A Secure Storage System for Sensitive Data Protection Based on Mobile Virtualization

Su-Wan Park, JaeDeok Lim, and Jeong Nyeo Kim

Cyber Security Research Division, Electronics and Telecommunications Research Institute, Daejeon, Republic of Korea

Correspondence should be addressed to Su-Wan Park; parksw10@etri.re.kr

Received 12 October 2014; Accepted 4 February 2015

Academic Editor: Hwa-Young Jeong

Copyright © 2015 Su-Wan Park et al. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

1. Introduction

Today the mobile phone is the first truly pervasive computer device in ubiquitous computing environments. It achieved to help the fusion of real space and cyber space based on the networks among existing things in the real world. In particular, the popularity of smart handheld devices, including smartphone and tablets, is really skyrocketing with advances in hardware and software technologies such as a wide range of communication, computing, and storage capabilities. In addition, wireless technology [1] has been embraced worldwide as it continues to expand in a growing number of applications, both indoors and outdoors. Wireless communications customers continue to demand increased services in terms of voice, video, and data communications. These growing demands are driving cellular/wireless carriers and service providers to expand their wireless infrastructure to achieve higher data rates and increased capacity to serve the increased requirements of a growing customer base.

However, mobile devices which are important components in ubiquitous computing environments have been exposed in security threats such as denial service attack exploiting low information processing capability of low-powered CPU, malicious code attacks exploiting vulnerabilities of mobile platforms and application programs, and exposure of information by unauthorized users. Recently, incident cases caused by attack exploiting vulnerabilities of mobile devices have occurred all over the world. It has been expected that damage of such attacks is serious as Hacking Group has developed and announced worm and virus exploiting vulnerabilities of mobile devices; the number of users of mobile devices and services provided by them has been increased.

In addition, cloud computing [2, 3] is in the spotlight as key IT trend. Users access cloud computing using networked client devices, such as desktop computers, laptops, tablets, and smartphones. Some of these devices such as smartphones rely on cloud computing for all or a majority of their applications so as to be essentially useless without it. Cloud resources are usually not only shared by multiple users but also dynamically reallocated per demand. Thus, cloud computing also has numerous thread problems, since almost every resource, for example, disk, memory, network, and so forth, is shared. As a result, sensitive data of smartphone is easily exposed to a higher risk by option functions of cloud application such as...
automatic updates, and users can steal this information from other users without leaving detectable trails [4].

Furthermore, workplace resources are drifting to personal mobile device in order to conduct business recently. To protect these resources, the Bring Your Own Device (BYOD) policy [5] was introduced. It allows one physical device to be used simultaneously for personal and business needs resulting in a need of rigid separation between these two environments in order to guarantee user privacy while conforming to enterprise policies. The solutions presenting separate environments to the end-user improve usability of device as well as access control mechanisms. To control and protect the data and configuration settings for all mobile devices in the network, mobile device management (MDM) [6] is introduced with BYOD policy. MDM aims to optimize the functionality and security of a mobile communications network while minimizing cost and downtime.

Despite these solutions, users of smartphone still face numerous threats such as sensitive data of user being exposed by unauthorized data access. Thus, users are requiring more smart and secure devices which can securely manage separating workplace resource and personal sensitive data in the device itself. In this paper, therefore, we first discuss the efforts towards solving the above problem in virtualization. Then, we propose a secure storage system based on trusted environments separated from existing platform of smart electronic devices and present secure functions to securely manage sensitive data in trusted environments.

The paper is organized as follows. Section 2 describes virtualization. In Section 3, we propose system framework for secure storage based on trusted virtual domain and suggest secure functions to be operated in trusted virtual domain. Section 4 presents the experimental results about the performance overhead of secure storage system. Lastly, we summarize and conclude in Section 5.

2. Related Works

Virtualization [7–10] uses a single piece of software, which operates in kernel mode: the hypervisor, which is at least two orders of magnitude smaller than general purpose OSs and less likely to have failures [11]. According to [12], there are two main approaches to implement the virtualization technique, by using hypervisor either type 1 or type 2. In hypervisor type 1, also known as hardware level virtualization, the hypervisor itself can be considered as an operating system, since it is the only piece of software that works in kernel mode. Its main task is to manage multiple copies of the real hardware just like an OS which manages multitask. On the other hand, in type 2 hypervisors, also known as operating system level virtualization, the hypervisor itself can be compared to another user application that simply “interprets” the guest machine.

