rights management (DRM), and secure boot mechanisms.

NAND Flash. Mainstream mobile computing devices use NAND flash as external storage. NAND flash stores information in an array of memory cells grouped into blocks. Each block contains a few pages, and every page usually has a small spare out-of-band (OOB) area. NAND flash exhibits several unique 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 updates. (4) A flash block has a finite number of program-erase cycles and will be worn out if the number of programs/erasures performed on it exceeds a certain threshold.

Flash Translation Layer. The most popular approach to managing flash memory is to emulate it as a block device using a flash translation layer (FTL). Traditional file systems (e.g., FAT32, EXT4) can be directly implemented on top of the FTL. The FTL is responsible for handling several crucial tasks: (1) Wear leveling: As flash memory cells have a limited endurance for P/E (Program/Erase) cycles, the FTL implements strategies to evenly distribute write operations across the physical memory. This is done to extend the serving life of the flash. (2) Mapping management: Flash memory does not support in-place updates. Therefore, the FTL maintains a mapping table to translate logical block addresses (as viewed by the file system) to physical block addresses in flash memory. (3) Garbage collection: When the data are rewritten, the FTL writes the new data to a new location and marks the old location as invalid. Consequently, numerous invalid pages are generated and dispersed across different blocks over time. To mitigate this, the FTL periodically reclaims these blocks by relocating any valid data to new locations and subsequently erasing the entire block. (4) Bad block management: Over time, certain blocks may become unreliable and therefore are marked as bad. The FTL handles the management of these impaired blocks to ensure that data are not written to them.

Since its inception, the FTL has undergone numerous improvements and optimizations, evolving alongside the flash memory technology itself. Despite this, it continues to provide the same fundamental services, serving as the vital link that allows flash memory to be used in a broad range of applications, from tiny USB drives to enterprise-grade SSDs.

3 SECURITY MODELS AND ASSUMPTIONS

System Model. We consider a mobile computing device which is equipped with an ARM architecture processor with TrustZone enabled. A flash memory-based storage device is utilized as external storage. Flash memory is managed by the flash translation layer (FTL), which transparently handles the unique characteristics of flash memory hardware, exposing a block device access interface to the operating system.

Adversarial Model. We consider a computationally bounded adversary. The adversary is capable of capturing a victim user, along with his/her computing device, at various intervals over the course of time, i.e., a *multi-snapshot adversary*. Upon capturing a victim device, the adversary can have access to both its external storage and internal memory and may coerce the device owner to reveal the decryption key for data. The access to the external storage can cover different storage sub-layers including application, block device, and raw flash memory. The adversary may play with the victim device and conduct forensic analysis over the storage and the memory. The adversary may also perform reverse engineering over the binary files in the victim device.

Assumptions. Mobile devices are ubiquitously equipped with ARM TrustZone and, using TrustZone and the secure OS itself will not raise any concerns about deniability, as they are widely used for confidential computing applications [1]. We assume that TrustZone is secure, which is a reasonable assumption in the domain of TrustZone technologies [21, 28]. We also assume that TrustZone is capable of protecting trusted application (TA) programs, aligning with other TEE-assisted proposals [22, 25, 31, 33]. We also assume that the encryption scheme we use is secure. In addition, we rely on a few additional assumptions that are common in the designs of the PDE systems [29, 38, 43]:

- The adversary is rational and will cease to coerce the device owner upon being convinced that the decryption key is disclosed.
- Upon being captured by the adversary, the victim user will not process hidden sensitive data. In addition, the user will refrain from logging into the hidden mode in the presence of the adversary. This assumption is necessary since the adversary will immediately compromise the deniability if the victim is found processing the hidden sensitive data or running the hidden mode.
- The mobile device is assumed to use system-on-chip, where the adversary cannot get physical access to the internal memory by cracking the chip.
- The mobile OS, kernel, and bootloader are assumed to be malware-free. In addition, the victim user does not use malicious applications controlled by the adversary. In other words, the attacks on the PDE only happen when the adversary captures the victim device. This assumption is necessary. Otherwise, by stealthily monitoring the system, the adversary may immediately compromise the deniability when the user processes the hidden sensitive data. This assumption is not unreasonable in practice, considering the PDE user is typically cautious, who will periodically use anti-virus tools to scan his/her device. In addition, the cautious user usually will not use unknown/untrusted apps.
- The user will only process the hidden sensitive data in the hidden mode, as otherwise, the hidden sensitive data may be present in the public mode which is hard to be denied.

4 OVERVIEW OF FSPDE

4.1 Design Goals

The design goals of FSPDE are elaborated below:

- **Goal 1: Resistance to Reverse Engineering Attacks**. The binary files associated with the PDE system must exhibit a robust defense against reverse engineering attacks, ensuring that even if the binary files are accessed by the adversaries, the underlying logic and functionality of the PDE system remain secure and impenetrable (analyzed in Section 7.1).
- **Goal 2: Concealed Entry Point**. The entry point of the PDE system should be concealed effectively so that it cannot be uncovered, even through reverse engineering (analyzed Section 7.2).
- **Goal 3: Resistance to Multi-Snapshot Attacks**. The PDE system should be able to resist against the multi-snapshot adversary, which can have access to both the volatile memory (RAM) and the non-volatile external storage multiple times over time (analyzed in Section 7.3).
- **Goal 4: Fast Mode Switching**. The design should support rapid transitions between the hidden mode and the public mode, accommodating the dynamic operational needs of mobile users (analyzed in Section 7.4).

4.2 Design Overview

The overall design of FSPDE is illustrated in Figure 1. It comprises two deniability layers: the execution layer, protected by TrustZone, and the storage layer, protected by the FTL-assisted dummy writes. The device owner is represented here as $\mathcal{P}$, who possesses two keys: 1) a secure key $\mathcal{K}_s$, which allows access to the secure OS to handle hidden data, and 2) a decoy key $\mathcal{K}_d$, which allows access to the non-secure OS to handle decoy data. A salient feature of FSPDE is its sophisticated data storage strategy. When the non-secure OS writes decoy data (m'), it concurrently disperses random dummy data (m_d) across various unused storage blocks. On the contrary, the secure OS embeds hidden data (m) into blocks already populated with m_d. In other words, the hidden data m is plausibly denied using the dummy data m_d.

