Playing Attack and Defense with Trusted Storage

Javier González - jgon@itu.dk
Philippe Bonnet - phbo@itu.dk
Luc Luc Bouganim - Luc.Bouganim@inria.fr

IT University Technical Report Series **TR-2014-TR 2014-174**

ISSN 1600–6100 **February 2014**

ISSN 1600–6100

ISBN 978-87-7949-312-4

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**

Playing Attack and Defense with Trusted Storage

Abstract

It is often convenient to assume in a data management platform that one or several computing devices are trusted, specially when the goal is to provide privacy guarantees over personal data. But what does it take for a computing device to be trusted? More specifically, how can a personal device provide trusted storage? This is the question we tackle in this demonstration. We describe how secure devices, equipped with a trusted execution environment, differ from general purpose devices. We illustrate with our demonstration scenario, that it is much more difficult to attack a storage service running on a secure device, than to attack the same service running on a general purpose device.

1 Introduction

In [1], we proposed the vision of a decentralized data platform based on personal data servers, denoted trusted cells, running on personal devices equipped with trusted execution environments at the edges of the Internet. A cloud infrastructure provides storage, computing and communication services, thus expanding the resources of a single trusted cell and providing the glue between trusted cells. By definition, the infrastructure does not benefit from the hardware security of the trusted cell and is therefore considered untrusted. Such a distinction between trusted and untrusted components is at the heart of a range of data platforms aiming to provide security and privacy [7]. But what does it take for a computing device to be trusted? What kind of attacks can be deflected if a data server relies on a trusted execution environment to protect its sensitive data? What is a trusted execution environment in the first place? These are the questions we tackle in this demonstration.

2 Trusted Execution Environment

According to the GlobalPlatform consortium[1], a trusted execution environment (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. Put differently, computing devices fall in two categories: (i) *general purpose* devices that do not provide any guarantee for authorized software, and (ii) *secure devices*, where authorized software can execute in a secure area which is not accessible by the rest of the system. Secure devices separate a secure area from a non-secure area, which we denote rich area in the rest of this paper, following the terminology from the GlobalPlatform consortium, while general purpose computing devices merely provide a rich area.

A potential large number of applications are expected to be installed and run concurrently in the rich area, therefore sharing software and hardware resources (e.g., libraries, drivers, peripherals, memory, etc.). As in any modern rich execution environment, access control is in effect to protect shared resources, with a user space distinct from a kernel space. Software running in kernel space has access to all resources (e.g., data and programs in memory or on secondary storage). The rich area is in constant exposure to the outer world, thus attacks are to be expected. The assumption should be that with enough time and expertise an attacker can obtain root access to kernel space. As a consequence, all programs and data stored in or handled in the rich area are at risk of being exposed, including sensitive data and encryption keys.

[1] `http://www.globalplatform.org/mediaguidetee.asp`

As we stated above, a secure device provides a hardware separation between the secure area and the rich area. Note that this is a slight generalization of GlobalPlatform's definition, which mentions that rich and secure areas should run on a same processor. We extend this definition so that a secure co-processor, or a smart card are considered trusted execution environments. In fact, we consider that any device equipped with a processor for the rich area, and another one for the secure area is a secure device as long as there is a guarantee that only authorized software runs in the secure area. We even consider that a virtual machine (VM) is a trusted execution environment as long as the hypervisor relies on hardware primitives to isolate each VM. Defining an ontology of these secure devices and precisely qualifying how secure they are is beyond the scope of this demonstration. It is a topic for future work.

Whenever rich and secure areas share physical memory, it is mandatory that software running in the rich area by no means is able to access memory allocated to the secure area. If peripherals are shared Secure must have prioritized access to them in such a way that if the rich area is compromised, the peripheral can still be accessed from the secure area even when a DoS attack is launched against it. More generally, the only assumption made in the secure area is that the authorized code that is run there can be trusted to protect the integrity and confidentiality of the data. This is a big assumption, but model-based development and formal methods can provide interesting guarantees. Again, exploring how model-based development can be leveraged in the context of secure data management is a fascinating topic for future work.

