Malware Tolerance

Report 3: Thesis Proposal

Michael Denzel <m.denzel@cs.bham.ac.uk>

April 17, 2015

Supervisor:	Prof. Mark Ryan
Thesis group:	Dr. Flavio Garcia,
	Dr. David Parker (RSMG Member)

Abstract

Defending a system against all possible attacks is extremely difficult. Most research in this area has attempted to harden the architecture but malware developers are still one step ahead. Thus, new approaches are needed to overcome the difficulties in handling compromised systems.

We propose the idea of a "malware-tolerant" system which detects or prevents attacks even if the underlying system is malicious. Ideally, it also guarantees certain security properties.

Our hypothesis is that multiple devices can interact in such a way that any individual device cannot meaningfully tamper with the resources (e.g. data or computations). An adversary would have to compromise several devices to be successful.

For example, one way would be to distribute trust (e.g. cryptographic keys) between an encryption-capable keyboard, a smart-card, and a computer. Assuming the devices are from different manufacturers, it is particularly complicated for an adversary to control all three items. Obtaining only parts of the secret is insufficient to compromise the protected data. Fundamental for this principle is the distribution of trust over multiple devices.

In summary, this research aims to securely use a potentially infected system for security critical tasks. We build security on several independent devices and technologies to tolerate malicious behaviour of some of them. The first step will be to expand our idea of using a smart-card and encryption-capable keyboard to implement trusted input. Afterwards, we plan to examine ARM TrustZone and trusted output. Our goal is to create a system that detects and prevents attacks even if M of the N devices of the system are malicious.

key-words: Malware Tolerance, Trusted Computing, Distributed Systems, Trusted Execution Environment (TEE), IT Security

CONTENTS

List of Figures

List of Tables

List of Abbreviations

AMBA	Advanced Microcontroller Bus Architecture
AMD	Advanced Micro Devices
API	Application Programming Interface
APT	Advanced Persistent Threat
ARM	Advanced RISC Machine
AXI	Advanced Extensible Interface
BIOS	Basic Input/Output System
CAPTCHA	Completely Automated Public Turing test to tell Computers and Humans Apart
CPU	Central Processing Unit
DEV	Device Exclusion Vector
DMA	Direct Memory Access
DoS	Denial of Service
DRM	Digital Rights Management
DRTM	Dynamic Root of Trust Measurement
EPC	Enclave Page Cache
GPU	Graphics Processing Unit
IDS	Intrusion Detection System
IEE	Isolated Execution Environment
I/O	Input/Output
IOMMU	Input/Output Memory Management Unit
IPI	Inter Processor Interrupt
IPT	Intel Identity Protection Technology
ISP	Internet Service Provider
IT	Information Technology
LAPIC	Local Advanced Programmable Interrupt Controller
MAC	Message Authentication Code
MITB	Man-in-the-Browser
MITM	Man-in-the-Middle
MMIO	Memory Management Input Output
MMU	Memory Management Unit
MSI	Message Signalled Interrupt

MTM	Mobile Trusted Module
NFC	Near Field Communication
NIDS	Network Intrusion Detection System
NPT	Nested Page Table
NVRAM	Non-Volatile Random Access Memory
OEM	Original Equipment Manufacturer
OS	Operating System
OTP	One-Time Password
PAL	Pieces of Application Logic
PAVP	Protected Audio and Video Path
PC	Personal Computer
PCB	Privileged Code Blocks
PGP	Pretty Good Privacy
PIN	Personal Identification Number
QR	Quick Response
RAM	Random Access Memory
RISC	Reduced Instruction Set Computing
RSA	Rivest-Shamir-Adleman Cryptosystem
SD	Secure Digital
SEM	Security Mediator
SE	Security Enhancements
SGX	Intel Software Guard Extensions
SoC	System on a Chip
SRTM	Static Root of Trust Measurement
SVM	Secure Virtual Machine
TAN	Transaction Authentication Number
TCB	Trusted Computing Base
TCG	Trusted Computing Group
TC	Trusted Computing
TEE	Trusted Execution Environment
TLR	Trusted Language Runtime
TPM	Trusted Platform Module
TTP	Trusted Third Party

