Data Loss Prevention and Storage Utilization Improvement of the Hidden Volume on Mobile Devices

Wendi Feng*, Chuanchang Liu*, Zehua Guo†, Thar Baker‡, Bo Cheng* and Junliang Chen*
*Beijing University of Posts and Telecommunications
†University of Minnesota Twin Cities
‡Liverpool John Moores University

***Abstract*—Sensitive data protection is vital for mobile users. An effective way is to store sensitive data in the hidden volume of mobile devices with Plausibly Deniable Encryption (PDE) systems. Typically, PDE creates a hidden volume inside the outer volume. However, existing PDE systems could lose data, due to overriding the hidden volume, and significantly waste physical storage, because of the fixedly reserved area for the hidden volume. In this paper, we present MobiGyges to solve the above problems. MobiGyges leverages the Thin Pool to coordinate the storage allocation for both the outer volume and the hidden volume which avoids data override and prevents data loss. Moreover, it improves the storage efficiency by virtualizing the total physical storage into small storage blocks and utilizing the blocks for the outer volume and the hidden volume, which eliminates the reserved area. We implement MobiGyges prototype on Google Nexus 6P with LineageOS 13 operating system. Experimental results show that MobiGyges avoids data loss and improves storage utilization up to 31%.**

***Index Terms*—mobile storage security, sensitive data protecting, override prevention, device mapper, thin provisioning**

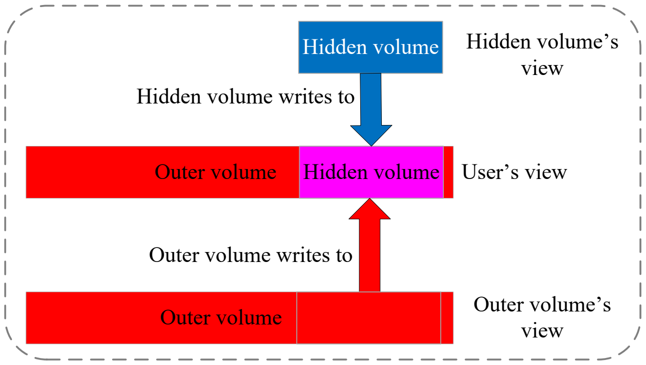

Fig. 1. Data loss caused by data override. The hidden volume occupies the blue block of the physical volume as its own resource while the outer volume occupies the whole disk (red block) as its own resource. When both the outer volume and the hidden volume commit data to that area, the storage block taken by the hidden volume is conflict (the purple block), and thus sensitive data lose.

I. Introduction

Protecting private and sensitive data on mobile devices (e.g., smartphones, tablets) is very important to mobile users. One intuitive solution is to use Full Disk Encryption (FDE) [1]. FDE encrypts all user data of a device with an encrypting key. However, FDE is not secure because the data would be exposed if the encryption key is given.

Recent works adopted Plausible Deniable Encryption (PDE) [2] to improve the security of the devices. PDE is a data protection paradigm that can conceal sensitive data on a device. Hidden volume mechanism is a typical solution for PDE and divides the physical volume of the device into one *outer volume* and the *hidden volume* logically. The outer volume is visible to everyone and stores the public data, while the hidden volume is visible only to the user and stores the sensitive data.

Previous hidden volume solutions have three problems. The first problem is data loss. Typically, a hidden volume is concealed inside the outer volume. Since the location of the hidden volume is unknown, the outer volume data could be written to the storage blocks occupied by the hidden volume and thus overrided the sensitive data in the hidden volume. Therefore, these important sensitive data could be lost. Fig. 1 shows an example of data override between the outer volume and the hidden volume. Recent works [3]–[6] solve this problem by reserving a large storage space for the hidden volume. The second problem is storage space waste. The reserved area is usually larger than the capacity of the hidden volume. This reserved method enables the hidden volume ”floats” inside the reserved area to hide the real location of the hidden volume. Consequently, a huge amount of physical storage space is wasted. In Fig. 2, (a) depicts an example of storage space waste. We investigate existing works [3], [4], [7] and find such a design introduced up to 45% storage space waste. The third problem is exposing the hidden volume. Normally, one device only has an outer volume, and its capacity is the same as the physical volume's capacity. However, introducing the hidden volume reduces the outer volume's capacity, potentially revealing the hidden volume. For example, a 32GB device uses 5GB for the hidden volume, thus the outer volume capacity becomes 27GB. The inconsistency of capacity is prone to compromise the hidden volume. Existing works cannot solve the three problems at the same time.