In this demonstration, our trusted execution environment relies on ARM TrustZone[2]. In [3] we presented a framework that combines commercially available hardware and open source software and enables the development of applications for TrustZone's secure area. In this framework, rich and secure areas are running on a single processor. TrustZone provides a hardware mechanism to separate the secure and rich areas. This mechanism relies on the so-called NS bit, an extension of the AMBA3 AXI Advanced Peripheral Bus (APS), a peripheral bus that is attached to the system bus using an AXI-to-APB-bridge. 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. 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). TruztZone is implemented in ARM popular processors: Cortex-A9 and Cortex-A15 (e.g., powering platforms such as Samsung Exynos and Nvidia Tegra series). More detailed information about TrustZone can be found in the TrustZone white paper [11]. The description of the platform (hardware, software combination) that we are using for this Trusted Cell prototype can be found in [3].

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.

Here are the most relevant successful attack perpetrated against TrustZone. In October 2013, TrustZone's implementation for a set of Motorola devices running Android 4.1.2 was bypassed. The attack reached the National Vulnerability Database and scored an impact of 10.0 (out of 10.0)[3]. The attack was possible because of a flaw in the communication interface between rich and secure areas. Indeed a function executing in the secure area was communicating physical addresses from the secure area back to the rich area. This permitted the attacker to access a specific flag in the secure area and to overwrite its value. The attacker was thus granted access to the bootloader and he could consequently obtaining full control of the system. The full attack is explained in the attacker's blog[4]. This attack

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

[3] http://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2013-3051

[4] http://blog.azimuthsecurity.com/2013/04/unlocking-motorola-bootloader.html

could have been avoided by keeping the mapping from physical to logical addresses within the secure area. It is vital to define an appropriate communication interface when designing secure applications. In this demonstration, we focus on trusted storage.

3 Trusted Storage

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 [5]. Existing work has evaluated the security of storage systems [8], and techniques have been propose to enhance it [6] [10] [9]. 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 systems is best done by utilizing proactive system engineering rather than reacting with ad hoc countermeasures to the latest attack du jour."* [5] .

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. Note that availability and durability come at the cost of performance. Exploring this trade-off is a topic for future work. In this demonstration, we illustrate how we achieve data integrity and confidentiality on top of TrustZone.

4 Home Energy Management

In this demonstration, we focus on a specific type of personal data, i.e., the smart meter data that powers efficient and flexible home energy management. Smart meters are being rolled out so that utilities can augment their ability to monitor and control their distribution infrastructure. Increasingly, smart meter data is made available to users at home either directly from the smart meter, or via digital electric boards. Smart meter data can thus be integrated with home automation systems, energy aggregators or online social games that generally require data to be transmitted to cloud-based services[6].

Most electrical appliances have a distinctive energy signature. It is possible to infer from smart meter data which appliances were used, and thus which activities people were involved in at specific points in time [12]. This is called load disaggregation, which is important to enable flexible energy management (as it makes possible for innovative services to recommend how activities could be adapted to reduce energy consumption or to match the available green energy. We consider however that it is very important that people retain control over how they share information about their activities at home!

The general setup to connect a smart meter to the cloud involves the use of local gateway. Many existing gateways are proprietary.But, they could also be open source. The emergence of cheap boards working out-of-the-box such as the Raspberry Pi[7], equipped with open energy management software (e.g., smartmeter[8], measureit[9], and online communities[4] makes it increasingly easy to assemble a custom home energy management solution. It has been shown that, whether proprietary or custom, gateways are the weak link for privacy-preserving smart metering [2]. The question is whether you can trust that a gateway will not leak your smart data to third parties? What does it take for a program running on the gateway to leak smart meter data? What does it take to provide trusted storage for smart meter data?

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

[6] This work is done in the context of the PDS4NRJ project: `http://www.pds4nrj.eu/`

[7] `http://www.raspberrypi.org`

[8] `https://github.com/biemond/smartmeter`

[9] `http://code.google.com/p/measureit/wiki/RaspberryPi`

We consider two scenarios. In scenario (1), the gateway forwards the time series (TS) generated by the smart meter to the cloud, where a service is in charge of load disaggregation. In scenario (2), it is the gateway that carries out the load disaggregation and encrypts the data before passing it on to the cloud, which is used as redundant storage. In this scenario the cloud is untrusted. Applications using disaggregated data need to establish a connection to the local gateway in order to be granted access to the views they need to provide their services.

While scenario (1) is the current choice for many *do-it-yourself* enthusiasts, it is obvious that successful attacks are easy to conceive that compromise the integrity and confidentiality of the data; not to mention illegitimate access from the cloud services providers themselves[10]. Centralized attacks to the cloud would expose all stored data. Since data is not encrypted and a potential large number of users store their data in this way, the revenue of elaborating an attack of this magnitude would be worth the time and money. Also, attacks to the gateway can be designed. There exists plenty of *rootkits* (e.g., WAT[11], Aircrack[12]) that make this a simple task. These kits explode the default configuration of many classes of devices when connected to the Internet: default passwords, open ports, remote root login, permissions, etc.