TVD	Trusted Virtual Domain
TVMM	Trusted Virtual Machine Monitor
TXT	Intel Trusted Execution Technology
USB	Universal Serial Bus
VLAN	Virtual Local Area Network
VMM	Virtual Machine Monitor
VM	Virtual Machine
VPN	Virtual Private Network
VT	Virtualisation Technologies
XOM	Execute-Only Memory

Definition of Words

Centralised system	Since there is no formal term to describe the opposite of a distributed system, we use "centralised" system to refer to this. In practice, there are also parallel systems but we do not need to distinguish between centralised and parallel systems as we only want to express that the system consists of one computer.
Device	Our definition of the word "device" includes classical computers and servers but also small external devices like online banking tokens or peripheral devices as e.g. keyboards and mice.
System	"System" is ambiguous since it can refer to a computer system as in System Management Bus, to software (e.g. operating system, system call), or to a mathematical term (cryptographic system, system of equations) or it can include everything as a whole ("concept"). We tried to use the word in the sense of a whole system but could not avoid double meanings due to technical terms where "system" usually is a synonym for computer.

1 Introduction

In reality, it is impossible to guarantee honesty of a platform: see e.g. Debug Register rootkit [3], Zeus/ZitMo trojan [35], Stuxnet [39], and Superfish [79]. Thus, new approaches are necessary to deal with malicious systems. We invented the term *Malware Tolerance* to describe techniques that securely use computers, which are suspected to have malware, for security-critical applications.

This work will be the first approach to deal with malicious software and hardware instead of removing it or protecting a system from malware before its installation. This seems impossible, but there are already systems which build on tolerating malware – without particularly aiming at it – namely online banking and electronic voting.

In online banking, the bank has no control over the computers of its customers. Therefore, usually a second channel is used to pass critical information. E.g. an authentication code is send to phones or generated with an online banking token. This prevents malware on the client computer to interfere with the transaction (integrity). Electronic voting is similar but has higher requirements, e.g. confidentiality and anonymity.

We are interested in transferring this principle to further scenarios like messaging, e-mails, and data storage (this includes document/cloud storage). The main difference to online banking and electronic voting is the frequency and data throughput. While voting takes place once a year or less often, 10-50 e-mails a day are common. With messaging being even more frequent, utilising a token with a number pad is unacceptable for those two scenarios. New techniques are needed to protect messages, e-mails, and storage.

Solutions to the mentioned scenarios would have impact on a wide range of applications and would help securing daily tasks.

Our goal is to draft a system of several devices (smart-cards, encryption-capable keyboards, Trusted Execution Environment (TEE), etc.) that tolerates compromised sub-systems. Roughly, we identified three parts to secure: input, processing, and output (details in Chapter 4.2.4).

As our scenarios represent daily tasks, we target the PC and smartphone/tablet architecture – i.e. normal home and work environments. Our techniques are likely portable to embedded systems (e.g. Raspberry Pi) since those are comparable to smartphones.

2 Literature Review

2.1 Overview of Previous Research

2.1.1 Distributed Systems

Distributed systems are devices connected by some form of network, e.g. the internet or a (system) bus. In this work, we refer to the opposite of a distributed system (a single device) as "centralised" system to emphasise the difference.

When a system is connected by a network, critical failures in a single device should not lead to a crash of the entire distributed system. There were attempts, already in 1991, to harden a system against faults. [20, 73] This was constantly developed towards Byzantine Fault Tolerance.

The idea of Byzantine Fault Tolerance is to replicate servers. If more than 50% of the servers are working correctly, the most frequent result is right. In the literature this is referred to as $f + 1$ computers can compensate f faulty machines ($2 * f + 1$ systems in total). Note that Byzantine Fault Tolerance considers faults, meaning computation errors, not malicious behaviour.

The replicas should be diverse in their software, Operating Systems (OS), as well as their physical location and need to communicate with each other to correct faults. [78] Crucial for this kind of tolerant systems is diversity. Garcia et al. [25, 26] tested 11 different operating systems of a period of 15 years to analyse their variation. The results show that there are nearly none to no common vulnerabilities and even different Linux distributions vary significantly, in contrast to the expectation.

