COMPUTERS & SECURITY 72 (2018) 163–174

Available online at www.sciencedirect.com

ScienceDirect

journal homepage: www.elsevier.com/locate/cose

Computers & Security

User-friendly deniable storage for mobile devices

Bing Chang [a,b,c,d], Yao Cheng [e], Bo Chen [f], Fengwei Zhang [g], Wen-Tao Zhu [a,b,*], Yingjiu Li [d], Zhan Wang [h]

[a] State Key Laboratory of Information Security, Institute of Information Engineering, Chinese Academy of Sciences, Beijing 100093, China
[b] Data Assurance and Communications Security Research Center, Chinese Academy of Sciences, Beijing 100093, China
[c] University of Chinese Academy of Sciences, Beijing 100049, China
[d] School of Information Systems, Singapore Management University, Singapore 178902
[e] Institute for Infocomm Research, Agency for Science, Technology and Research (A*STAR), Singapore 138632
[f] Department of Computer Science, Michigan Technological University, Houghton, MI 49931, USA
[g] Department of Computer Science, Wayne State University, Detroit, MI 48202, USA
[h] RealTime Invent, Inc., Beijing 100102, China

ARTICLE INFO

Article history:
Received 15 March 2017
Received in revised form 15 August 2017
Accepted 7 September 2017
Available online 21 September 2017

Keywords:
Plausibly deniable encryption
Mobile security
Near field communication
Thin provisioning
Coercive attack

ABSTRACT

Mobile devices are prevalently used to process sensitive data, but traditional encryption may not work when an adversary is able to coerce the device owners to disclose the encryption keys. Plausibly Deniable Encryption (PDE) is thus designed to protect sensitive data against this powerful adversary. In this paper, we present *MobiPluto*, a user-friendly PDE scheme for denying the existence of sensitive data stored on mobile devices. A salient difference between MobiPluto and the existing PDE systems is that any block-based file systems can be deployed on top of it. To further improve usability and deniability of MobiPluto, we introduce a fast switching mechanism and incorporate the widely-used Near Field Communication (NFC) technology. Users can securely switch from the public mode to the hidden mode within 10 seconds, which is a significant improvement compared to previous solutions. Users can also store strong passwords on NFC cards and tap them to enter the system, which significantly liberates them from the burden of memorizing and typing strong passwords. Most importantly, the users can deny the existence of the hidden data without the skill to camouflage as long as the NFC cards are used properly.

1. Introduction

Mobile devices are frequently used to process personal or mission critical data. Encryption can be utilized to protect those sensitive data. However, it may not work for all the cases. For example, in certain geopolitical areas with tensions, the border inspector may compulsively require the passengers to reveal the content stored encrypted on their mobile devices. In extreme situations, the device owner may be tortured to surrender decryption key. This could prove detrimental and may compromise security of particular professionals such as human rights activists, who may possess evidence of violence.

Plausibly deniable encryption (PDE) was explored to maintain the privacy of communicated data against a coercive attacker, who can approach and coerce either the sender or the receiver into revealing the decryption keys (Canetti et al., 1997). This practice should not be confused with encryption,

* Corresponding author.
E-mail address: wtzhu@ieee.org (W.-T. Zhu)
https://doi.org/10.1016/j.cose.2017.09.005
0167-4048/

as regular encryption is overt, while PDE is covert. PDE can be applied to storage encryption and a variety of PDE systems have been published for PC platform, including Rubberhose (Assange et al., 2012), TrueCrypt (2012), etc.

As the first PDE system implemented for mobile devices, Mobiflage (Skillen and Mannan, 2013) requires a physical or an emulated FAT32 SD card which is not necessarily present in mobile devices. Mobiflage is then extended (Skillen and Mannan, 2014) to support Ext4 file system by modifying the driver of Ext4. Although the extended Mobiflage no longer requires a physical or an emulated FAT32 SD card, its extensive modifications to the Ext4 driver may indicate the use of PDE which may lead to compromise of deniability. MobiHydra (Yu et al., 2014) improves Mobiflage by adding support to multiple levels of deniability and mode switching without rebooting, but it still requires a physical or an emulated FAT32 SD card.

All the prior solutions (Assange et al., 2012; McDonald and Kuhn, 2000; Ragnarsson et al., 2012; Skillen and Mannan, 2013, 2014; TrueCrypt, 2012; Yu et al., 2014) are not suitable for mobile devices, due to their poor performance, getting inadequate support from the mobile operating systems, or being forced to modify the associated file systems as a last resort. In this work, we propose *MobiPluto*, a user-friendly PDE solution for mobile devices. MobiPluto can achieve a user friendly design, providing several salient features:

- **File-system-friendly.** As data hiding is achieved at the block level, any block-based file systems can be deployed without modifications. To the best of our knowledge, no prior work can provide such a novel feature.
- **Short switching time.** Users can switch from the public mode to the hidden mode within 10 seconds. Previous solutions require more than 1 minute which may lead to missing the best timing to collect sensitive data.
- **Deniability as a side-effect.** MobiPluto achieves deniability as nothing but a "side-effect" of combining thin provisioning with encryption.
- **Better usability and deniability.** Users can store the passwords in NFC cards instead of memorizing them. Moreover, users can tap NFC cards to input passwords, eliminating the need of typing passwords. More importantly, by destroying the NFC card, users can eliminate the possibility of the sensitive hidden data of being disclosed, significantly improving deniability.

This article extends our conference version (Chang et al., 2015) in the following aspects:

- We present a new switching method which eliminates the need of rebooting the device when the user wants to switch from the public mode to the hidden mode. The user can enter the hidden password in the screen lock and the system will switch to the hidden mode. The switching time is less than 10 seconds. Note that previous solutions need to reboot the device which takes more than 1 minute. Security analysis shows that fast switching does not bring deniability issues.
- We take advantage of the NFC technology so that users can store strong passwords on NFC cards and use these cards for admission, relieving themselves of memorizing and entering complicated passwords. Users can also deny the existence of any hidden data as long as the adversary does not possess usable cards.
- We implement the fast switching mechanism and the NFC card support on our system. We also provide a thorough security analysis on the fast switching mechanism and the adoption of the NFC technology. In addition, we conduct experiments to demonstrate the usability improvement owing to the introduction of NFC. The results show that using NFC cards is 18 times faster than typing a lengthy password. We also conduct a more comprehensive discussion on and comparison with related work.

The rest of the paper is organized as follows. Section 2 presents the background. In Section 3, we introduce the system model and the threat model. In Section 4, we describe MobiPluto design. In Section 5, we discuss the implementation for Android. We present the evaluation results in Section 6, including security analysis and performance evaluation. Section 7 presents related work and Section 8 is the conclusion.

2. Background

2.1. *Deniable encryption*

Plausibly deniable encryption (PDE) was first explored to maintain the privacy of communicated data against a coercive attacker, who can approach and coerce either the sender or the receiver into revealing the decryption keys (Canetti et al., 1997). When being applied to storage encryption, PDE allows a data owner to decrypt a ciphertext to a plausible and benign decoy plaintext with a different key, such that the data owner is able to deny the existence of the original sensitive data (Skillen and Mannan, 2013). To provide plausibility, a PDE system usually requires that (Ragnarsson et al., 2012), 1) the decoy plaintext can be normally found on a computer; 2) all the ciphertexts should be "accounted for", i.e., having a plausible explanation. Existing PDE systems rely on either steganography (Anderson et al., 1998; McDonald and Kuhn, 2000; Pang et al., 2003) or hidden volumes (Skillen and Mannan, 2013; TrueCrypt, 2012; Yu et al., 2014) to achieve deniability.