Recently, virtualization is in the spotlight as a possible security tool in addition to its usefulness for resource sharing. By providing a high degree of isolation between individual environments, containment of an entire operating system and all processes beneath it suddenly becomes much more possible than in a traditional single OS environment [4].

In use of virtualization towards confinement, the Terra virtual machine platform [13] and QubesOS [14] were introduced. Terra is a virtual machine monitor (VMM), that is, software that manages multiple virtual machines, in order to provide isolation and privacy between them. What distinguishes Terra from normal VMs is that there are two defined types of virtual machines that can run on Terra: “open boxes,” which have no real distinction with traditional VMs, and “closed boxes,” that are highly isolated to ensure privacy and integrity of their contents. QubesOS serves as a hybrid VMM and operating system managing processes and providing each one with its own virtualized view of the underlying operating system. The main difference with Terra is that processes from different VMs can be viewed on a single screen as if they were operating in the same environment. QubesOS does however provide a mechanism for transferring files between security domains on the machine, partially improving the usability drawbacks of such strong isolation. These two mechanisms use Trusted Platform Module [15–17] hardware to ensure isolation of the closed VMs from all other VMs. Although the startup process is verified via the Trusted Platform Module, it still has the vulnerability to malicious software since software subsequently runs on the guest is not. Storage Capsules [18] aim to adapt virtualization-based solutions like Terra to a more convenient interface for users. Namely, a VMM manages a single untrusted guest operating system, which can switch between secure and insecure modes in order to gain and lose access to sensitive data, which is stored in a secure VM with a communication channel to the guest VM. One important point worth noting is that, even in the event of a compromise of the operating system, capsules are designed to still prevent the leak of data. Capsules [19] also simplify usability while maintaining strong isolation, where users can still access the files available to them in insecure mode and are only unable to access the secure data outside of secure mode.

Virtualization solutions have been around for years in the PC/laptop and server markets. Recent technology advancements have permitted the use of virtualization in mobile devices. As this technology gains acceptance over the next few years, enterprises will have yet another workspace management tool available to them for managing devices in a BYOD world. Mobile virtualization also uses a type-1 or type-2 hypervisor to create virtual work and personal profiles on an employee’s mobile device, which provide greater control and security for the enterprise workspace while the personal area is kept private. Despite obvious similarities between enterprise/desktop virtualization and its mobile counterpart, mobile phone use cases present key differences: smaller memory capacities demand slimmer embedded hypervisor footprints, current mobile processors lack virtualization support in hardware requiring paravirtualization, and hosted guest software span the gamut from enterprise OSEs to embedded RTOSes to stand-alone device drivers. Adding to that, the small form factor and limited battery life impose new usability restrictions. This has motivated research into lighter weight virtualization systems [20–26].

MOSES system [25] presents a policy-based framework for Android that enables the separation and isolation of
applications and data. A crucial one in MOSES is the notion of security profiles. Each security profile represents a unit of isolation enforcing that applications can only access data of the same security profile. MOSES uses a combination of taint propagation, reference monitors, and filesystem namespaces to achieve strong information isolation in a lightweight manner. The policies control the behavior of the reference monitors and taint trackers. Furthermore, the policies can be configured to switch automatically based on the physical context of the device. Cells [26] proposes a lightweight virtual smartphone architecture for Android. Cells introduces a usage model of having one foreground virtual phone and multiple background virtual phones. This model enables a new device namespace mechanism and novel device proxies that integrate with lightweight operating system virtualization to multiplex phone hardware across multiple virtual phones while providing native hardware device performance. Hence, Cells provides the same illusion as traditional desktop virtualization and has similar information confinement guarantees. Information from one virtual phone cannot leak to another. However, any vulnerability in a program that needs real superuser privileges running in a virtual phone will compromise the base kernel. This is a fundamental difference between Cells and traditional virtualization mechanisms that contain separate kernels.

