Received March 5, 2017, accepted April 15, 2017, date of publication April 18, 2017, date of current version May 17, 2017.

Digital Object Identifier 10.1109/ACCESS.2017.2695561

Personal Privacy Protection Framework Based on Hidden Technology for Smartphones

SHUANGXI HONG, CHUANCHANG LIU, BINGFEI REN, YUZE HUANG, AND JUNLIANG CHEN
State Key Laboratory of Networking and Switching Technology, Beijing University of Posts and Telecommunications, Beijing 100876, China

Corresponding author: Shuangxi Hong (hongshuangxi_2008@126.com)

This work was supported in part by the Natural Science Foundation of China under Grant U1536112, in part by the National Key Technology Research and Development Program of China under Grant 2012BAH94F02, in part by the National High-tech Research and Development Program of China (863 Program) under Grant 2013AA102301, and in part by the Project of New Generation Broadband Wireless Network under Grant 2014ZX03006003.

ABSTRACT Smartphones are increasingly used for storing a large amount of sensitive used data. Users can protect their sensitive data through encryption and locking screen. A pattern lock is one of the ways to lock a screen. This method is used by most users because of its ease of use and memorization. However, a pattern lock with low security level is inadequate to protect the sensitive data of the user when it encounters a brute force or other physical attack (e.g., smudge attack). Furthermore, it bypasses all of the protection measures of mobile device when users are coerced into disclosing their passwords. Steganographic techniques and deniable encryption are designed to protect the sensitive data of the user as well as secure communications and can hide sensitive data on a disk or during communication with other devices. To overcome these deficiencies that mobile devices present, we present a novel, practical safe framework called MobiMimosa that is based on plausible deniable encryption. MobiMimosa enables multiple hidden encryption volumes and dynamic mounting of hidden volumes, which facilitates the transfer of sensitive data from a normal volume to a hidden volume. Simultaneously, to meet the personalized needs of the security of the mobile device, MobiMimosa enables a strategy to be set that can trigger the uninstalling of a sensitive app and the destruction of sensitive data. MobiMimosa also greatly alleviates corruption of the cross-volume boundary that is present in previous smartphone PDE schemes. We implemented a prototype system on the android device.

INDEX TERMS System service, deniable encryption, mobile device, privacy protection.

I. INTRODUCTION

In recent years, the number of smartphones has presented a trend of fast growth. Approximately one-fifth of the worldwide population owned a smartphone in 2013, and this proportion is expected to increase to 34 percent by 2017. Considering all of the smartphone operating systems, the android OS accounted for approximately 85 percent of the market share worldwide in 2015 [1]. These figures show that the android system is ubiquitous on smartphones.

With the increasing computational capability and functions of smartphones, users can perform tasks while on a bus or while traveling, such as chatting through Social Networking Software or online shopping. Smartphones have become essential tools for both amusement and communication, and thus, they store a large amount of sensitive information (e.g., photo and chatting records) of the end user. Similarly, an increasing number of companies provide mobile versions of their PC applications to enable working anytime, anywhere. Research has shown that allowing access to enterprise services with smartphones improves working efficiency for employees [2]. This arrangement leads to having smartphones store not only personal sensitive data but also enterprise confidential information. In this way, the security of smartphones has become a public focus. For this reason, all of the major mobile operating systems now offer some measures of protecting sensitive data, such as full disk encryption and file-based encryption.

The android system occupies the largest share of the market and provides some protective measures, such as FDE (full disk encryption) [3], pattern-locked, and PIN. Among these options, pattern-locked has become the most frequent method of protection [4] due to its convenience and ease of memorization. As far as we know, the latest android version enables pattern lock full disk encryption (i.e., encrypting a master key by wrapping a key generated by a pattern password). However, a pattern-locked phone has a lower resistance because it has a smaller key space and lower password entropy [5]. Therefore, it is easy for the password of a pattern-locked phone to be acquired through brute force or physical attacks. Specifically, in certain situations, all of the protection measures of an android system could be disabled. For example, a robber could capture a smartphone and enter its system when the user of the smartphone is operating his/her device while walking (device in the unlocked state). Another situation is when the user is required to unlock his/her mobile device when travelling across a border checkpoint. Both situations can result in the leakage of sensitive information that belongs to the owner of the smartphone.

At present, research on security for android devices mainly focuses on how to grant fine-granularity permissions to applications, detect malicious applications and prevent sensitive data leakage from running programs on the smartphone. However, there is less focus on preventing the leakage of sensitive data from external physical attacks. Smartphones still have weak resistance to external attacks in a certain environment, while full disk encryption can prevent the leakage of sensitive data after a smartphone is lost or stolen. For example, sensitive data could be fully exposed to an adversary when the owner of a smartphone is coerced. In these protection measures, pattern-locking as the main protection of a smartphone is more vulnerable to attacks. Aviv *et al.* [6] present a smudge attack measure on smartphone touch screens that can easily acquire a pattern password by extracting the track of the remaining smudge. FDE will lose its protection ability after the pattern password of a smartphone is cracked, in such a way that all of the sensitive data are presented to the adversary. Therefore, the smartphone should provide a set of functions that implements hiding and encryption of data to prevent the adversary from finding any sensitive information that was hidden by the user, even if the adversary gains full control of the smartphone.

Plausible Deniable Encryption (PDE) [7] can be adopted to protect sensitive information against powerful adversaries who can coerce users or utilize other black technology to reveal their sensitive data. This practice should not be confused with encryption technology because it hides data on the basis of encrypted data. Several existing solutions support full disk encryption with plausible deniability in the released desktop operating system. TrueCrypt [8] is a famous system among the PDE tools. To the best of our knowledge, Mobiflage [9] is the first PDE system for mobile devices with android system, and it is implemented based on a physical or emulated SDcard with a FAT32 file system. Recently, Mobiflage was extended to support the Ext4 file system on the built-in emulated SDcard by modifying the driver of Ext4 [10] (achieving the inode sequential allocation). MobiHydra [11] has expanded Mobiflage by supporting multiple levels of deniability, which enables data transfer between two modes across a specific cache partition and mitigates the booting time attack of Mobiflage. MobiPluto [12] is another PDE original system that implements linear allocation space using a LVM (logic volume management) tool to avoid modifying the Ext4 driver, and it also enables multiple levels of deniability (multiple hidden volumes). However, these original systems have some common defects because they share the same pedigree, such as the boot time being longer than the normal android boot time (i.e., booting time attack), corruption of the across boundary of the volume, and so on. These PDE systems are not suitable for common users due to their inconvenience of use and the aspects mentioned above. Consequently, in our work, we propose MobiMimosa, which is a more practical analogous PDE solution. To achieve deniability, MobiMimosa uses data partitions in its outer volume (or called public volume) to store the regular data of users. The hidden volume locates the second half of the public volume and is used to store the sensitive data of users. When enabling multiple hidden volumes, a hidden volume could locate another partition (e.g., a cache partition). Users are allowed to use any encryption algorithm that the android kernel enables to encrypt the hidden volume. We also improve mode of data transmission between two volumes. This is necessary due to android security framework. In the android system, data of an application can only be stored into its installing directory or emulated Sdcard.