In this paper, we propose MobiGyges to address the aforementioned problems. MobiGyges is a hidden volume based PDE security system with the Volume Management module and the FDE module. Volume Management module allocates different storage blocks to the outer volume and the hidden volume, which avoids data loss caused by the data override problem. The Volume Management module utilizes the Device Mapper (DM) and Thin Provisioning to virtualize the physical storage space into small blocks and store data for the outer volume and the hidden volume in the unit of the blocks, thus

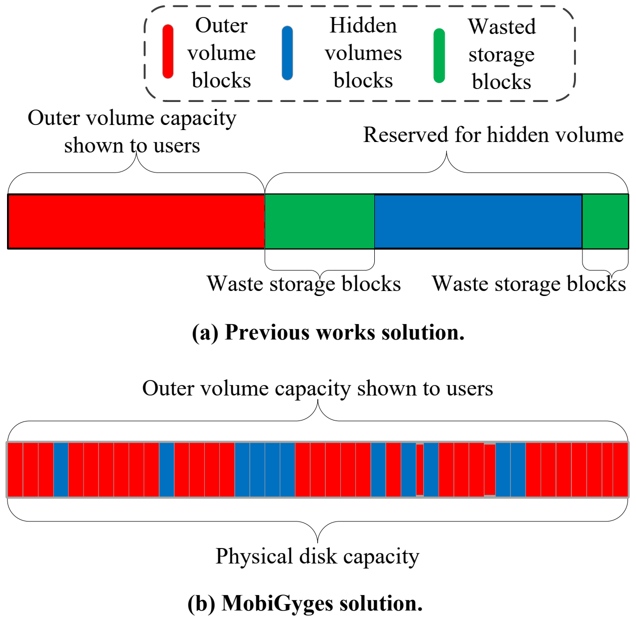

Fig. 2. Placing hidden volume into a reserved area to avoid data override. Previous solutions could waste a large amount of space. MobiGyges can use all the storage space for data except some metadata.

eliminating the reserved area (Shown in Fig. 2b). We leverage Thin Provisioning to use the capacity of outer volume as that of physical volume and prevent observing the volume capacity difference between the outer volume and the physical volume to identify the hidden volume.

Our contributions are summarized as follows:

- We propose MobiGyges to address the data override problem and improve the storage utiliztion with Thin Provisioning and Thin Pool. By using the method of Shrunk U-Disk, we reduce the possibility of system compromise due to the significant differences between the outer volume showing capacity and physical volume capacity.
- We implement the MobiGyges prototype system on Google Nexus 6P with LineageOS [8] 13 Operating System. To implement our system, we port two tools (i.e., pdata_tools, LVM2) into the Android system. We conduct experiments on storage space waste and IO performance. The results show MobiGyges prevents the data override problem and improve the utilization up to 31%.

The rest of the paper is organized as follows. Section II, introduces the threat model background and related works. Section III overviews the design of MobiGyges. In Section IV, we detail the key components of MobiGyges. In Section V, we introduce how we implement MobiGyges. Section VI evaluates the performance of MobiGyges. Finally, we conclude the paper in Section VII.

II. Threat Model and Related Works

In this section, we discuss threat model and related works.

A. Threat Model

The key to design a hidden volume based PDE system is concealing the hidden volume. We make the following threat model for MobiGyges according to [3].

1) An attacker knows that userdata of a mobile device is encrypted by FDE and also knows the key to this encryption but lacks the knowledge of MobiGyges's design. Alternatively, An attacker knows the design of MobiGyges but cannot get the Thin Pool raw data.
2) An attacker is able to get root privilege and the physical storage of the phone or dump the raw data from the storage medium but cannot decrypt the raw data and cannot recover the MobiGyges layout.

B. Related Works

FDE is an elementary way to prevent sensitive data from attacks. It encrypts all the data on a volume. It has now been a standard configuration for most of the security system [9]. There are many mature FDE solutions like TrueCrypt [10], [11](deprecated to use), BitLocker [12], [13], LUKS(Linux Unified Key Setup) [14],FileVault [15], etc. However, FDE fails to provide users the deniability for the sensitive data.