FTL-assisted Dummy Writes. In the public mode, decoy data is encrypted via a decoy key and randomly placed to the block device. To confuse the attacker, a dummy write is performed with a certain probability when a data block is allocated to the decoy data at the block device. Note that the dummy data can be generated by encrypting garbled data using the same symmetric encryption algorithm which is used to encrypt the hidden data and, the encryption key is discarded once the dummy data are generated. This dummy writes are executed only when Equation 1 [13] is met. Here, x represents a fixed positive integer, while $stored\_rand$ denotes a random number periodically refreshed. Furthermore, $rand$ is an integer

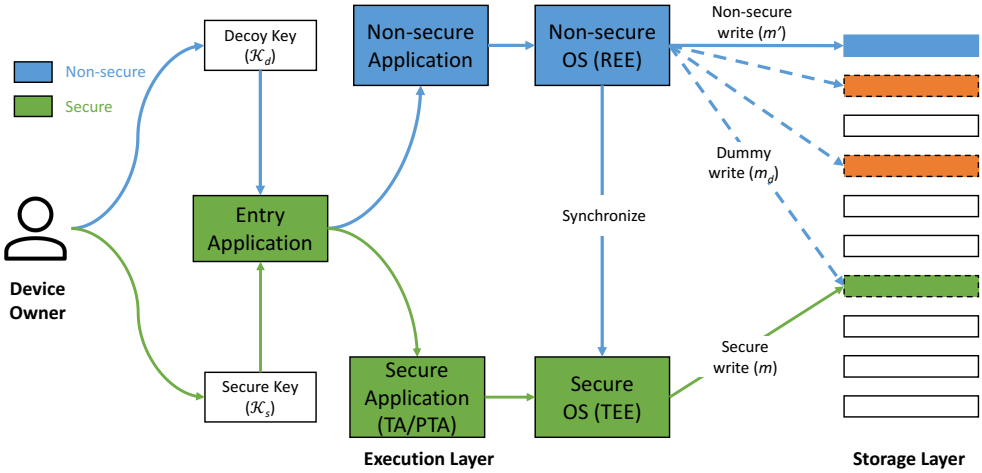

Figure 1: The design overview of FSPDE. The PDE operations in the TEE are shaded in green, while the operations in the REE are shaded in blue. On the data storage, the REE manages the decoy data m' along with the dummy data m_d, while the TEE handles the hidden data m.

randomly chosen from a uniform distribution ranging between 1 and $2 \cdot x$ for each instance of a dummy write. This methodology is implemented to ensure that the probability of executing a dummy write remains consistently below 50%.

$$rand \leq stored\_rand \mod x \tag{1}$$

Once the dummy writes are performed, a total of m blocks are randomly assigned within the layer of the block device. These newly allocated blocks are filled with randomized data. The specific value of m is determined by Equation 2. Furthermore, f represents a randomly generated number within the range of 0 to 1, and λ denotes the rate parameter.

$$m = \lfloor m' \rfloor, \text{ where } m' = -(\ln(1-f))/\lambda \tag{2}$$

The random placements with dummy writes on the block device will not be sufficient if the adversary can peek into the raw flash memory. This is because, the FTL typically implements a log-structured writing mechanism [37] by which the data are written sequentially to the flash memory to accommodate its unique hardware nature (Section 2); even if the data are placed randomly on the block device, the FTL, staying between the block device and the flash hardware, will convert the random pattern on the block device back to a sequential pattern on the raw flash and, such a sequential pattern may lead to deniability compromises if the adversary gains access to the raw flash. For example, the user first writes some decoy data in the pubic mode, and then switches to the hidden mode to write a hidden file; by peeking into the flash memory, the adversary will observe some decoy data together with some dummy data, followed by a large amount of undecryptable ciphertext which is corresponding to the hidden file. In other words, the sequential pattern involuntarily places bounds between the decoy and the hidden data, leaking the existence of the hidden data.

To mitigate this deniability compromise, we modify the log-structured writing in the FTL so that all the data, including decoy, dummy, and hidden data, will instead follow a random writing strategy. Utilizing the random writing is exclusively feasible for the flash memory as addressing in the flash memory is not based on mechanical rotations which discourage random seeking. In addition, the random writing may result in a better wear-leveling effectiveness which is essential for the flash memory. Note that the FTL does not differentiate decoy, dummy, and hidden data. Therefore, only one mapping table is needed to be maintained in the FTL.

Handling Mapping Tables. To avoid the potential overwriting of hidden data by decoy and dummy data, our design maintains three tables on the block device: the *global mapping table*, the *public mapping table*, and the *hidden mapping table*. These tables collectively serve to effectively manage data allocation and preservation. The *global mapping table* keeps track of locations that are occupied by all types of data. The *public mapping table* only records the locations of decoy data. It should be noted that a dedicated mapping table for dummy data is deliberately omitted, as its existence could inadvertently allow the attacker to differentiate the hidden data from the dummy data. Both the *global mapping table* and the *public mapping table* are visible to the adversaries and, FSPDE will transfer them to the TrustZone upon switching to the hidden mode. The *hidden mapping table* is used to keep track of the hidden data at the block layer. This table should be invisible to the adversaries.

PDE Binary Protection (Goal 1). To protect the binary files of the PDE system while concealing its existence, we use symmetric encryption algorithms to encrypt all TA binary files. These files are only decrypted after they are loaded into the TEE. Each TA's encryption and decryption key is uniquely generated from a root key built into the TEE system. When a TA is loaded into the TEE for execution, the TEE recalculates the appropriate sub-key based on the TA's identity and uses this key to decrypt the TA. The encryption and decryption keys originate from the root key within the TEE and are securely managed there.

Key Protocols: `MUTE` (Goal 2) and `MIST` (Goal 3). To ensure full-stack deniability, FSPDE introduces the `MUTE` and `MIST` protocols:

- MUTE Protocol: To protect the PDE system's entry point, the `MUTE` protocol operates across the secure (TEE) and non-secure (REE)

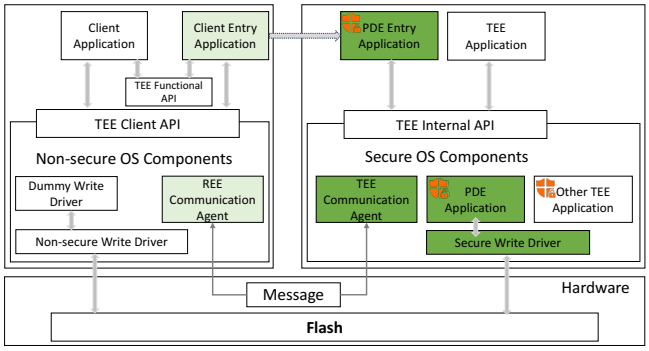

Figure 2: The structure of the MUTE protocol. The shaded in green are the core components of the MUTE protocol, while the light green parts are the MUTE components that reside in the non-secure OS. Note that all TAs are encrypted before being loaded into Trustzone.

environments. In the secure environment, it includes specialized applications within TrustZone, while appearing as conventional software in the non-secure environment, devoid of any PDE-related indications. This design effectively shields against memory attack strategies and reverse engineering, ensuring that PDE entry remains undetected and secure. The MUTE protocol is elaborated in Section 5.