Our contributions mainly include the following:

- We add the DataSafe system service to the android system to provide uniform management for the hidden encryption volume. This service ensures that our prototype system has outstanding expandability.
- Similar to TrueCrypt, we dynamically mount a hidden encrypted volume to overcome the defect of requiring a rebooting system to switch modes when transferring data between normal volume and hidden encrypted volume of previous prototype systems.
- We adopt a novel data storage method to alleviate data corruption across the volume boundary and improve the utilization of the storage space in mobile devices.
- Our prototype system avoids booting time attacks, which previous prototype systems allow, and simplifies the complexity of operating the hidden encryption volume for the user. We also address several challenges and compensate for some inadequacies of the countermeasures that are adopted by previous PDE systems.

The remainder of this paper is organized as follows: Section II presents the related work. In section III, we describe design of MobiMimosa. We detail the system implementation in section IV. In section V, we evaluate the system performance and discuss the experimental results. We analyze the leakage of sensitive data and with respect to countermeasures in section VI, and we present the conclusions in section VII.

II. RELATED WORK

The android system is divided into four layers, the application layer, application framework layer [13], libraries and android runtime layer and Linux kernel. Among the four layers, the application framework layer provides the fundamental system function and the uniform API interfaces for the application layer. For example, the PackageManager service provides installing/uninstalling of an application. All types of services of the application framework layer are registered with the process of the system server while booting the system in such a way that an application can call the system service by querying the system server process. Therefore, we can provide interfaces for operating the hidden encryption volume for applications by adding new system services to the application framework layer.

Deniable encryption was first applied in network communications [14] and was applied to disk storage and cloud storage [15] later. In disk storage, there are two main types of PDE systems: hidden volume and Steganographic file systems.

A. STEGANOGRAPHY-BASED

Anderson *et al.* [16] first propose a PDE encryption scheme-based file that includes two solutions: hiding data blocks within cover files and hiding data blocks within random data. McDonald and Kuhn [17] proposed a new steganographic file system by modifying the Ext2 file system driver based on the second approach in [16]. Pang *et al.* [18] proposed a different design in which blocks occupied by hidden files are marked by fulfilling the corresponding block bitmap table, and they used "abandoned blocks" and "dummy blocks" to acquire plausible deniability. Unfortunately, these designs have some mutual defects, such as a high I/O overhead and low utilization of storage space, and these defects cannot be tolerated in a mobile environment.

B. HIDDEN VOLUMES-BASED

TrueCrypt and FreeOTFE [19] are two well-known PDE tools that are based on a hidden volume. TrueCrypt is an open source project and enables only a single hidden volume, which can be a file (a virtual disk) or a disk partition. It also enables plausible deniability: it allows a user to create a hidden volume inside a normal TrueCrypt volume's free space. VeraCrypt is a fork of the discontinued TrueCrypt project. Many security improvements have been implemented and issues raised by TrueCrypt code audits have been fixed [20]. By contrast, FreeOTFE enables multiple hidden volumes within the free space of a FreeOTFE volume. However, the users must specify the offset of the hidden volumes in which the hidden data are to be written. Users can mount a hidden volume by specifying the corresponding offset of the hidden volume. Elettra [21] is an early PDE tool that allows users to add many files with different passwords into an archive. The user can input a password to decrypt the corresponding file. These tools work on the mainstream desktop operating system. In addition, Mobiflage [10] and MobiHydra [23] are two PDE original systems that are based on the android FDE. Mobiflage is the first PDE scheme for a mobile device, and it creates a hidden volume at the offset that is computed in accordance with the input password of the user to achieve plausible deniability. Mobiflage usually works in normal mode (i.e., the mode of daily use) and mounts the normal volume (i.e., the outer volume of Mobiflage). By contrast, the hidden encryption volume is mounted while in the PDE mode. Compared to Mobiflage, MobiHydra enables multiple hidden volumes and transfers sensitive data from the normal mode to the PDE mode via a cache area that is located behind multiple hidden volumes. The author of MobiHydra finds a booting time attack in which Mobiflage presents and alleviates this attack. Booting time attack: there are a longer booting times for mobile devices that use Mobiflage than normal mobile devices that use FDE when verifying the password while booting the mobile device. This relationship can compromise the plausibly deniability that is offered by Mobiflage. MobiPluto [12] is a more practical PDE scheme, and it utilizes LVM (logical volume management) to create the outer volume and hidden volume. The LVM can implement a linear allocation of the storage space to avoid modification of the file system driver. However, these prototype systems on mobile devices have the same origin and present some mutual defects. 1) Transferring sensitive data is inconvenient from a normal volume to a hidden volume. However, TrueCrypt is the reverse and can mount the outer volume and hidden encryption volume simultaneously. 2) Mounting the hidden volume requires rebooting the android system. 3) These systems present cross-border corruption and low utilization of the storage space.

The device-mapper is a module that is provided by the Linux kernel for mapping a physical block device to higher logical level virtual devices, starting with Linux kernel version 2.6. This module provides linear, snapshot and mirrors mapping modes. The dm-crypt [22] is a subsystem of a device-mapper, and it implements transparent disk encryption. The android system introduces full disk encryption based on the dm-crypt subsystem of the Linux kernel to protect the data of the user. However, our prototype system adopts the linear mode of the device-mapper and dm-crypt to implement plausible deniability and transparent encryption.

III. MOBImIMOSA DESIGN

In this section, we describe our design and explain our design choices. Our MobiMimosa provides a more practical PDE scheme that is different from previous PDE schemes on mobile devices (e.g., Mobiflage). Previous original systems [10]–[12] cannot simultaneously use outer a volume and hidden volume, and they can only mount one of the two choices: the hidden volume or public volume. However, similar to the TrueCrypt tool, our prototype system is able to mount the hidden volume and outer volume simultaneously, which is convenient for transferring user data between the outer volume and hidden volume. MobiMimosa is named after Mimosa due to the features of our system. When the mobile device of a user suffers a brute force attack, an operating system that enables MobiMimosa will automatically protect the sensitive app and data stored in the hidden volume (i.e., uninstalling the app and destroying the sensitive data to prevent leakage to the adversary). This feature is similar to the automatic protection function of Mimosa when touching its leaf by hand.

A. OVERVIEW

We implement MobiMimosa by hiding the encrypted volume in free space (sequential or non-sequential) within the internal flash storage and manage the hidden encrypted volume via the system service of the application framework layer. For practicality, we enable three volumes in our prototype system: An outer volume is enabled for storing regular data, in which we use the default /data partition as the outer volume. Two hidden volumes are created to store sensitive data that are denied by the user, and they are separately located in the second portion of the userdata partition and cache partition. If multiple hidden volumes are required, then the number of hidden volumes can be varied by the DataSafe system service before initializing the hidden volume. We use the terms ''hidden'' and ''deniable'' when referring to the hidden volume.

