MobiHydra: Pragmatic and Multi-Level Plausibly Deniable Encryption Storage for Mobile Devices*

Xingjie Yu[†,‡,♮], Bo Chen[♯], Zhan Wang[†,‡,**],
Bing Chang[†,‡,♮], Wen Tao Zhu[‡,†], and Jiwu Jing[†,‡]

† State Key Laboratory of Information Security, Institute of Information Engineering, Chinese Academy of Sciences, CHINA
‡ Data Assurance and Communication Security Research Center, Chinese Academy of Sciences, CHINA
♮ University of Chinese Academy of Sciences, CHINA
♯ Department of Computer Science, Stony Brook University, USA
Email: xjyu@is.ac.cn, chen@chenirvine.org, zwang@is.ac.cn, changbing12@is.ac.cn, wtzhu@ieee.org, jing@is.ac.cn

Abstract. Nowadays, smartphones have started being used as a tool to collect and spread politically sensitive or activism information. The exposure of the possession of such sensitive data shall pose a risk in severely threatening the life safety of the device owner. For instance, the data owner may be caught and coerced to give away the encryption keys so that the encryption alone is inadequate to mitigate such risk.

In this work, we present MobiHydra, a pragmatic plausibly deniable encryption (PDE) scheme featuring multi-level deniability on mobile devices, to circumvent the coercive attack. MobiHydra is pragmatic in that it remarkably supports hiding opportunistic data without necessarily rebooting the device. In addition, MobiHydra favourably mitigates the so-called booting-time defect, which is a whistle-blower to expose the usage of PDE in previous solutions. We implement a prototype for MobiHydra on Google Nexus S. The evaluation results demonstrate that MobiHydra introduces very low overhead compared with other PDE solutions for mobile devices.

Keywords: Mobile security, plausibly deniable encryption (PDE), data secrecy, coercive attack, countermeasures

1 Introduction

Many people today perform a majority of their daily communications, web browsing, and financial transactions via their mobile devices, and leave a large amount of sensitive data (e.g., phone call history, financial account secrets, and even evidences of an encountered crime) stored in those devices. Particularly, as mobile phones spread across the globe, human rights activists are increasingly turning to mobile technology for help in documenting, visualizing, and prosecuting human rights abuses. Unfortunately, human rights violators often harness similar technologies to silence activists [2]. To protect sensitive data, major mobile operating systems now provide different levels of encryption [3–6]. This simple encryption-based solution may work well when a mobile device gets lost or stolen. However, when a mobile device owner is caught and coerced into disclosing his/her decryption keys (which is known as a coercive attack), encryption alone becomes inadequate for protecting the owner's sensitive data. An example of coercive attack

* This is the full version of the paper that appears in the proceedings of ISC'14 [1].
** This author is the corresponding author.

against mobile devices is that a human rights worker uses his/her mobile device to collect evidences of atrocities in a region of oppression, but unfortunately he/she is captured and forced to hand over the evidences.

Plausibly deniable encryption (PDE) [7] is a promising tool that helps to circumvent coercive attack and allows the data owner to deny the existence of certain data. PDE allows the owner to decrypt a same ciphertext but present a different (innocuous yet plausible) plaintext, such that the attacker cannot differentiate between the real plaintext and the presented plaintext, and thus can be utilized to counteract the coercive attack [8]. A PDE solution can offer plausible deniability, which is an unusual security property allowing somebody to claim to others that certain information is not in his/her possession or that certain transaction has not been conducted [7]. In the literature, PDE has been investigated extensively for regular desktop operating systems [9–13]. However, there are very few developments of PDE on mobile devices where coercive attacks are more likely to occur.

Mobiflage [14] was the first PDE solution designed for mobile devices by customizing Android full disk encryption (FDE) [6]to offer plausible deniability. A mobile device which is equipped with Mobiflage works in two operation modes, the standard mode and the PDE mode. The standard mode is used to manage the regular data, which are less sensitive and thus can be encrypted normally without plausible deniability. This standard mode is accessed by entering a public password, which can be disclosed in emergency. The PDE mode, however, is used to manage sensitive data that can be denied of their existence. The PDE mode can only be activated by entering a hidden password. When facing a coercive attacker, the smartphone owner can simply disclose the public password, such that the attacker is only able to access the standard mode. Since the PDE-enabled device does not have any indication of the existence of hidden deniable files, the attacker will likely be convinced that the device owner has not kept any sensitive data and release the owner.

Although Mobiflage [14] initiates the research of PDE on mobile devices, it has some limitations. First of all, hiding data requires booting the device into the PDE mode, which may bring inconvenience to users and even worse, a user may not have enough time to reboot his/her device when an attacker suddenly appears. Therefore, Mobiflage is not so pragmatic in the real world. Second, Mobiflage can only support one deniability level, that is, a Mobiflage user can either keep or disclose all the sensitive data in the presence of a coercive attack. This will be problematic if an attacker insists the existence of the sensitive data. Third, Mobiflage is vulnerable to a new attack due to its design flaw during booting, which may compromise plausible deniability.

In this work, we take a holistic view over the existent PDE solutions, and propose MobiHydra, a novel PDE system for mobile devices, in which we alleviate the limitations of Mobiflage [14]. Our MobiHydra can support secure data hiding in the standard mode for emergency, offering a convenient feature to MobiHydra users. This way, device owners can work at the standard mode and transfer the sensitive data to the PDE mode without rebooting. In addition, MobiHydra introduces another feature, multiple deniability levels, with which users can choose to store sensitive data at different deniability levels. To the best of our knowledge, we are the first to design a PDE storage system for mobile devices offering both features of hiding data without rebooting and multi-level deniability. Moreover, we observe that all the previous PDE solutions for mobile devices are vulnerable to the booting-time attack, and we integrate certain countermeasures into MobiHydra to mitigate the defect. A feature comparison between Mobiflage and MobiHydra is summarized in Table 1.

Features	Mobiflage	MobiHydra
Hiding volumes in external storage	Yes	Yes
Hiding data w/o rebooting	No	Yes
Multi-level deniability	No	Yes
Booting-time attack resistance	No	Yes

Table 1. Comparison between Mobiflage and MobiHydra

In this paper, we make the following technical contributions:

- We identify a previously unreported booting-time defect, which can be exploited as a whistle-blower to compromise plausible deniability offered by previous PDE solutions for mobile devices. We also develop countermeasures to mitigate this attack.
- We propose MobiHydra, in which we design novel techniques to support hiding data without rebooting and multiple deniability levels. Particularly, we utilize an additional *shelter* volume to support hiding data in the standard mode, and we implement multiple hidden volumes to support multiple deniability levels, such that multi-level deniability can be achieved.
- We theoretically analyze MobiHydra's security guarantee. We also implement MobiHydra on a Google Nexus S smartphone powered by Android 4.0 and experimentally evaluate its performance.