To conclude, virtualization technologies have been mainly provided in network-based high performance environment like cloud computing or server system. Mobile virtualization techniques have focused on limited range such as policy-based framework, running on an identical OS, and switching secure and insecure modes. This still leaves much to be desired. Therefore, it is time for the development of novel mobile security system based on virtualization providing usability to user and lightweight secure schemes optimized by device characteristics are required desperately.

3. The Proposed Scheme

Providing isolation of data and execution through virtualization is a very effective security technique. In this chapter, thus, we describe a system framework for secure storage, which is a way to protect sensitive data of smartphone, based on domain separation on virtualization, and present secure functions to be operated on virtual secure domain.

3.1. System Framework for Secure Storage. The proposed system makes virtual secure domain (SD) separated from general domain (GD) using hypervisor of paravirtualization. SD means a virtual machine created by the virtualization mechanism and novel device proxies that integrate with lightweight operating system virtualization to multiplex phone hardware across multiple virtual phones while providing native hardware device performance. Hence, Cells provides the same illusion as traditional desktop virtualization and has similar information confinement guarantees. Information from one virtual phone cannot leak to another. However, any vulnerability in a program that needs real superuser privileges running in a virtual phone will compromise the base kernel. This is a fundamental difference between Cells and traditional virtualization mechanisms that contain separate kernels.

Figure 1 shows the system framework for secure storage of smartphone. It greatly consists of four layers: application, platform, virtualization, and hardware, and application layer and platform layer are separated into android part of GD and secure part of SD. In platform layer including OSs and middleware, secure platform is run by μC/OS [27] which is an embedded and real-time operating system and is composed of secure functions, while android platform is run by the existing android OS and functions. Also, all user apps can be downloaded in GD and secure service (SS) apps distinguished from personal android apps can be managed by container. SS apps call SS APIs to manage the data of app in SD and only these requests by SS APIs can attempt interdomain communication (IDC) to transmit the data to SD.

IDC is permitted through two authentication steps. One is user authentication using user PIN which is registered by user at the early stage of app use and is stored after encrypting in SD. The other one is app authentication utilizing AppID. A unique AppID assigned to each app makes it possible to block
the access or use of data generated by app A from app B. Since the data sharing between apps may be required to provide flexibility of data access, we solve this problem through policy setting prescribed for access control in SD.

If these authentications are successful, IDC opens up channel and session between domains to message communication. The app requests to reach in SD through IDC are mapped to security functions by SS abstraction. Namely, SS abstraction represents secure APIs in SD to call secure functions. As the secure functions mean the detailed secure features to help data protection, in addition, they include access control, secure filesystem (SFS), encryption, and key generation.

As a result, security functions in SD can be accessed by only authorized SS APIs accessed through IDC. This is because our system does not permit user applications to directly access SS abstraction or security functions in SD since user applications are not allowed in SD. In addition, security functions in SD have mutual relation with each other. SFS calls encryption function to encrypt the file data received from GD, and encryption function calls the key management function to generate the file encryption key. Besides, the access control and authentication are applied in combination with these functions on platform of SD.

In paper [28], authors classify goals of the attacker: container compromise, denial of service, and privilege escalation. Then, they mentioned that these attack groups can be further arranged into two classes based on the type of underlying mechanism: attacks via communication mechanisms and attacks via storage mechanisms. From this classification, they derived a set of security requirements. Thus, we analyze our system through these security requirements. Here, we denote existing android apps by $D_i$ ($i = 1, 2, \ldots, N$) and SS apps by $S_j$ ($j = 1, 2, \ldots, M$). $N$ and $M$ mean the maximum number of apps and $D_i$ and $S_j$ do not permit overlapping.

(i) Separation of processes aims to isolate processes running in distinct containers. $S_j$ of the proposed system are separated from $D_i$ through container, and it allows only $S_j$ in secure container to call the SS API. The container is managed by special user PIN, and use of PIN and AppID prevents the illegitimate access through IDC from GD to SD even if $D_i$ are possible to directly access SS APIs of $S_j$. And most importantly, SD is operated by distinct embedded OS, and it means that processes in SD are completely separated from those of GD.

(ii) IDC isolation is needed to prevent $D_i$ from accessing or modifying data of $S_j$ being transmitted over IDC channels. To prevent illegal access of data in IDC, IDC can perform encrypted communication and message queue for IDC can keep the encrypted messages. Besides, IDC does not allow channel open if user and App authentication fail in modifying message for communication between domains.