There is hardly any new research on distributed systems in the security context. Narayanan et al. [53] gave an overview about strengthening personal data through decentralisation. Storage can be classified into four groups: remotely centralised, managed by an infomediary, distributed (e.g. peer-to-peer), and local on the user's computer. "Infomediary" stands for a Trusted Third Party (TTP) which negotiates between the user and a commercial entity. The authors conclude that software needs to be open-source and audited by a third party. As hardware might have backdoors, independent trust is needed.

2.1.2 Intrusion Detection and Tolerance

The closest research area to malware tolerance is intrusion detection and intrusion tolerance. It developed from distributed systems but instead of focusing on faults, it concentrates on attacks. An Intrusion Detection System (IDS) tries to block or rate-limit attacks by scanning the network traffic, i.e. mostly connections between internal and external identities. Adaptive filtering additionally considers the system status as attacks vary in time. It focuses on CPU consumption and blocks requests if a certain threshold is exceeded. This way, low-rate "stealth" Denial of Service

(DoS) attacks can be blocked. [23]

Another approach is dialog-based correlation which tracks two way communications to reveal internal malicious computers (bots). The technique uses three Intrusion Detection Systems (statistical payload anomaly detection, statistical scan anomaly detection, rule-based detection) and merges their outputs to get better results. [30]

Just et al. [34] use machine learning techniques to learn unknown attacks. Their idea is that attacks are repeatable and can, thus, be re-tested in a sandbox environment. Possible responses are tested and refined before fed back into the IDS. The attacks are then automatically further examined, e.g. for the minimum length of the attack string in case of buffer overflows, to prevent attack variations.

As Intrusion Detection Systems gained popularity they were also target to attacks, disabling them before the actual exploit took place. Then, intrusion tolerance for IDS developed. Even when components of the IDS fail due to attacks it must stay operable. For Network Intrusion Detection System (NIDS) this meant that multiple independent instances are run. If one component fails, a copy is installed automatically and started again. [38]

2.1.3 Sandboxing

A sandbox is a closed environment in which malicious or dubious software can be tested without harming the actual system. Qualitative filtering is differentiated from quantitative filtering. While in the former only certain resources are accessible, the latter limits the share of the resource (e.g. only 20% of the CPU). E.g. the sandbox of Chang et al. [18] restricts CPU, memory, and network usage by a fine-grained monitoring and control architecture. They implemented a heuristic to estimate the waiting time of an application as a progress metric to limit the resource.

Early approaches for qualitative filtering sandboxes used the Linux *chroot* command. This way, a new file system environment is created but it can only restrict accesses to resources in the file system itself. [60]

As sandboxes are meant to analyse malicious software, automated detection is desired. The main part is behaviour-based dynamic analysis. One idea is to compare before- and after-images of the sandbox, but this only gives an overview about the modifications. If malware creates and deletes the same file during the execution, this would not be detected. The second concept is to monitor the actions of the malware by Application Programming Interface (API) hooking, system service hooking, dynamic linked library injection, etc. This gives a more detailed result but requires more interaction. A disadvantage of dynamic analysis techniques is that it only examines one execution path. Appearance detection on the opposite focuses on syntactic markers rather than the performed actions. [81, 33]

The recent literature researched online sandboxes [43] and virtual networks in sandboxes [84]. Virtual networks use Trusted Virtual Domains (TVDs), stateful firewalls, Virtual Local Area Network (VLAN) tagging, and Virtual Private Networks (VPNs) to create and control the sandbox.

2.1.4 Trusted Execution Environment

Trusted Execution Environments (TEEs) are environments that guarantee the correctness of computations. In general, there are three sources for this trust: software, integrated hardware, and external hardware. We will introduce each of them in the following sections.

Software: Virtualisation and Hypervisors

The simplest TEEs are software solutions, as they are easy to deploy even for inexperienced users. Virtualisation is a technique which simulates a system instead of running it directly. The simulated system is called Virtual Machine (VM) and the controlling part is a hypervisor or Virtual Machine Monitor (VMM).