The rest of the paper is organized as follows. The next section introduces the background and related work. Section 3 presents our threat model and assumptions. Section 4 introduces the booting-time attack. In Section 5, we describe the design of MobiHydra. We perform security analysis in Section 6. In Section 7, we present the implementation of MobiHydra on Android along with performance evaluations. Section 8 concludes the paper.

2 Background and Related Work

Android full disk encryption (FDE). The Android encryption layer is implemented with dm-crypt [6]. First of all, a master volume key used in encryption is chosen randomly. This master (volume) key is encrypted with another key, which is derived from 2000 iterations of PBKDF2 (a password-based key derivation function) [15] digest of the user's screen-unlock password and a salt value. Both the encrypted master key and the salt value are stored in the footer, which is located in the last 16KB of the userdata partition. When an Android device is booted and cannot find a valid file system on the userdata partition, it will require the user to enter the password. If a valid file system is then found in the dm-crypt target, it will be mounted and the system will boot regularly.

Mobiflage. Mobiflage [14] implements PDE for a mobile device by hiding a secret volume in the empty space of the device's external storage. Mobiflage initially fills the disk with noise-like random bits, and then creates two volumes, an outer volume and a hidden volume. The outer volume uses the entire disk while the hidden volume is created in a secret offset within the disk. The regular data will be stored in the outer volume encrypted by the public password (following full disk encryption), while the sensitive data will be stored in the hidden volume encrypted by the hidden password. A Mobiflage user can operate in both the standard mode (by providing the public password, which allows access to the outer volume) and the PDE mode (by providing the hidden password, which allows access to the hidden volume). The deniability of Mobiflage comes

from the fact that without knowing the hidden password, an adversary cannot tell whether an empty space embeds the encrypted hidden volume or simply the noise-like random bits.

Other work on PDE. Deniable encryption was firstly introduced by Canetti et al. [7]. Deniable encryption allows an encrypted message to be decrypted to different meaningful plaintexts, some of which can be used as decoy messages depending on the key or passphrase being used. Anderson et al. [13] designed the first file encryption scheme with PDE support, which is termed as steganographic file system. The steganographic file system can achieve the same objective as a PDE cipher by hiding files in random data. Rubberhose [16] for Linux is the first known instance of a PDE-enabled storage system. Other steganographic file systems [10, 12, 17, 18] focused on improving the efficiency and reliability. Moreover, a few desktop disk encryption tools [9,11] can support PDE by allocating hidden volumes on the storage devices. The deniability of hidden volumes relies on the fact that no evidence is available for identifying the presence of hidden volumes or data hidden in them. Moreover, some efforts investigated the deniability of desktop PDE tools based on forensic methods [19, 20] .

3 Threat Model and Assumptions

In this section, we discuss our threat model and operational assumptions.

Threat model. We consider an adversary who is able to fully control a mobile device after capturing it, including a root-level access to the device and a full control over the device's internal and external storage. In addition, the adversary can coerce the device's owner for the secrets to access the device.

Assumptions. We rely on several assumptions, all or portion of which are also required in the previous PDE solutions [14]:

- The adversary cannot capture a mobile device working in the PDE mode and has no knowledge of the PDE key and password which can allow accessing to the PDE mode. This assumption is necessary; otherwise, there is no solution since the adversary can directly acquire the sensitive data from the PDE mode.
- The adversary is not able to snapshot a mobile device's encrypted physical storage before having captured the device. An active attacker can continuously monitor a suspicious user, and periodically snapshot his/her mobile device's storage in a stealthy way. In this work, however, we only consider a passive attacker who will only take action after capturing a device.
- The adversary is rational: it has motivations to coerce the mobile device's owner to reveal the encryption keys and passwords (e.g., secrets for unlocking the screen of the mobile device), but will stop forcing the owner once it is convinced that the secrets have been revealed; it will not hold the user indefinitely or simply punish the user without any evidence of the existence of hidden data.
- MobiHydra must be merged with the default Android code stream, such that its capability is widespread, i.e., an attacker cannot simply distinguish a device with PDE capability.
- The mobile device has a physical or an emulated FAT32 external (SD or eMMC) storage partition. In Section 5.6, we will discuss more on this limitation.
- The device is malware-free in both the standard mode and the PDE mode.

4 Booting-time Attack

We found that previous PDE solutions for mobile devices (e.g., Mobiflage [14]) are vulnerable to a new attack, which we call the booting-time attack. Although in this section we use Mobiflage as an example, this booting-time vulnerability is common for *all the PDE solutions* which rely on hidden volumes to offer plausible deniability. In the following, we elaborate the principle of the booting-time attack, and demonstrate an experiment to validate the effectiveness of this attack on Mobiflage.

4.1 The Booting-time Attack

For a mobile device enabling the regular full disk encryption (FDE) without PDE support, when the device's user enters a password, the device will try to mount a valid file system (Section 2). It will test the correctness of the password by decrypting the master volume key to mount a valid file system. The system will boot successfully if the password is correct, otherwise, it will require the user to enter the password again. Thus, in FDE, the pre-boot authentication for both the valid password and the wrong password will not differentiate too much. For a mobile device enabling PDE (i.e., where Mobiflage is installed), when a valid public password is provided, it will boot into the standard mode by successfully mounting the outer volume. However, when a wrong password is provided, the system will first try to boot into the standard mode, and will then try to boot into the PDE mode by trying to mount the hidden volume. Only after the password has failed to activate the PDE mode, the user will be required to enter the password again. A wrong password in Mobiflage is tested for both the public password and the hidden password, which requires much more pre-boot authentication, and hence takes much longer time than when a valid public password is provided.