2.2. *Thin provisioning*

Thin provisioning (Kernel, 2017) has been designed to optimize storage utilization by eliminating the need for installing unnecessary storage capacity. With thin provisioning, a storage administrator only allocates logical storage space to an application and the system will not release the physical storage capacity until it is actually required. This "on-demand" storage avoids pre-allocating physical storage capacity, eliminating the waste caused by unused capacity. In the Linux kernel, thin provisioning has been implemented by the `dm-thin-pool` module, which works with two devices, a data device and a metadata device. The data device contains blocks from various volumes, allocated sequentially from the beginning, while the metadata device contains the block mappings. Thin provisioning is added to LVM and can provide much more flexible storage management. Logical Volume Manager (LVM (Levine, 2016)) has gained

popularity on Android for being able to flexibly handle storage (CyanogenMod; Xda-Developers, 2012).

This work tries to adapt thin provisioning to build file-system-friendly deniable storage for mobile devices, for the following reasons: 1) The `dm-thin-pool` module has been added to the kernel, and we can simply rely on the existing kernel features to build PDE systems for mobile devices; 2) A thin volume can be used to install any block-based file systems, and thin provisioning can transform non-sequential allocation on the thin volume to sequential allocation on the underlying storage. This makes it possible to combine both thin provisioning and hidden volumes to build file-system-friendly PDE systems.

2.3. *Near field communication (NFC)*

NFC (Curran et al., 2012) is a set of short-range (typically 10 cm or less) wireless technologies which is used widely nowadays (e.g., in mobile payments). Most recent smartphones are equipped with NFC technology. This work tries to make use of NFC technology to improve usability and deniability of PDE systems. Since NFC cards and NFC-compatible phones are widely used, we do not need to introduce any extra device which may lead to compromise of deniability. NFC can improve the usability, because users can store strong passwords on NFC cards and tap the NFC cards to enter the system. This releases users from the burden of memorizing and typing strong passwords. In addition, NFC can help improve the deniability offered by a PDE system, because users can deny the existence of the hidden data as long as the adversary does not obtain the NFC cards used to encrypt/decrypt the hidden data. More importantly, users can eliminate the possibility of the sensitive hidden data of being disclosed by destroying the NFC cards (e.g., burn, cut or break the NFC cards).

3. Assumptions

3.1. *System model*

We mainly consider mobile devices equipped with storage media that expose a block-based access interface. Such block-based storage media are used extensively as the internal or external storage for mobile devices nowadays (Skillen and Mannan, 2013), e.g., eMMC (Samsung, 2015), microSD cards, etc. MobiPluto needs to be merged with Android code stream, such that the PDE capability is widespread, and the availability of PDE will not become a red flag. For a mobile device that uses MobiPluto, we assume the mobile OS, the bootloader, as well as the firmware and the baseband OS are all malware-free (i.e., trusted). Especially when the device is in the PDE mode, the user will not use malicious apps controlled by the adversary.

3.2. *Threat model*

We consider a computationally bounded adversary, who can fully control a mobile device after having captured the device's owner. The adversary can get root privilege of the device, and fully control the device's internal and external storage. Additionally, the adversary can coerce the device's owner to surrender keys or NFC cards, in order to decrypt the storage and obtain sensitive data. As mobile devices usually communicate with the external environment, the adversary may also collude with the wireless carriers or the ISPs to collect the network activity logs of victim devices.

We do not consider an adversary who can continuously monitor a victim device, and can stealthily take periodic snapshots of the device's storage. This would be practical as the adversary usually can have access to a mobile device only after seizing the user (Skillen and Mannan, 2013). The adversary does not capture a mobile device which is working in the PDE mode. Otherwise, he/she can trivially retrieve the sensitive data from the PDE mode. In addition, the adversary may know the design of MobiPluto. However, he/she does not have any knowledge on the keys, passwords, and the offset of the hidden volume. Finally, the adversary will stop coercing the device's owner once being convinced that the decryption keys have been revealed.

4. MobiPluto design

In this section, we present MobiPluto, a user-friendly PDE system for mobile devices by utilizing hidden volumes and thin provisioning. MobiPluto is named after the *Helmet of Pluto*, which is capable of turning its wearer invisible according to classical mythology (Crystalinks, 2015).

4.1. *Overview*

MobiPluto is able to deny the existence of sensitive data by hiding volumes (storing sensitive data) in the empty space of the storage medium. For simplicity of presentation, we consider a simple case which has only two volumes: a public volume created for storing regular data, and a hidden volume created for storing sensitive data. The data stored in the hidden volume are those whose existence the owner wants to deny. If multi-level deniability is required, the number of hidden volumes can be varied accordingly (Yu et al., 2014). By utilizing the interesting properties offered by thin provisioning (Section 2.2), we build thin logical volumes ("thin volumes" for short) to achieve a file-system-friendly deniable storage solution for mobile devices.

In MobiPluto, the public volume is protected by a decoy password and the hidden volume is protected by a hidden password. Specifically, we use a randomly generated decoy key to encrypt the public volume and use the decoy password to encrypt the decoy key, which will be stored in the encryption footer (located in the last 16 KB of the userdata partition). We use a randomly generated hidden key to encrypt the hidden volume, and use the hidden password to encrypt the hidden key. The encrypted hidden key will be stored at a secret offset which is determined by the hidden password. To achieve a file-system-friendly PDE design, we choose to create thin volumes in the public volume and the hidden volume, respectively. A thin volume created on top of the public volume allows to deploy any type of block-based file systems for managing public data. Similarly, a thin volume created on top of the hidden volume can be used to deploy any block file systems for managing

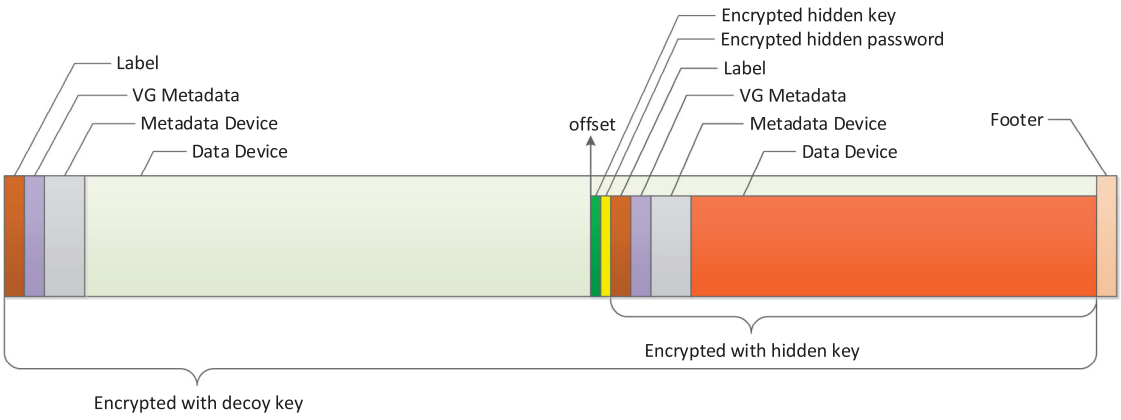

Fig. 1 – MobiPluto storage layout.

hidden sensitive data. Thin provisioning can ensure that the data in the public thin volume will not over-write the data in the hidden thin volume. Upon booting, the password will be used to decrypt the key and the decrypted key will be used to decrypt the corresponding volume. Specifically, if the owner provides the decoy password, the system will boot into public mode, in which the content of the public volume will be present. The hidden volume part of the storage looks no different from random data. In this fashion, the adversary (Section 3) will be convinced that no sensitive data exist. To process sensitive data, the owner should use the hidden password and switch to the PDE mode, in which the hidden volume will be located and then decrypted using the hidden key.