In MobiMimosa, the outer volume is protected by a decoy password (the screen-locked password that the user usually uses). The decoy key to encrypt the outer volume is wrapped by the decoy password and stored in the footer structure, which is placed in the end of the outer volume. Two hidden volumes are protected by two different hidden password, and we encrypt the hidden volumes according to hidden keys that are stored at the beginning of the hidden volumes. The DataSafe system service is in charge of managing the hidden volumes, which includes initialization, creation or deletion of the hidden volume and uninstalling sensitive app. Upon booting, the android system mounts the outer volume, and the user can store regular data in the outer volume. The hidden volume can be mounted by the settings app (the inherent app of the android system), which calls the DataSafe system service when the user needs to store sensitive data. The user can mount a hidden volume by inputting the appropriate password. The user can write data into the hidden volume directly or transfer data from the public volume to the hidden volume after the hidden volume is mounted. To protect sensitive data stored in the hidden volume, the DataSafe system service will unmount the hidden volume automatically after the device is idle for a period of time. Furthermore, DataSafe will destroy the sensitive data that is stored in the hidden volume when a number of failed passwords attempts reaches the threshold set by the user.

B. FILE SYSTEM CONSIDERATIONS

Currently, the android system enables the Ext4 file system by default. The Ext volume is divided into block groups and contains meta-data (super block, block/inode bitmaps and inode table) and the data block. To create a hidden volume in the free space of the Ext volume, we must avoid overwriting the occupied blocks of the meta-data. Otherwise, the adversary can distinguish the normal Ext volume from the hidden volume. Furthermore, when creating the inode of the file/directory in the root of an Ext4 file system [23], the inode is placed into a more vacant block group by the Orlov algorithm [24], which can be spread over the entire partition. While the hidden encryption volume exists, the data written to the outer volume will likely corrupt the sensitive data. Therefore, a subtle modification of the Ext4 file system was necessary for MobiMimosa and refers back to Mobiflage.

C. DATASAFE SYSTEM SERVICE

DataSafe is a system service that is combined with the android source code, and it is registered to the system server daemon during booting. The settings system application and Initmimosa app can interact with DataSafe by calling the API: context.getSystemService (DATASAFE_SERVICE). Therefore, the user can operate the hidden volume and set the trigger strategy by using Initmimosa application. The application should be stored on a personal PC. When the user needs to create a hidden volume, the application can be installed by the adb install command or other installing way. To ensure deniability, the application must be uninstalled after initializing the hidden volume. Mounting/unmounting of the hidden volume is performed by the settings application (see part III-D). The service framework is as follows (see Figure 1): The four modules of DataSafe consist of operating the hidden encryption volume, the trigger strategy, destroying data and uninstalling the application. The meta-data of DataSafe Manager are stored in a specific directory, which mainly includes the mapping table of the hidden volume and the list of names of the applications that the user wants to uninstall when the device is in danger.

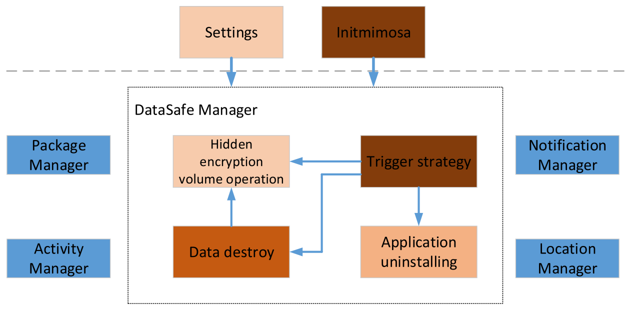

FIGURE 1. DataSafe system service framework.

1) HIDDEN ENCRYPTION VOLUME OPERATION MODULE

This module includes the creation, deletion, mounting and unmounting of the hidden volume. A user can create a hidden volume, specify its size and format and mount the hidden volume. During creation of the hidden volume, the user must set its password, which is used to generate the key for encrypting the hidden volume, and the user must specify the number (one or two) and size (less than half of /Cache or /Userdata) of the hidden encryption volume. Formatting and mounting are performed automatically through the DataSafe system service. However, unmounting and deletion of the hidden volume are triggered by the trigger strategy.

TABLE 1. Trigger strategy list by user.

trigger condition	action	Description	protection level
errnum>=5	1) destroy sensitive data and meta-data. 2) uninstall sensitive apps	When the number of failed attempts to unlock the screen (errnum) reaches at least five over an interval of time, DataSafe will destroy the meta-data and sensitive data stored in the hidden volume	1
message="data_ destroy@$%"	1) destroy sensitive data and meta-data. 2) uninstall sensitive apps	When the mobile device receives a message with the content "data_destroy@$%", DataSafe will destroy the meta-data and sensitive data stored in the hidden volumes	1
errnum>=3&& errnum<5	1) unmount and delete hidden volumes 2) uninstall sensitive apps	when meeting the trigger condition, DataSafe will unmount and delete the hidden volumes and uninstall specific apps	2
wait_time>10 (min)	1)Unmount and delete hidden volumes	If the time (wait_time) of the black screen of the mobile device is more than ten minutes, DataSafe unmounts the hidden volume	3

Note that the deletion operation only deletes the device file (e.g., /dev/block/dm-1) that was created by DM-crypt. These actions eliminate all traces of the hidden encryption volumes and ensure deniability. Furthermore, the data stored in the hidden encryption volume is still intact.

2) TRIGGER STRATEGY MODULE

The module builds a trigger strategy. When the trigger condition is satisfied, corresponding actions will be performed. The module listens to a broadcast to capture the device state. The strategy is as shown below (see table 1). To better prevent leakage of sensitive data, the secure defense is divided into three levels. The leakage probability of the sensitive data is in turn increased from level one to level three. The first level denotes that the mobile device is suffering the most serious threat, such as an adversary capturing the mobile device and attempting a brute force attack on the screen-lock. When the number of inputs of a wrong password reaches at least five, the operating system will destroy all of the sensitive data and meta-data and uninstall specific app (errnum>3). In addition, when the mobile device is lost, the user can send a message (message="data_destroy@$%") to the missing mobile device using another smartphone to destroy all of the sensitive data. Furthermore, the operating system will uninstall all of the app that the user specifies. In the second level, when the number of failed attempts to unlock the screen reaches at least three, the operating system will unmount and then delete the hidden volume (i.e., the device file that corresponds to the hidden volume is deleted) and will uninstall sensitive apps. For example, when the adversary coerces the user to hand over his/her password, the user could deliberately provide an incorrect password to meet the trigger condition. In the third level, when the standby time of the mobile device is more than ten minutes, we assume that the user has not used the mobile device for a long period of time. Thus, the operating system will unmount and delete the hidden volume automatically to prevent the leakage of sensitive data. Sensitive data are protected by default in the third level after the user reboots the android system. The trigger strategy not only prevents leakage of sensitive data but also ensures deniability of our prototype system.