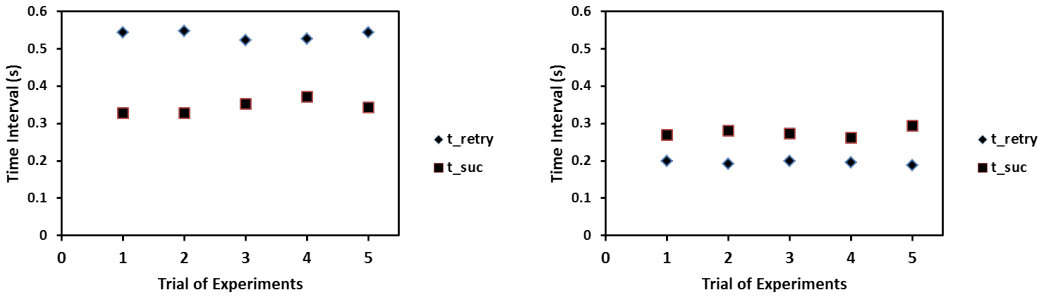
Based on the above observations, we identify a novel booting-time attack, which can compromise the plausibly deniability offered by Mobiflage. Suppose a mobile device is equipped with Mobiflage, and the device's owner (i.e., the Mobiflage user) has been captured by a coercive attacker $\mathcal{A}$.

1. $\mathcal{A}$ coerces the owner to disclose the password. The owner gives away the public password (which is allowed in Mobiflage design). $\mathcal{A}$ then boots the system with the public password several times and records each time interval between the moment when the password is entered and the moment when the operating system starts to boot up. We denote this time interval as *t_succ*.
2. $\mathcal{A}$ then uses wrong passwords to boot the system multiple times and records each time interval between the moment when the wrong password is entered and the moment when the retry screen appears. We denote this time interval as *t_retry*.
3. $\mathcal{A}$ makes a decision on whether PDE is present based on the statistical analysis of *t_succ* and *t_retry*. If significant deviation is observed between instances of *t_retry* and those of *t_succ*, then $\mathcal{A}$ can conclude that the device is protected by PDE.

4.2 Experimental Validation

To evaluate the effectiveness of the booting-time attack on Mobiflage, we targeted a Google Nexus S Android phone powered by Mobiflage. We depict the instances of *t_succ* and *t_retry* in Figure 1(a). As a comparison, we perform the same experiment on the same device with FDE

enabled (specifically, we first recovered the device to its initial state without Mobiflage, and then enabled FDE) and depict the instances of *t_succ* and *t_retry* in Figure 1(b). We observe that for Mobiflage, *t_retry* is at least 50% longer than *t_succ*, i.e., it takes significantly more time to try a wrong password than booting the system with a valid public password. However, for a regular FDE-enabled system, *t_retry* is approximately 30% shorter than *t_succ*. The difference between Mobiflage and FDE, unfortunately, will offer the adversary a clear indication of the existence of PDE. Note that for different devices, the specific time characteristics may be a little different, but the statistical time characteristics will be similar.



(a) Booting time of Mobiflage (b) Booting time of an Android FDE system

Fig. 1. A booting-time attack on Nexus S

5 MobiHydra Design

In this section, we propose a secure PDE system for mobile devices called MobiHydra, which supports both pragmatic data hiding under standard mode and multi-level deniability. The resistance against the booting-time attack is also featured in MobiHydra.

5.1 Overview

MobiHydra works in two modes: the standard mode for daily operations, providing encrypted storage without deniability; and the PDE mode for storing sensitive data with deniable encryption. The standard mode and the PDE mode are activated by a public password and one of the multiple hidden passwords, respectively. When booting the device, a password is entered by the user. As shown in Figure 2, the system first makes an attempt to decrypt the master public volume key with a password-derived key for mounting the public volume. If it fails, MobiHydra will calculate an offset with the supplied password, and try to mount a hidden volume onto the file system mount point where physical storage would be normally mounted.

MobiHydra enables encryption storage with plausible deniability by hiding sensitive data in hidden volumes. Such volumes are located in the empty space on the mobile device's external storage, and each of those is coordinated to one deniability level. To cover up such hidden volumes, the external storage is first filled with randomly generated bits, ensuring that the encrypted hidden volumes are indistinguishable from empty space. Then MobiHydra will format and encrypt the public and hidden volumes as specified by the user. Meanwhile, the offsets of

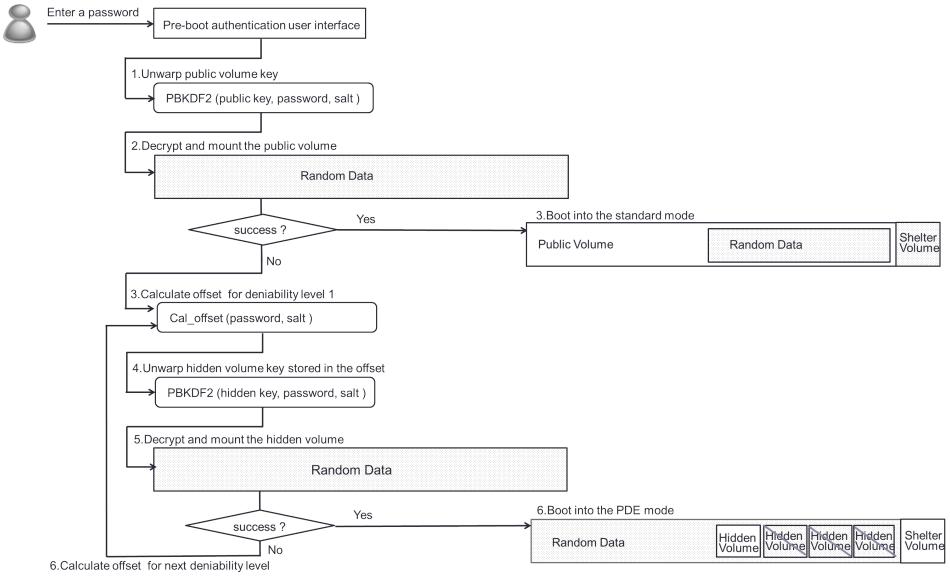


Fig. 2. MobiHydra workflow

hidden volumes are derived from user passwords. Moreover, a special partition call shelter volume on the external storage is also allocated and used as a temporary storage for data transfer from the standard mode to the PDE mode.

MobiHydra provides a safe mechanism for hiding opportunistic files when emergency events take place but the smartphone is under the standard mode. In the standard mode, files which are needed to be denied of their existence are stored in the *shelter* volume. Such files are encrypted and cannot be decrypted in the standard mode. When the device is booting into the PDE mode in a safe environment, the opportunistic files will be retrieved and decrypted in the PDE storage. After a successful retrieval, the *shelter* volume will be wiped out securely, to ensure data secrecy and leave enough storage space for future opportunistic files.