To further improve MobiPluto's usability, we introduce a fast switching mechanism and propose to use NFC cards to store strong passwords. The existing PDE solutions for mobile devices (Skillen and Mannan, 2013, 2014; Yu et al., 2014) all require rebooting to switch modes, which is unfortunately time-consuming (need more than 1 minute). Our new fast switching mechanism eliminates the need for rebooting the device and is able to reduce the switching time to less than 10 seconds (Section 6.2). To protect sensitive data, we usually need a strong password for the hidden volume. However, if the password is chosen by the user, security cannot be ensured as the user tends to choose weak password. In addition, if the user needs to memorize the password, when he/she is facing coercion, the deniability will really depend on his/her willpower and ability to camouflage. To address these concerns, we first generate a strong password for the user and then store the password in an NFC card with security features, by which we can allow the user to choose a strong password without struggling to memorize it. Utilizing NFC technology is practical as most of the modern mobile devices are equipped with NFC features. In addition, by destroying the NFC card, the user can eliminate the possibility that the password for the hidden sensitive data will be leaked, and deniability can be enhanced in some sense.

4.2. *Storage layout*

Public volume: we create an *encrypted block device* using the decoy key over the entire disk. We then use LVM to create a *physical volume* on the *encrypted block device*. A physical volume label will be placed in the second 512-byte sector. We further create a *volume group* within this *physical volume*. The metadata of the *volume group* will be placed right after the physical volume label (see Fig. 1). Next, we create a small logical volume, which is used as the *metadata device* for the *thin pool*. We also create a large logical volume in the remaining space of the *volume group*, which is used as the *data device* for the *thin pool*. LVM allocates space in the *data device* to *thin volumes*, and the corresponding mapping information is kept in the *metadata device*. Each *thin volumes* can allow to be deployed any block-based file systems (e.g., Ext4).

Hidden volume: we first calculate an offset using the hidden password. At the offset, the encrypted hidden key (encrypted using a key derived from the hidden password) and the encrypted hidden password are stored. Note that the hidden password is encrypted by the hidden key. We create another *encrypted block device* within the space between the offset and the end of the storage medium (see Fig. 1). A *thin volume*, which is used to store the sensitive hidden data, is created in this block device following the aforementioned steps. Note that any block-based file systems can also be deployed on this thin volume.

4.3. *File-system-friendly deniability*

Different file systems may use different allocation strategies. FAT32 has the nature of supporting hidden-volume mechanism due to its concentrated metadata and sequential allocation. For a FAT32 formatted device, we can simply place the hidden volume somewhere in the second half of the storage medium, by which the data in the hidden volume will not be easily overwritten by the data in the public volume. However, it will be problematic to place hidden volumes in an Ext4 formatted device, because: First, the data in the hidden volumes may overwrite all or part of the public volume's metadata. By observing this abnormal overwrite, the adversary may suspect the existence of hidden volumes; Second, the data from the public volume may easily overwrite the data in the hidden volumes. In general, if the device is formatted with a file system having similar characteristics like Ext4, we cannot simply create hidden volumes within it.

Thin provisioning can help address the aforementioned concern. By using thin provisioning, we can first create a thin

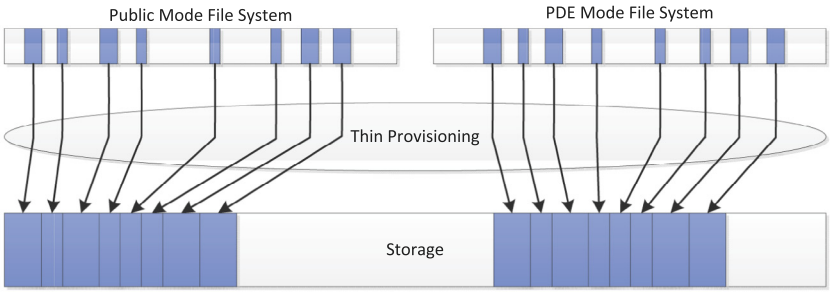

Fig. 2 – Thin provisioning and the hidden volume.

pool and then create thin volumes in the thin pool. The thin pool ties together a small metadata device and a data device, and the latter occupies most of the storage space. Interestingly, the data device in thin provisioning is used linearly (i.e., the space maps allocate space linearly from front to back). To avoid fragmenting free space, the allocation always goes back and fills the gaps in the data device. In this way, thin provisioning can transform the possibly non-sequential allocation on the thin volume to sequential allocation on the data device. This renders it possible to combine hidden volumes and thin provisioning to achieve a PDE solution that allows any block-based file systems to be deployed.

Thin provisioning provides a logical block device (a thin volume), which allows to deploy an arbitrary block-based file system. No matter how the file system uses the logical device, the changes on the data device are linear (see Fig. 2). On the thin volume, the non-sequential addresses are mapped to sequential addresses on the data device through mappings in the metadata device. As a result, we can create hidden volumes on the second half of the public volume's data device. Since the data in the public volume are always written sequentially to the data device (regardless of the allocation strategies of the deployed file system on the thin volume), it is very unlikely that they will overwrite the hidden volumes. In other words, the file-system-friendly feature can be achieved.

4.4. *Size calculation*

Next, we describe how to calculate the offset and the size of the hidden volume. For deniability purpose, the thin volume size should be set the same as the total disk size in the public mode.

The hidden volume starts at a specific offset on the storage medium. MobiPluto generates this offset using hidden password in the following way (Skillen and Mannan, 2013):

$$offset = 0.75 \times vlen - (H(pwd \| salt) \bmod (0.25 \times vlen)).$$

Here, *vlen* denotes the number of 512-byte sectors on the physical block device; H is a PBKDF2 iterated hash function (Kaliski); *pwd* is the hidden password; *salt* is a random salt value for PBKDF2 and it is the same as the one stored in the encryption footer. Thus, we do not need to store another salt.

The hidden volume is stored after the encrypted hidden key (stored at the offset). The hidden volume size can be calculated as follows:

$$S_{hid} = vlen - offset - S_{key} - S_{footer}.$$

Here, S_{key} and S_{footer} denote the sizes of the encrypted hidden key and of the encryption footer respectively.

4.5. *Fast switching mechanism*

Fast switching is a desired feature for PDE systems on mobile devices. Imagine that a device owner just faces an emergency and he/she wants to instantly switch the device to the hidden mode to collect some sensitive data, which requires a short switching time. However, the existing PDE solutions for mobile devices (Skillen and Mannan, 2013, 2014; Yu et al., 2014) all require rebooting to switch modes, which is time-consuming. The main concern is how to switch fast from the public mode to the hidden mode without compromising deniability. We find it possible to restart the Android framework rather than the entire device, which can help significantly reduce the switching time. We describe our new fast switching mechanism in the following. The default screen lock app of Android is chosen to be the entrance of the hidden mode, because it can allow the user to enter the password and its widespread usage would not become an indication of the existence of deniability. The fast switching process is shown in Fig. 3. After the user enters a password, the system first tests whether it is the screen lock password. If so, the screen is unlocked. Otherwise, the system calculates the offset and tests whether the password is the hidden password. If so, the system will shut down the Android framework, unmount the public volume, decrypt and mount the hidden volume, and then restart the Android framework. Otherwise, the system will ask the user to enter another password.

