TEE-Based Trusted Storage

Javier González
Philippe Bonnet

IT University Technical Report Series **TR-2014-TR-2014-172**

ISSN 1600–6100 **February 2014**

ISSN 1600–6100

ISBN 978-87-7949-310-0

Copies may be obtained by contacting:

IT University of Copenhagen
Rued Langgaards Vej 7
DK-2300 Copenhagen S
Denmark

Telephone: +45 72 18 50 00
Telefax: +45 72 18 50 01
Web `www.itu.dk`

TEE-Based Trusted Storage

Abstract

Today, it is safe to assume that any program or data can be compromised, if they are not protected by hardware within a secure area. Systems based on crypto-processors (e.g., a trusted platform module, a smart card or a hardware security module) rely on the properties of tamper resistant hardware to establish a tight security parameter around a reduced set of predefined functionalities. Such systems are very secure, but they impose strong constraints on the functionalities, the connectivity or the resources available within the secure area. They have not proven versatile enough to provide mainstream trusted storage for personal data. We believe that this role can be taken over by systems equipped with Trusted Execution Environments (TEE), such as ARM's TrustZone. Indeed, even if TEEs provide weaker security guarantees than crypto-processors, they already provide a secure area on many personal devices. With the advent of programming frameworks for TEEs, interesting services can now be provided on top of a basic encryption/decryption service. In this paper, we describe our implementation of a trusted storage service within a TEE, we analyze its security and show that it can scale. We derive our design from a set of general principles for TEE-Based trusted storage, which we believe constitutes a promising avenue for future research.

1 Introduction

As the US department of Homeland Security explains on its home page: *Cyber intrusions and attacks have increased dramatically over the last decade, exposing sensitive personal and business information, disrupting critical operations, and imposing high costs on the economy*[1]. Put differently, it is commonplace for intruders to obtain the identity or rights of the administrator of any computer. As a consequence, it is safe to assume that any data or program can be compromised on the server side, on the client side or on any third-party site. This assumption is not new. It was already clearly formulated more than ten years ago [9]. Its far reaching consequences have been discussed thoroughly, most notably by Cory Doctorow [2]. And even if it has been discussed for years, trusted computing in general, and trusted storage in particular, are not mainstream on personal devices. Today, the emergence of programmable trusted execution environments embedded on smart phones, tablets or laptops, makes it possible to envision a widespread deployment of trusted storage solutions on personal devices. In this paper, we focus on the potential of TEE-Based trusted storage, and we propose an initial building block.

It is generally accepted that a system providing trusted storage should guarantee data *confidentiality, integrity, availability, and durability* [4, 9]. Today's security policies rely on data encryption to support confidentiality and integrity. However, if encryption keys can be compromised or stolen on the client computer, then there is not much protection left. A solution is to rely on a secure crypto-processor (e.g., a trusted platform module, a smart card or a hardware security module) to provide encryption and decryption services. Such devices provide strong guarantees based on tamper resistant hardware. Here, the encryption keys never leave the secure area. On the downside, these devices have strong limitations in terms of the functionality that they embed, and in terms of the amount of data that they can store. As importantly, these devices are physically separated from the client computer; they are connected via a narrow client-server interface. As a consequence, the client computer can neither be monitored nor controlled from the secure area. For example, a smart card equipped with an embedded database, can securely manage up to 1GB of data. However, running a machine learning algorithm on this data requires that the data is moved to the client computer outside of the crypto-processor secure area. Also, any replication of the encrypted data on the cloud will rely on a program running on the client computer to copy the encrypted data onto the cloud.

[1] `https://www.dhs.gov/cybersecurity-overview`

An alternative is to rely on a trusted execution environment (TEE) to encapsulate encryption and decryption services. According to the GlobalPlatform consortium[2], a TEE is a secure area of a computing device that ensures that sensitive data is only stored, processed and protected by authorized software. The secure area is separated by hardware from the device's main operating system and applications, which is denoted rich area. Rich and secure areas share the same processor. Trusted execution environments embedded on personal devices are ideal to support trusted storage systems for personal data. Indeed, personal data is largely produced and consumed via mobile devices or set-top box and gateways at home. The rich area provides the environment for innovative services and mobile apps that rely on trusted storage primitives. Authorized storage components running in the secure area can encrypt data and store keys in a tamper-resistant chip (e.g., smart card), thus guaranteeing data confidentiality. They can verify that the encrypted data that has been stored corresponds to the data that was written, thus guaranteeing data integrity. They can replicate data stored locally on several remote instances (e.g., on the cloud), thus providing availability and durability.