5.2 User Steps

At the initialization, the user needs to specify the number of hidden volumes, and select a public password for activating the standard mode and hidden passwords for accessing each level of hidden volumes in the PDE mode.

At the pre-boot authentication, the user enters the public password to activate the standard mode. The daily operations (e.g., calls, short messages) should be performed in the standard mode where operational logs and traces of usage are recorded as the innocuous materials. Consequently, if the user get caught with the device working in the standard mode or powered-off, he/she can feign compliance by relinquishing the public password. In some circumstances, the adversary may not be convinced by the public password, and thus continue intimidating the user to reveal the hidden passwords. The user can give away one or two hidden passwords which are associated with relatively lower deniability levels to avoid safety threat and keep more sensitive data in secret. Of course, the user should assess the importance of each hidden file, and save it in an appropriate hidden volume.

When the user boots the device into the PDE mode, he/she should provide a hidden password at the pre-boot authentication. After booting into the PDE mode, the user can transfer documents with another device, or take photos and videos and all the data are stored on deniable volumes. However, once the necessary operations are completed, the users are recommended to reboot and switch the device to the standard mode for the sake of safety.

5.3 Pragmatic Hiding Support

Pragmatic hiding allowing data transfer from the standard mode to the PDE mode without rebooting the device is not supported in previous PDE system for mobile devices. However, in some cases, the user has no time to reboot the device before saving opportunistic files. MobiHydra implements a pragmatic mechanism for hiding data under the standard mode.

Principle. The basic idea for hiding data without rebooting the device into the PDE mode is that a sensitive file can be saved in a *shelter* volume temporarily in the standard mode, and will be transferred to the hidden volume automatically when the user boots the device into the PDE mode. For the purpose of denying the existence of such file in the standard mode, the opportunistic file is protected by the same symmetric encryption function used for hidden volumes (which is also the same function used for filling the empty area with random numbers) to keep it undistinguishable from the noise-like random numbers. Moreover, in order to keep such file not readable in the standard mode, the symmetric key is encrypted by an asymmetric public key, and the private key is kept in hidden volumes which cannot be accessed from the standard mode.

In the standard mode, if the user has to hide opportunistic data for emergency (i.e., taking photos of an encountered crime as evidences) and has no time to rebooting the device into PDE mode, MobiHydra will encrypt such data with a randomly generated symmetric key and save the ciphertext in a *shelter* volume. The random symmetric key will be encrypted with a public key which is generated at the initialization of MobiHydra, and saved in the *shelter* volume. When the PDE mode is activated, MobiHydra will decrypt the symmetric key of the *shelter* volume with the private key saved in the mounted hidden volume to decrypt the opportunistic data saved in the *shelter* volume. After that, such decrypted data will be transferred to the hidden volume and then securely deleted from the *shelter* volume to release storage space. In our design, the private key is saved in every hidden volume, so that the opportunistic data will be transferred to any hidden volume mounted at the first time of activating the PDE mode after storing these data in the standard mode. Therefore, the user could choose which deniability level to keep such data in by entering an associated password. Moreover, a possible way to decide the deniability level of an opportunistic file in the standard mode is discussed in Section 5.6.

In the standard mode, the hidden data may need to be operated by customized apps with our pragmatic hiding support and an option to save files to *shelter* volume. Note that, to be capable of hiding files, an app should give no indication of the existence of such files in its operation logs. The customized apps may be a red flag for the usage of PDE tool, thus such apps should be designed with deniability. In this work, we focus on the PDE storage scheme itself, while the secure management of any related apps is beyond the scope of the paper. We suggest that a customized app can be designed as a hidden app that works in the backstage and could be activated in emergency.

***Shelter* volume.** A small part of the external storage is allocated specifically for the *shelter* volume while other remaining storage is used for the public volume and hidden volumes. To avoid overwriting the hidden volume by writing opportunistic data in the *shelter* volume across the volume border, the offset of this *shelter* volume is close to the end of the external storage. The *shelter* volume is mounted as a block device in the standard mode.

However, since *shelter* volume is mounted as an independent block device in the standard mode, the available space of the external storage is less than its real physical storage in the file system. It may make the adversary suspect the existence of a *shelter* volume. A deniable explanation for the unavailable storage could be given by attributing it to system influence. In this way, the *shelter* volume should be limited to a size that small enough to be reasonable for system influence, which may reduce the data transfer capability. However, since the *shelter* volume is used for emergency data transfer, there is no need to assign a massive storage space for such volume. We suggest that the user need to transfer the opportunistic data to hidden volume as soon as convenient to ensure enough space for next emergency.

To allocate storage area for the *shelter* volume without size limitation and thus improve the data-transfer capability, some dummy files could be maintained in the *shelter* volume and updated periodically. In this case, the unavailable storage can be explained as being occupied by such dummy files. However, the existence of dummy files will result in a waste of storage space. Since an extra block device may arouse suspicion of an adversary about the existence of opportunistic data, instead of mounting a *shelter* volume as the temporary storage, such data can also be stored in some special areas of the public volume, and encrypted by another symmetric key before being encrypted with the master public volume key. The locations of such areas are recorded in the hidden volumes together with the public volume key and another symmetric key. However, this solution requires either the modification of the access privilege of the whole SD card or rooting the mobile device, which brings additional security risk to the system.

5.4 Multi-Level Deniability

Previous PDE system (i.e., Mobiflage) keeps all sensitive data in a single hidden volume, which may risk in all or nothing exposure. In contrast, MobiHydra offers multiple hidden volumes which are corresponding to different deniability levels so that the user is able to selectively relinquish one of the multiple hidden volumes when it is necessary. This is how the name MobiHydra comes where "Hydra" is a many-headed serpent in Greek mythology [21].

Storage layout. The storage space can be regarded as the concatenation of independent encrypted volumes, including one public volume, one shelter volume and multiple hidden volumes. We formalize the storage layout as follows:

$$E_{K_p}(Vol_{pub})||E_{K_1}(Vol_{h1})||E_{K_2}(Vol_{h2})||...||E_{K_n}(Vol_{hn})||E_{K_s}(Vol_{shel})$$

Here, $E_K(\cdot)$ represents a symmetric encryption function with key K and $||$ represents concatenation. Vol_{pub}, Vol_{shel} and Vol_{hi} denote the public volume, *shelter* volume and the i-th hidden volume, respectively. K_p and K_s represent a master volume key of the public volume and *shelter* volume. n represents the required number of hidden volumes specified by the user, and K_1 to K_n represent the master keys of hidden volumes.