StegFS [16] is a steganography-based PDE file system. It is based on the Ext2 file system and its key idea is to hide data in a bunch of cover files. Its shortcoming includes: (1)It costs space; (2)The performance is low especially when writing; (3)The Possibility of data loss is high; (4)The modification of Ext2 may lead to compromise of deniability. All these shortcomings make it not suitable for mobile devices.

TrueCrypt [10] and FreeOTFE [17] are PC PDE solutions. They can create hidden volume(s) as files or physical volumes. However, both TrueCrypt and FreeOTFE can only create PDE hidden volume on its resource file. Therefore, if the resource file is lost, all the data stored on the hidden volumes will be lost, which is prone to compromise.

Mobiflage [3] is the first implementation of the mobile PDE prototype system by modifying the FDE and Ext4 file system on Android. In Mobiflage, the hidden volume is concealed inside the outer volume. The outer volume data could be written to hidden volume occupied blocks.Thus, the data override issue exists and may result in sensitive data loss. Based on Mobiflage, MobiHydra [7] implements multi-level deniability and it allows the user to store secret data without rebooting the system. However, similar to Mobiflage, MobiHydra also fails to solve the problem of data loss.

MobiPluto [4], [5] is the first file system friendly PDE solution built on The Android operating system. It uses DM and Thin Provisioning technologies to create virtual volume on the Android system. Any block-based file system can run on top of the virtual volume without modification. Although it addresses the data loss problem by placing the hidden volume into a reserved area, but the storage space is not efficiently used.

MobiMimosa [18] is a hidden volume PDE solution for Android. It records the hidden volume occupied blocks and simply skips those blocks while writing data. So, the hidden volume can be easily exposed. Thus, this solution is prone to compromise.

MobiCeal [6] is another recent hidden volume PDE solution, its key contribution is to defend against strong coercive multi-snapshot adversaries. they do not consider the storage space waste and capacity inconsistency problems faced with previously hidden volume solutions.

III. MobiGyges Overview

In this section, we first introduce the overview design of MobiGyges starting at its components. Then we propose a so called Shrunk U-Disk method to hide the hidden volume. Finally, we introduce Thin Pool to overcome data loss and storage space waste problems.

A. *MobiGyges Workflow*

MobiGyges consists of three components: Volume Management, FDE, and physical volume. Fig. 3 depicts the bird's eye view of the MobiGyges system. As shown in the Fig., (1) Apps generates or receives data and make data flow redirected between the Volume Management; next (2) the Volume Management component process the data with DM, and Thin Provisioning and sends Input/Output (IO) redirects to the FDE; finally, (3) PDE encrypts or decrypts the data and communicates with the physical volume and commit data on it or read data from it.

B. *MobiGyges System Design*

Our key contribution of MobiGyges is threefold: (1) designing the Volume Management module by leveraging Thin Pool to coordinate storage space allocation for the outer volume and hidden volume to avoid data loss; (2) Using Thin Pool to eliminate the storage space waste by virtualizing the physical storage space into small blocks, and allocated for the outer volume and hidden volume separately. (3)leveraging the shrunk U-Disk method to hide the hidden volume.

Shrunk U-Disk is a kind of disk, which is labeled with a bigger capacity. When users plug it into a computer, the shown capacity is the same as the label. However, the actual physical capacity is much smaller. Normal users cannot discover the trick. We find this can be used to address the capacity inconsistency problem between the outer volume and physical volume. For example, given a mobile phone labeled with 128GB storage if the user logs into the system and the system show that the device has approximately 128GB storage, users could believe no other storage spaces on it. Since Thin Provisioning can create a virtual logical volume with arbitrary capacity, if the outer volume capacity is set to the whole physical capacity, people will not discover the hidden volume. Moreover, with the help of Thin Pool, the real capacity of the outer volume is able to change dynamically. Thus, even if filling the outer volume until a full error occurs, the attack can still hardly find tricks because the outer volume allocates storage space on demand from the Thin Pool as long as it has storage space available. The real capacity of the outer volume could reach the physical capacity.