- MIST Protocol: Designed for managing hidden data in the mobile devices, the MIST protocol leverages both theTrustZone and the FTL technologies. It segregates the hidden data operations from non-secure systems, ensuring that no residual traces in the insecure memory. The protocol classifies data into public, dummy, and hidden categories, ingeniously blending hidden data within dummy data blocks for concealment. In addition, the MIST protocol incorporates a PDE Mapping Table to link hidden data to dummy data, and an FTL Mapping Table to manage physical and virtual flash addresses. The MIST protocol is elaborated in Section 6.

5 HIDING THE PDE SYSTEM VIA MUTE

The obfuscation of the PDE entry point serves as an imperative shield against the coercive adversaries. In this light, the MUTE protocol, as depicted in Figure 2, is presented as an integral component of FSPDE. Its primary objective is to obscure the entrance of the PDE system, rendering potential reverse engineering efforts futile.

The MUTE Protocol Paradigm. The MUTE protocol bifurcates its operations into two realms: the secure world, executed within TrustZone or TEE, and the non-secure world, operating within the REE.

- **Secure World (TEE):**
  - *PDE Entry Application (TA):* Resides and functions within the TrustZone user stratum.
  - *PDE Core Program (PTA):* Manifests within the TrustZone system layer.
  - *Other Dummy Application (PTA):* Ensures the holistic operation of the system.
- **Non-secure World (REE):**
  - *Entry Application:* Externally, it appears as standard software devoid of any PDE-specific code or data.

Protocol Description. Algorithm 1 shows the MUTE protocol detail. In the context of FSPDE, the user initiates access by launching an entry program in REE and entering a specific secret key. Externally, this program appears as standard software devoid of any overt PDE-related information. Upon key entry, the system initiates a transition that leads to the secure *TrustZone* execution environment. Here, the PDE entry program takes over the task of verifying keys. If the key is verified and matches the key designed for the PDE system, the system invokes the *OP-TEE* environment to activate and run the PDE's PTA, thereby initiating the core functionalities of the PDE system. Conversely, if the key is recognized as a dummy (decoy), the system opts to load the dummy PTA, providing a non-PDE response as a decoy to mislead potential attackers. Crucially, all operations within the TrustZone are performed under strict encryption, ensuring that TA and PTA are decrypted and executed exclusively within this secure space, adding a layer of protection against reverse engineering. Simultaneously, the absence of PDE-related code in the REE entry program further ensures the system's resilience against reverse detection. This sequential and integrated operation flow guarantees the integrity and security of FSPDE.

Algorithm 1 MUTE protocol for PDE loading

```
Initialization:
Encrypt both the Trusted Application (TA) and the Pseudo-Trusted Application (PTA) securely.
Launch the Entry Application within the REE.
Prompt the user to enter the privileged key.
if the provided key is valid then
    Transition the environment to TEE.
    Decrypt the TA and PTA.
    Verify the privileged key through the PDE Entry Application.
    if the key corresponds to the PDE then
        Engage the PDE Core Program as a PTA for execution.
    else if the key is associated with a dummy request then
        Activate Dummy Application as a PTA.
    end if
    Upon task completion, transition back to REE.
else
    Remain within the confines of the Entry Application and notify the user.
end if
```

6 SECURING DATA VIA MIST

Previous PDE systems [9, 13, 14, 30] that utilized dummy write operations segmented data into different partitions. Then these PDE partitions were concealed within partitions dedicated to dummy writes. However, such obfuscation methods heavily relied on full-disk encryption techniques, leading to significant system overhead. Furthermore, performing these complex operations directly in a non-secure environment inevitably generated numerous memory traces that were susceptible to analysis, increasing the risk of PDE system files being detected. To address these challenges, we propose the "MIST Protocol", denoted as $\Pi_{\texttt{MIST}}$, which uses TrustZone and FTL to provide an efficient and low overhead mechanism to handle hidden data. The protocol ensures complete decoupling of hidden data operations from non-secure systems, guaranteeing that no footprints or traces of hidden data operations remain in insecure systems or memory. Figure 3 illustrates the architecture of the MIST protocol. In this section, we will dive into the details of our MIST protocol.

Due to the prevalence of mobile devices based on the ARM architecture and that utilize flash storage embedded with the FTL, an innovative approach to PDE is needed that ensures optimal

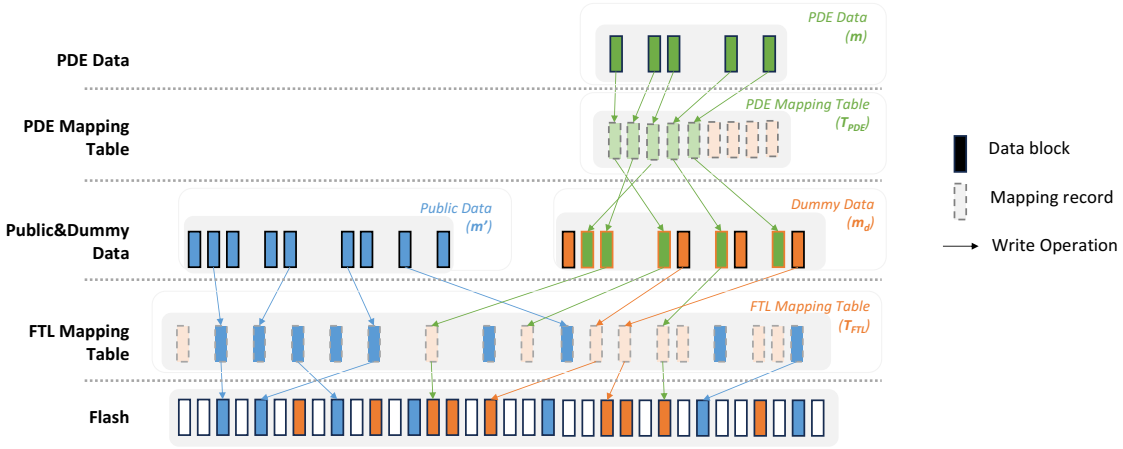

Figure 3: The MIST protocol's storage layout is structured such that solid blocks indicate data storage, and dashed blocks are for data block address mapping records. Arrows in the diagram represent the write operation processes, showing data flow direction. The data is categorized into: public data (blue, m'), dummy data (orange, m_d), and hidden data (green, m). It features two mapping tables: PDE (T_{PDE}) for mapping hidden data to dummy data, and FTL (T_{FTL}) for mapping virtual to physical flash block addresses. The color of the mapping records matches the data types: blue for public, orange for dummy, and green for hidden data writing. The hidden data in MIST is not stored separately but hidden within dummy data, making the PDE Mapping Table crucial for identifying hidden data in dummy blocks. Some mapping arrows are omitted in the figure for simplicity.

security. Thus, we introduce the MIST protocol, which is designed to operate within ARM's TrustZone. This design choice minimizes interaction with the Android system, thus mitigating the risk of leaving unintended forensic footprints in the system memory.