(iii) Filesystem isolation is required to prevent illegitimate access to filesystem objects. SFS of our system is located in SD, and it can be only accessed by $S_j$ having legitimate user authority and app authentication information.

(iv) Device isolation should protect device drivers shared between different containers or domains. Our system is based on the paravirtualization, on which hypervisor resides on top of the hardware and operates through a set of low-level routines with the hardware called hypervisor calls. We maintain independence and safety of device drivers in SD on the assumption that the embedded OS of SD can interface with the hypervisor through these hypervisor calls.

(v) Network isolation aims to prevent attacks by $S_j$ via available network interfaces. Since SD does not permit any direct network communication such as app download, system, and security policy, our system is guaranteed safety from attacks through network. Instead, settings for secure platform can be securely supplied from GD.

(vi) Resource management is needed to limit the amount of resources available to each domain depending on the system load. The resources for SD in our system are allocated appropriately considering embedded OS through hypervisor. The closed SD can protect the physical resources available from exhausting works by intentional attacks.

3.2. Secure Functions in Secure Platform. In this subsection, we introduce secure functions run on the secure platform in SD and mention concretely the SFS closely associated with secure storage of data [29, 30]. Secure functions of the proposed system can be classified into three modules: authentication/access control [31, 32], encryption/key management, and SFS. The role and method of functions are as follows.

(i) Authentication and Access Control Module. The proposed system performs access control function by requiring user and app authentications for communication channel open in IDC. First, the unique AppID given to each SS app is utilized to verify whether app has the legal authority to access and to use other secure functions in SD. If AppID is successfully verified, our system identifies user PIN to check whether SS app is being used by legitimate user. This module is linked with SFS module for managing AppID and PIN information. In addition, new secure policies can be set in SD to provide the strict access control between domains, apps, and functions, and policy data should be also managed through SFS.

(ii) Encryption and Key Management Module. All data transmitted from GD is encrypted before storage through SFS. Also, the encrypted data is decrypted when it is called for use for other secure functions. The encryption module can provide various algorithms and operation modes depending on symmetric or asymmetric key. We assume that keys for encryption/decryption are provided on root of trust, which
(iii) Secure Filesystem (SFS) Module. Sensitive data transmitted from GD is managed by SFS. This module is connected with encryption module to request data encryption/decryption and is linked with authentication and access control module to provide information related to authentication or secure policy.

Here, we describe SFS in detail. As shown in Figure 2, SFS consists of four parts divided into volume, bitmap, objects, and files. Volume supports the system information like boot record, and bitmap is used to manage total blocks composed by filesystem. Here, the used block is expressed as “1” in a bit. Objects, which are called inodes in general filesystem, include information to find and access the file. Lastly, files mean the real file data to be stored in the storage. For simplicity of expression in this paper, the scope of volume, bitmap, and objects is referred to as a filesystem metadata, and each object is called a file's metadata.

The secure features for SFS have two aspects as follows.

(i) First is file data protection.

(a) Each file is encrypted by various algorithms and operation modes. The encryption key can be generated by root key of hardware chip and the seed value for generating the encryption key is stored in the corresponding file metadata. Whether the file encryption is applied or not, it is marked to “Crypto flag” field in file metadata.

(b) When a file is recorded, a hash value of each file is generated by hash functions like MAC (Message Authentication Code) or MD5, and it is stored in the corresponding file metadata with length 20 bytes. Also, file integrity is verified whenever the file is read.

(ii) Second is filesystem information protection.

(a) Exposure of filesystem metadata can still threaten the stored file data in attack. Thus, bitmap and objects in filesystem metadata are encrypted to securely protect data in hardware. At this time, volume is excluded for encryption since its values should be used to mount SFS to memory. If the filesystem metadata size imposes loads, nevertheless, some parts of objects can be selectively used for filesystem metadata encryption.

(b) Integrity of SFS is also checked when mounting or remounting. Verification value is generated using bitmap and objects by the hash function at the unmounting point, and it is restored in volume.