4.6. *Activation by NFC cards*

4.6.1. *DESFire EV1*

In our design, we use the DESFire EV1 card (NXP, 2016), which provides advanced security features, as the activation of MobiPluto. A DESFire EV1 card could hold up to 28 applications, each of which can provide 32 files. Hardware DES and AES algorithms can be applied to both card level and application level. Integrity check is carried out for every communication message.

A three-pass authentication is used to establish a secure communication channel between a mobile device and a card before each access. As shown in Fig. 4, k denotes the shared

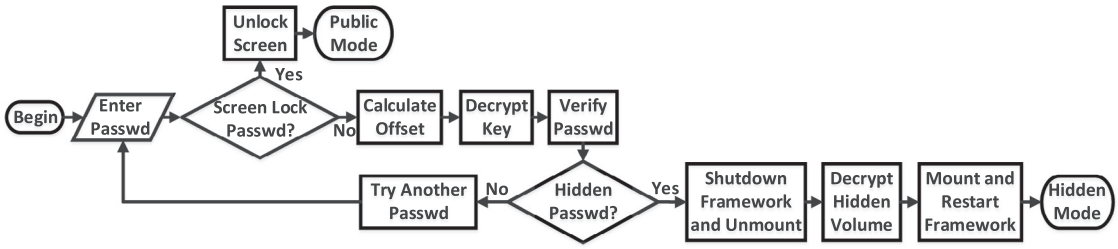

Fig. 3 – MobiPluto fast switching process.

symmetric card key between a mobile device and a DESFire card, $E_k(x)$ means the encryption of x using key k. R_a and R_b denote the random numbers chosen for each round of communication. R'_a and R'_b are the results of left rotation of R_a and R_b, respectively. When a mobile device requests to access the card storage, it sends out the authentication request containing the index number of the specific key on the card. Then, the card uses the key indicated by the key number to encrypt a random number R_b, and sends the cipher text back to the mobile device. The mobile device decrypts the data using the key, and rotates the decryption result to the left by 8 bits, which is then denoted as R'_b. After that, the mobile device chooses a random number R_a, and encrypts the concatenation of R_a and R'_b. The card compares the received R'_b with R_b to check whether R'_b is the result of R_b rotating left by 8 bits. If so, it indicates that the mobile device indeed shares the same key with the card. Later, the card rotates the received R_a to the left by 8 bits, which is then denoted as R'_a, and sends it back to the mobile device. The mobile device compares the received R'_a to R_a. If R'_a equals to the result of R_a rotating left by 8 bits, it means that the card is the target card instead of any fake card which tries to replay valid messages. Finally, after the computation and comparison, both sides can verify each other's possession of k. A session key is generated after each authentication, and will be used later to encrypt communication messages. DESFire cards support a set of cryptographic algorithms, each of which computes their session keys based on R_a and R_b using different formulas. For example, the AES session key is generated by combining R_a and R_b as the formula shown below.

$$SessionKey = R_a[0,3] \| R_b[0,3] \| R_a[12,15] \| R_b[12,15].$$

4.6.2. *NFC mode initialization*

The initialization of MobiPluto in NFC mode, which mainly generates the keys and sets up the card, should be performed in a secure and isolated environment. It requires one NFC card for each mode, including the decoy card for the public mode and the hidden card for the PDE mode.

We build an assisting application for Android which fulfils the task of initializing NFC cards. This application can be used to generate strong passwords. After the PDE storage is created and the hidden mode is booted, this application will be installed separately in the public and the hidden mode to initialize the corresponding NFC card. The application writes the generated password to the public and hidden NFC card, respectively. After successful writes, users can use the NFC card to activate the corresponding mode. Note that the application does not need to be deleted after having activated the corresponding NFC card. This will not compromise deniability, because the assisting application in the public mode is purely used to activate the NFC card for the public mode, and has nothing to do with the hidden mode. In addition, the assisting application in the hidden mode will not create any trace to the public mode, and the adversary who can have access to the public mode, is not able to learn anything about the assisting application of the hidden mode.

When the device is booted but fails to find a valid Ext4 file system on the userdata partition, the system would ask for an NFC tap instead of a password. It depends on the NFC card provided by the user whether the device boots into the public mode or the PDE mode. If the NFC card provided by the user is the decoy card, the device would successfully decrypt the key in the footer and further decrypt the public storage with this key. If the NFC card provided by the user is the hidden card, the system would calculate the offset and boot into the PDE mode. The system would not boot until a valid card is provided.

During the use of NFC-enabled MobiPluto, users do not need to memorize the passwords with high entropy. Instead, they just need to take care of the NFC cards. Also, users can make duplicate cards and store them in a safe place for backup. In critical situations during the use of PDE mode, we highly recommend users to dispose the hidden NFC card or destroy it with a little twist, and reboot the device into public mode to keep the deniability.

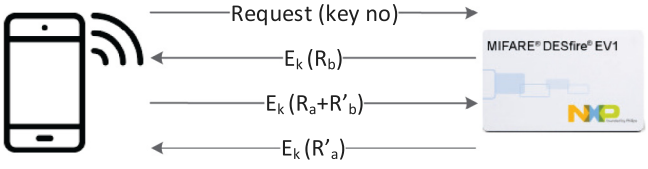

Fig. 4 – Three-pass authentication between a mobile device and an NFC card.

5. Implementation

We implement a prototype of MobiPluto on an LG Nexus 4 phone and its Android version is 4.2.2. We modify the Android volume daemon (VOLD) and the Android screen lock app to implement the basic function, the fast switching mechanism and NFC cards support. We also change some of the default

kernel configurations so that we can use the required features (e.g., thin provisioning in the kernel). In addition, we compile Logical Volume Manager (LVM) and thin provisioning tools (Thornber, 2015) for Android and put them in the boot image.

5.1. *Thin provisioning on android*

Now, we describe how we run thin provisioning on Android. Thin provisioning is available in the Linux kernel since version 3.2. Android 4.2.2 uses the Linux kernel 3.4, but the default configuration disables this feature. Therefore, we have to enable it and recompile the kernel. In addition, since we use XTS-AES (Martin, 2010), the `xts` and `gf128mul` kernel crypto modules should be enabled, too. It is not enough to enable only the thin provisioning feature in the kernel. We have to use LVM to setup logical volumes. Furthermore, we use thin provisioning tools to activate the thin volumes. Thus, we compile LVM and thin provisioning tools for Android. The compiling process requires a specific environment for Android. Besides the `gcc` and the `g++` tool chains for Android, both tools need to be statically linked. For static compiling, we add "`–enable-static_link`" when configuring LVM and we add "`LDFLAGS=-static`" to the makefile of thin provisioning tools. Furthermore, the default LVM configuration does not enable the thin provisioning, so we add "`–with-thin=internal`" in the LVM configuration for that. Next, we add both tools to the `boot.img` using `unpackbootimg` and `mkbootimg` which are provided by AOSP (Android, 2017). Note that we modify the access permissions of these files by adding "`chmod`" command to `mako.init` in the `boot.img`. Otherwise, we are not able to use them. After enabling thin provisioning feature in the kernel and adding the tools to the `boot.img`, we can use thin provisioning on Android.