The MIST Protocol Paradigm. The figure provided illustrates the intricate storage layout intrinsic to the MIST protocol. Solid blocks represent the actual data storage units, while dashed blocks visually depict the address mapping records associated with these data blocks. The arrows illustrate the dynamic nature of write operations, showing the directionality and flow of data transmission.

The data architecture within the MIST protocol can be classified into three main categories: (1) **Public Data (m'):** Represented by blue, this category encompasses data that are openly accessible and do not require special protection considerations. (2) **Dummy Data (m_d):** Illustrated in orange, this category of data is essential for the MIST framework. Acting as a smokescreen, it emulates genuine data patterns and serves as a protective veil for genuine encrypted content. (3) **Hidden Data (m):** Described by green blocks and arrows, this category houses the data that require PDE protection. It is cleverly interspersed among the dummy data blocks, ensuring its protection by blending in with their indistinctness.

Central to the MIST protocol are two pivotal mapping tables: (1) PDE Mapping Table (T_{PDE}): This table establishes a relationship between hidden data and dummy data, facilitating the identification of hidden data concealed within dummy blocks. This table is the *hidden mapping table* mentioned in Section 4.2. (2) FTL Mapping Table (T_{FTL}): This table serves to bridge the virtual and physical addresses of the blocks, a necessity for streamlined flash storage management.

Protocol Description. Algorithm 2 shows the MIST protocol detail. Within the non-secure system, public data is directly committed to storage, denoted as m'. To enhance the deniability of the system, the non-secure system sporadically executes dummy write operations, labeling the resulting data as m_d. The metadata from these dummy write operations is then securely transmitted to the TrustZone, ensuring its immunity from external access or tampering.

However, within the secure world, all sensitive data writing operations occur under the protective umbrella of the TrustZone, with such data designated as m. When the system needs to write sensitive data, it consults T_{PDE}, a meticulously maintained table within the TrustZone that chronicles the dummy write operations. The system searches T_{PDE} for an unoccupied dummy write block. Once identified, m is securely written on this block, and the block is marked as occupied within T_{PDE}. This strategy ensures that sensitive data are indistinguishable from dummy data at the physical storage level, thus achieving data deniability.

A notable characteristic of the MIST protocol is its unconventional approach to deny the existence of hidden data. Instead of creating a separate storage volume for hidden data, the protocol cleverly integrates it into the dummy data, resulting in a higher level of obfuscation. This strategic amalgamation poses significant challenges for potential adversaries, making it difficult for them to distinguish genuine encrypted data from decoys.

7 SECURITY ANALYSIS

7.1 Reverse Engineering Defense (Goal 1)

The FSPDE's security architecture intricately combines encrypted TAs with the exclusive decryption capabilities of the TrustZone system [35]. This setup ensures that only the TrustZone can decrypt the TAs, thus maintaining strict control over access to these sensitive applications. Moreover, users are not in possession of the decryption keys, a deliberate design choice that mitigates the risk of key compromise, even in scenarios where users might be coerced by attackers. This lack of user access to the decryption keys, paired with the TrustZone's exclusive decryption authority, forms a comprehensive barrier against reverse engineering attacks.

Algorithm 2 MIST Protocol Data Handling

```
function Encrypt(data, key)
    return encrypted_data
end function
function Decrypt(encrypted_data, key)
    return data
end function
procedure WriteData(data, type, key)
    if type == Public then
        Write data to the non-secure system as m'
    else if type == Dummy then
        encrypted_data ← Encrypt(data, key)
        Write encrypted_data to the non-secure system as m_d
        Update T_PDE with dummy block information
    else
        encrypted_data ← Encrypt(data, key)
        Retrieve an unoccupied dummy block from T_PDE
        Write encrypted_data to the retrieved block as m
        Mark the block as occupied in T_PDE
    end if
end procedure
procedure ReadData(block, key)
    encrypted_data ← Retrieve data from the block
    if the block is in T_PDE then
        return Decrypt(encrypted_data, key)
    else
        return encrypted_data          ▷ Assuming dummy data is decrypted elsewhere if needed
    end if
end procedure
```

7.2 Entry Point Protection (Goal 2)

FSPDE utilizes the MUTE protocol for PDE entry point protection. By strictly separating REE and TEE, it can effectively isolate sensitive operations. The REE entry program serves as a decoy, meticulously designed to mimic typical software and shield any clues about the concealed PDE system. Upon key validation, a seamless and encrypted transition is initiated into the TrustZone, where the key verification logic is exclusively executed, further reinforcing the system's security. The MUTE protocol improves its defenses with a dummy PTA, acting as a deceptive mechanism against unauthorized TrustZone activations. Additionally, TA and PTA operations within the TrustZone are encrypted, providing a formidable shield against reverse engineering. The communication modules that cross REE and TEE have been engineered to resist data manipulation and interceptions. Collectively, these security mechanisms ensure the effectiveness of the MUTE protocol, allowing securely concealing the entry point of the PDE hidden mode.

7.3 Mitigating Multi-Snapshot Attacks (Goal 3)

7.3.1 Definitions. In this analysis, we use several key symbols and functions: $\mathcal{S}_{\text{flash}}$ represents the state of the storage medium; $\mathcal{S}_{\text{RAM}}$ represents the state of the memory; $\mathcal{A}$ represents a probabilistic polynomial-time adversary; $\Delta\text{flash}^{t_i,t_j}$ indicates the difference between storage snapshots at two different times t_i and t_j; and $\Delta_{\text{RAM}}^{t_i,t_j}$ indicates the difference between memory snapshots at two different times t_i and t_j.