Most virtualisation environments need hardware support and commonly use Intel Virtualisation Technologies (VT) or AMD Secure Virtual Machine (SVM). In the security context, the VMM is part of the Trusted Computing Base (TCB) together with the underlying hardware. Efforts were made to minimise implementations and thus the TCB. [46]

The following gives an overview about some virtualisation techniques.

Terra [27] uses a Trusted Virtual Machine Monitor (TVMM) to partition the system into multiple, isolated VMs. It provides the applications with encrypted and integrity-checked disks for storage and is also able to attest the VM. To implement this, hardware support for attestation, sealed storage, virtualisation, secure Input/Output (I/O), secure counter, and device isolation (protection from Direct Memory Access (DMA)) is needed.

Apart from managing the VMs, hypervisors were also used to secure further properties as e.g. a trusted path. This was done in the beginning by shielding device drivers and I/O. [19] Small pieces of code, so-called Privileged Code Blocks (PCB), achieve a more fine-grained differentiation of program parts to secure. Especially those accessing I/O devices were protected against the untrusted guest OS. The disadvantage, however, is that PCBs need to be identified manually in the code of drivers and other software. Furthermore, PCBs are not restricted to certain I/O ports and can access any other device which violates access controls. [85]

Zhou et al. redesigned the previous approach, refining the properties a hypervisor has to offer. First, the hypervisor has to protect accesses to I/O ports, device memory, and device configuration space. To do so, it has to scan I/O port mappings and Memory Management Input Output (MMIO) memory mappings. Second, device interrupts have to be filtered which was done by creating a redirection table for hardware interrupts and isolating Message Signalled Interrupt (MSI) and Inter Processor Interrupt (IPI) with Input/Output Memory Management Unit (IOMMU) and Local Advanced Programmable Interrupt Controller (LAPIC). The authors further suggested some architectural changes to improve security in this context. [85]

Software: Compiler

Another technique to offer TEEs is a special compiler in combination with a hypervisor framework. Memory access is controlled through protecting the program counter. Agten et al. [2] claim that this way one can only interact with values through the public API apart from side-channel hardware attacks. They created a special compiler to produce secured assembly language from a high-level language which is then run in a legacy VM ("normal mode") besides a secure VM ("protected mode") handling special calls. The Memory Management Unit (MMU) is set up to generate traps into the hypervisor and ensures that security sensitive functions are managed correctly. The additional compiler is needed to preserve the security properties of the high level language in the low level language, so-called "fully abstract compilation".

The authors further developed their approach to handle function pointers to trusted modules and included an update mechanism for modules after they were deployed. The benefit of function pointers is that software modules can choose which other programs they trust. Untrusted third party programs do not affect the security of the secure software module. [69] In their third paper, the authors managed to fully support object orientation. They implemented dynamic memory allocation, exceptions, inheritance, and inner classes. [57]

Integrated TEE: Overview

Since all software-based techniques are vulnerable to firmware- or hardware-based attacks, hardware vendors designed integrated hardware chips which supply cryptographic measurements to deal with this problem.

Murdoch [52] compared all recent integrated hardware techniques, namely, Trusted Platform Module (TPM), Dynamic Root of Trust Measurement (DRTM), Flicker, Intel Intel Identity Protection Technology (IPT), ARM TrustZone, Trustonic, Samsung Knox, and Intel Intel Software Guard Extensions (SGX). In addition, he considers smart-cards versus integrated TEE.

The following gives a brief description of the four underlying technologies (see Table 1).

Table 1: Comparison of Trusted Execution Environments

Base technique	Implementation	used by
TPM	separate chip	DRTM, Flicker
IPT	separate CPU	-
ARM	33rd bit for secure mode	Trustonic, Knox
SGX	integrated in CPU	-

According to Murdoch, the advantages of the TPM is that it has an open specification and a generic platform but is slow and inflexible. In theory, the security should be good due to its isolation as a separate chip but in practice the TCB is too large for Static Root of Trust Measurement (SRTM). Its updated version enhanced the TPM with DRTM, enabling it to start the trusted environment at any time. Flicker built on top of this and allows arbitrary code to execute while suspending the OS temporarily. It enables trusted modules to run on the highest privilege level but leaves the OS vulnerable to attacks of the modules.