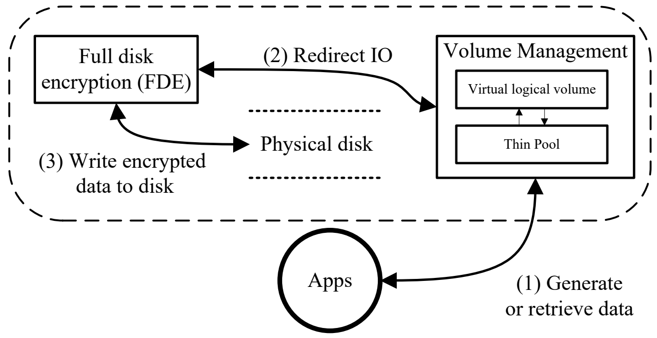
Full disk encryption (FDE)
(2) Redirect IO
Volume Management
Virtual logical volume
Thin Pool
Physical disk
(3) Write encrypted data to disk
Apps
(1) Generate or retrieve data

Fig. 3. MobiGyges key components and their relationship.

IV. MobiGyges Components

This section illustrates the detailed design of MobiGyges key components. We do not mention the physical volume component because it is a part of a device hardware and we do not change it.

A. *Full Disk Encryption (FDE)*

MobiGyges first performs FDE by creating an encryption layer on the `userdata` partition. Thus, it is easy to encrypt everything on the `userdata` partition. MobiGyges uses 128-bit Advanced Encryption Standard (AES) [19] with cipher-block chaining (CBC) and ESSIV:SHA256 to perform encryption. And the master key is encrypted with 128-bit AES via calls to the OpenSSL library. As shown in Fig. 4, (2) stands for the encrypted logical volume created on top of the physical volume. It also shows the logical position in MobiGyges. FDE is created using DM technology. DM maps existing block devices into another logical block device. Furthermore, DM redirects or filters IO request from logical block devices to the mapped device. FDE is applied to MobiGyges that encrypts everything on the encrypted logical volume including the Thin Pool and the MobiGyges structure thus protects the system.

B. *Volume Management*

Volume Management module leverages the Thin Provisioning technology to reduce the storage space waste and uses virtual logical volume to make the outer volume like a "Shrunk U-Disk". Virtual logical volumes are built on Thin Pool and the creation of Thin Pool requires Thin Provisioning technology. Thin Provisioning is a kind of storage virtualization technology [20]. Originally, it solves the problem of storage space wastes. Normally, computers especially servers tend to provide disks whose capacity is larger than the actual need for further use. That is called Thick Provisioning. However, most of the time, disks would not store data as much as its capacity, which results in wastes of storage space. Thin Provisioning can configure the capacity of the disk in advance, which allows the assigned virtual capacity can be larger than the real capacity. Provisioned volume will not occupy the physical disk space until data is written to. In the rest of this section, we will introduce how Thin Pool and virtual logical volume are applied to MobiGyges.

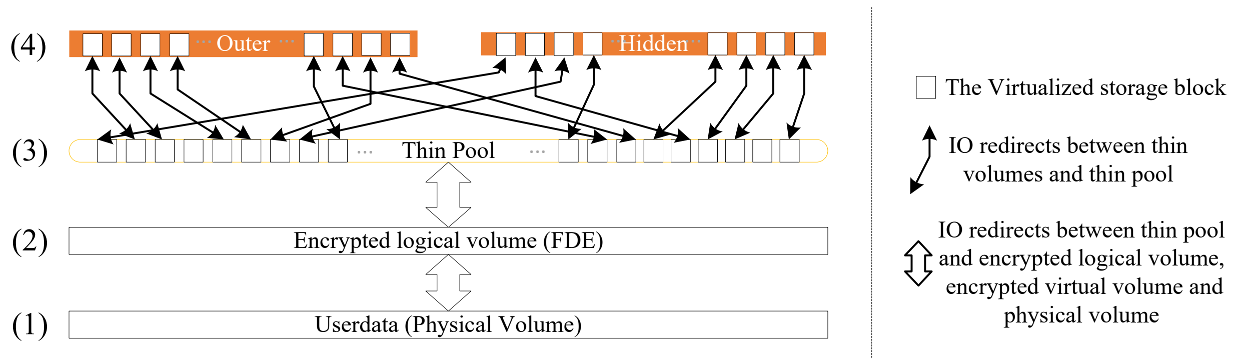
(4)
Outer
Hidden
(3)
Thin Pool
(2)
Encrypted logical volume (FDE)
(1)
Userdata (Physical Volume)
The Virtualized storage block
IO redirects between thin volumes and thin pool
IO redirects between thin pool and encrypted logical volume, encrypted virtual volume and physical volume

Fig. 4. Disk layout of MobiGyges. (1) The `userdata` is the physical volume used for storing user-generated data. Next, (2) FDE is applied to encrypt both data and the structure atop. Then, (3) all the userdata storage is virtualized as a Thin Pool. Finally, (4) virtual logical volumes are created as outer volume and hidden volume, respectively.