The deniability of Vol_{hi} enhances along with the increment of i. When a right password (either a public password or a hidden password) is supplied for system boot, to avoid a visible limit on the mounted volume, the volume decrypted by a given key will appear to consume all remaining space (except the space allocated for the shelter volume), and other volumes will appear to random noise. Thus, the existence of $Vol_{hi,i>1}$ could be denied by relinquishing the password associated with $Vol_{hx,x\in[1,i-1]}$, and the existence of Vol_{h1} can be denied if the user only gives away the public password. In addition, the exposure of $Vol_{hi,i>1}$ may expose the existence of $Vol_{hx,x\in[1,i-1]}$ which is detailed in Section *Offset Calculation*. Therefore, if the adversary forces the user to reveal hidden password besides the public password, the user can fake compliance by giving away one or two hidden passwords associated with relatively lower deniability levels, and the data saved in the hidden volumes relatively associated with higher deniability levels are still secret.

Note that, the hidden volumes may be overwritten by writing to the currently mounted volume past the volume boundary. Although overwriting the sensitive data incidentally can be avoided by saving appropriate size of data in each hidden volume on purpose, an adversary still can destroy the hidden data by overlapping them. This issue is inherent to PDE storage solutions, which is usually addressed by keeping data replicas on desktop [12, 16]. However, considering the limited storage space on mobile devices, MobiHydra only protects data secrecy rather than data integrality.

Offset calculation. The offset is the mounting point for each volume at booting time. The offset is derived from the password provided by the user. In order to keep the operations indistinguishable between PDE and Android FDE, the calculation of each offset should not rely on any stored variables that are not introduced by FDE, since the existence of such variables may expose the existence of hidden volumes. The offset for public volume is just the start of the storage space; and the offset for shelter volume is specified by the user at the PDE installation time. The offset calculation for the i-th hidden volume is formalized as follows:

$$offset(i) = m \times vlen + (i-1)/N \times (1-m) \times vlen + H(pwd_i||salt) \bmod \lfloor 1/kN \times (1-m) \times vlen \rfloor$$

Where *vlen* denotes the number of 512-byte sectors on the logical block storage device; m represents the proportion of the storage space that is only occupied by the public volume; H is a PBKDF2 iterated hash function; pwd_i is the password of the hidden volume with the deniability level of i; N is the largest number of hidden volumes supported by the system; *salt* is a random value for PBKDF2 and k is a real number larger than 1 and helps to randomize the offset point. We further explain each variable as follows:

- m is recommended to be no less than 50%, on the account of that the public volume is used frequently for daily operations than the hidden volumes;
- The *salt* value has been also used for the public volume key derivation (i.e., stored in the default encryption footer);
- N is a constant decided by the system, and each hidden volume's size is approximately $(1-m)/N$ of external storage size except for the n-th one (consuming the storage space between its offset and the *shelter* volume's offset) regardless of the number of needed hidden volumes specified by the user (represented by n in Section *Storage Layout*). If N is replaced by n, the size of each hidden volume can be specified by the user to some extent. As a prerequisite, n should be encrypted with different password-derived key, respectively, to ensure

that any hidden password could unwrap n for offset calculation. Therefore, the adversary who has already obtained a hidden password can decrypt n, thus can ascertain whether any extra hidden volumes exists. Note that, in our design, if the user gives away the passwords associated with Vol_{hi}, the adversary can calculate the space between the offset of this volume and the end of the public volume which may indicate the presence of other hidden volumes ($Vol_{hx,x\in[1,i-1]}$).

- i is not supplied by the user for system boot for the sake of keeping consistence with Android FDE, since any different user interaction will give an indication of the usage of PDE tool. Instead, i will be traversed from 1 to N for offset generation until a valid volume is mounted. Moreover, this design ensures that the required time for testing a wrong password gives no indication of the value of n.
- k should be specific based on a trade-off between security requirement and storage utilization. The offset of a certain hidden volume randomizes in the range of 0 to $\lfloor 1/kN\times(1-m)\times vlen\rfloor$, therefore, to avoid overwriting the next volume, we suggest that the data stored in a hidden volume should be within $(1-m)/N\times vlen\times(1-1/k)\times 512$ bytes except for the n-th hidden volume. However, it results in a waste of storage space. For the purpose of reducing the waste of storage, k should be set to a relatively large integer which will result in a corresponding small offset random range.

5.5 Mitigating the Booting-time Attack

In order to make the time characteristic of password verification identical between MobiHydra and a regular Android FDE system, there are two options: reducing the time for wrong password verification or increasing the verification time for correct password in MobiHydra implementation.

Wrong password verification involves the following essential operations: calculating offset from password, retrieving and decrypting the encrypted master key. Each of above operations requires executing PBKDF2 digest algorithm 2000 times. We firstly attempt to accelerate the wrong password verification by reducing the iterations of PBKDF2 digest to 1000 times. However, the results turn out that it not only weakens the security and may suffer dictionary attack, but also has no significant change in time characteristics of password verification in MobiHydra.

Furthermore, we try to mitigate the booting-time defect by manipulating the verification time for correct password. According to our experiment, if $3\times N$ extra invocations of PBKDF2 are performed as a dump operation after a password is authenticated as the right password (where N is the supported level of deniability in MobiHydra), the characteristics of *t_retry* and *t_succ* will be identical in MobiHydra and an Android FDE system as shown in Section 7.

5.6 Known Issues

MobiHydra shares parts of limitations of Mobiflage scheme, including the need of a separate physical FAT32 storage partition and the choice between allowing the user to configure the volume size and avoiding the need to store the offset. Besides, our current MobiHydra design has the following unsolved issues:

1. The size of the *shelter* volume cannot be changed once it has been set at the initialization of the PDE system, in that the *shelter* volume is located between the public storage area and the hidden one physically and dynamically scaling the size of the *shelter* volume may

overwrite the data stored in the hidden volumes. In addition, it is noted that a relatively smaller size of *shelter* volume will lower the risk of comprising deniability because the *shelter* volume is viewed as unavailable storage space under the standard mode.