An example of TEE is ARM's TrustZone[3]. TrustZone relies on the so-called NS bit, an extension of the AMBA3 AXI Advanced Peripheral Bus (APS), to separate rich and secure areas. The NS bit distinguishes those instructions stemming from the secure area and those stemming from the rich area. Access to the NS bit is protected by a gatekeeper mechanism in the Operating System. The operating system thus distinguishes between user space, kernel space and secure space. Only authorized software is running in secure space, without interference from user or kernel space. While TrustZone was introduced 10 years ago; it is first recently that Trustonic, Xilinx, SierraWare and others have proposed hardware platforms and programming frameworks that make it possible for the research community [3] as well as industry to experiment and develop innovative solutions with TrustZone.

TrustZone defines a communication abstraction for the interaction between programs running in the rich area that act as clients, and programs running in the secure area that act as servers. This client-server communication is session-based (each session is bound to programs in the rich area); no state is kept in the secure area across sessions (any state must be explicitly stored in memory or on secondary storage). Unlike secure crypto-processors, TEEs make it possible to (i) run complex authorized software on the secure area (e.g., a machine learning algorithm); (ii) to access to all the computer's peripherals from the secure area, thus enabling secure interactions between users and programs running in the TEE's secure area; and (iii) to monitor and control the rich area from the secure area. We present in Section 2 an example of how a TEE could be leveraged for trusted storage.

TrustZone does not offer tamper resistance or even tamper evidence. The assumption here should be that with enough time, money and expertise and attacker could (i) steal the secrets in the secure area by means of a sophisticated physical attack, (ii) get legitimate access to the secure area by obtaining control of the gatekeeper by means of a sophisticated software attack, and (iii) having control over the rich area, and attacker could use legitimate interfaces to establish a communication with programs running in the secure area. A successful physical attack would result on all secrets stored or handled by the secure area being exposed. However, the result of a successful software attack depends on how the relation between rich and secure areas is defined.If a program running in the secure area is able to react against the suspicion of a threat, it could take measures to logically isolate itself, thus giving up on availability but still maintaining data integrity, confidentiality and durability.

In this paper, we illustrate the potential benefits of TEE-based trusted storage and we compare our approach with existing work. We then report on the design of a trusted storage module (TSM) for TrustZone. We analyze its security and show that it can scale. We derive our design from a set of general principles for TEE-Based trusted storage. While preliminary, we believe that our work shows that the advent of programmable trusted execution environments can open up new opportunities for the research community.

2 Application Tiering

Here are a couple of examples of how our TSM module could be used for trusted storage, followed by a discussion of how we envisage application tiering on top of TEE-based trusted storage.

[2] http://www.globalplatform.org/mediaguidetee.asp

[3] http://www.arm.com/products/processors/technologies/trustzone/index.php

2.1 Password Management

Existing software, such as 1Password, LastPass or KeePass, allow users to generate one semi-random strong password for a range of services. They also allow to store personal information such as credit cards and passport numbers. All passwords are stored in one encrypted file that is then stored in the user's file system. The key to decrypt that file is a master password that the user needs to remember. There are three issues with the current incarnation of these applications in terms of privacy: passwords accessed at runtime, plain master password protection and direct encryption. First, the file containing all passwords is decrypted and located in memory whenever a password is accessed at run-time. This makes it possible for an attacker to run a piece of software that gets the decrypted file from memory, therefore exposing all the contents of the file. TSM solves this issue by design as the encrypted file never leaves the secure area. Second, the master password is necessarily another password that the user knows. If the password is not strong enough, then a brute force attack would be feasible; if the password is lost then all the encrypted contents are lost. The TSM could rely on other methods of authentication such as a smart card, fingerprint or biometric sensors, or a combination of them to authenticate a user. Since these sensors would be only accessed form the secure environment (Principle 3 in Section 4.1), the possibility of external applications snooping or modifying the sensor interaction with the software carrying out the authentication is eliminated. Third, the file containing all passwords and other personal information is directly encrypted using the master password. If that master password is exposed (either overheard in a conversation or brute forced) an attacker could remotely access the encrypted file and access its contents without the user even noticing. What is more, most of the applications mentioned above provide a method for the encrypted file containing all passwords and personal information to be shared along different devices using a cloud service (e.g., iCloud, Dropbox. etc.), increasing the chances of an attack. With TSM the file is encrypted with a key managed by the TSM, within the secure environment. An attacker would have to either break a symmetric key to access the file, or hack into the user's system and introduce the user's master password. This intrusion, even when possible, would leave a log of last access to the application stored in the secure environment, giving the user the chance to at least notice the attack and act accordingly (i.e., change passwords, call the bank, etc.).