(1) **Thin Pool** is created on top of the encrypted logical volume. As shown in Fig. 4, Thin Pool virtualizes the encrypt logical volume as a resource pool, which consists of small storage blocks splited from the whole storage block. In our case, each block is 64k in size. Every block is used for the outer volume and the hidden volume solely, without introducing the reserved area thus no storage space waste introduced. At first, motivated by the limitations of previous solutions [3]–[7], we attempt to split all the hidden volume blocks and randomly place the hidden volume blocks on the whole physical volume. To avoid data override, our early attempt marks the hidden volume occupied block as bad blocks for the outer volume. Thus, the outer volume will not allocate those blocks for the new data. However, the way to record the location of these blocks remains a problem. [21] proposes a simple method of storing these metadata to the secure cloud, but it is not realistic for the devices to connect to the network all the time. Finally, we find Thin Pool to solve the problem.

The creation of Thin Pool consists of two parts. In the first part, two logical volumes are created, one stores data and another stores the metadata. The metadata is used for virtual logical volumes. In the second part, those two logical volumes are converted into a Thin Pool. The size of the metadata volume and the data volume can be formalized as:

$$S_m = \frac{S_p}{S_c} \times 64. \quad (1)$$

where S_m denotes the size of metadata volume, S_p denotes the size of pool volume, S_c denotes the chunk size of pool volume.

(2) **Virtual logical volume** is the volume allocating storage space from Thin Pool and. The topmost part of Fig. 4 depicts the logical position in the MobiGyges system. MobiGyges uses virtual logical volume as the outer volume and the hidden volume. The usage of the virtual logical volume has no differences between that of a physical volume. Virtual logical volume has a virtual capacity when it is created. It does not occupy thin pool blocks until they need write data. In MobiGyges, the capacity of the outer volume is configured to be the same as the physical volume. The hidden volume is assigned with a capacity, which is the minimum to the total size of sensitive data. According to the Shrunk U-Disk trick, people will believe no other devices on the device apart from the outer volume. Notably, DM is used to create the virtual logical volume from the Thin Pool.

In conclusion, the operating mechanism of MobiGyges comes as follows: (1) the mobile device has a physical volume used to store user data; next, (2) FDE is applied to create an encryption layer to encrypt everything atop including the structure of Thin Pool, virtual logical volume and the data; (3) Thin Pool is created to dynamically coordinate the storage space for different volumes and eliminate storage space waste on the physical storage and finally; (4) virtual logical volumes are created to serve as the outer volume and the hidden volume, furthermore, the outer volume capacity matches the physical capacity, which conceals the hidden volume.

V. Implementation

We implement the MobiGyges prototype system with LineageOS 13 on Google Nexus 6P. We add about 500 Lines of Code to the LineageOS and port LVM2 (Logical Volume Management version 2) and pdata_tools [22] to Android. In this subsection, we introduce our main challenges face while implementing MobiGyges and how we manage to overcome them.

(1)Manipulating the `userdata` partition is hard on Android. The creation of Thin Pool and virtual logical volume requires an unoccupied partition. However for stability and data integrity consideration, the `userdata` partition cannot be unmounted while Android is running. Because certain files are stored on the `userdata` partition and are used by the system. To this end, the Thin Pool and virtual logical volume creation procedure code run at the system startup stage, before the `userdata` partition is mounted. We assemble our code right after the FDE in Android Volume Daemon (VOLD) in `cryptfs.c`.

(2)Running toolsets on Android is another challenge. The toolsets are usually for desktop and server Linux distributions. They do not have Android versions. But

We can port those tools to Android without much modification because Android is based on Linux Kernel. We use `gcc-arm-linux-androideabi` to conduct the cross-compile. Note that the tools have dependency libraries that should be cross compiled in advance. Besides, `-enable-static_link` and `LDFLAGS=-static` flags should be set for the compiler to make sure the compilation target is a static executable because target devices usually do not have these dependencies. Otherwise, it cannot run successfully when pushed to Android.