2. For the sake of simplicity, the opportunistic data stored in the *shelter* volume are visible to all the hidden volumes, so that the user cannot set the deniability level of them. A possible solution is to generate a unique pair of RSA keys for each deniability level, save the private key in each corresponding hidden volume and save the public keys in the *shelter* volume. When a file is stored in the *shelter* volume, MobiHydra will encrypt the file with a random master key and wrap the random master key with its corresponding public key. In this fashion, only the hidden volume with correct private key can retrieve the opportunistic data appropriately.

6 Security Analysis

The security of MobiHydra is captured in Lemma 1 and Theorem 1. Compared to Mobiflage [14], MobiHydra is enhanced with additional security properties as follows:

- Mitigate the booting-time attack: Most of the previous PDE solutions based on hidden volumes [8] are vulnerable to the booting-time attack (Section 4). Our MobiHydra, however, mitigates this attack by introducing additional delay to the booting process of the standard mode. Thus, an attacker cannot suspect the existence of the PDE mode by observing the time difference between a booting process where a correct public password is provided and another where a wrong password is provided.
- Support multiple deniability levels: Previous PDE solutions for mobile devices [8] can only support one deniability level, by which a mobile device owner can either keep or disclose all the sensitive data upon a coercive attack. This will be problematic if an attacker insists the existence of the sensitive data. In MobiHydra, we allow the device owner to convince such an attacker by disclosing a portion of the less sensitive data from the lower deniability levels, such that the more sensitive data from higher deniability levels can remain secret.

Lemma 1 (Security Guarantee of Mobiflage). *Mobiflage [14] can offer plausible deniability for mobile devices.*

Proof. (sketch) In terms of PDE paradigm, Mobiflage is receiver-deniable [7]. In addition, Mobiflage maintains deniability against threats from both the desktop and the mobile systems. We refer the reader to [8] for more detailed security analysis.

Theorem 1. *MobiHydra can offer plausible deniability for mobile devices.*

Proof. (sketch) MobiHydra implements multiple deniability levels based on multiple hidden volumes, and utilizes additional techniques to support pragmatic hiding without rebooting and mitigate the booting-time attack. Let S be a sub-system of MobiHydra, which only implements multiple deniability levels based on multiple hidden volumes. A MobiHydra system is equivalent to a system which is formed by adding to S additional techniques to support pragmatic hiding without rebooting and mitigate the booting-time attack. In the following, we show: 1) the techniques used to support pragmatic hiding and mitigate the booting-time attack will not introduce additional security breaches in terms of plausible deniability; 2) S is equivalent to a Mobiflage system.

To support pragmatic data hiding, we utilize a *shelter* volume (which can simply be extended to multiple *shelter* volumes), which is encrypted and is not able to be differentiated from a regular empty volume. Thus, the supporting of hiding data without rebooting does not add any indication of the existence of PDE. In addition, to mitigate the booting-time attack, we add additional delay to the booting process of the standard mode, which does not add any indication of the existence of PDE either (based on our assumptions in Section 3, MobiHydra should be merged with the default Android code stream, and thus additional delay to the booting process of the standard mode will not become a vulnerability).

S (i.e., a sub-system of MobiHydra) implements multiple deniability levels, each of which has a hidden volume accessible by a different hidden password. Let n be the number of deniability levels. MobiHydra system has 1 outer volume, 1 public password, n hidden volumes, and n hidden passwords. We can view the n hidden volumes as 1 large hidden volume, and can view the n hidden passwords as 1 combined password by which we can access every portion in this large hidden volume. Thus, S can be viewed as a PDE system which has 1 outer volume, 1 public password, 1 hidden volume, and 1 hidden password, which is equivalent to a Mobiflage system.

We conclude that MobiHydra can achieve Mobiflage's security guarantee, and thus can offer plausible deniability for mobile devices (Lemma 1).

7 Implementation and Evaluation

We implemented a prototype of MobiHydra on a Google Nexus S smartphone based on Android 4.0 (ICS). Our Nexus S has 1 GB internal and 15GB eMMC external storage (i.e., eMMC partition). In our prototype implementation, we use almost half of the extern storage for hidden volumes. In the following, we first elaborate the implementation details, and then provide the experimental results.

7.1 Prototype Implementation

Pragmatic hiding support. We mount a *shelter* volume as a block device, which is linked to */dev/shelter_vol* through a hard link. This volume consumes the last 128MB of the external storage. We modify the access privilege of such storage area, such that apps are able to write files to it. The content of the *shelter* volume is shown in Figure 3. For experimental purpose, we develop a camera app which can save images to this *shelter* volume when the device works in the standard mode.

MobiHydra generates a pair of 1024-bit RSA keys during its initialization, and saves the public key in the *pubkey* field. Meanwhile, the private key is saved in a file /SDcard/prkey, which will be stored in each hidden volume, and is not accessible in the standard mode. In the standard mode, once the camera app has captured an opportunistic image, MobiHydra will encrypt the image by a randomly chosen AES-XTS key, and save the encrypted image in the *Data* field. It will then encrypt this AES-XTS key by the public key and save the ciphertext in the *AESkey* field. After that, the plaintext of the AES-XTS key will be discarded. The number of files will be kept in the *int count* field. For simplicity, our current prototype only supports 16 files, but it can be easily extended to support more files. In addition, the *AESkey* field in the current prototype can only store one AES-XTS key, i.e., the user can only hide data in the standard

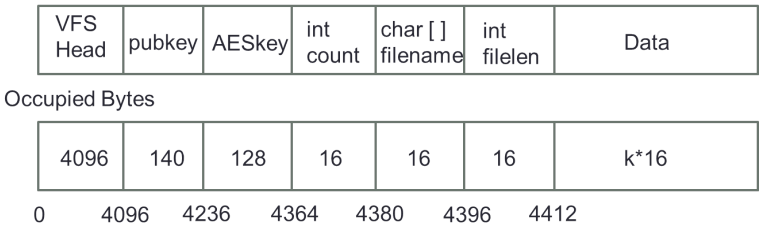


Fig. 3. Content of the *shelter* volume

mode once before booting it into the PDE mode, since it cannot access to the old AES-XTS key in the standard mode. This limitation can simply be addressed by extending the *AESkey* field to be able to store multiple randomly generated keys, such that for a new opportunistic data hiding attempt, MobiHydra can generate a new AES-XTS key without overwriting the old one.