5.2. *User interface and pre-boot authentication*

Users can use a command-line utility, `vdc`, to activate MobiPluto PDE; the command is as follows: "`vdc cryptfs pde wipe ⟨decoy_pwd⟩ ⟨hidden_pwd⟩`". The default Android shell does not maintain history, and thus the commands or the passwords cannot be retrieved from a captured Android device.

To make the hidden volume indistinguishable, we first wipe the entire internal storage with random noise. For the public volume of MobiPluto, we use a random key to create an encrypted block device and store the encrypted key and the salt in the encryption footer. We then create a thin volume on the encrypted block device and create an Ext4 file system on the thin volume. We have described the procedure of initializing a public volume and a hidden volume in Section 4.2. The size of the metadata device is calculated according to Section 4.4. S_{req} is chosen as 50 for now, and a more reasonable value will be investigated in our future work.

When the device is booted but fails to find a valid Ext4 file system on the userdata partition, the system will ask the user to enter a password. The default Android will use this password to decrypt the key in the footer and decrypt the storage medium with this key. If a valid Ext4 file system can be mounted, the system will continue to boot. However, MobiPluto creates a thin volume instead of an Ext4 file system on the encrypted block device. It would be time consuming if MobiPluto enables the thin volume to check the existence of Ext4 file system by mounting it. To reduce the time of checking, we just test whether there is a thin volume on the encrypted block device. When the user enters a password, the system will check whether there is a thin volume on the public volume. If so, the password is the decoy password and the system will boot into the public mode. Otherwise, the system will calculate an offset and check whether there is a thin volume on the hidden volume. If so, the password is the hidden password and the system will boot into the PDE mode. Otherwise, the system asks for another password.

5.3. *Fast switching support*

5.3.1. *Changes to Android screen lock*

The original screen lock checks whether the password being entered is the screen lock password. If so, the system simply unlocks the screen. Otherwise, the system will request a correct password. We modify the Android screen lock as an entrance of the hidden mode. The system first checks whether the password is the screen lock password. If not, the system will pass the password to VOLD which will check whether the password is the hidden password. If so, the system will switch to the hidden mode. Otherwise, the user is requested to enter another password.

5.3.2. *Changes to Android Volume Daemon*

We implement a switching function in Android Volume Daemon (VOLD) to verify the password and switch to the hidden mode. The input of this function is a password and it returns –1 if the password is not the hidden password. Otherwise the system switches to the hidden mode. To verify the password, the system first reads the salt from the encryption footer. Then an offset is derived using the password and the salt. After that, the system reads the encrypted master key and the encrypted password at the offset. Note that the encrypted password is derived by encrypting the password using the hidden key. Then the system decrypts the master key using the password and salt. To verify the password, the system encrypts the password using the master key. If the result is the same as the previous encrypted password, the password is correct and the system begins to switch to the hidden mode. Otherwise the password is wrong and the function simply return –1. To switch to the hidden mode, the system first shuts down the Android framework to unmount "/data" partition. Then the volume group will be deactivated and the encrypted block device will be deleted. After that, a new encrypted block device will be created beginning at the offset using the hidden key. The thin volume on the encrypted block device will be activated and mounted to "/data". Then the Android framework is restarted and the hidden mode is activated.

5.4. *NFC card support*

Besides activation using passwords, MobiPluto provides activation using NFC cards for both the public mode and the hidden mode. At the initialization stage, which is supposed to be in a secure and isolated environment, users can initialize the

NFC-enabled MobiPluto with the assistance from the application we have designed.

The initialization takes two steps with the help of our assisting application. The first step is to use a command-line utility, `vdc`, to activate MobiPluto PDE. This step requires two passwords for both the public mode and the hidden mode. Our assisting application will generate high-quality passwords for the user to activate MobiPluto PDE. The second step is that, after the user activates MobiPluto PDE, he/she needs to boot into the hidden mode to complete the NFC card initialization. Our assisting application is used to initialize the NFC card configuration, including configuring the encryption mode and card key, and writing the passwords to the corresponding NFC cards.

During the use of NFC-enabled MobiPluto, when the device is booted but fails to find a valid Ext4 file system in the userdata partition, the system will wait for the user to tap an NFC card. When the user taps a card, the activity showing on screen reads the stored passwords. The system will boot into different modes according to the card that the user presents.

We face several challenges in providing NFC support. First of all, when the device is booted but fails to find a valid Ext4 file system, it is a minimal system that only provides some core functions on Android. Unfortunately, NFC module is not considered as a core function. We manage to add the NFC module to the core application list by modifying the attribute "`coreApp`" to "`true`" in its manifest file. In this case, MobiPluto can use NFC module at the early stage of the boot.

Second, we have to consider the security of password stored on the card. Any unauthorized access should be denied. During the initialization, the card key of the DESFire card is activated. Meanwhile, we configure the card to the fully enciphered mode and use AES encryption. This configuration means that any subject accessing to the card is required to possess the card key, and at the same time, all the communications between the device and the card are encrypted using AES session keys which are different for different authentication sessions.

Third, it is important to store the card key securely. The card key to access the NFC card is shared between the device and the card. Anyone with the card key can get access to the card and obtain the password. However, the challenge is that before Android finds a valid Ext4 file system, i.e., when Android is expecting a password or an NFC card tap, the accessible storage is very limited. The card key, which is required to read the NFC card, cannot be stored in normal storages because they are not mounted yet. In this situation, a solution is to use the storage in the encryption footer. We use the encrypted bytes in the encryption footer as the card key, i.e., the first 16 bytes are used as the card level key, and the latter 16 bytes are used as the application level key. In our assumption, the adversary cannot get the NFC card for the PDE mode after getting the mobile device. Therefore the card key can prevent any adversary who gets the hidden card from accessing the stored data.

Last, in our implementation, we *cannot* use Mifare advanced SDK (Mifare, 2016), which encapsulates primary operations on DESFire card on Java level, to perform operations on the NFC card. MobiPluto needs to be merged into Android code. However, Android cannot accept the obfuscation characters in Mifare SDK during the compiling. Thus, we process the command and data in DESFire protocol at byte level, and further implement the whole process of three-pass authentication, communication message encryption, message authentication code calculation, and NFC card read/write operation.

6. Evaluation results

6.1. Security analysis

6.1.1. Deniability provided by hidden volumes

In general, storage units are not filled with random data when coming from the manufacturers. In addition, the operating system installation procedures do not fill the entire storage with random data. Thus, the adversary may suspect the existence of PDE after decrypting the disk with a decoy key, as it can find out random data which is not "accounted for" (Section 2.1). A plausible explanation from the device owner can be that he/she always fills the disk with random data before putting file systems on it. Although the adversary has the full knowledge of MobiPluto design (Section 3.2), without knowing the secrets, it cannot prove the existence of hidden volumes, as they are encrypted by FDE and are indistinguishable from the initially filled random data (note that MobiPluto uses the encryption function for FDE as the pseudorandom number generator). To prevent the adversary from identifying hidden volumes without recovering any hidden plaintexts, MobiPluto uses XTS as the block cipher mode, which has been designed for disk encryption, and is able to prevent attacks such as ciphertext manipulation and cut-and-paste (Martin, 2010).