VI. Evaluation

In this section, we first conduct experiments for the storage utilization. Then, we evaluate the IO performance of MobiGyges and make a comparison to the original Android FDE. We use `dd` [23] and bonnie++ [24] to conduct the experiments. Unfortunately, We can not make comparisons between MobiGyges and former works because neither of them provides a precompiled package nor concrete documentation on building the system. We also cannot use their experiments result directly, because the devices used in their experiments are not exactly the same model as ours. Thus, in the storage utilization evaluation parts, we make a comparison between the experimental results of our solution and the theoretical results of former solutions.

A. Storage Utilization Evaluation

MobiGyges uses Equation 1 to calculate the size of metadata volume. The Thin Pool is created by converting the metadata volume and the data volume. In addition, the size of Thin Pool equals to the size of the data volume. Therefore, the size of the data volume is the actual size for users to store their data on the device. Consequently, the utilization of physical storage η can be calculated with the Equation 2. The meaning of the notations here is the same as that of Equation 1.

$$\eta = \frac{S_p}{S_p + S_m} = \frac{S_p}{\frac{S_p}{S_c} \times 64 + S_p} \times 100\% = 99.9024\%. \quad (2)$$

We use `dd` to perform the experiment. After 50 trials of tests, the average utilization is 79.81%. Normally, Ext4 file system occupies about 10% storage for its system use [25], [26]. Since we have two virtual logical volumes (a outer volume and a hidden volume), so there is a 20% loss resulting from file system. Consequently, the storage space utilization of MobiGyges is 99.76% excluding the 20% occupied by file systems.

Mobiflage proposes an equation in [3] to calculate the offset of placing the hidden volume inside the outer volume. Thus, the minimum size of the MobiGyges hidden volume is 25% of the total disk size, and the reserved area is 50%. Therefore, the storage space waste introduced by the architecture is 45%[1]. Most of the related works [4]–[7] follow the method of Mobiflage. As shown in Fig. 5, the storage utilization for these former solutions is down to 68.75%. Therefore, MobiGyges has a 31% performance improvement thus greatly reduces physical storage space waste.

[1]Including the 20% caused by the two file systems.

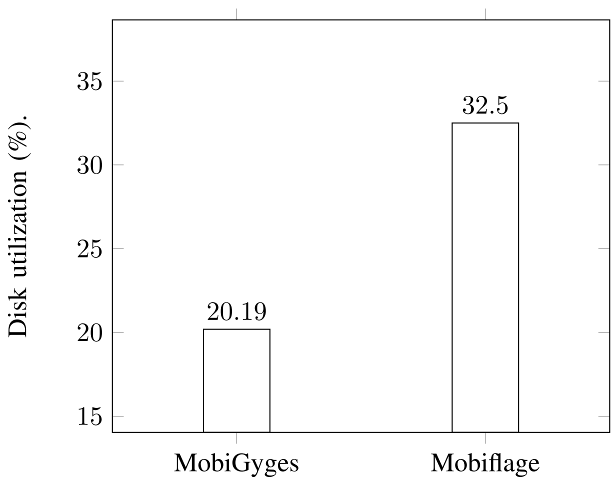

Fig. 5. Disk utilization result of MobiGyges and Mobiflage.

B. IO Performance

IO performance evaluate is used for illustrating the performance differences between the original Android FDE and our MobiGyges. Because significant differences may expose the hidden volume, which compromises the phone.

(1) Bonnie++ test: We first use Bonnie++ V1.97.3 to conduct the experiment. We compare the performance of Android FDE rather than the physical disk because Android compulsorily apply FDE to `userdata` since Android 5.0, and users cannot disable it. We run 100 trials of sequential block tests on a 6GB file[2] on each system.