2.2 Rich and Secure

Trusted execution environments embedded on personal devices are ideal to support trusted storage systems for personal data. The question is how to split the processing that is to take place on decrypted data between rich and secure areas. All the data that leaves the secure area to be manipulated by complex algorithms in the rich area is at risk of being compromised or stolen. This is why we have proposed in [1] that a usage control framework should be incorporated within the secure area. Ideally, the rich area is equipped with a sandboxing framework which is tied with the secure area, so that the side effects linked to data usage (e.g., screenshots, local storage or communication on the network) can be controlled from the secure area. Alternatively, authorized secure tasks can be installed in the secure area to operate directly on decrypted data. This way, only higher level data products are exposed to the rich area. Note that Trustonic[4] supports the virtualization of TrustZone's secure area so that different tasks can be executed concurrently and safely in the secure area. Our TSM could be a basic service in such an environment. Exploring these design alternatives is beyond the scope of this paper; it is a topic for future work. Before we focus on the storage primitives designed for the secure area of a TEE, let us review how our approach compares to existing work.

3 Related Work

Given their nature, storage systems have been the focus of a great deal of attacks, and the center of major scandals (e.g., Google loosing Gmail data[5]). Threat models have been specifically defined for storage systems [4]. Existing work has evaluated the security of storage systems [9], and techniques have been propose to enhance it [5] [11] [10]. These works have two aspects in common. First, they all agree that a trusted storage system should provide data integrity, availability, durability and confidentiality. Second, it is generally accepted that *"Designing protection for storage*

[4] http://www.trustonic.com

[5] http://news.cnet.com/8301-1023_3-20037554-93.html

systems is best done by utilizing proactive system engineering rather than reacting with ad hoc countermeasures to the latest attack du jour." [4] .

In 2002, Ganger et al. [9], introduced the notion of self-securing storage devices based on the insight that storage devices should view hosts and their users, as *questionable entities for which they work*. In this paper, we adopt this assumption. Ganger et al. defined a security perimeter for self-securing storage devices as a form of Hardware Security Module: *the security perimeter consist of self-contained software that exports only a simple storage interface to the outside world and verifies each command's integrity before processing it*. The authors evaluated their solution in the context of a networked file system. In this paper, we define the basic building block (i.e. the trusted storage module) that could enable embedding self-securing storage devices within personal devices.

In [8], Ganger et al. proposed host-based intrusion detection systems (HIDS). The goal there is to detect intrusions in a system by observing sequences of system calls [6], patterns [7] or storage activity [8]. The main issue for host intrusion detection is that a smart attacker could feed the HIDS component with false information about the system and therefore bridge its security. Approaches such as [8] where the storage activity is monitored by looking at system primitives allow to decouple the OS processes from the processes running on the storage device. Illegitimate accesses can be detected even when the OS is compromised. This approach assumes, however, that the storage device and the console to which the intrusion alerts are sent to, cannot be compromised. But it is leap of faith. When an intrusion is detected the HIDS component notifies the administrator of the system, and in the best case enforces some rigid policies (e.g., stop the activity of the user). As a result, users might be banned in case of false positive (the paper mentioned that it occurs very often). With our approach the security is driven by the secure environment. Put differently, we provide a concrete means to protect the storage system. A HIDS component implemented on top of our TSM could implement a machine learning algorithm that classifies the attacks and learn from the traces (who launches the attack, when, how the process got installed, by whom, who gave it enough privileges, etc). Such analysis is possible because (a) the TEE allows for complex logic to be embedded in the secure area, and (b) the TEE's secure area has complete view of the client beyond storage primitives.

4 Trusted Storage Module

4.1 Design Principles

We conceive a TEE as a secure area that distrusts its environment. Based on this basic observation, we define three principles for the design of authorized software for the secure area:

Principle 1: Self preservation first Under the suspicion of a threat, the secure environment isolates itself logically and gives up availability in order to protect data integrity, confidentiality and durability. In addition to damage contention, this principle ensures that the logs contained in the secure environment cannot be tampered with (unless a successful hardware attack is perpetrated), therefore helping a post analysis of the threat.