4. Experiment Results

We implement the secure storage system for sensitive data of smartphone and measure the performance overhead imposed
### Table 1: The performance of SFS in SD.

| Operations Unit | Data size | 256 bytes | 1 KB | 2 KB | 4 KB |
|------------------|-----------|-----------|------|------|------|
|                  | Write     | Read      | Write | Read | Write | Read |
| None             | µsec      |           |       |      |       |      |
|                  | 3031      | 71        | 3098  | 117  | 3242  | 184  |
|                  | MB/s      |           |       |      |       |      |
|                  | 0.08      | 3.60      | 0.33  | 8.79 | 0.63  | 11.16|
|                  | %         |           |       |      |       |      |
|                  | 100       | 100       | 100   | 100  | 100   | 100  |

| Encryption       | MB/s      |           |       |      |       |      |
|                  |           |           |       |      |       |      |
|                  | 0.08      | 1.20      | 0.27  | 1.43 | 0.47  | 1.55 |
|                  | %         |           |       |      |       |      |
|                  | 105       | 300       | 121   | 613  | 135   | 722  |

| Integrity        | MB/s      |           |       |      |       |      |
|                  |           |           |       |      |       |      |
|                  | 0.06      | 0.55      | 0.26  | 1.03 | 0.43  | 1.18 |
|                  | %         |           |       |      |       |      |
|                  | 133       | 649       | 128   | 855  | 147   | 944  |

| Enc. + Integrity | MB/s      |           |       |      |       |      |
|                  |           |           |       |      |       |      |
|                  | 0.06      | 0.42      | 0.21  | 0.61 | 0.35  | 0.72 |
|                  | %         |           |       |      |       |      |
|                  | 138       | 865       | 159   | 1441 | 182   | 1548 |

by secure features of secure storage system [33]. Our experiment is performed in two sides. One is to measure the performance of SFS in SD; the other is to measure the performance of SFS through overall system based on IDC. Namely, the first way means the overhead caused only by secure features such as encryption/decryption and integrity verification as the yellow arrows shown in Figure 1. The second way means the overhead imposed by IDC and secure functions, while data of SS apps in GD are transmitted to SD and stored by SFS as the black arrows shown in Figure 1.

All experiments are performed on Odroid-Q2 which is an open development platform based on Exynos4412 Prime ARM Cortex-A9 Quad Core 1.6 GHz with 2 GB memory. In addition, we used AES algorithm and counter mode for encryption and used HMAC function for integrity validation. The write, read, and delete operations of SFS are performed iteratively between 100 and 10000 times, and the performance times are measured in microsecond (µsec).

We first evaluated the performance of SFS with 256 B, 1 KB, 2 KB, and 4 KB data in SD as yellow dotted box shown in Figure 1. In Table 1, “none” row shows write and read results of SFS conducted without secure features; the following rows show the results performed only file encryption, only file integrity, and the combination of encryption and integrity, respectively. Basically, write operation takes some time, as our write operation includes sequential search for checking the existence of the identical file name. Thus, the performance time of write operation has a low increasing rate in additional application of secure features while read operation grows steeply. The delete operation with 2 KB data has average 273 µsec (7.49 MB/s), and it is maintained at similar values because it is not affected by secure features. As the amount of data increases, it improves the efficiency of data processing while it spends more processing time.

Figure 3 also shows the performance comparison results of write, read, and delete operations of SFS applied or unapplied secure features through data throughput (MB/sec). Write operation has slight difference under 1 MB/sec between none and secure features. Moreover, delete operation has proportional increasing rates in none and secure features since it is not affected by secure features. However, the performance of read operation is significantly degraded by adding secure features. Therefore, it means that read operation is affected greatly on performance by secure features in comparison with write and delete operations.

Next, we tested the performance time of SFS by overall system based on IDC as blue dotted box shown in Figure 1. Here, we only use 2 KB data. As shown in “none” row of Table 2, the performance time for IDC itself takes a long time fundamentally in these experiments. Thus, the time increasing rate by secure features is the performance based on IDC is relatively low.

Finally, we summarize briefly Tables 1 and 2 mentioned earlier. The values in italic font of Table 3 become reference value for write and read, respectively. The values of MB/s mean the data throughput, and values of ratio show the increasing rates according to the reference value. And to conclude, the performance overhead by secure features has little effect on our system by IDC, while the performance increasing rate by secure features is relatively high.