Intel IPT utilises a separate CPU to run Java programs. It has similar functionality as the TPM and has access to the display. Arbitrary code can be executed but has to be licensed. At the moment, Intel supports applications for key generation and storage, one time password generation, and secure PIN entry via mouse and secure display.

Compared to the other techniques, TrustZone only extends the CPU bus with a 33rd bit and creates two “worlds” by this mean, the normal and the secure world. This bit flag is also distributed to the peripherals and RAM to enable securing them. Since TrustZone only provides one secure enclave and provides no implementation, Trustonic and Samsung Knox were developed, To overcome the shortcomings of TrustZone, Trustonic splits the single secure enclave into several by a custom OS. Samsung Knox is similar but additionally provides a trusted boot.

In 2013, Intel announced Intel SGX[1], a TEE integrated in the CPU. It differentiates from other approaches by not only protecting enclaves from other (e.g. OS) code but also the vice versa. According to the limited information online, it offers isolated execution, secure storage, remote attestation, and secure provisioning. One disadvantage is that a trusted path is missing until now.

Murdoch concluded that, in comparison to integrated TEE, smart-cards offer better security but disadvantages in implementing some special applications.

Integrated TEE: Trusted Platform Module

The original approach for TEEs was the TPM chip, an additional co-processor, which was frequently used in research projects due to its open implementation.

[1]Intel SGX was not released publicly yet.

Sailer et al. created an integrity measurement mechanism on the TPM for dynamic executables. Their main goal was to ensure remote attestation without changing CPU or OS. The integrity measurement covers the entire system from Basic Input/Output System (BIOS) to the application layer including dynamic loaders, shared libraries, and kernel modules. In the end, the authors conclude that security guarantees do not require new technologies but depend on the availability of an independent trusted entity (as the TPM). [64]

Another secure technique based on the TPM is Open Secure LOader (OSLO) [36] which uses the DRTM feature of the TPM to create a trusted environment during run-time. DRTM was introduced after attacks on bootloaders, BIOS, and TPM reset attacks breached the static root of trust the TPM used in its first version. AMD as well as Intel support a DRTM with their *skinit* and *senter* instruction respectively. By removing BIOS and bootloaders from the TCB the mentioned attacks are successfully countered.

Flicker [47] is another TPM based technology which uses DRTM. It removes the application and the OS from the TCB and only executes secure computation in a minimal working environment. This is done by abruptly switching from legacy operating system to the trusted environment, executing the secure functions, and switching back to the original OS. During a Flicker session the OS and interrupts are disabled. This means the system will not respond to any user interaction during this time, like e.g. keyboard inputs and interactions with storage devices as hard disk and USB sticks. This represents a major drawback of the technique and forces Flicker sessions to be very short (< 1 sec) in order to preserve system stability.

Based on Flicker, Filyanov et al. [24] built an uni-directional trusted path. They focused on integrity and created secure input by implementing a browser plugin which uses Flicker to display a confirmation dialog. The technique was aimed at online banking to replace Completely Automated Public Turing test to tell Computers and Humans Aparts (CAPTCHAs) or transaction authentication.

The authors of Flicker continued their research on the TPM with TrustVisor [49] which also uses DRTM to minimise the TCB. Security critical code blocks, Pieces of Application Logics (PALs), are protected by the hypervisor which has three modes. “Host mode” executes the hypervisor, “legacy guest mode” is for the untrusted guest OS, and “secure guest mode” isolates PALs from guest OS and from the hypervisor. Each PAL entity is supplied with its own microTPM implemented in software which resides in TrustVisor. The necessary hardware features are provided by the TPM chip. TrustVisor focuses on isolation between itself, the untrusted OS, and the PALs but requires a couple of hardware features, namely AMD SVM, 2D page walks, DRTM, Nested Page Table (NPT), and either Device Exclusion Vector (DEV) or a full IOMMU. The root of trust is provided by a three layered bootloader, starting with an untrusted loader (“TLoader”) to relocate the guest OS and initialise AMD SVM. The second part of the bootloader is already TrustVisor but it had to be split into two portions to fit with AMD’s *skinit* instruction (up to 64 KB): an initialisation part (< 64 KB) and a run-time part. Former sets up DEV to protect against DMA accesses and starts the third bootloader after verifying it. Then the VM of the guest OS is activated.