7.3.2 Plausibility Analysis. Plausibility refers to the concept that when accessing a system using particular credentials, the accessed dataset could be either genuine or fabricated. From an external evaluator's perspective, the authenticity of both datasets remains ambiguous. By using specific credentials to access a system, access is granted to a dataset that can be either genuine or manufactured, depending on the credentials used.

The fundamental basis for the plausibility of the FSPDE system lies in the indistinguishability of the datasets. It is computationally challenging for an adversary to determine the authenticity of the datasets.

A PDE system satisfies plausibility if all probabilistic polynomial-time (PPT) adversaries have a negligible advantage in distinguishing between the genuine and fabricated datasets, given access to the storage and memory states, and the credentials used to access the datasets.

The design of FSPDE inherently supports plausibility by generating multiple datasets that appear authentic regardless of the credentials used, making it difficult for an external evaluator to definitively determine the validity of the data. If an adversary could distinguish between the genuine and fabricated datasets with a non-negligible probability, it would imply that the adversary could break the security of the underlying encryption scheme. However, since the encryption scheme is assumed to be secure, the existence of such an adversary would lead to a contradiction.

Therefore, FSPDE achieves plausibility if the underlying encryption scheme is secure.

7.3.3 Deniability Analysis. Deniability guarantees that users can present a decoy dataset while denying the authenticity of the original dataset, upon being coerced.

In other words, deniability ensures that users have the ability to plausibly deny the existence of sensitive or genuine data by presenting a convincing decoy dataset. The decoy dataset is crafted in such a way that it is indistinguishable from the genuine dataset, making it extremely difficult for an adversary to determine the true nature of the disclosed information.

A PDE system satisfies deniability if all PPT adversaries have a negligible probability of guessing the genuine datasets and genuine credentials, given access to the fabricated datasets.

The deniability of FSPDE is based on three key defensive mechanisms:

1. **Dummy Write Mechanism**: This mechanism introduces intentional obfuscation by concealing hidden write operations within dummy writes. As a result, it becomes computationally challenging for an adversary to distinguish them. Note that performing dummy writes itself does not imply PDE existence, as it is also used for other purposes, such as access pattern protection and erase operations in storage.

2. **Isolated Execution Environment**: The decision logic and metadata essential to PDE executions are housed within an isolated execution environment. This ensures that the traces of hidden mode will not appear in the memory region accessible to the adversaries.

3. **Ambiguity of The Encryption Outputs**: The system's ability to generate multiple authenticated datasets ensures that the concealed datasets remain hidden, even when a user exposes one using the genuine credentials.

If an adversary could guess the genuine datasets and genuine credentials with a non-negligible probability, given access to the fabricated datasets, it would imply that the adversary could either break the security of the encryption scheme or compromise the isolated execution environment. However, since the encryption

scheme and the isolated execution environment are assumed to be secure, the existence of such an adversary would lead to a contradiction.

Therefore, FSPDE achieves deniability if the underlying encryption scheme is secure and the decision logic and metadata are securely isolated in the TrustZone.

7.3.4 Multi-Snapshot Resistance. In real-world scenarios, adversaries may have the ability to collect multiple snapshots of the storage medium over time. If not designed properly, this multi-snapshot capability can compromise the deniability aspect of encrypted systems. Analyzing these snapshots could reveal patterns or metadata that hint at the presence of hidden data or the use of deniability techniques. Therefore, it is crucial for the PDE system to maintain its deniability properties even in the face of multi-snapshot adversaries.

Let $\mathcal{S}_{\mathrm{B}}^{t_1}, \mathcal{S}_{\mathrm{B}}^{t_2}, \ldots, \mathcal{S}_{\mathrm{B}}^{t_n}$ denote the series of snapshots taken at different instances $t_1, t_2, \ldots, t_n$, where $\mathrm{B} \in \{RAM, flash\}$.

Deniability Across Snapshots. For an adversary $\mathcal{A}$ which has access to a series of snapshots, we want to ensure that for all snapshots at any time instance (t_i), the adversary remains unable to determine the presence or absence of the authentic dataset, even if they have prior knowledge of earlier snapshots. The probability of the adversary distinguishing between the genuine and fabricated datasets should be approximately equal, regardless of the credentials used to access the system.

Consistency Across Snapshots. Consistency of concealed data is a critical aspect to consider for multi-snapshot deniability. The system must ensure that the hidden data remain consistent across different time instances or that any patterns align with standard data operations. Any sudden changes or patterns that deviate from standard operations could raise suspicion.

Formally, let $\Delta_{\mathrm{B}}^{t_i,t_j}$ be the difference between two snapshots $\mathcal{S}_{\mathrm{B}}^{t_i}$ and $\mathcal{S}_{\mathrm{B}}^{t_j}$. For genuine operations between times t_i and t_j, the difference $\Delta_{\mathrm{B}}^{t_i,t_j}$ should be indistinguishable from the difference for contrived operations using the decoy key:

$$\Delta_{\mathrm{B}}^{t_i,t_j}(\mathcal{K}_s) \approx \Delta_{\mathrm{B}}^{t_i,t_j}(\mathcal{K}_d) \tag{3}$$

where $\mathrm{B} \in \{RAM, flash\}$.

Memory Multi-Snapshot. Memory multi-snapshot attacks aim to capture the state of a system's RAM at various moments, with the intent to uncover patterns, secrets, or operations that suggest existence of hidden data. The PDE system uses TrustZone to counter such threats. TrustZone provides a secure execution environment that isolates critical operations and data from standard memory regions. It offers several key features:

• *Isolation from Main Memory*: Operations associated with the PDE mechanism occur within the secure confines of the TrustZone. Even if $\mathcal{A}$ captures the state of the main memory, the contents and operations within the TrustZone remain undisturbed and undisclosed. This isolation ensures that RAM snapshots do not provide any indication of PDE activities.

• *Secure Metadata Storage*: Metadata linked to PDE operations are securely stored within the TrustZone, making it difficult for $\mathcal{A}$ to discern patterns or behaviors from successive RAM snapshots.

• *Integrity of Operations*: TrustZone ensures not only confidentiality but also the integrity of operations. Even if other parts of the system are compromised, PDE-related operations remain untampered with.