Principle 2: Lead all communications Even though the secure environment does not initiate a communication, it defines all parameters that define this communication: protocol, certificates, encryption keys, etc. Following this principle, the secure environment interprets each message coming from the rich area and establishes communication under its own terms.

Principle 3: Secure all interactions Much of the input outside the security perimeter comes from users. In this way, the secure environment needs to be sure that any interaction stemming from the richa area is legitimate: the user responsible for the interaction should be authenticated and no man-in-the-middle attack should take place. As a consequence, the secure area should have priority to obtain exclusive access to secure peripherals.

4.2 Communication Abstraction

From a design point of view, each secure module can be defined as a black box exposing a narrow communication interface. This communication interface is used by rich applications to request the execution of a secure task (i.e., a program running in the secure area). Rich applications are however not guaranteed to receive the services they request since it is always the secure environment that decides how to follow-up on the interactions initiated from the rich area (Principle 2).

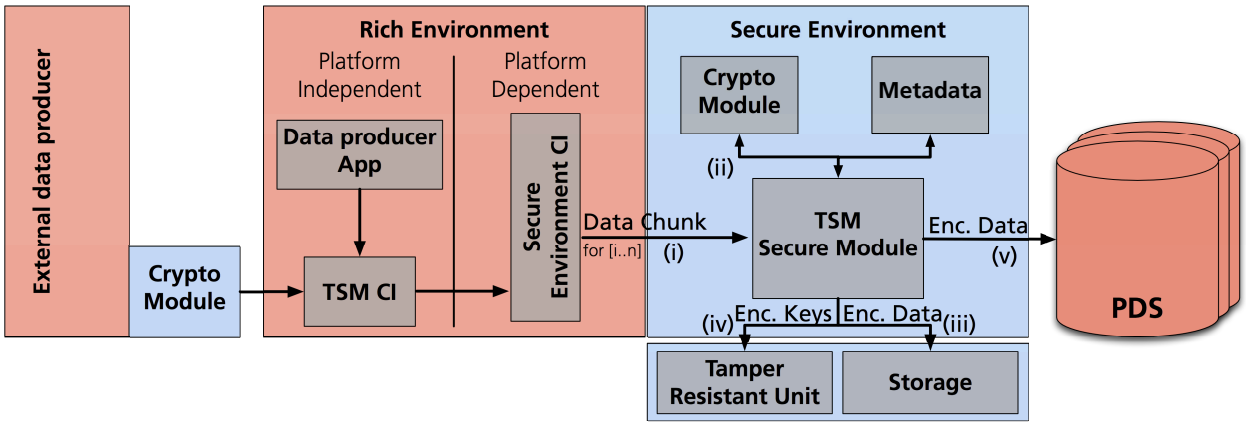

Figure 1: TSM architecture. Red: Rich environment and untrusted components. Blue: Secure environment and trusted components. CI stands for Communication Interface.

The TSM is thus split in two parts: the *TSM communication interface* used by rich applications, and the *TSM secure module* that takes care of encryption and storage within the secure environment. With TrustZone, certain parts of the memory are set to allow sending parameters in both directions. The maximum payload for these parameters can vary, introducing a per-call size restriction. To deal with this limitation, we introduce a third component that partitions the payload sent between the rich and the secure environment in chunks of the maximum manageable size. This component affects neither the TSM operation, nor its implementation.

The TSM optionally replicates encrypted data on a remote data server. Such a remote data server implements a protocol whose definition is not relevant for this paper. A remote data server is not part of the secure area, so it is likely that the availability, durability and confidentiality of such a server can be compromised. However, the integrity is guaranteed as long as the TSM that acts as a replication master is operational.