Fig. 6 shows the outer volume has an approximately 10% performance improvement on sequential reading. Thus for example, when reading a 1GB file, MobiGyges reduces about 500ms compared with the original Android FDE. This is mainly because when reading sequentially, the system can merge different IO requests, cache them and read adjacent storage blocks together [27], which improves the performance. The result shows that the writing performance is reduced by about 12% due to three IO redirects procedure but writing operations cannot be merged. Similarly, it takes 600ms longer than the original Android FDE for writing a 1GB file. By comparing the hidden volume and outer volume of MobiGyges, the writing performance is only reduced by 3%, and the hidden volume takes 150ms longer for writing a 1GB file. Since half a second delay does little influence on the user experience of IO consuming tasks [28], the structure of Thin Provisioning and device mapper do not have a significant influence on the performance of IO operations.

(2) `dd` test: `dd` is another IO tool running on UNIX-like systems. It allows user to write data directly to disk without using cache by defining `fsync` flag. Both read[3] and write

[2]Bonnie++ requires to test on a file whose size is twice as big as the device RAM to decrease the influence of system cache.

[3]We do not use `conv=fsync` option while going through reading tests. Because it will cause an error when the output file is `/dev/null`.

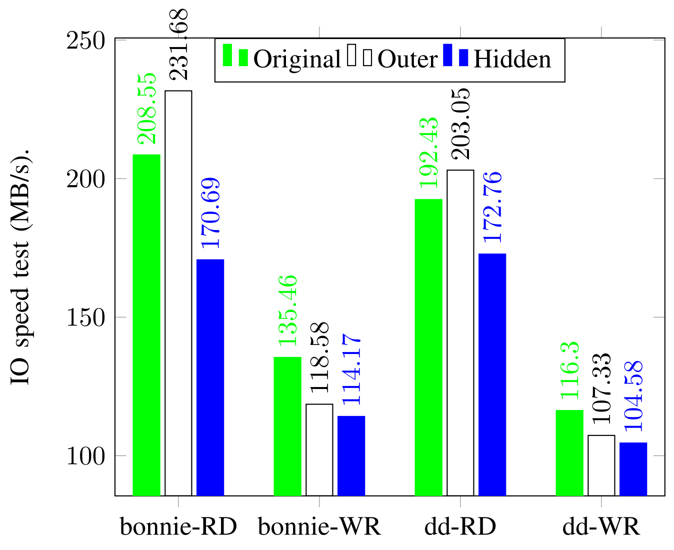

Fig. 6. Bonnie++ Sequential read and write speed (MB/s) and dd read and write speed (MB/s) comparing between outer volume, hidden volume, and original Android FDE.

experiments are conducted. We have to manually clear the memory cache[4] every time before running reading tests. This will clear the error caused by system cache. As we can see from Fig. 6, the test result is similarly to bonnie++.

Compared with the original Android FDE, the reading performance of MobiGyges outer volume increases by approximately 5-10%. While the reading performance of MobiGyges hidden volume has reduced by 10-15%. Therefore, the encryption layer is the key factor that affects reading performance. Hidden volume data need to be decrypted first before using, which results in the performance loss of hidden volume reading. Besides, the writing performance changes 7-10%. The writing performance of MobiGyges outer and hidden volume only have a 2% difference. As we illustrated in Bonnie++ test evaluation, performance differences under 15% will not compromise the device.

VII. Conclusion

In this paper, we present the hidden volume based PDE system MobiGyges. It leverages the DM and Thin Provisioning with the Shrunk U-Disk method to address the problem of data loss, storage space waste and capacity inconsistency between the outer volume and the hidden volume. We port toolsets to Android and implement the MobiGyges prototype system with LineageOS 13 on Google Nexus 6P. Finally, we conduct storage utilization and IO performance experiments, and the experiment evaluations show that MobiGyges improves storage utilization up to 31% without significantly losing performance. We have many lessons learned from the designing and implementing MobiGyges, the lessons will be the experiences for our further study and research life. We will further research on the user experience and the use of hidden volume.