6.1.2. Thin provisioning/LVM specific security issues

MobiPluto uses both thin provisioning and LVM tools. Thin provisioning/LVM in either the public mode or the hidden mode will have its own label, VG metadata, metadata device and data device (Sec. 4), which are stored in its own userdata partition, encrypted by `dm-crypt` with different keys (i.e., decoy key and hidden key). For the hidden mode specifically, the location of the userdata partition is secret and can only be derived when knowing the hidden password. Thus, when the adversary looks into the public volume (i.e., in the public mode), he will not have any clues of the data related to thin provisioning/LVM in the hidden volumes.

6.1.3. Fast switching related issues

MobiHydra (Yu et al., 2014) proposes to use "the shelter volume" to store sensitive data temporarily, but it suffers from the side channel attack (Czeski et al., 2008). Due to the shared OS, the information of the hidden data may be recorded in the public volume (Czeski et al., 2008), leading to compromise of deniability. In addition, MobiHydra (Yu et al., 2014) needs a special app to process the hidden data, the existence of the special app may make the user suspicious. However, our design can prevent the information leakage problem, since we isolate the hidden volume from the public volume. Although the hidden password is entered in the public mode, the Android screen lock does not record the entered password. As a result, the security of the hidden password is ensured. In addition, after the hidden mode is off, the RAM should be cleared to

prevent information leakage from RAM. To achieve this goal, we only support fast switching from the public mode to the hidden mode. The user has to reboot the phone to switch from the hidden mode to the public mode, so as to clear the information of the hidden data in the RAM.

6.1.4. *NFC-related issues*

It would not weaken the security features of MobiPluto due to the use of NFC cards. Since the deniability provided by hidden volumes is already discussed, we focus on the impacts of **deniability** introduced by NFC cards, including the assisting application, the card key in the encryption footer, and the operation that reads data from the card. First, the existence of the hidden card cannot be inferred by the access to the decoy card. The user could easily deny the existence of the hidden NFC card, as well as the hidden storage. Even in a critical situation that the leakage of sensitive data leads to severe consequences, the user could dispose or destroy the hidden NFC card to prevent the adversary from having access to any sensitive data. Users can retrieve the data in hidden storage once they obtain the backup cards, which are easy to make with the help of our assisting application. Regarding the card key in encryption footer, we use the encrypted key bytes as the card key. We do not modify the encryption footer at all, so it would not decrease deniability. As for the operations, the decoy card and hidden card are initialized in the same way. Therefore, accesses to the decoy card and the hidden card are indistinguishable.

6.1.5. *Cloning*

Cloning is one of the most serious threats to card-based systems. If the hidden card is cloned before the adversary obtains the mobile device, the adversary can access the hidden storage once he/she obtains the mobile device. To prevent a card from being cloned, we can adapt two anti-cloning precautions. First, the access to the password stored on the card is protected properly by the card key that is 128 bits long. Even if the adversary could forge a card of the same unique ID as our original card using the state-of-art techniques (Kasper et al., 2010), he/she cannot have access to our deniable storage due to the absence of the card key. Second, we use DESFire EV1 to prevent the non-invasive side-channel attack. This attack extracts secret information from contactless cards by measuring the electromagnetic emanations of a card while it carries out a cryptographic operation (Kasper et al., 2009). DESFire EV1 cards in our implementation are not affected by such an attack. As a consequence, it is not practical for the adversary to produce a cloned card to access our deniable system.

6.1.6. *Eavesdropping*

To prevent eavesdropping, a session key is generated during the three-pass authentication before the data transit begins. The communications in the establishment procedure of the session key are encrypted by the card key. This is configured during the initialization in a secure and isolated environment. Moreover, the card key is never transferred outside the card. It is the key number that is transited from the device to the card. The key number can be set to any number between 0 and 13 during the initialization, which is meaningless to the adversary. As such, the establishment message does not leak any information about R_a, R_b, and hence the session key. The subsequent communication is encrypted by this session key. In addition, there is little chance for the adversary to carry out the ciphertext-only attack, because the session key is different after each authentication.

6.2. *Performance evaluation*

6.2.1. *Throughput performance*

The main difference between MobiPluto and the default Android FDE is that MobiPluto uses 1) XTS-AES and 2) thin volumes, so we intend to understand how these two points impact the performance. We use three experiments to understand the performance differences among 1) the default Android FDE, 2) the XTS Android FDE (i.e., only replace the CBC-AES of Android FDE with XTS-AES), 3) the public mode of MobiPluto, 4) and the PDE mode of MobiPluto. We conduct experiments on the internal storage of an LG Nexus 4. We use a popular Linux command line tool, "`dd`", to measure the storage performance of the above four systems. For the write speed, we execute the command, "`time dd if=/dev/zero of=test.dbf bs=400M count=1 conv=fdatasync`". It measures the time for writing a 400 MB file to the storage. In addition, we use "`time dd if=234.mp4 of=/dev/null bs=400M`" to measure the read speed. Here "`234.mp4`" is a multimedia file and its size is 3 GB. Note that "`dd`" command tests the sequential I/O performance. Additionally, we use a popular benchmark, Bonnie++ (Coker, 2009), to test the sequential I/O operations. We conduct each experiment 10 times, and the average results and standard deviations are shown in Fig. 5. We can see that the XTS-AES has a small impact on the read speed, and the use of thin volumes has little influence on the performance.

6.2.2. *Fast switching performance*

To test the performance of the fast switching mechanism, we record and analyze the time for switching from the public mode to the hidden mode. By reading the system logs, we obtain the details about the switching time. We repeat the experiment 10 times and the results show that the average switching time is less than 10 s. Specifically, the switching process includes

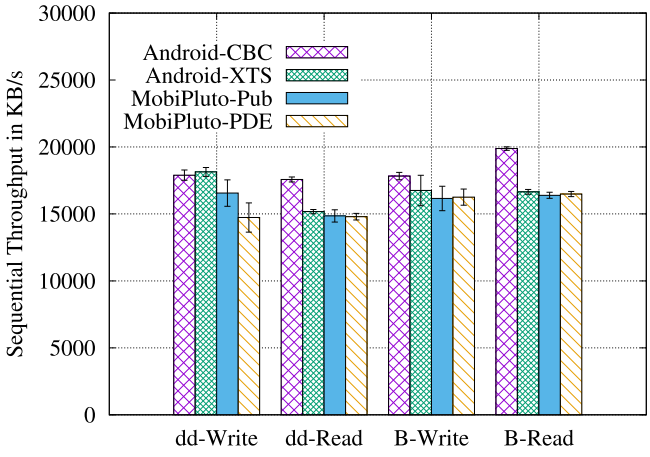

Fig. 5 – Average throughput and standard deviation in KB/s (B: Bonnie++).

Table 1 – Time for getting the password from the NFC card.