3) DATA DESTROY MODULE

This module is in charge of destroying the meta-data and sensitive data. When the module is called, a random number is entered in the hidden partition to overwrite the data stored in the hidden volume [19]. Furthermore, the meta-data (i.e., the mapping table and key) of the hidden volume is also destroyed. The deletion of flash storage is discussed in detail in section 6. This strategy ensures that the adversary cannot capture the data stored in the hidden volume by analyzing the physical disk.

4) APPLICATION UNINSTALLING MODULE

This module calls the system service of the package manager to implement the uninstalling function. The user can pass the names of application packages to the system service by Initmimosa application. When the trigger condition is met, the module uninstalls these apps. This ensures that account information is not revealed from sensitive apps (e.g., banking, social software) that are used often, because apps(especially being social software) usually save account and password of user.

In summary, DataSafe not only provides a uniform interface for the upper application to operate on a hidden volume but also enables uninstalling of apps that the user specifies. DataSafe also listens to all types of broadcasts from the system and user-defined aspects to capture the state of the device. When these broadcasts meet a certain trigger condition, a corresponding action will be performed. Therefore, this approach not only improves the scalability of our prototype system but also reduces the possibility of leakage to a great extent.

D. STORAGE LAYOUT

In the android system, the storage layout usually includes six main partitions (i.e., boot, recovery, system, data, cache, and misc). Among these partitions, a hidden volume can

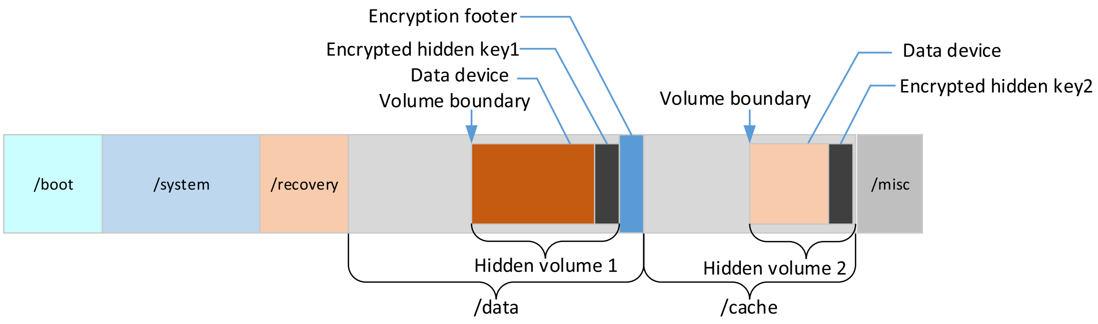

FIGURE 2. Storage layout and hidden volumes

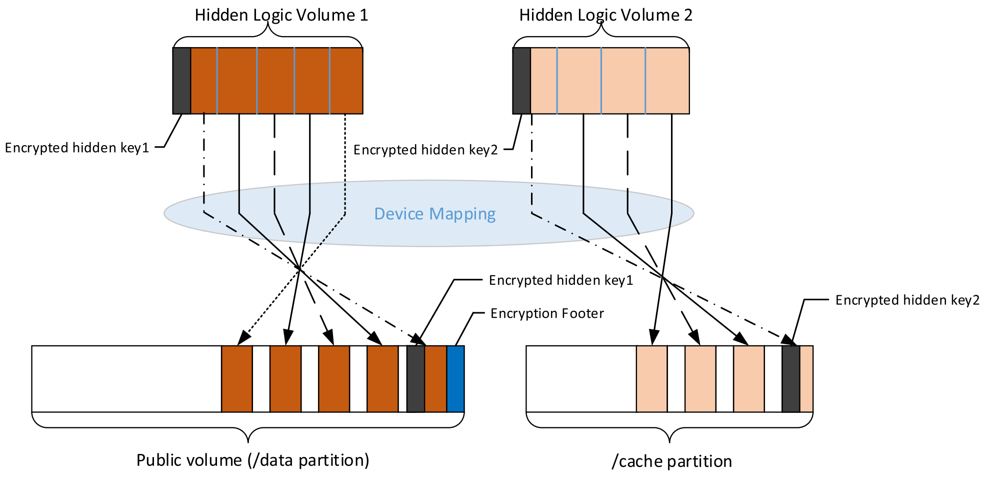

FIGURE 3. Hidden volume mapping framework.

be created over a data/cache partition. Currently, the data partition (i.e., the outer volume) is encrypted and formatted for regular use by default in the latest version of android. The cache partition is used to cache the application or update the system via OTA updates. The hidden volume can be created at an offset within the free space of the outer volume and can be encrypted with different keys (see Figure 2). There are no distinguishing characteristics between the free blocks and hidden encrypted volumes because the data/cache partition is filled with random numbers before formatting. While the hidden volume is not mounted, the adversary cannot locate it in the outer volume. The outer volume shows all free blocks (including those occupied by hidden volumes) when mounting. In the TrueCrypt volume, it is possible that the data that are written to the mounted outer volume can corrupt the master key and the data stored in the hidden volume can breach the boundary of the volume.

This defect is also present in previous prototype systems [15], [26], [28]. For this reason, in our prototype, the hidden volume is divided into multiple groups that are larger (e.g., 50 M per block group); these groups are mapped to corresponding physical block groups that have the same size. The first block group with a hidden logical volume is mapped to the corresponding physical block group that lies at the end of the physical device (data/cache partition). The subsequent logic block group is in turn mapped to the corresponding physical block group, which is close to the beginning of the physical device (see Figure 3). While writing the data to the disk, the direction of the growth of data stored in both the public volume and hidden volume is in the direction of the boundary of the hidden volume (see Figure 2). Therefore, when the hidden volume has free space, sensitive data cannot be overwritten even if the regular data stored in the outer volume occasionally crosses the boundary of the volume. Thus, the method effectively reduces the probability of corrupting the cross-border volume. To avoid the stored data and meta-data in the hidden volume overwriting blocks that are occupied by the meta-data of the outer volume (i.e., the public volume), the block groups of the hidden volume are mapped to blocks that store data from the public volume. Thus, the physical blocks that correspond to the hidden volume are non-sequential.

E. CALCULATION OF THE VOLUME BOUNDARY

The hidden volume starts at a specific offset in the flash storage. MobiMimosa generates the offset, using the following method:

$$\text{offset} = vlen - hlen - flen - (ram/(0.001 * hlen))$$

where vlen denotes the sectors number of /data partition, 512 bytes per sector; hlen is the number of sectors in the hidden volume and is less than 50% of vlen; flen is the number of sectors of the footer structure, which is usually 16 KB in the android system; and ram is the random number generated by reading /dev/urandom. The intention of using random blocks is that there is enough extra space to avoid overwriting the metadata block of the outer volume. The method is the same as above while creating hidden volume 2.