The mentioned 2D page walk of TrustVisor is a technique introduced by Bhargava et al. [13] to reduce the memory management overhead of virtualisation. It proposes a different computation and caching model for the standard x86 page walk using nested page tables in a matrix ("2D"). The advantage is that large page sizes can be handled which reduces the need for hypervisor interventions on cache misses.

Following their approach, the authors of TrustVisor build the modular hypervisor framework "XMHF" [77] to achieve modular extensibility, automated verification, and high performance. Vasudevan et al. derived six properties to ensure memory integrity and verified the code, mostly by the automated tool "CBMC". Considered were the hypervisor itself, the guest OS, external devices (like USB-sticks), and the memory. Guest and devices were assumed malicious.

The following enumeration describes the conditions that need to be fulfilled for each of the six properties:

1. Modularity
   First, during initialisation the hypervisor has to be called and, second, intercepts must trigger the correct intercept handler.
2. Atomicity
   Atomicity is given when the initialisation routine finishes before any other code runs and intercept handlers run atomic, that means single-threaded without interruptions.
3. Memory access control protection
   As guest OS and external devices are possibly controlled by an attacker, memory accesses have to be restricted for all untrusted sources.
4. Correct initialisation
   After the hypervisor starts, memory has to be protected and intercepts must call to the correct handler.
5. Proper mediation
   Memory needs to be protected when attacks could take place. For the external devices this is always the case. In contrast, the guest OS can only interfere after it was started.
6. Safe state updates
   No updates to system state or memory break the protection mechanism, modify the intercept entry point, or change the code of the hypervisor.

All of those properties were verified for XMHF (6018 lines of code) with the C model checker CBMC (5208 lines) or by manual examination (810 lines). The only disadvantage of the implementation is that only a single guest system is supported, but another hypervisor as guest is possible due to direct access to devices.

While all other approaches secured the execution or memory integrity, "Memoir" [56] guards memory continuity with the TPM chip. It stores a log (instead of only a counter) of the corresponding software module in protected Non-Volatile Random Access Memory (NVRAM). Non-volatile memory is safe but slow and limited in its size. To improve the performance, the TPM buffered write accesses which required an uninterruptible power supply to save data in case of power failures. Furthermore, the authors created a management module in NVRAM which handles and secures sub-modules. This way an unlimited amount of modules could be supported with the limited safe memory. The history of every snapshot is identifiable and guarantees safety against replays or roll-backs. As proof of concept, three modules relying on Memoir were implemented.

Integrated TEE: TrustZone

Adopting the idea of the TPM, various hardware vendors were creating TEE: ARM produces TrustZone [5, 6, 7], Samsung is developing Knox [66, 67], and Intel has announced SGX [31].

Contrary to SGX and Knox, TrustZone is already available and mostly deployed in mobile phones and embedded systems. It introduces two new operating modes, secure world and normal world, which orthogonally split privileged and unprivileged mode, creating four co-existing modes. Critical resources, like interrupt flags and special system control co-processor registers, are controlled from the secure world. Between the two worlds, a special secure monitor mode mediates requests and can be activated by a secure monitor call instruction. The AMBA AXI bus signals the state, secure or normal world, also to the System on a Chip (SoC) peripherals to control access. [82]

Winter ported the TCG model for trusted computing to TrustZone. The goal was to create a secure bootloader and arbitrary trusted environments from only two worlds. The secure bootloader relies on a hardware Mobile Trusted Module (MTM), if applicable, or has to combine a software MTM with other hardware roots of trust. To divide the two worlds into multiple parts, a hypervisor supervises VMs. Interrupts as well as user and kernel space have to be secured over both worlds. [82]

The author also experimented with an ARM development board capable of secure boot and support for TrustZone. [83] Since some features were first missing due to incomplete documentation, he proposed the idea of "untrustlets", untrusted pieces of software running in the normal world while the OS is running in secure world. After reverse-engineering the infrastructure and revealing the missing features, he produced a micro-kernel to utilise all elements of the FriendlyARM Mini6410 chip.