Action	Android-NFC	Read Key	Session Key
Time	0.13 ± 0.03 s	0.06 ± 0.03 s	0.07 ± 0.02 s
Action	Read Password	Decrypt Password	Total
Time	0.12 ± 0.04 s	0.05 ± 0.02 s	0.50 ± 0.06 s

three main steps: password verifying, volume switching and Android framework restarting. It takes the system 0.57 ± 0.03 s to verify the password and 3.21 ± 0.24 s to switch the volume. Meanwhile, it takes the system 6.19 ± 0.16 s to restart the Android framework. In total, the system needs 9.97 ± 0.18 s to switch from the public mode to the hidden mode. Previous solutions (Skillen and Mannan, 2013, 2014; Yu et al., 2014) all require rebooting to switch modes which is time-consuming. To reboot the phone, a user should first shutdown the phone, which usually needs about 16 s on an LG Nexus 4. After that, the user turns on the phone and the phone will show an interface for the user to enter the password. This usually needs about 33 s on an LG Nexus 4 phone. After the user enters the hidden password, the system boots into the hidden mode, which needs about 13 s. In total, it takes the previous solutions more than 1 minute to switch the modes. Our solution does not need to reboot the phone and the switching time of our system is less than 10 seconds.

6.2.3. *NFC performance*

To understand the performance impact due to the use of NFC cards, we record and analyze the time for reading passwords from NFC cards. By reading the logs of the system, we can get the details about the reading time. We repeat the experiment 10 times and the results are shown in Table 1, where Android-NFC means the time interval between the moment when the system detects the NFC card and the moment when the system dispatches the operation to our application. It takes 0.13 ± 0.03 s. Then the keys are read from the encryption footer, which takes 0.06 ± 0.03 s. After the keys are obtained, the system conducts a two-round authentication with the NFC card. Each authentication takes about 0.07 s. Then the system reads the password from the NFC card, which takes 0.12 ± 0.04 s. Note that the password is encrypted and the decryption process takes 0.05 ± 0.02 s. The total time for reading the password from the NFC card is 0.50 ± 0.06 s. If the user needs to type the password when rebooting the system, it takes about 0.56 s to type each character of the password (Von Zezschwitz et al., 2014). If the length of the password is 16, it will take about 9 s to type the password. However, if the user uses the NFC card, it just takes about 0.5 s. A more intuitive comparison is shown in Fig. 6.

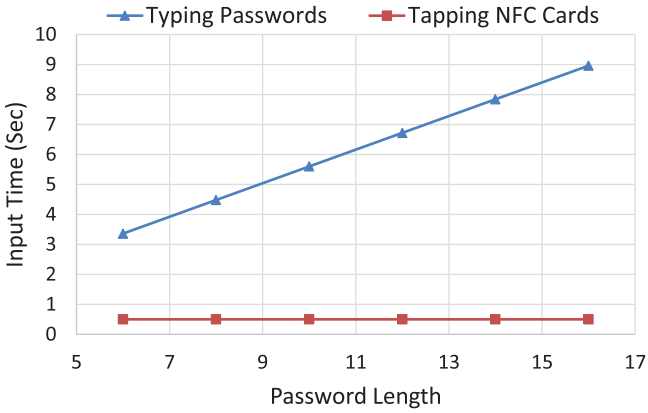

Fig. 6 – Time for entering passwords.

7. Related work

Deniable encryption is an emerging security paradigm in network communications (Canetti et al., 1997), disk storage, cloud storage (Gasti et al., 2010), etc. In disk storage, most of the existing work relies on either steganography or hidden volumes to achieve deniability.

7.1. *Steganography-based*

Anderson et al. (1998) propose the first file encryption scheme with PDE support. They present two solutions: Hiding blocks within cover files and hiding blocks within random data. However, both solutions are not suitable for performance-sensitive mobile devices due to high storage and I/O overheads. StegFS (McDonald and Kuhn, 2000) is a deniable-encryption version of the work of Anderson et al. (1998). It uses the second approach in Anderson et al. (1998) to work on Ext2 file system. However, the existence of the modified Ext2 driver and the external block table may expose the PDE system to the adversary. In addition, the disk usage rate is low due to the collision avoidance. Pang et al. (2003) propose a different design that blocks used by hidden files are marked as occupied in the bitmap, and it uses “abandoned blocks” and “dummy blocks” to achieve deniability. Unfortunately, their design is disk space inefficient.

7.2. *Hidden volumes-based*

FreeOTFE (2017) and TrueCrypt (2012) are two well-known PDE tools relying on hidden volumes. Compared to TrueCrypt, MobiPluto decouples file system from the underlying storage medium, achieving file-system-friendly feature. Mobiflage (Skillen and Mannan, 2013, 2014) builds the first PDE scheme for mobile devices. It is implemented in two versions: one for FAT32 file system in external storage (Skillen and Mannan, 2013), and the other for Ext4 file system in internal storage (Skillen and Mannan, 2014). The FAT32 version is not suitable for mobile devices without external storage; the Ext4 version needs to significantly modify Ext4 file system that introduces a large attack surface against PDE. MobiHydra (Yu et al., 2014) improves Mobiflage by introducing multi-level deniability and shelter volume to store sensitive data temporarily. In Table 2, we provide a comparison among PDE systems implemented for mobile devices.

Blass et al. (2014) present HIVE, a desktop PDE scheme that can defend against a multiple-snapshot adversary. HIVE relies on write-only oblivious RAM, which suffers from a high performance overhead.

Table 2 – Comparison among PDE systems implemented for mobile devices.

	MF-SD (Skillen and Mannan, 2013)	MF-MTP (Skillen and Mannan, 2014)	MobiHydra (Yu et al., 2014)	Our scheme
FAT32 support	Yes	No	Yes	Yes
Ext4 support	No	Yes	No	Yes
Other block file systems support	No	No	No	Yes
Fast switching	No	No	No	Yes
Free from memorizing pwd	No	No	No	Yes

7.3. *Others*

Ragnarsson et al. (2012) propose to use thin provisioning to provide deniability. However, their solution requires significant modifications of thin provisioning. In addition, they do not provide any proof-of-concept implementation. Peters et al. (2015) introduce DEFY, a deniable encrypted file system based on YAFFS (2016). DEFY is the first deniable file system specifically designed for flash-based, solid-state drives. It follows a log-structured design, motivated by the technical constraints of flash memory.

8. Conclusion

In this paper, we present MobiPluto, a user-friendly PDE solution for modern mobile devices. MobiPluto utilizes thin provisioning to build an additional layer that can transform the non-sequential allocation on the thin volumes to sequential allocation on the underlying storage medium. This renders it feasible to achieve file-system-friendly PDE using hidden volumes. In addition, we introduce a fast switching mechanism which can support switching to the hidden mode within 10 seconds. This is beneficial since it allows the user to capture the sensitive information in time from the hidden mode. We also incorporate NFC technology to improve MobiPluto's usability and deniability. We implement a prototype of MobiPluto on an LG Nexus 4 Android phone and our extensive evaluations show that MobiPluto only introduces a small performance overhead.

Acknowledgment

This work was partially supported by the National Research Foundation Singapore under NCR Award Number NRF2014NCR-NCR001-012.