In our prototype implementation, all the deniability levels share the same RSA private key. Note that differentiating the hiding data for each different deniability level can be achieved by generating a RSA key pair for each deniability level, and storing each private key in its associated hidden volume. When the PDE mode is activated, MobiHydra will decrypt the *AESkey* field using the private key stored in */SDcard/prkey*. It then decrypts the *int count* field using the AES-XTS key (acquired by decrypting the *AESkey* field), acquiring the number of files. If the number of files falls within the range of 1 to 16, those files saved in the *shelter* volume will be decrypted and saved to the corresponding hidden volume. After that, the data stored in the *shelter* volume, starting from the *AESkey* field, will be wiped and replaced by random data. If the number of files does not fall within the range of 1 to 16, no further process for data transfer will be performed.

Multi-level deniability. For simplicity, our prototype only supports up to 5 deniability levels, and it is easy to be extended to support more deniability levels. The storage layout for our prototype is shown in Figure 4. In our prototype, 50% of the external storage is only occupied by the public volume on the account of frequent usage of it for daily operations. A user can determine the number of deniability levels during initialization (i.e., from 1 to 5). If the user initializes 5 hidden volumes, the size of each hidden volume is around 1GB. If the user needs more space for one hidden volume, he/she can choose less hidden volumes during initialization. For this case, the last hidden volume can consume the remaining external storage starting from its own offset (except the space allocated for the *shelter* volume). The master volume key for each hidden volume is stored at the storage sector located at the corresponding hidden volume's offset (recall from Section 5.4 that the offset is computed based on secrets, e.g., the hidden password). The master volume key for the public volume is encrypted and stored in a footer located in the last 16KB of the userdata partition (located in internal storage). The random salt is also stored in this area. We fix k (a variable used in computing the offset as described in Section 5.4) at 8 in our prototype, which can achieve a good tradeoff between the security requirement and the storage utilization (only resulting in a waste of 1GB storage space).

Pre-boot authentication in MobiHydra. When a MobiHydra device boots, it will require the user to enter a password. It then derives a key from the password, and uses this key to decrypt the master volume key of the public volume, attempting to mount the public volume. If it fails, it will try to mount the first hidden volume based on the given password. It first

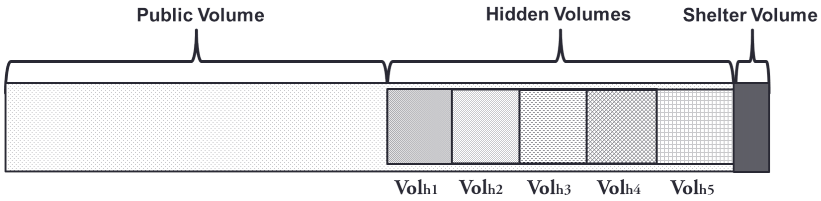


Fig. 4. Storage layout of our MobiHydra prototype

system	Key length (bits)	Initiali -zation (s)	IO speed (MB/s)	Boot time (wrong password) (s)	Boot time (public password) (s)
Default FDE	128	85.4	8.5	0.19	0.27
Mobiflage	512	7399	8.2	0.54	0.35
MobiHydra	512	7421	8.1	1.49	1.98

Table 2. Performance comparison between default Android FDE, Mobiflage and MobiHydra

calculates the offset of this hidden volume by initializing the variable i (a variable representing the deniability level as described in Section 5.4) as"1". It then obtains the corresponding master volume key by decrypting the storage sector at this offset with a key, which is derived for this hidden volume from the given password. It tries to mount this hidden volume starting at the next storage sector after the calculated offset. If a valid file system is found, the corresponding hidden volume will be mounted, and the system boot will continue as usual. Otherwise, it will try to mount another hidden volume by increasing i by 1, up to all the available hidden volumes (at most 5 in our prototype implementation). If it cannot find a valid file system after having tried all the available hidden volumes, it will ask the user to re-enter a password.

7.2 Experimental Evaluation

We evaluate MobiHydra in terms of initialization performance, I/O and system boot performance, and the effectiveness of mitigating the booting-time attack. We also provide a performance comparison among MobiHydra, Mobiflage and the default Android FDE system.

Initialization performance. To compare the time required for initialization among MobiHydra, Mobiflage and Android FDE system, we initialize each system three times, and average the results in Table 2. We observe that the initialization time significantly increases in both MobiHydra and Mobiflage compared to Android FDE. This is because, for PDE purpose, both MobiHydra and Mobiflage require an expensive two-pass random wipe of the external storage (each pass with a new random key), and encryptions over the hidden volumes. However, Android FDE only needs to encrypt the whole external storage with the master volume key one time. We claim that the high cost of initialization is not a significant issue for MobiHydra because initialization is a one-time operation.

PDE system	Booting time with hidden password(s)	
Mobiflage	10.7	
MobiHydra	pde_1	0.62
	pde_2	0.86
	pde_3	1.09
	pde_4	1.31
	pde_5	1.62

Table 3. System boot time in the PDE mode, in which pde_n represents the time for booting the device into the PDE mode for deniability level n employing the n-th hidden password.

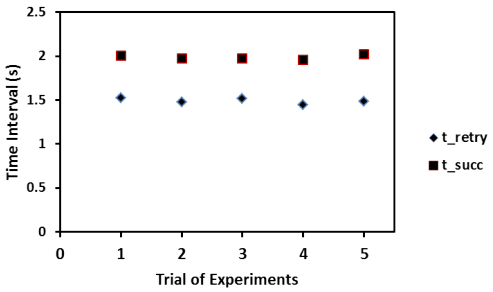


Fig. 5. Booting time of MobiHydra

I/O performance. We use the adb-shell tool to test the I/O performance for each aforementioned system. We run 10 trials, during each of which we write a 100MB file to the external storage (i.e., eMMC partition). The results in Table 2 show that MobiHydra has a similar I/O performance compared to Mobiflage because, first of all, for both MobiHydra and Mobiflage, read/write the hidden volume will result in similar operations; secondly, in MobiHydra, we adopt the same encryption algorithm as in Mobiflage.