F. USER STEPS

Here, we describe how users interact with MobiMimosa, including initialization and usage.

Users do not manually enable encryption in android version 5.0 because the android system automatically enables encryption when a new smartphone boots the system for the first time. Users must set the login password immediately for security after entering the system. The initialization process is as follows:

Step 1: user installs the Initmimosa app (see Figure 1) that is used to set the parameters (e.g., the space size, password) and then begin initializing /data partition by Initmimosa.

Step 2: user installs Initmimosa again and set size and password of hidden volume.

Step 3: user chooses apps installed which were uninstalled in time when meeting trigger strategy.

Step 4: user chooses "finish" button after steps above are completed.

Step 5: users need to uninstall the Initmimosa application to prevent compromising of the deniability of the hidden volume after completing initialization and reboot.

Specific initialization process as shown figure 4. Pure color frames need operations by user. However, other color frames are operated by android system.

When the hidden volume must be mounted, users can run settings application, choose the "security" option and then chose the "lock-screen" option. The users inputs the password of the hidden volume when a password-box interface appears. A broadcast that carries the encrypted password is sent to DataSafe at this time. DataSafe attempts to mount the hidden volume using the received password. If the password is correct, the hidden volume is mounted successfully. Otherwise, mounting fails.

IV. MOBIMIMOSA IMPLEMENTATION

We implement a prototype system, MobiMimosa, on the latest Nexus 6P device with android 6.0. To reduce the difficulty of the experiment, we adopt the source code of CM13 provided by CyanogenMod (the largest android third-party compiler team). We add the application framework to the source code of the android system and customize the Ext4 driver. In addition, to create the hidden volume, we need to static cross-compile the DmSetup [25] and CryptSetup [26] tools and then push them to the android device by using the adb-push command.

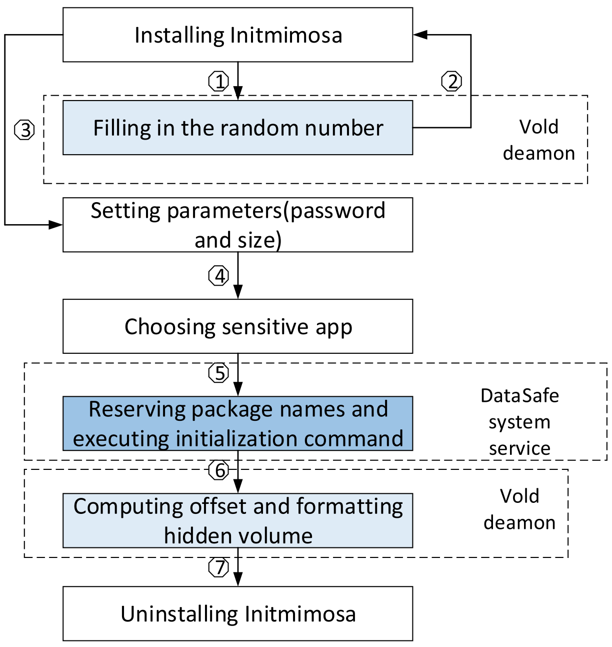
Installing Initmimosa
①
②
③
Filling in the random number
Vold deamon
Setting parameters(password and size)
④
Choosing sensitive app
⑤
Reserving package names and executing initialization command
DataSafe system service
⑥
Computing offset and formatting hidden volume
Vold deamon
⑦
Uninstalling Initmimosa

FIGURE 4. User initialization process.

A. ADDING THE SYSTEM SERVICE

In this section, we explain the implementation of the DataSafe system service. Usually, each type of hardware that is supported by android has a corresponding system service and HAL (hardware abstract layer) definition. There is a lightweight service and a lightweight HAL definition, for example, a Wifi Service and a Wifi HAL definition. However, our system service does not need to match HAL definitions because are not truly adding hardware to the android device. We implement our system service according to the methods provided by Yaghmour [27]. In android version 6.0, the android system enhances the security of the application framework. These methods are already very different.

In the operation module of the hidden encrypted volume, first, a linear mapping table is created, which builds a map from a logic block to a physical block [25]. The Dmsetup tool passes the mapping table to the driver of the kernel and then creates a corresponding logical block device. Second, a hidden encrypted volume is created based on the logical block device that was just created by the Dmsetup tool using the CryptSetup tool. The hidden encrypted volume can be formatted by the make_ext4fs tool, which is included in the android OS. By contrast, the hidden encrypted volumes are also deleted by the same tools (i.e., the DmSetup and CryptSetup tools run the detection command) in such a way that all of the clues are eliminated, which compromises the deniability of the hidden volume. Thus, the user can deny the existence of the hidden volume.

In the trigger strategy module, DataSafe listens to four types of broadcasts, SMS_RECEIVED, USER_PRESENT, SCREEN_OFF [28] and NUM_ERROR. Among these broadcasts, the NUM_ERROR broadcast is user-defined to inform the receiver of a broadcast regarding the entry

TABLE 2. Observed throughput and file operations of default Ext4 and Ext4 with Linear Inode Allocation.

Prototype system	Multiple hidden volume	Dynamic mounting	Overlay probability	Utilization of space	scalability	interactivity
Mobiflage	N	N	H	L	L	L
MobiHydra	Y	N	H	L	L	L
Mobipluto	Y	N	H	L	L	L
MobiMimosa	Y	Y	L	H	H	H

of an incorrect screen-lock password. When DataSafe continuously receives three broadcasts of NUM_ERROR (errnum=3), the corresponding strategy is conducted. Here, wait_time denotes the time interval between receiving the SCREEN_OFF broadcast receiving the USER_PRESENT broadcast. When wait_time is more than ten minutes, the module performs the corresponding action. When DataSafe receives a SMS_RECEIVED broadcast, the message that the mobile device receives will be compared to a preset string character, and the corresponding strategy will be triggered if the match is successful.

The data destroying module is in charge of thoroughly eliminating sensitive data and meta-data (e.g., the encrypted mapping table). We fill all of the free space of the flash storage with a random number generated by the encryption function to implement secure deletion. However, the deletion operation for meta-data is simple because we only discard the key of the encrypted meta-data. The module of the uninstalling application is in charge of uninstalling applications that are specified by the user.

B. FILE SYSTEM AND USER INTERFACE

1) SEQUENTIAL INODE ALLOCATOR

Because the inode-spread feature of the Ext file systems likely causes block collisions between the outer volume and hidden volume, MobiMimosa does not require this feature. Our Ext4 driver is modified to implement the linear inode allocation in such a way that the first free inode is always allocated when creating a file. Because the block allocator always attempts to place data blocks within the block group and includes the inode of its file, this arrangement is sufficient to ensure that the disk is filled linearly. In addition, because the cache partition is also the Ext4 file system, inode allocation is also allocated linearly. Thereby, it is possible to place hidden volume 2 in the second half of the /cache partition.