By ensuring consistency of operations within the TrustZone over time, regardless of the unlocking key in use, it becomes computationally infeasible for adversaries to differentiate between genuine and decoy operations, even with multiple snapshots. Once an unlock operation is performed, whether using the secure key $\mathcal{K}_s$ or the decoy key $\mathcal{K}_d$, the resulting dataset undergoes processing within the secure environment of TrustZone. This approach guarantees heightened ambiguity; even if $\mathcal{A}$ seizes the RAM's state post-decryption, the TrustZone operations reveal no hint about the possible existence or absence of another dataset. That is

$$\Delta_{\mathrm{RAM}}^{t_i,t_j}(\mathcal{K}_s) \approx \Delta_{\mathrm{RAM}}^{t_i,t_j}(\mathcal{K}_d) \tag{4}$$

. Therefore, FSPDE achieves execution layer deniability against multi-snapshot.

Storage Multi-snapshot. The adversary cannot detect the existence of hidden data by performing a multi-snapshot attack on the flash storage. When analyzing the storage, two types of data can be observed: 1) data that can be decrypted by the public key; and 2) random data encompassing both dummy data and hidden data. Although the attacker can successfully decrypt the encrypted public data using the corresponding public key after coercing the device owner, he/she cannot differentiate hidden sensitive data from random data. Furthermore, any observations of alterations within the random data can be denied as stemming from dummy write operations. That is

$$\Delta_{\mathrm{flash}}^{t_i,t_j}(\mathcal{K}_s) \approx \Delta_{\mathrm{flash}}^{t_i,t_j}(\mathcal{K}_d) \tag{5}$$

, and makes FSPDE achieving storage layer deniability against multi-snapshot.

7.4 Extended Analysis

Fast Mode Switching (Goal 4). FSPDE's transitions between its hidden and public modes efficiently utilize ARM TrustZone's capability to rapidly switch between the secure and non-secure worlds. This process is initiated when the ARM CPU shifts to the Secure World upon PDE activation and returns to the non-secure world when exiting PDE. The seamless transition is primarily due to TrustZone's hardware-level integration and tailored design, which notably reduces the processing overhead for state transitions. These strategic design choices allow TrustZone to uphold robust security while ensuring efficient performance, thus facilitating FSPDE in achieving a fast mode switching.

Secure Cross-layer Communication. In the `MUTE` protocol, a secure storage driver is implemented within TrustZone, allowing FSPDE to interact directly with the underlying storage without passing through the non-secure world. This design ensures that no traces or logs are left in the non-secure world's memory, cache, file system, or OS. Consequently, it establishes secure communication between the execution layer and the storage layer, effectively preventing potential security breaches that could occur if sensitive operations were exposed to non-secure world.

Resisting against Side-channel Attacks. Side-channel attacks present a significant challenge in cybersecurity due to their wide range of attack vectors, spanning both software and hardware

layers. These attacks exploit inherent vulnerabilities [10]. Consequently, it's nearly impossible to defend against all side-channel attacks. However, in the context of FSPDE, all execution operations are confined within TrustZone, significantly mitigating the risk of side-channel attacks at the memory and cache levels. This confinement within TrustZone effectively shields FSPDE from many common side-channel vulnerabilities, thereby improving its overall security posture against these sophisticated attack methods.

8 IMPLEMENTATION AND EVALUATION

8.1 Implementation

To support TrustZone, we utilized OP-TEE. To support FTL, we relied on an open-source NAND flash controller OpenNFM [23]. We chose OpenNFM due to its modular and layered architecture, which lends itself well for customization and integration of advanced mechanisms. Our implementation comprises three major components: (1) modifying the OP-TEE OS to support the PDE execution layer, and (2) modifying the OpenNFM to support the FTL sub-layer, and (3) developing a PDE-enabled application within the REE. We encrypted the PDE TAs by enabling the build flag `CFG_ENCRYPT_TA=y` [5]. In addition, a dummy write operation is performed on the block device when operating in the public mode. Random numbers from /dev/urandom are used to determine the probability of a dummy write. A variable *rand*, generated from /dev/urandom, is generated between 0 and 100. To perform a dummy write, random data is encrypted using the same algorithm as that for public data. A free location is then selected using random allocation, and the encrypted random data is written to that location. The corresponding location in the global mapping table is marked as "allocated".

8.2 Evaluation

We evaluated FSPDE using a Raspberry Pi 3 Model B development board. This board boasts a Broadcom BCM2837 System-on-Chip (SoC) that features a 64-bit CPU architecture, complemented with 1 GB of Random Access Memory (RAM). There resides a 1.2 GHz quad-core ARM Cortex A53 processor in the SoC, adeptly paired with the DesignWare USB 2.0 controller [40]. This SoC has built-in support for TrustZone technology. On the software front, we have implemented OP-TEE, version 3.19, as the foundation of our secure software platform, as documented by OP-TEE, 2019 [35]. In addition, the non-secure software realm was anchored in the Linux kernel with version rpi3-optee-5.17 [24]. For external flash memory storage, we used another electronic board LPC-H3131. This board is equipped with an ARM9 32-bit ARM926EJ-S processor, clocked at 180MHz, 32MB RAM and 512MB NAND flash. OpenNFM was modified and ported to the LPC-H3131 [41], converting it to a regular flash storage device.

Secure Read/Write. The performance of secure read/write can be found in Table 1 and 2. To write the hidden sensitive data, FSPDE needs to encrypt them in the TEE, invoking the disk driver[1], and writing them to the external flash storage (managed by the FTL). Similarly, to read the hidden data, the TEE needs to invoke the disk driver, retrieving them from the external flash storage, which will then be decrypted in the TEE. The FTL needs to randomize the data placements and we measured how this randomization may affect the performance (Table 2). This was measured indirectly by comparing the throughput with/without the FTL randomization. The throughput degradation indicates the extra overhead caused by the FTL randomization. From Table 1, we can observe that the time delay caused by the encryption/decryption increases linearly when the data size is increased. In addition, the time delay caused by the TEE for both read and write operations increases when the data size is increased (but not linearly). We also observe that the time delay caused by the TEE is not insignificant. The reason is that Raspberry Pi-3B does not support general interrupt controller (GIC), making it hard to implement an efficient storage driver within the TrustZone. This could be optimized if the implementation is migrated to platforms supporting GIC which will be studied in our future work. From Table 2, we can observe that the randomization in the FTL has little impact on the read operations, but does have some impacts on the write operations. Using Raspberry Pi-3B, the performance degradation on the write operations is around 70% (this varies for different types of hosting computing devices, e.g., we observed around 50% degradation in a personal computer and around 60% in another embedded board Firefly AIO-3399J [3]). In addition, writing the LPC-H3131 in Raspberry Pi-3B is slow in general.