That leaves us with scenario (2). This solution is however not bullet proof, and attacks to the gateway are still feasible. In our demonstration scenario, we play attack and defense on a gateway, showing how our implementation of a trusted storage on top of a trusted execution environment allows to protect data integrity and confidentiality.

5 Demo Scenario

We focus on software attacks against the gateway in the scenario (2)) described above, i.e., the target of the attacks is a computing device that runs a program (load disaggregation) on a time series (smart meter data). We will assume that the attacker has *root* access to the rich area of the device. We consider two forms of attacks: attacks whose goal is to steal or compromise data (SC attack) and denial of service (DoS). DoS is possible for the load disaggregation (DoS LD attack) by impeding processing on the gateway, and for the archival (DoS Arch attack) by impeding gateway - cloud communication.

The idea is to show how a general purpose device (Raspberry Pi) is susceptible to all three attacks while a secure device, equipped with a trusted execution environment (Xilinx Zynq-7000) can resist SC and DoS Arch attacks[13]. Both devices run Linux.

Figure 1 illustrates the demo scenario. Each device receives an encrypted data stream (via a local Ethernet connection) from a smart meter. This data stream is decrypted (using the smart meter key). The decrypted data stream is then encrypted again with a local key and stored on secondary storage together with a checksum. The decrypted data stream is also fed to the continuous processing component that implements a simple form of load disaggregation and displays the result [14]. Concurrently, the encrypted data stream stored on secondary storage is read, decrypted and its checksum is checked to control its integrity. The result of the integrity check is visualized from the rich area.

The demonstration consists of four phases. In a first phase, data streams are received, processed and stored on both devices as expected. We show the positive result of the integrity checks. In a second phase, we ask the attendee to formulate an attack on either device, based on a range of building blocks commonly available on Linux (scanmem, gameconqueror, ldd, shared library overload with LD_PRELOAD) and some custom scripts. We provide attendees with a short description of these tools and scripts; we even provide ready-made attacks for busy attendees. In a third phase, we actually proceed to the attack designed in the previous phase and observe whether it succeeds in compromising the integrity or confidentiality of the data stream. In the last phase, we try and apply a similar attack on the other device. Attendees thus experience how both devices react to a specific attack.

[10] `http://edition.cnn.com/2010/TECH/web/09/15/google.privacy.firing/`

[11] `http://sourceforge.net/projects/piwat/`

[12] `http://aircrack-ng.org`

[13] Secure devices cannot resist DoS Arch attacks that compromise availability. Indeed, since the attacker has *root* access to the device, an attack against the communication link that exist between the rich and secure areas will result in successful DoS LD attacks. Even when it is possible to directly connect the smart meter to the secure area by making it a secure peripheral, the general assumption should be that it is possible for an attacker to prevent programs in the rich area to communicate with programs in the secure area, thus compromising availability.

[14] Note that in this demonstration, we consider that the result of the processing can be visualized from the rich area. There is thus no way to ensure that these results are not leaked. Ideally, we would consider a form of usage control to enforce that data communicated from the secure to the rich area cannot be leaked. Such usage control should rely on sandboxing of programs running in the rich area in order to control how they share (e.g., print, display, transmit, or store) the data they obtained from the secure area. This is a topic for future work.