REFERENCES

Anderson R, Needham R, Shamir A. The steganographic file system. In: Information hiding. Springer; 1998. p. 73–82.
Android. AOSP: Android open source project. 2017. Available from: http://source.android.com/.
Assange J, Weinmann RP, Dreyfus S. Rubberhose filesystem. 2013. Available from: http://web.archive.org/web/20130613060251/http://marutukku.org/.
Blass E-O, Mayberry T, Noubir G, Onarlioglu K. Toward robust hidden volumes using write-only oblivious RAM. In: Proceedings of the ACM SIGSAC CCS. 2014.
Canetti R, Dwork C, Naor M, Ostrovsky R. Deniable encryption. In: Advances in Cryptology-CRYPTO'97. 1997.
Chang B, Wang Z, Chen B, Zhang F. MobiPluto: file system friendly deniable storage for mobile devices. In: Proceedings of the 31st annual computer security applications conference. ACM; 2015.
Coker R. Bonnie++ file system benchmark suite. 2009. Available from: http://www.coker.com.au/bonnie++/.
Crystalinks. Pluto-King of the underworld. 2015. Available from: http://www.crystalinks.com/plutorome.html.
Curran K, Millar A, McGarvey C. Near field communication. Int J Electr Comput Eng 2012;2(3):371.
CyanogenMod. Consider LVM on Android. Available from: http://web.archive.org/web/20141018080847/http://forum.cyanogenmod.org/topic/4226-has-anyone-considered-lvm-on-android/.
Czeski A, Hilaire DJS, Koscher K, Gribble SD, Kohno T, Schneier B. 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.
FreeOTFE. FreeOTFE – free disk encryption software for PCs and PDAs. 2017. Available from: http://sourceforge.net/projects/freeotfe.mirror/.
Gasti P, Ateniese G, Blanton M. Deniable cloud storage: sharing files via public-key deniability. In: Proceedings of the 9th annual ACM workshop on privacy in the electronic society. 2010.
Kaliski B. PKCS 5: password-based cryptography specification, version 2.0, RFC 2898 (informational).
Kasper T, Oswald D, Paar C. Em side-channel attacks on commercial contactless smartcards using low-cost equipment. In: Information security applications. Springer; 2009. p. 79–93.
Kasper T, von Maurich I, Oswald D, Paar C. Cloning cryptographic rfid cards for 25$. In: 5th Benelux workshop on information and system security. Nijmegen, Netherlands. 2010.
Kernel. Thin provisioning documentation. 2017. Available from: https://www.kernel.org/doc/Documentation/device-mapper/thin-provisioning.txt.
Levine S. LVM administrator guide. 2016. Available from: https://access.redhat.com/documentation/en-US/Red_Hat_Enterprise_Linux/6/html/Logical_Volume_Manager_Administration/.
Martin L. XTS: a mode of AES for encrypting hard disks. IEEE Secur Priv 2010;3:68–9.
McDonald AD, Kuhn MG. StegFS: a steganographic file system for Linux. In: Information hiding. Springer; 2000.
Mifare. Mifare advance sdk. 2016. Available from: https://www.mifare.net/en/products/tools/mifare-sdk/mifare-sdk-advanced/.
NXP. MIFARE DESFire EV1 8K. 2016. Available from: http://www.nxp.com/products/identification-and-security/mifare-ics/mifare-desfire/mifare-desfire-ev1-contactless-multi-application-ic:MIFARE_DESFIRE_EV1_8K.

Pang H, Tan K-L, Zhou X. StegFS: a steganographic file system. In: Proceedings of the 19th international conference on data engineering. 2003.
Peters TM, Gondree MA, Peterson ZN. DEFY: a deniable, encrypted file system for log-structured storage. In: 22th NDSS. 2015.
Ragnarsson B, Toth G, Bagheri H, Minnaard W. Desirable features for plausibly deniable encryption. 2012. Available from: https://www.os3.nl/_media/2012-2013/courses/ssn/desirable_features_for_plausibly_deniable_encryption.pdf.
Samsung. Samsung eMMC memory. 2015. Available from: http://www.samsung.com/global/business/semiconductor/product/flash-emmc/overview.
Skillen A, Mannan M. On implementing deniable storage encryption for mobile devices. In: 20th NDSS. 2013.
Skillen A, Mannan M. Mobiflage: deniable storage encryption for mobile devices. IEEE Trans Dependable Secure Comput 2014;11(3):224–37.
Thornber J. Thin provisioning tools. 2015. Available from: https://github.com/jthornber/thin-provisioning-tools.
TrueCrypt. Free open source on-the-fly disk encryption software. Version 7.1a. 2012. Available from: http://www.truecrypt.org/.
Von Zezschwitz E, De Luca A, Hussmann H. Honey, I shrunk the keys: influences of mobile devices on password composition and authentication performance. In: Proceedings of NordiCHI '14. 2014.
Xda-Developers. "partitioning" your Nexus S using LVM. 2012. Available from: http://forum.xda-developers.com/nexus-s/general/howto-partitioning-nexus-s-using-lvm-t1656794.
YAFFS. Yaffs overview. 2016. Available from: http://www.yaffs.net/yaffs-overview.
Yu X, Chen B, Wang Z, Chang B, Zhu WT, Jing J. MobiHydra: pragmatic and multi-level plausibly deniable encryption storage for mobile devices. In: International conference on information security. 2014.

Bing Chang received his Ph.D. degree in information security from University of Chinese Academy of Sciences in 2017, and the B.E. degree in automation from Tsinghua University in 2012. He is currently a research fellow at School of Information Systems, Singapore Management University, Singapore. He was a research assistant at Singapore Management University from 2015 to 2016. His research interests include mobile devices security, biometric authentication and plausibly deniable encryption.

Yao Cheng received her Ph.D. degree from University of Chinese Academy of Sciences in 2015. She was a research fellow at School of Information Systems, Singapore Management University till 2017. She is currently a scientist at Institute for Infocomm Research, A*STAR, Singapore. Her research interests are in the information security area, focusing on vulnerability analysis, privacy leakage and protection, malicious application detection, and usable security solutions.

Bo Chen received his Ph.D. degree from the Department of Computer Science, New Jersey Institute of Technology in 2014. He was a postdoc in Stony Brook University (2014–2015) and Pennsylvania State University (2015–2016). He is currently an Assistant Professor in the Department of Computer Science, Michigan Technological University. He is interested in all aspects of information security and cyber security, with an emphasis on data security, cloud computing/storage security, mobile devices security, and network security.

Fengwei Zhang is an Assistant Professor at Wayne State University. He received his Ph.D. degree from George Mason University in 2015. His research interests are in the areas of systems security, with a focus on trusted execution environments, transparent malware analysis, hardware-supported security, transportation security, and plausibly deniable encryption.

Wen-Tao Zhu received his B.E. and Ph.D. degrees in 1999 and 2004, respectively, both from Department of Electronic Engineering and Information Science at University of Science and Technology of China. Since July 2004, he has been a faculty member with State Key Laboratory of Information Security, which is now part of Institute of Information Engineering, Chinese Academy of Sciences, where he has been a full professor since Oct. 2011. Dr. Zhu is a senior member of the IEEE. His research interests include network and system security, applied cryptography, and privacy protection. Since Aug. 2011, he has been on the editorial board of Journal of Network and Computer Applications published by Elsevier.

Yingjiu Li is currently an Associate Professor at the School of Information Systems, Singapore Management University. He has published over 130 technical papers in international conferences and journals, and served on the program committees for over 80 international conferences and workshops. His research interests include RFID security and privacy, mobile and system security, applied cryptography and cloud security, and data application security and privacy. He is a Senior Member of ACM and a member of the IEEE Computer Society.

Zhan Wang received the Ph.D. degree from University of Chinese Academy of Sciences in 2013. She visited George Mason University from 2011 to 2013. Since July 2013, she has been a faculty member at State Key Laboratory of Information Security, which is now part of Institute of Information Engineering, Chinese Academy of Sciences. Her research interests include cloud and mobile security, authentication and deniable encryption. She founded security startup Realtime Invent in 2015.