System boot performance. Compared to Mobiflage, MobiHydra performs different operations in system boot procedure, especially in pre-boot authentication. We evaluate the boot time of MobiHydra, Mobiflage and Android FDE under a wrong password, a correct public password (a correct password for Android FDE since it does not have a public password) and a correct hidden password (not applicable to Android FDE), respectively. We average the time in Table 2 and 3. We have several observations: First of all, MobiHydra takes longer time when testing a wrong password compared to either Mobiflage or Android FDE (Table 2). This is because MobiHydra needs to try the wrong password for the public volume and all the hidden volumes during pre-boot authentication. Meanwhile, MobiHydra takes longer time to boot the standard mode compared to the other two systems (Table 2), which is resulted from the countermeasures used in mitigating the booting-time attack. Second, the time needed to boot the PDE mode in MobiHydra increases with the deniability levels (Table 3), i.e., the higher the deniability level, the more time is needed to boot the PDE mode. This is because, MobiHydra tests a given password starting from the lowest deniability level, and keeps testing until it can find a target deniability level. Third, the time needed to boot the PDE mode of MobiHydra ranges around 1 second in MobiHydra, which is significantly shorter than 10.7 seconds as it is in Mobiflage (Table 3). This is because, Mobiflage always tries to un-mount a persistent log partition when booting a system which, unfortunately, is very inefficient for a device that does not have this partition (e.g., Google Nexus S). We optimize this process by judging whether or not the device has such a partition before trying to un-mount it, which results in a saving of approximately 9 seconds when booting a device without such a partition.

Mitigating the booting-time attack. To mitigate the booting-time attack, MobiHydra performs 15 additional invocations of PBKDF2 as dump operations when a correct public password is provided. Figure 5 shows the statistical characteristics of *t_retry* (refer to Section 4) and *t_succ* (refer to Section 4) for MobiHydra. *t_retry* is approximately 30% shorter than *t_succ*, i.e., the deviation between *t_retry* and *t_succ* is similar to that of a default FDE system (Figure 1(b)). Booting the standard mode in MobiHydra now takes approximately 2 seconds. However, this will not degrade plausible deniability because, first of all, taking several seconds to boot a mobile

device is very common in daily life; second, upon being suspicious of, the device's owner can simply ascribe the slow boot to the poor device performance. Thus, MobiHydra can mitigate the booting-time defect.

8 Conclusion

The mobile device owners may become the target of a coercive attack if their devices store certain sensitive content such as evidences of an encountered crime and uprising. To help them circumvent such attack, we propose MobiHydra, a pragmatic PDE storage with multi-level deniability for mobile devices. MobiHydra provides a secure mechanism for pragmatic data hiding, allowing a user to save deniable data in the standard mode without rebooting the device. In addition, MobiHydra supports multi-level deniability, which reduces the possibility of losing all the sensitive data at a time. Moreover, MobiHydra can mitigate the booting-time attack, which is a common vulnerability for all the previous PDE solutions relying on hidden volumes to offer plausible deniability.

References

1. X. Yu, B. Chen, Z. Wang, B. Chang, W. T. Zhu, J. Jing. MobiHydra: Pragmatic and Multi-Level Plausibly Deniable Encryption Storage for Mobile Devices. In *17th Information Security Conference (ISC'14)*, 2014.
2. SXSW Schedule. Caught in the Act: Mobile Tech & Human Rights. *[www Document]:* `http://schedule.sxsw.com/2014/events/event_IAP21063`, 2014.
3. Windows Inc. BitLocker Drive Encryption. *[www Document]:* `http://windows.microsoft.com/en-us/windows7/products/features/bitlocker`, 2014.
4. App Inc. OS X: About FileVault 2. *[www Ducument]:* `http://support.apple.com/kb/ht4790`, 2014.
5. Google Inc. Linux Unified Key Setup. *[www Document]:* `https://code.google.com/p/cryptsetup/`, 2014.
6. Google Inc. dm-crypt: Linux kernel device-mapper crypto target. *[www Document]:* `https://code.google.com/p/cryptsetup/wiki/DMCrypt`, 2014.
7. R. Canetti, C. Dwork, M. Naor, and R. Ostrovsky. Deniable Encryption. In *CRYPTO'97*, 1997.
8. A. Skillen. *Deniable Storage Encryption for Mobile Devices.* PhD thesis, Concordia University, 2013.
9. FreeOTFE. FreeOTFE - Free disk encryption software for PCs and PDAs. version 5.21. *Project website:* `http://www.freeotfe.org/`, 2012.
10. A. D.McDonald and M. G. Kuhn. Stegfs: A Steganographic File System for Linux. In *International Workshop on Information Hiding (IH'99)*, 1999.
11. TrueCrypt. Free open source on-the-fly disk encryption software.version 7.1a. *Project website:* `http://www.truecrypt.org/`, 2012.
12. H. Pang, K. lee Tan, and X. Zhou. StegFS: A Steganographic File System. In *19th International Conference on Data Engineering (ICDE'02)*, 2002.
13. R. Anderson, R. Needham, and A. Shamir. The Steganographic File System. In *International Workshop on Information Hiding (IH'98)*, 1998.
14. A. Skillen and M. Mannan. On Implementing Deniable Storage Encryption for Mobile Devices. In *20th Annual Symposium on Network and Distributed System Security (NDSS'13)*, 2013.
15. B. Kaliski. PKCS 5: Password-based cryptography specification,version 2.0. *RFC 2898 (informational)*, 2000.
16. J. Assange, R.-P. Weinmann, and S. Dreyfus. Rubberhose: Cryptographically Deniable Transparent Disk Encryption System. *Project website:* `http://marutukku.org`, 1997.
17. J. Han, M. Pan, D. Gao, and H. Pang. A Multi-user Steganographic File System on Untrusted Shared Storage. In *26th Annual Computer Security Applications Conference (ACSAC'10)*, 2010.
18. X. Zhou, H. Pang, and K.-L. Tan. Hiding Data Accesses in Steganographic File System. In *20th International Conference on Data Engineering (ICDE'03)*, 2003.
19. A. Czeski, D. J. S. Hilaire, K. Koscher, S. D. Gribble, T. Kohno and B. Schneier. Defeating Encrypted and Deniable File Systems: TrueCrypt v.5.1a and the Case of the Tattling OS and Applications. In *3rd USENIX Workshop on Hot Topics in Security (HotSec'08)*, 2008.
20. A. Irwin. Forensic Methods and Techniques for the Detection of Deniable Encryption. *[www document]:* `http://www.cosc.canterbury.ac.nz/ray.hunt/deniable_encryption_tool_a_survey`, 2008.
21. Wikipedia. Hydra. *[www Document]:* `http://en.wikipedia.org/wiki/Hydra`, 2014.