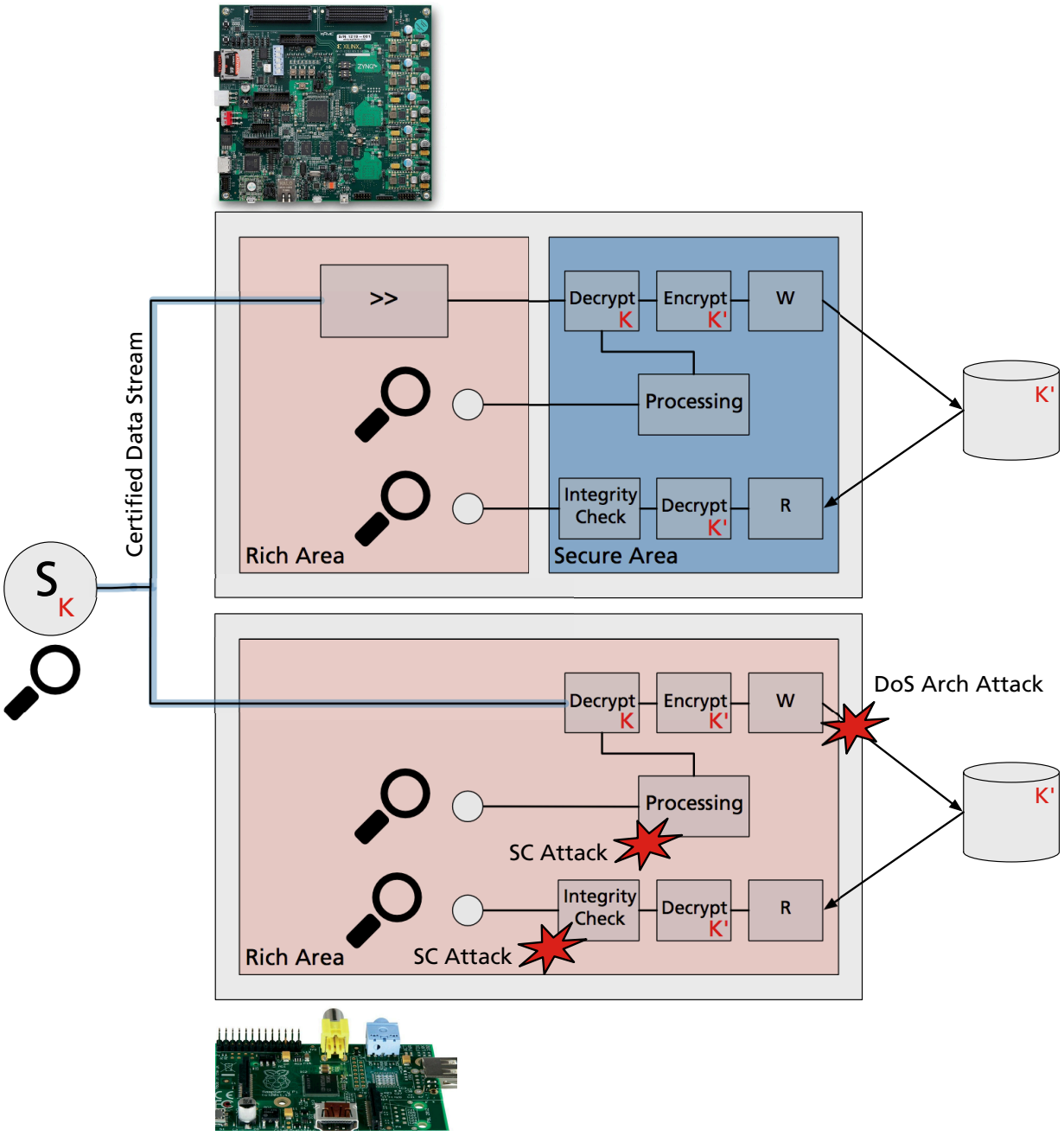

Figure 1: Energy data in the cloud: a) gateway forwards TS and the cloud carries out load disaggregation, b) gateway carries out load disaggregation and the cloud is used for archival. Blue: trusted components. Red: untrusted components. (*) means encrypted data.

6 Conclusion

How can be sure of the integrity and confidentiality of our own energy data then? We argue that secure devices, acting as trusted gateways between smart meters and the cloud are a necessary component of any solution that attempts to tackle this question. By providing a secure area, separated from the rest of the system secure devices can - by design - resist sophisticated software attacks that can easily compromise general purpose devices such as the Raspberry Pi. In the demonstration scenario we show: (i) how vulnerable a general purpose gateway is and how data can be either compromised and corrupted, and (ii) how a secure device can resist the same class of software attacks. We identified a range of important topics for future research on trusted data management.

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] G. Danezis. Privacy technology options for smart metering. Microsoft Research White Paper, 2011.

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

[4] S. Goodwin. *Smart Home Automation with Linux and Raspberry Pi*. Apress, 2013.

[5] 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.

[6] 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.

[7] S. Mehrotra, editor. *Special Issue on Security and Privacy in the Cloud*, volume 35. IEEE Data Engineering Bulletin, December 2012.

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

[9] 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.

[10] 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.

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

[12] M. Weiss, A. Helfenstein, F. Mattern, and T. Staake. Leveraging smart meter data to recognize home appliances. In *Percom*, 2012.