2) FDE CHANGE

The android encryption layer is implemented in the logical volume manager (LVM) of the device-mapper crypto target: dm-crypt [22]. Encryption takes place below the file system and is hence transparent to the operating system and application. The AES cipher is used in the CBC mode with a 128-bit key. ESSIV is used to generate unpredictable IVs to protect water-marking attacks [29]. In our prototype system, we implement transparent encryption using dm-crypt and use the XTS-AES cipher instead of CBC-AES. XTS-AES is chosen as a precaution against known attacks (e.g., copy-and –paste attacks) [30].

C. USER INTERFACE

In previous prototype systems, the initialization was very inconvenient to user, and thus, the user usually used the vdc command line tool to interact with the system. However, the user can interact with MobiMimosa through Initmimosa and settings application. The programmer can develop an application with an analogous function by calling the DataSafe system service. This approach enhances the scalability of our prototype and simplifies the complexity of operating the hidden volume to the user.

V. PERFORMANCE EVALUATION

In this section, we test the linear Ext4 file system and compare the throughput with the default android Ext4 file system. In the linear Ext4 file system, we describe the experimental results under the two encryption modes of MobiMimosa and full disk encryption, and then, we explicate the performance impact on the mobile device. To determine the performance impact of the modified Ext4 file system, we use the popular benchmark tool bonnie++ [31] to test the I/O performance and compare the default Ext4 against the linear Ext4 on a Google Nexus 6P running android-6.01. Because FDE is enabled by default, we do not test the I/O performance under the unencrypted state. We run 10 trials on a 900 M file and compute the average size according to the test results, as shown in table 2. We observed that the throughput was slightly affected and that the sequential read/write operations were almost equal, but was is a small amount of performance degradation under random operations. Under the linear Ext4 file system, all of the Ext4 partitions of the android system allocate inode through the linear inode allocator. Thus, we can create the hidden encryption volume in a partition formatted in the Ext4 file system. Two hidden volumes are created by the same mapping method. We tested the performance impact of the hidden volume in the /data partition. We used the bonnie++ and dd tools to test the I/O performance of the default android system and MobiMimosa.

We ran 10 trails on three files that were between 40 M and 200 M. We evaluated the performance of the linear Ext4 file system.

We observed that the throughput of MobiMimosa appears to not decrease compared with the default android system, but there is some improvement in the sequential read (See Figure 5). The read operation improvement is due to the encryption mode of the cipher and secondary device mapping. Some clues can be observed in Figure 5, using the two test tools under the linear Ext4 file system leads to the

TABLE 3. Feature comparison between previous prototypes and MobiMimosa (N: Not, Y: Yes, H: High, L: Low) (Seq: Sequential; Rnd: Random).

	Throughput(KB/s)		File system operation/s				
	Seq read	Seq write	Rnd seek	Seq create	Seq delete	Rnd create	Rnd delete
Ext4 default	1692243	134676	15239	2855	23007	2907	6903
Ext4 linear	1705643	134926	14992	2857	22627	2792	6645

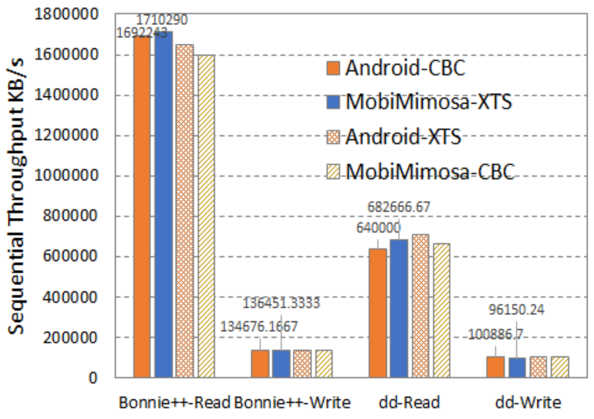

FIGURE 5. Observed Sequential Throughput per second with the Bonnie++ and dd tools.

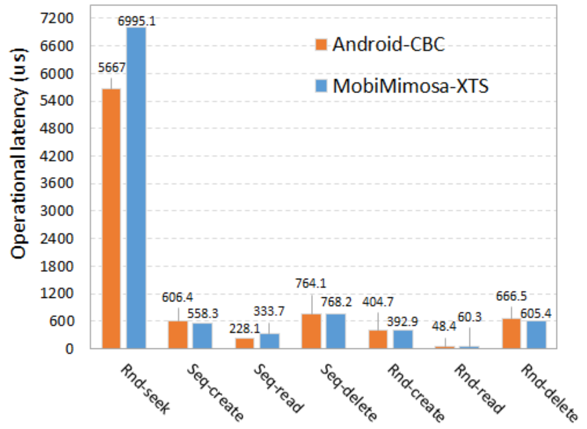

FIGURE 6. Observed Operational Latency with Bonnie++ (Rnd: Random; Seq: Sequential).

same result (i.e., MobiMimosa-XTS is greater than or almost equal to android-CBC in terms of throughput).

In addition, we also test the operational latency and operational number of the file system on the android platform. The test results are shown in Figures 6 and 7. We observed that MobiMimosa slightly reduces the system performance due to operational latency and the operational number of the file system. Most notably, the operational latency of the random seeking of MobiMimosa is longer than that of the default android (i.e., Android-CBC mode). This finding should be attributed to secondary device mapping. Specifically, the I/O request is forwarded many times in the kernel when the system reads/writes the hidden encryption volume. On the whole, MobiMimosa only slightly reduces performance compared with the default android system, which is acceptable to the user.

Finally, we compare our system with previous prototype systems, and the specific results are shown in table 3. By observing table 3, we find that MobiMimosa enables not only the multiple hidden volume but also dynamic mounting of the hidden volume (i.e., the hidden volume and outer volume are mounted simultaneously).

Through a novel storage method (see III-*D*), our prototype reduces the corruption of the cross volume boundary and improves space utilization (i.e., the data written into the outer volume can occasionally be stored in the free area of the hidden volume). We can easily add the new function module into DataSafe to expend MobiMimosa and provide a friendly

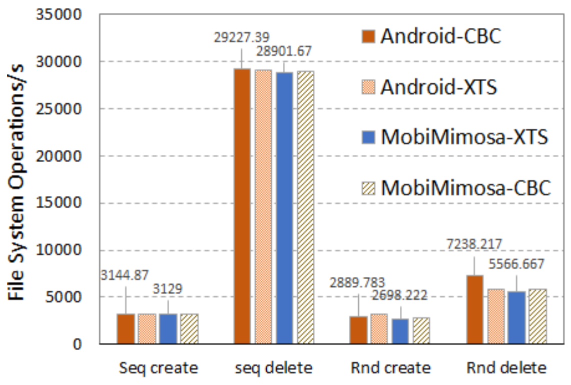

FIGURE 7. File system operation with Bonnie++ (Seq: Sequential; Rnd: Random).