Non-secure Write/Read. The performance for the non-secure read/write is shown in Table 3. We can observe that the time needed for both the non-secure read and write operations grows linearly when the data size is increased. In addition, the non-secure write is significantly slower than the non-secure read, due to two potential reasons: 1) The write operation incurs extra dummy writes and random placements on the block device. 2) The raspberry Pi-3B itself is significant slow in writing the LPC-H3131 compared to the read operations.

Mode Switching. The transition time between the hidden mode and the public mode is dominated by the time needed to switch between the secure and the non-secure worlds. It takes approximately 110 μs when switching from the non-secure world to the secure world, and around 47 μs in the opposite operation, as documented by Amacher et al. [6].

9 RELATED WORK

The Block Device-based Mobile PDE Systems. Skillen et al. proposed Mobiflage [38, 39], the first mobile PDE system by integrating the hidden-volume-based TrueCrypt [42] with Android platform. Yu et al. [43] improved Mobiflage by mitigating a new booting-time attack, supporting multi-level deniability and fast data transfer to the hidden mode without rebooting. Mobiflage and MobiHydra both assumed the existence of a physical or an emulated FAT32 external storage partition, which limits their usage. Chang et al. [11, 12] removed this impractical assumption, by proposing a filesystem friendly PDE design that is compatible with any block file systems. Another design by Chang et al. [13], MobiCeal, tried to defend against a strong multi-snapshot adversary which was not

[1]To confirm the feasibility of allowing TrustZone to directly access the flash storage, we modified an SD driver from an open-source bare-metal project for Raspberry Pi 3 [2]. We maintained the driver's design and integrated it into the TrustZone hardware by specifying the correct translation between the virtual and physical addresses of the involved registers.

data size (KB)	1	4	8	16	32	64	128	256	512
Encryption/Decryption (μs)	160	781	2,019	4494	9945	19347	39,150	78,753	157,965
Inside TrustZone write (μs)	10,294	11,251	14,680	18,993	19,345	29,063	48,077	79,815	146,797
Inside TrustZone read (μs)	17,378	17,828	19,762	22,877	27,214	37,978	56,455	99,476	177,503

Table 1: Performance of secure read/write in the TrustZone secure world of Raspberry Pi-3B. "Encryption/Decryption" captures the time needed for encrypting/decrypting the hidden data in the secure world. "Inside TrustZone Read/Write" captures the other time in the secure world when reading/writing hidden data. Note that both do not capture the performance impact by the modified FTL, which will be evaluated in Table 2.

	SR	RR	SW	RW
Throughput of original OpenNFM (KB/s)	1138	978	55	43
Throughput of modified OpenNFM with random writes (KB/s)	1086	966	14	12
Estimated overhead caused by the FTL randomization	4.6%	1.2%	74.5%	72.1%

Table 2: Performance impact caused by FTL randomization. The throughput was obtained by running the fio [27] benchmark in Raspberry Pi-3B, performing different read/write patterns on top of LPC-H3131. SR: sequential read; RR: random read; SW: sequential write; RW: random write.

data size (KB)	32	64	128	512
non-secure read (s)	0.045	0.089	0.177	0.711
non-secure write (s)	1.423	2.961	5.603	23.474

Table 3: Performance of non-secure read/write in the non-secure world of Raspberry Pi-3B with LPC-H3131 as external storage. The time for the non-secure read was measured by reading the decoy data from the block device and decrypting them. The time for the non-secure write was measured by encrypting the decoy data, placing the resulted ciphertext to a random location of the block device, and performing the dummy writes (conditionally based on Equation 1).

considered in prior mobile PDE systems. Feng et al. [26] proposed MobiGyges which can improve storage usage, avoid rebooting the device to enter the hidden volume, and mitigate the capacity comparison attack and the fill-to-full attack.

The Flash Memory-based Mobile PDE Systems. Jia et al. [29] identified the deniability compromises when deploying hidden volumes at the block device. They instead designed DEFTL which integrates the hidden volumes into the flash translation layer. Having observed that the DEFTL can only mitigate the single-snapshot adversaries, Chen et al. [16] designed a new PDE system that can defend against multi-snapshot adversaries on NAND flash using write-once memory (WOM) codes. DEFY [36] was another PDE system which can mitigate multi-snapshot attackers by integrating steganography with a special flash file system YAFFS. INFUSE [15] also integrated a hiding scheme into YAFFS, by hiding sensitive data through the flash memory hardware side channels.

Other Mobile PDE Systems. Chen et al. have designed CrossPDE [20], which spreads the PDE functionality across multiple sub-layers (flash memory, block device, and file system) of the storage system to achieve a secure yet efficient mobile PDE system design. Another PDE system [18] by Chen et al. is hiding sensitive data in the non-sensitive images leveraging image steganography and watermarking. This however, can only hide a small amount of data due to the limited hidden space of the watermark images.

All the aforementioned mobile PDE systems can only partially achieve deniability, as they exclusively focus on concealing sensitive data within the storage layer. We have previously proposed a preliminary mobile PDE system [30] aiming for a full-stack design. However, [30] performs dummy writes on the FTL, resulting in both security and performance issues: 1) Performing dummy writes on the FTL cannot defend against the adversary which can access the block device. 2) Having the low-power flash device to handle the entire PDE will be very inefficient. On the contrary, FSPDE can address both concerns by performing dummy writes on the block layer, minimizing operations imposed to the FTL. In addition, [30] does not protect PDE binaries, leaving the design vulnerable to reverse engineering attacks. On the contrary, FSPDE can mitigate such attacks, and conceal entry points. Note that HiPDS [17] designed by Chen et al. does not work for the mobile systems as it relies on the Intel SGX to isolate the sensitive data in the memory.

10 CONCLUSION

In this work, we have designed FSPDE, a full-stack PDE system framework unique for mobile platforms. Leveraging the robust security attributes of ARM TrustZone and coordinating multiple storage sub-layers, FSPDE introduces a series of enhancements spearheaded by the `MUTE` and `MIST` protocols. For the first time, FSPDE works towards mitigating the deniability compromises at both the execution and the storage layers as well as the cross-layer communication for mobile systems. Security analysis and experimental evaluations on real-world mobile hardware confirm that FSPDE can meet the desired security goals with acceptable overheads.

ACKNOWLEDGMENTS

This work was supported by US National Science Foundation under grant number CNS-2313139, CNS-1928331 and CNS-1928349. Niusen Chen and Bo Chen were also supported by US National Science Foundation under grant number CNS-2225424.

REFERENCES

[1] [n. d.]. https://www.arm.com/products/silicon-ip-security.