5. Conclusion
Since sensitive information stored in an insecure manner is vulnerable to theft, the ways to safely store and manage data
have been the focus. Thus, we proposed a system framework to protect sensitive data of smartphone. It provides general domain (GD) and secure domain (SD) in mobile device utilizing domain separation technique of virtualization. Namely, GD means general android execution area, and SD can provide a secure execution environment that runs secure functions to securely manage all data input by secure service apps in GD. These sensitive data by secure service apps can be called through only secure service API and can be transmitted through interdomain communication (IDC). As IDC requires user and app authentications, the secure communication between domains satisfies isolation of filesystem, network, and IDC.

In addition, we suggested the secure functions such as authentication/access control, encryption/key management, and secure filesystem, and especially secure filesystem is discussed as a key function for secure storage. Thus, we evaluated the performance of secure filesystem imposed by security features in SD and by overall system based on IDC. While the SFS imposed a lot of time overhead in SD, the performance based on IDC is almost not affected since IDC consumes a pretty long time for basic communication. It will provide many possibilities about security functions based on virtualization domain.

The target data of our system were small size data set as sensitive private information. As amounts of memory of latest smartphone are occupied by pictures or videos, processing methods for big size data should be considered together in our system. Therefore, we first will enhance scalability and efficiency of our system, and then we will suggest various secure functions which can be combined with our system and describe each security function in detail.

### Conflict of Interests

The authors declare that there is no conflict of interests regarding the publication of this paper.

### Acknowledgment

This work was supported by the ICT R&D program of MSIP/IITP (2014(10043959), development of EAL 4 level military fusion security solution for protecting against unauthorized accesses and ensuring a trusted execution environment in mobile devices).

### References

[1] R. Singh, P. Singh, and M. Duhan, "An effective implementation of security based algorithmic approach in mobile adhoc networks," Human-Centric Computing and Information Sciences, vol. 4, article 7, 4 pages, 2014.

[2] J. Kim, H. Jeong, I. Cho, S. M. Kang, and J. H. Park, "A secure smart-work service model based OpenStack for Cloud computing," Journal of Cluster Computing, vol. 17, no. 3, pp. 691–702, 2014.

[3] Y. Song and Y. Pang, "How to manage cloud risks based on the BMIS model," Journal of Information Processing Systems, vol. 10, no. 1, pp. 132–144, 2014.

[4] A. Crowell, B. H. Ng, E. Fernandes, and A. Prakash, "The confinement problem: 40 years later," Journal of Information Processing Systems, vol. 9, no. 2, pp. 189–204, 2013.

[5] J. P. Shim, D. Mittleman, R. Welke, A. M. French, and J. C. Guo, Bring Your Own Device (BYOD): Current Status, Issues, and Future Directions, AIS Electronic Library, 2013.

[6] K. Rhee, W. Jeon, and D. Won, "Security requirements of a mobile device management system," International Journal of Security and its Applications, vol. 6, no. 2, pp. 353–358, 2012.

[7] T. Garfinkel and A. Warfield, "What virtualization can do for security," The USENIX Magazine, pp. 28–34, 2007.

[8] G. Heiser, "The role of virtualization in embedded systems," in Proceedings of the 1st Workshop on Isolation and Integration in Embedded Systems (IES '08), pp. 11–16, April 2008.

[9] A. Aguiar and F. Hessel, "Embedded systems’ virtualization: the next challenge?" in Proceedings of the 21st IEEE International Symposium on Rapid System Prototyping (RSP '10), pp. 1–7, IEEE, Fairfax, Va, USA, June 2010.

[10] K. Mahajan, A. Makroo, and D. Dahiya, "Round robin with server affinity: a VM load balancing algorithm for cloud based infrastructure," Journal of Information Processing Systems, vol. 9, no. 3, pp. 379–394, 2013.

[11] A. S. Tanenbaum, Modern Operating Systems, Prentice Hall, Upper Saddle River, NJ, USA, 2007.

[12] M. Rosenblum, "The Reincarnation of Virtual Machines," ACM Queue, vol. 2, no. 5, pp. 34–40, 2004.