GUI to facilitate the interaction between the user and our system. Therefore, our prototype is more practical than previous prototypes.

VI. SECURITY ANALYSIS

In this section, we detail four leakage vectors for sensitive data stored in the hidden volume. The four vectors exist only when the security strategy of MobiMimosa experiences complete failure (the probability is very low). Specifically, a powerful adversary can enter the operating system or kidnap the owner of the smartphone, and then, they could gain full control of the mobile device. Furthermore, sensitive data stored in the hidden volume is not destroyed in time. For this scenario, we analyze the leakage vectors and present corresponding countermeasures.

Flash storage leakage. A smartphone usually stores data that belongs to a user using flash storage. Flash memory is not divided into sectors in the same way as magnetic disks. The writing operation takes place on the page level (e.g., a 4-kb page), and it takes place on an empty page. However, the erasing operation takes place on a group of pages, called an erasing block (e.g., 64 pages per block). This arrangement results in the production of multiple versions of the erased data after performing the deleting operation many times [32]. To implement secure deletion, Mobiflage adopts the method of filling the free space of the flash storage two times using random numbers. The limitations of of this method are that filling the flash storage with random numbers will take a large amount of time and power. These constraints are not acceptable in mobile devices. In addition, the authors of the other two prototype systems do not explain how their systems delete sensitive data in a secure way. To reduce the wear of the flash storage, when erasing data, we advise the user to use a combination of file-based encryption and transparent encryption. When a file must be deleted, the user needs to only delete the corresponding key or discard the corresponding password [32]. Key security is outside the scope of this paper. When the flash storage has less free space, we can securely delete data by filling random numbers into the free space two times. MobiMimosa adopts the AES-XTS encryption mode, which can resist a copy-and-paste attack [30]. While transferring data from the outer volume to the hidden volume, previous prototype systems (e.g., Mobiflage and MobiHydra) can produce a copy of the sensitive file. For example, MobiHydra first stores a sensitive file that was created in the normal mode into a cache partition. MobiHydra then transfers the file from the cache partition to the hidden volume, after switching to the PDE mode. This method results in two copies of the file and compromises the deniability of the hidden volume. To avoid the production of one copy or multiple copies, we can mount the hidden volume to the corresponding directory in which the application stores data using the ‘‘bind’’ parameter of the mount command.

File system leakage. Currently, the android system mainly enables the Ext4 file system, which is divided into a multiple block group. In previous PDE prototypes, the hidden volume is usually located in a continuous area in the outer volume. When the user writes sensitive data to the hidden volume, these data can overwrite the blocks that are occupied by the meta-data (e.g., the inode bitmap table) of the outer volume. A powerful adversary can find an abnormal result using the DumpE2fs tool or the file system checking tool. This circumstance will compromise the plausible deniability of the hidden volume. To avoid overwriting the meta-data blocks, we map the hidden volume to the data blocks of the outer volume by building a mapping table. We currently create the mapping table through manually computation. In the future, we will implement automatic creation of a mapping table using the procedure according to size of hidden volume, for adapting various types of smartphones.

Password attack. The user needs to select a strong password to protect their encryption key. Currently, the pattern authentication of the android system has a 30 second timeout after ten failed password attempts, which alleviates an online guessing attack. An offline dictionary attack is also possible on the flash storage of an android system. The salt of an android FDE is stored in an android footer structure and exists in the form of plaintext. Thus, the adversary can pregenerate dictionaries and rainbow tables through the salt of the FDE. However, MobiMimosa adopts the aes-xts-plain64 encryption mode. To enhance the security of the key, we set 5000 iterations of PBKDF2 and at least six for the length of the password. Similarly, a user can also specify another cipher (e.g., a twofish cipher) to enhance the security of the data stored in a hidden volume.

Software issue. The MobiMimosa enables us to dynamically mount the hidden encrypted volume, which can coexist with the public volume. It is convenient to transfer sensitive data from the outer volume to the hidden volume. However, the probability of leakage is raised. First, an android application usually caches operated data in a certain environment. For example, a PDF reader usually caches the information of a file (e.g., the filename) that was browsed previously to facilitate reading the file again. This arrangement is a leakage source of sensitive data. Therefore, the user is required to reduce operations with the hidden volume as much as possible. Second, the log submodular of an android system is in charge of recording all of the types of logs. The adversary can find some clues regarding the hidden volume by searching all types of system logs. To prevent exposure of the hidden volume from the log, the system logs and the mounting records must be deleted in a timely manner.

VII. CONCLUSIONS

The smartphone has become an indispensable communication tool, and it stores a large amount of sensitive data for the user. The Android system provides some protection measures. A pattern password is the most popular protection measure as is easily remembered and used. However, its security level is also the lowest among the protection measures of an android system. Therefore, we present MobiMimosa, which enables two hidden volumes and application uninstallation. MobiMimosa also enables dynamic mounting of a hidden volume. It is convenient to directly store sensitive data into a hidden volume. Dynamic mounting reduces the leakage of sensitive data from flash storage and the file system. Simultaneously, this arrangement reduces that assumption mode that previous prototype systems present, i.e., that the mobile device cannot be captured in PDE mode. We also provide automatic uninstalling app function, which can prevent leakage of account and password when smartphone of user is lost or forced to take away by the adversary. Specially, we alleviate corruption of the cross-boundary volume and improve the utilization of flash storage. Thereby, the prototype system can satisfy most ordinary users as well as enterprise customers. However, even if MobiMimosa provides relatively comprehensive protection, the leakage of sensitive data is still possible. In our future work, we plan to further solve the corruption of cross boundaries and attempt to add cloud backup for better preservation of sensitive data in the hidden volume.

REFERENCES

[1] (2015). *Statistics and Facts About Android*. [Online]. Available: http://www.statista.com/topics/876/android/

[2] (2015). *Are Your Sales Reps Missing Important Sales Opportunities?* [Online]. Available: http://m.sybase.com/files/White_Papers/Solutions_SAP_Reps.pdf

[3] (2016). *Full Disk Encryption, Android Encryption Technology*. [Online]. Available: http://source.android.com/devices/tech/security/encryption/rity/encryption/

[4] E. Von Zezschwitz, P. Dunphy, and A. De Luca, "Patterns in the wild: A field study of the usability of pattern and pin-based authentication on mobile devices," in *Proc. 15th Int. Conf. Human-Comput. Interact. Mobile Devices Services*, Munich, Germany, 2013, pp. 261–270.

[5] S. Uellenbeck, M. Dürmuth, C. Wolf, and T. Holz, "Quantifying the security of graphical passwords: The case of android unlock patterns," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Berlin, Germany, 2013, pp. 161–172.