The unit of storage in the TSM is an object. An object is a stream of bytes of a determined size that is to be securely written on persistent storage. From the point of view of a rich application, the process of storing an object is reduced to: creating an object, storing an object, and receiving the acknowledgment that the object has successfully being stored. This corresponds to the following sequence of operations (illustrated in Figure 1): (i) the stream of bytes forming the object is partitioned in chunks of the maximum size that can be shared between the rich and the secure environment. This task is carried out by a component located in the rich environment, which as mentioned above, is implementation specific. When the data chunks are received by the TSM in the secure environment, (ii) each chunk is encrypted with a newly generated symmetric key and (iii) stored locally. The same key is used to encrypt all the chunks that form an object. When all the bytes forming the object are encrypted and stored, (iv) the key is stored in a tamper resistant unit only accessible from the secure environment. At that point, the acknowledgment that the object has successfully been stored is sent to the rich application. Finally, (v) the TSM sends the encrypted object to a remote data server, if one has been indicated by the user. For the rich application, the encryption process, key handling, and metadata handling is transparent, just as if the object had been directly stored in the rich environment. Note that any piece of software running under the secure environment must be certified, hence trusted tasks trust each other. If this requirement cannot be met then each secure module must have its individual tamper resistant storage unit to store its encryption keys. Since the TSM is a storage manager in itself and therefore transversal to all rich applications and secure tasks, it only needs access to one of the tamper resistant storage units.

4.3 Security Analysis

There are three types of attacks that can be perpetrated against the TSM: (i) software attacks against the rich - secure communication interface, (ii) software attacks against the secure user interaction interface, and (iii) hardware attacks against the tamper resistant hardware storing the keys, and TrustZone itself.

Most of the interaction that the secure environment has outside the security perimeter is through the the rich

environment. Hence the communication interface between the rich and the secure environments is a point subject to attacks. The TSM interface exposes the object name space and maps it to secure data, the name space is exposed to rich, while the mapping and the secure data remain secure - it is impossible for an attacker to build the mapping or the secure data from the exposed name space. Following Principle 1, and assuming that an attacker could supplant the identity of a legitimate user, no data is deleted or replaced immediately, rather objects are marked as deleted or updated and both metadata and data are stored locally and remotely, as long as there is space left. An attacker could still try to brute-force the TSM interface. However, following Principle 1 again, the TSM would isolate itself under the suspicion of an attack. It is then up to the rules under which the TSM and the secure environment in general operate, to contemplate a massive use of the interface as a possible thread. Evaluating threats and learning from them to prevent future attacks from the secure environment is work in progress.

Attacks to the secure user interaction interface can fall into two categories: supplanting of a legitimate user and man-in-the middle between the user and the trusted task using the TSM. Although user authentication is out of the scope of this paper, in Section 2.1 we propose the use of biometric or fingerprint sensors that, handled by trusted tasks, can validate a user. Since peripherals can be acquired on the fly by secure tasks, then the overall security relies on the tamper resistance of the hardware. When a secure task is using a peripheral to interact with a user, the peripheral is only visible from within the secure environment (Principle 3). Assuming that the sensor does not fail, if it authenticates a user then the secure task can be sure that the user is not being supplanted. The same argument applies for man-in-the-middle attacks.

Finally, TrustZone is not tamper resistant, which means that a hardware attack is feasible. For example, in our current implementation, the upper boundary to which the TSM can resist software attacks is indeed the level of tamper resistance leveraged by the Zynq-7000 series implementation of TrustZone. Bypassing the hardware would mean the collapse of the TSM security. Regarding the encryption keys, the upper boundary to which they can be safe is again, the level of tamper resistance of the hardware in which they are stored. Since the keys are stored in a secure element, hardware attacks need to be very advanced to succeed. The assumption should be that with enough time and money any security system can be bypassed. Nonetheless, we consider that client devices such as smart phones, tablets or laptops offer an intrinsic level of tamper evidence, giving users the chance to minimize the effects of a successful hardware attack.

In case of a failure in the secure environment, the effect on rich applications is the same as a system call failure. If the TSM crashes, the call issued inside the secure environment will not be catched and an error will be returned; if the whole TEE crashes, the system call issued by the gatekeeper in order to switch contexts will not be catched either, and a similar error will be returned.

4.4 Scalability

In this section we evaluate the overhead of writing or reading an object with TSM. For this experiment we are using a Global Platform compliant TEE implemented on top of ARM's TrustZone, i.e., the framework presented in [3]. The experiment consists on measuring the overhead of storing an object using the TSM implemented in the secure environment with respect to the TSM implemented in the rich environment. Since in our implementation of TrustZone both environments are running in the same processor, using the TSM in the secure environment entails: (i) initiating a Context for the communication in the rich environment, (ii) opening a Session[6], (iii) creating the payload and allocate shared memory for the rich - secure communication, (iv) loading the communication parameters and request the execution of the secure task associated with the TSM, (v) evaluating the system call and switch context to run in secure environment mode, (vi) recovering the parameters sent from rich, (vii) encrypting the object using OpenSSL and storing it, (viii) loading the return parameters and switching context to run in rich environment mode, and finally (ix) recovering the parameters and closing the Session and Context. Using the TSM in the rich environment only entails: (i) creating the payload and (ii) encrypting the object using OpenSSL and storing it.