[13] T. Garfinkel, B. Pfaff, J. Chow, M. Rosenblum, and D. Boneh, "Terra: a virtual machine-based platform for trusted computing," in Proceedings of the 19th ACM Symposium on Operating Systems Principles (SOSP ’03), pp. 193–206, ACM, October 2003.

[14] J. Rutkowska and R. Wojtczuk, "Qubes OS architecture," Tech. Rep., Invisible Things Lab, 2010.
[15] S. Berger, R. Cáceres, K. A. Goldman, R. Perez, R. Sailer, and L. Doorn, "vTPM: virtualizing the trusted platform module," in Proceedings of the 15th Conference on USENIX Security Symposium, vol. 15, article 21, July 2006.

[16] S. Moein, F. Gebali, and I. Traore, "Analysis of covert hardware attacks," Journal of Convergence, vol. 5, no. 3, pp. 26–30, 2014.

[17] K. Hwang and S. Nam, "Near real-time M2M communication for bidirectional AMR systems," Journal of Convergence, vol. 5, no. 2, pp. 1–7, 2014.

[18] K. Borders, E. V. Weele, B. Lau, and A. Prakash, "Protecting confidential data on personal computers with storage capsules," in Proceedings of the USENIX Security Symposium, pp. 367–382, 2009.

[19] P. Maniatis, D. Akhawe, K. Fall, E. Shi, S. McCamant, and D. Song, "Do you know where your data are?: secure data capsules for deployable data protection," in Proceedings of the 13th USENIX Conference on Hot Topics in Operating Systems, p. 22, Berkeley, Calif, USA, 2011.

[20] H. Tuch, C. Laplace, K. C. Barr, and B. Wu, "Block storage virtualization with commodity secure digital cards," ACM SIGPLAN Notices, vol. 47, no. 7, pp. 191–202, 2012.

[21] L. Catuogno, H. Löh, M. Manulis, A. Saeghi, and M. Winandy, "Transparent mobile storage protection in trusted virtual domains," in Proceedings of the 23rd Conference on Large Installation System Administration, pp. 1–14, November 2009.

[22] L. Catuogno, H. Löh, M. Winandy, and A.-R. Saeghi, "A trusted versioning file system for passive mobile storage devices," Journal of Network and Computer Applications, vol. 38, no. 1, pp. 65–75, 2014.

[23] J. Shu, Z. Shen, and W. Xue, "Shield: a stackable secure storage system for file sharing in public storage," Journal of Parallel and Distributed Computing, vol. 74, no. 9, pp. 2872–2883, 2014.

[24] L. Catuogno, A. Dmitrienko, K. Eriksson et al., "Trusted virtual domains—design, implementation and lessons learned," in Proceedings of the 1st International Conference on Trusted Systems, pp. 1–24, December 2009.

[25] G. Russello, M. Conti, B. Crispo, and E. Fernandes, "MOSES: supporting operation modes on smartphones," in Proceedings of the 17th ACM Symposium on Access Control Models and Technologies (SACMAT ’12), pp. 3–12, June 2012.

[26] J. Andrus, C. Dall, A. V. Hof, O. Laadan, and J. Nieh, "Cells: a virtual mobile smartphone architecture," in Proceedings of the 23rd ACM Symposium on Operating Systems Principles (SOSP ’11), pp. 173–187, October 2011.

[27] J. Labrosse, μC/OS the Real-Time Kernel, R&D Publications, Lawrence, Kan, USA, 1992.

[28] E. Reshetova, J. Karhunen, T. Nyman, and N. Asokan, "Security of OS-level virtualization technologies," in Secure IT Systems, Lecture Notes in Computer Science, pp. 77–93, Springer, Berlin, Germany, 2014.

[29] S. M. Diesburg and A.-I. A. Wang, "A survey of confidential data storage and deletion methods," ACM Computing Surveys, vol. 43, no. 1, article no. 2, 2010.

[30] Y. Qin, W. Tong, J. Liu, and Z. Zhu, "SmSD:A smart secure deletion scheme for SSDs," Journal of Convergence, vol. 4, no. 4, pp. 30–35, 2013.

[31] T. Truong, M. Tran, and A. Duong, "Improvement of the more efficient and secure ID-based remote mutual authentication with key agreement scheme for mobile devices on ECC," Journal of Convergence, vol. 3, no. 2, pp. 1–10, 2012.