[6] A. J. Aviv, K. L. Gibson, E. Mossop, M. Blaze, and J. M. Smith, "Smudge attacks on smartphone touch screens," in *Proc. WOOT*, vol. 10. 2010, pp. 1–7.

[7] L. M. Grupp, J. D. Davis, and S. Swanson, "The bleak future of NAND flash memory," in *Proc. 10th USENIX Conf. File Storage Technol.*, San Jose, CA, USA, 2012, p. 2.

[8] A. Czeskis, D. J. S. Hilaire, K. Koscher, S. D. Gribble, T. Kohno, and B. Schneier, "Defeating encrypted and deniable file systems: TrueCrypt v5. 1a and the case of the tattling OS and applications," in *Proc. HotSec*, 2008, pp. 1–7.

[9] A. Skillen and M. Mannan, "On implementing deniable storage encryption for mobile devices," in *Proc. 20th Annu. Netw. Distrib. Syst. Secur. Symp.*, 2013, pp. 1–17.

[10] A. Skillen and M. Mannan, "Mobiflage: Deniable storage encryption for mobile devices," *IEEE Trans. Depend. Sec. Comput.*, vol. 11, no. 3, pp. 224–237, May/Jun. 2014.

[11] X. Yu, B. Chen, Z. Wang, B. Chang, W. T. Zhu, and J. Jing, "MobiHydra: Pragmatic and multi-level plausibly deniable encryption storage for mobile devices," in *Proc. 17th Int. Conf. Inf. Secur. (ISC)*, Hong Kong, Oct. 2014, pp. 555–567.

[12] B. Chang, Z. Wang, B. Chen, and F. Zhang, "MobiPluto: File system friendly deniable storage for mobile devices," in *Proc. 31st Annu. Comput. Secur. Appl. Conf.*, Los Angeles, CA, USA, 2015, pp. 381–390.

[13] (2016). *Android System Architecture*. [Online]. Available: https://source.android.com/security/

[14] R. Canetti, C. Dwork, M. Naor, and R. Ostrovsky, "Deniable encryption," in *Advances in Cryptology—CRYPTO*, vol. 97. Berlin, Germany: Springer, 1997, pp. 90–104.

[15] P. Gasti, G. Ateniese, and M. Blanton, "Deniable cloud storage: Sharing files via public-key deniability," in *Proc. 9th Annu. ACM Workshop Privacy Electron. Soc.*, Chicago, IL, USA, 2010, pp. 31–42.

[16] R. Anderson, R. Needham, and A. Shamir, "The steganographic file system," in *Proc. 2nd Int. Workshop Inf. Hiding (IH)*, Portland, OR, USA, Apr. 1998, pp. 73–82.

[17] A. D. McDonald and M. G. Kuhn, "StegFS: A steganographic file system for Linux," in *Proc. 3rd Int. Workshop Inf. Hiding (IH)*, Dresden, Germany, Sep./Oct. 1999, pp. 463–477.

[18] H. Pang, K.-L. Tan, and X. Zhou, "StegFS: A steganographic file system," in *Proc. 19th Int. Conf. Data Eng.*, Mar. 2003, pp. 657–667.

[19] (2015). *FreeOtfe—Free Transparent Disk Encryption Software*. [Online]. Available: http://sourceforge.net/projects/freeotfe.mirror/

[20] (2016). *VeraCrypt Document Guidance*. [Online]. Available: http://veracrypt.codeplex.com/documentation

[21] (2015). *Elettra—Plausible Deniable File Cryptography*. [Online]. Available: http://www.winstonsmith.info/julia/elettra/

[22] C. Saout, "dmcrypt: A devicemapper crypto target," 2011. [Online]. Available: http://www.saout.de/misc/dmcrypt/

[23] (2016). *Ext4 Disk Layout*. [Online]. Available: https://ext4.wiki.kernel.org/index.php/Ext4_Disk_Layout

[24] J. CoBERT. *The Orlov Block Allocator*. [Online]. Available: https://lwn.net/Articles/14633/

[25] (2016). *dmsetup Tool and Mapping Table*. [Online]. Available: http://www.mediacollege.com/cgi-bin/man/page.cgi?section=8&topic=dmsetup

[26] (2015). *CryptSetup Tool Guidance*. [Online]. Available: https://gitlab.com/cryptsetup/cryptsetup/wikis/DMCrypt

[27] K. Yaghmour, *Embedded Android: Porting, Extending, and Customizing*. Farnham, U.K.: O'Reilly Media, Inc., 2013.

[28] (2016). *Android System Broadcast*. [Online]. Available: https://developer.android.com/reference/android/content/Intent.html

[29] C. Fruhwirth, *New Methods in Hard Disk Encryption*. Citeseer, Penn State Univ., PA, USA, Jul. 2005.

[30] L. Martin, "XTS: A mode of AES for encrypting hard disks," *IEEE Security Privacy*, vol. 8, no. 3, pp. 68–69, May/Jun. 2010.

[31] T. Bray, "Bonnie filesystem benchmark," 2016. [Online]. Avilable: http://www.textuality.com/bonnie/

[32] J. Reardon, D. Basin, and S. Capkun, "SoK: Secure data deletion," in *Proc. IEEE Symp. Secur. Privacy (SP)*, May 2013, pp. 301–315.

SHUANGXI HONG received the master's degree in computer application technology from the North China University of Water Resources and Electric Power, China, in 2013. He is currently pursuing the Ph.D. degree with the State Key Laboratory of Networking and Switching Technology, Beijing University of Posts and Telecommunications. His major research interests include mobile device security, computer network and privacy protection, storage encryption technology, and data mining.

CHUANCHANG LIU is currently an Associate Professor with the State key Laboratory of Networking and Switching Technology, Beijing University of Posts and Telecommunications. His current research interests mobile device security, cloud computing, and oriented-service computing.

BINGFEI REN received the B.S. degree in software engineering from the Beijing University of Posts and Telecommunications, China, in 2014, where he is currently pursuing the Ph.D. degree with the State Key Laboratory of Networking and Switching Technology. His main research interests include service-oriented computing, mobile security, and data management.

YUZE HUANG received the B.S. degree in applied physics from Hangzhou Dianzi University, China, in 2008, and the M.Eng. degree in computer technology from the Kunming University of Science and Technology, China, in 2013. He is currently pursuing the Ph.D. degree with the State Key Laboratory of Networking and Switching Technology, Beijing University of Posts and Telecommunications. His main research interests include service-oriented computing, data analysis, and management. He is a Student Member of ACM and CCF.

JUNLIANG CHEN is currently a Professor and the Academic Leader with the State key Laboratory of Networking and Switching Technology, Beijing University of Posts and Telecommunications. He is a member of the Chinese Academy of Science and the Chinese Academy of Engineering, and a fellow of the China Computer Federation. His current research interests include service-oriented computing and service generation system.