Figure 2 shows the overhead of using the secure environment. The interesting part is that the overhead is linear with respect to the size of the shared memory. This is explained because once a session is initialized and the shared memory is allocated, the cost of switching contexts is minimal. For the processor this context switch is not much different from

[6]The concepts of Context and Session are logical abstractions defined in the Client TEE API Specification which can be found at `http://www.globalplatform.org/specificationsdevice.asp`

Figure 2: Storing an object with TSM in the rich and in the secure environments. Hardware: Xilinx Zynq-7000 series (ZC702).

the normal user space / kernel space context switch where registers and memory are saved and loaded in runtime. The result is that the cost of storing arbitrarily large objects does not increase when the objects are partitioned in chunks of manageable size, as explained in Section 4.2. The overhead of using the TSM inside the secure environment using Xilinx's TrustZone implementation is minimal. The specific processes that TrustZone systems execute and the hardware support for them can be read in detail in the TrustZone white paper[12].

5 Conclusion

In this paper, we describe the design of TSM, a TEE-based trusted storage module. We describe a set of principles from which our design derives and show that our solution scales with the size of the objects stored in the secure area. The TSM is the primary building block for advanced trusted storage solutions that can be embedded on personal devices. While much work still need to be done to clarify how advanced trusted storage solutions can take its place in the nascent TEE software ecosystem, we consider that TSM is a necessary first step. We anticipate that a lot of the recent obtained for self-securing devices can be adapted and expanded on top of TSM. We also consider that the development of usage control models in secure area, on top of TSM is a great topic for future work.

References

[1] N. Anciaux, P. Bonnet, L. Bouganim, B. Nguyen, I. Sandu Popa, and P. Pucheral. Trusted cells: A sea change for personal data services. *CIDR*, 2013.

[2] C. Doctorow. Lockdown, the coming war on general-purpose computing.

[3] J. González and P. Bonnet. Towards an open framework leveraging a trusted execution environment. In *Cyberspace Safety and Security*. Springer, 2013.

[4] R. Hasan, S. Myagmar, A. J. Lee, and W. Yurcik. Toward a threat model for storage systems. In *StorageSS '05*, pages 94–102, New York, NY, USA, 2005. ACM.

[5] E. Haubert, J. Tucek, L. Brumbaugh, and W. Yurcik. Tamper-resistant storage techniques for multimedia systems. In *Storage and Retrieval Methods and Applications for Multimedia*, volume 5682 of *SPIE Proceedings*, pages 30–40. SPIE, 2005.

[6] S. Hofmeyr, S. Forrest, and A. Somayaji. Intrusion detection using sequences of system calls. *Journal of Computer Security*, 1998.

[7] S. Mukkamala, G. Janoski, and A. Sung. Intrusion detection using neural networks and support vector machines. *Neuronal Networks, IJCNN'02*, 2002.

[8] A. G. Pennington, J. L. Griffin, J. S. Bucy, J. D. Strunk, and G. R. Ganger. Storage-based intrusion detection. *ACM Trans. Inf. Syst. Secur.*, 13(4):30:1–30:27, Dec. 2010.

[9] E. Riedel, E. Riedel, M. Kallahalla, and R. Swaminathan. A framework for evaluating storage system security. *IN FAST '02*, pages 15–30, 2002.

[10] G. Sivathanu, C. P. Wright, and E. Zadok. Ensuring data integrity in storage: Techniques and applications. In *StorageSS '05*, pages 26–36, New York, NY, USA, 2005. ACM.

[11] J. D. Strunk, G. R. Goodson, M. L. Scheinholtz, C. A. N. Soules, and G. R. Ganger. Self-securing storage: Protecting data in compromised system. In *OSDI'00*, pages 12–12, Berkeley, CA, USA, 2000. USENIX Association.

[12] A. S. Technology. Buiding a secure system using trustzone technology. Technical report, ARM, 2009.