[2] [n. d.]. Bare Metal Programming on Raspberry Pi 3. https://github.com/bztsrc/raspi3-tutorial.

[3] [n. d.]. Firefly AIO-3399J. https://en.t-firefly.com/product/industry/aio_3399.

[4] 2004. *TrueCrypt: Free open-source disk encryption software.* https://www.truecrypt.org

[5] 2023. OP-TEE Trusted Applications. https://optee.readthedocs.io/en/latest/architecture/trusted_applications.html. Accessed: 2023-12-28.

[6] Julien Amacher and Valerio Schiavoni. 2019. On the Performance of ARM TrustZone: (Practical Experience Report). In *Distributed Applications and Interoperable Systems: 19th IFIP WG 6.1 International Conference, DAIS 2019*. Springer, 133–151.

[7] ARM. 2021. Arm Confidential Compute Architecture. https://developer.arm.com/architectures/architecture-security-features. accessed: 2021-03-31.

[8] Amr M Azab, Peng Ning, Jitesh Shah, Quan Chen, Rohan Bhutkar, Ganesh Ganesh, Jia Ma, and Wenbo Shen. 2016. SICE: A hardware-level strongly isolated computing environment for x86 multi-core platforms. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. 375–388.

[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 CCS*. 203–214.

[10] Sebanjila Kevin Bukasa, Ronan Lashermes, Hélène Le Bouder, Jean-Louis Lanet, and Axel Legay. 2018. How TrustZone could be bypassed: Side-channel attacks on a modern system-on-chip. In *Information Security Theory and Practice: 11th IFIP WG 11.2 International Conference, WISTP 2017*. Springer, 93–109.

[11] 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.

[12] 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*. ACM, 381–390.

[13] 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 *48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*. IEEE, 454–465.

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

[15] Chen Chen, Anrin Chakraborti, and Radu Sion. 2020. INFUSE: Invisible plausibly-deniable file system for NAND flash. *Proc. of PET* 2020, 4 (2020), 239–254.

[16] Chen Chen, Anrin Chakraborti, and Radu Sion. 2021. PEARL: Plausibly Deniable Flash Translation Layer using WOM coding. In *The 30th Usenix Security Symposium*.

[17] Niusen Chen and Bo Chen. 2023. HiPDS: A Storage Hardware-independent Plausibly Deniable Storage System. *IEEE Transactions on Information Forensics and Security* (2023).

[18] 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.

[19] Niusen Chen, Bo Chen, and Weisong Shi. 2022. The block-based mobile pde systems are not secure-experimental attacks. In *EAI International Conference on Applied Cryptography in Computer and Communications*. Springer, 139–152.

[20] Niusen Chen, Bo Chen, and Weisong Shi. 2022. A Cross-layer Plausibly Deniable Encryption System for Mobile Devices. In *International Conference on Security and Privacy in Communication Systems*. Springer, 150–169.

[21] 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.

[22] Raymond Cheng, Fan Zhang, Jernej Kos, Warren He, Nicholas Hynes, Noah Johnson, Ari Juels, Andrew Miller, and Dawn Song. 2019. Ekiden: A platform for confidentiality-preserving, trustworthy, and performant smart contracts. In *2019 IEEE European Symposium on Security and Privacy (EuroS&P)*. IEEE, 185–200.

[23] Google Code. 2011. OpenNFM. https://code.google.com/p/opennfm/.

[24] OP-TEE Contributors. 2021. OP-TEE/manifest. https://github.com/OP-TEE/manifest GitHub repository.

[25] Poulami Das, Lisa Eckey, Tommaso Frassetto, David Gens, Kristina Hostáková, Patrick Jauernig, Sebastian Faust, and Ahmad-Reza Sadeghi. 2019. FastKitten: practical smart contracts on bitcoin. In *28th USENIX Security Symposium (USENIX Security 19)*. 801–818.

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

[27] Freecode. 2014. fio. http://freecode.com/projects/fio.

[28] 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 ACSAC*. 339–349.

[29] 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.

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

[31] Jinghui Liao, Fengwei Zhang, Wenhai Sun, and Weisong Shi. 2022. Speedster: An Efficient Multi-party State Channel via Enclaves. In *Proceedings of the 2022 ACM on Asia Conference on Computer and Communications Security*. 637–651.

[32] Johannes Lind, Henry Grahn, Thomas Johansson, and Markus Karlsson. 2014. Tee based mobile plausible deniability. In *Proceedings of the 30th Annual Computer Security Applications Conference*. 346–355.

[33] Joshua Lind, Oded Naor, Ittay Eyal, Florian Kelbert, Emin Gün Sirer, and Peter Pietzuch. 2019. Teechain: a secure payment network with asynchronous blockchain access. In *Proceedings of the 27th ACM SOSP*. 63–79.

[34] Ildar Muslukhov, Yazan Boshmaf, Cynthia Kuo, Jonathan Lester, and Konstantin Beznosov. 2019. Know your enemy: Leveraging data analysis to expose potential attackers in mobile encrypted communication. *ACM Transactions on Privacy and Security (TOPS)* 22, 1 (2019), 3.

[35] OP-TEE. 2019. Open Portable Trusted Execution Environment. *https://www.op-tee.org/* (2019).

[36] Timothy M Peters, Mark A Gondree, and Zachary NJ Peterson. 2015. DEFY: A deniable, encrypted file system for log-structured storage. (2015).

[37] Mendel Rosenblum and John K Ousterhout. 1991. The design and implementation of a log-structured file system. In *Proceedings of the thirteenth ACM symposium on Operating systems principles*. 1–15.

[38] Adam Skillen and Mohammad Mannan. 2013. 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*.

[39] Adam Skillen and Mohammad Mannan. 2014. Mobiflage: Deniable Storage Encryptionfor Mobile Devices. *IEEE Transactions on Dependable and Secure Computing* 11, 3 (2014), 224–237.

[40] Synopsys. 2023. DesignWare USB 2.0 Digital Controllers. Available at: https://www.synopsys.com/dw/ipdir.php?ds=dwc_usb_2_0_digital_controllers.

[41] Deepthi Tankasala, Niusen Chen, and Bo Chen. 2022. *Creating a testbed for flash memory research via lpc-h3131 and opennfm-linux version.* Technical Report. Technical report, Department of Computer Science, Michigan Tech.

[42] TrueCrypt. [n. d.]. Free open source on-the-fly disk encryption software.version 7.1a. http://www.truecrypt.org/. Accessed: 2021-08-23.

[43] 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.