[4]`echo 3 > /proc/sys/vm/drop_caches`

References

[1] J. Götzfried and T. Müller, "Analysing android's full disk encryption feature." *JoWUA*, vol. 5, no. 1, pp. 84–100, 2014.
[2] R. Canetti, C. Dwork, M. Naor, and R. Ostrovsky, "Deniable encryption," in *CRYPTO'97*. Springer, 1997, pp. 90–104.
[3] A. Skillen and M. Mannan, "Mobiflage: Deniable storage encryptionfor mobile devices," *IEEE TDSC*, vol. 11, no. 3, pp. 224–237, 2014.
[4] B. Chang, Z. Wang, B. Chen, and F. Zhang, "Mobipluto: File system friendly deniable storage for mobile devices," in *ACSAC'15*. ACM, 2015, pp. 381–390.
[5] B. Chang, Y. Cheng, B. Chen, F. Zhang, W. T. Zhu, Y. Li, and Z. Wang, "User-friendly deniable storage for mobile devices," *Computers & Security*, vol. 72, pp. 163–174, 2018.
[6] B. Chang, F. Zhang, B. Chen, Y. Li, W. T. Zhu, Y. Tian, Z. Wang, and A. Ching, "Mobiceal: Towards secure and practical plausibly deniable encryption on mobile devices," in *DSN'18*, 2018, pp. 454–465.
[7] 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 *ISC'14*. Springer, 2014, pp. 555–567.
[8] J. John and C. K. Raju, "Design and comparative analysis of mobile computing software framework," in *ICICCT'18*, April 2018, pp. 1639–1644.
[9] L. Khati, N. Mouha, and D. Vergnaud, "Full disk encryption: bridging theory and practice," in *In Proc. CT-RSA'17*. Springer, 2017, pp. 241–257.
[10] Team, TrueCrypt, "Truecrypt-free open-source disk encryption software for windows vista/xp, mac os x, and linux, sept 2018," http://www.truecrypt.org/.[accessed 18 Sept-2018].
[11] 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 *HotSec'08*, 2008.
[12] The Windows Team, "Bitlocker drive encryption overview," https://technet.microsoft.com/en-us/library/cc732774(v=ws.11).aspx, 2010.
[13] N. Kumar and V. Kumar, "Bitlocker and windows vista," 2008.
[14] C. Fruhwirth, "Luks–linux unified key setup," 2009.
[15] X. OS, "About filevault 2," *Apple inc. Viitattu*, vol. 22, 2014.
[16] A. D. McDonald and M. G. Kuhn, "Stegfs: A steganographic file system for linux," in *IH'99*. Springer, 1999, pp. 463–477.
[17] S. Dean, "Freeotfe," https://en.wikipedia.org/wiki/FreeOTFE/.[accessed 18 Sept-2018].
[18] S. Hong, C. Liu, B. Ren, Y. Huang, and J. Chen, "Personal privacy protection framework based on hidden technology for smartphones," *IEEE Access*, 2017.
[19] E. Miles and E. Viola, "The advanced encryption standard, candidate pseudorandom functions, and natural proofs," in *ECCC'11*, 2011, p. 226.
[20] A. Veprinsky, O. E. Michael, and M. J. Scharland, "Data de-duplication using thin provisioning," Oct. 26 2010, uS Patent 7,822,939.
[21] W. Feng, C. Liu, B. Ren, B. Cheng, and J. Chen, "Trustgyges: A hidden volume solution with cloud safe storage and TEE," in *MobiSys'18*, 2018, p. 511.
[22] jthornber, "Thin provisioning tools," https://github.com/jthornber/thin-provisioning-tools, 2018.
[23] Wikipedia, "dd (unix)," https://en.wikipedia.org/wiki/Dd_(Unix). [accessed 23 Sept-2018].
[24] Coker, Russell, "Bonnie++ file-system benchmark," http://www. coker. com. au/bonnie+, 2018.
[25] A. Mathur, M. Cao, S. Bhattacharya, A. Dilger, A. Tomas, and L. Vivier, "The new ext4 filesystem: current status and future plans," in *the Linux symposium*, vol. 2, 2007, pp. 21–33.
[26] T.-Y. Chen, Y.-H. Chang, S.-H. Chen, N.-I. Hsu, H.-W. Wei, and W.-K. Shih, "On space utilization enhancement of file systems for embedded storage systems," *ACM TECS*, vol. 16, no. 3, pp. 83:1–83:28, Apr. 2017.
[27] Linux Kernel Organization, "The kernel documentaion of thin provisioning," https://www.kernel.org/doc/Documentation/device-mapper/thin-provisioning.txt, 2019, [Online; accessed 7-Feb-2018].
[28] N. Tolia, D. G. Andersen, and M. Satyanarayanan, "Quantifying interactive user experience on thin clients," *Computer*, vol. 39, no. 3, pp. 46–52, March 2006.