Pirker and Slamanig [59] examined anonymity for payments, a completely different setting to the other scenarios. They used TrustZone in combination with Near Field Communication (NFC) to create a payment system which protects the privacy of its customers. For the payments they distinguished four characteristics of payments:

- micro (<5$) and macro (>5$) payments
- local payments with a vending machine versus remote payments (internet)
- prepaid, post-paid, and in-time paid payments
- online (centralised to an entity) in contrast to offline (distributed) payments

The goal of the paper was to achieve anonymous and unlinkable payment. Prepaid mechanisms are the only method to hide the payer from the payee while simultaneously assuring the payee of receiving the requested amount. The process is divided into three steps. First, (1) *credit* has to be acquired through a prepaid procedure. The authors intended to provide scratch-off cards at central locations (e.g. supermarkets, petrol stations, etc.) with QR codes to top up the balance. Afterwards, to (2) *activate* the credit, communication with the service provider is necessary. This can either be handled over the internet or via NFC at the ticket terminals. Lastly, (3) the *payment* takes place. The user activates the app for public transport at departure, the normal world app requests the travel costs from the payment trustlet in TrustZone, and provides a bar-code as proof for ticket inspectors.

TrustZone was also used to facilitate confidentiality and integrity protection of .NET mobile apps by a Trusted Language Runtime (TLR). [68] The goal of this work was to enable coding as in normal environments while still providing a small TCB and compatibility with legacy software. The classical TLR consists of an untrusted execution environment for OS and apps and a trusted one for TLR and security sensitive app components. This bisection matches very well with the two worlds of TrustZone and can be easily ported. To proof their concept, Santos et al. implemented

- a one-time password app managing a TAN list,
- an user authentication system for bus or subway,
- a secure mobile transaction app for credit card details and banking,
- and an access control mechanism for e-health applications.

A major drawback of the paper is the missing trusted I/O in favour of a reduced TCB. Likely, this optimisation affects the security of the entire system since Vasudevan et al. [76] defined a trusted path as one of five essential properties of a secure centralised system (see below “Integrated TEE: Properties”).

TEE are also used to protect media data like music or films. [51] The problem is to allow honest users to play the content while protecting the clear text from dishonest ones. Digital Rights Management (DRM) is a term to describe approaches protecting this material and consists of mainly two parts:

- Licenses hide content from unauthorised users or protect it through cryptography.
- To trace misbehaving users, data is watermarked to make it identifiable.

Usually, a content server, licence server, and a DRM client are part of the scenario. DRM private keys, licence, and DRM temporary keys (e.g. content encryption key) need protection. Mohanty et al. [51] gave a survey over existing TEE which are suitable for this scenario. Intel offers three technologies on this, Intel Trusted Execution Technology (TXT) which is based on the TPM, SGX, and Protected Audio and Video Path (PAVP). The third one is different in the sense that it provides secure content path. A software decoder reads the content from the medium, encrypts it, and sends it to the graphics card, effectively tunnelling the kernel driver. From the Graphics Processing Unit (GPU) the data is securely delivered to the display device with high bandwidth digital copy protection. The authors also mentioned ARM TrustZone, ARM hypervisor mode, ARM Mali-V500, Texas Instruments M-Shield, AEGIS [70], and the virtualisation approach. The difference for DRM is, that the user is assumed untrustworthy and the hardware in contrast is trusted which only counts in this very limited scenario.

Integrated TEE: Properties

Vasudevan et al. [76] analysed TrustZone and its capabilities in depth considering Original Equipment Manufacturers (OEMs), telecommunications providers and carriers, app developers, and device owners. While security is important for developers and users, it is only a secondary concern for OEMs and carriers. The need for these features is visible through the new market of removable secure elements like universal integrated circuit cards and SD cards.

To illustrate the security of TEE, the authors defined five desired security features which became the building blocks for TEE:

1. **Isolated execution**
   The secrecy and, above all, integrity of the software module's code and data has to be secured at run-time. Possible implementations include OS (if not compromised), a hypervisor, or a separate co-processor. Environments securing this property are called Isolated Execution Environment (IEE).

   Isolation can either be achieved with a single execution environment multiplexed by a hypervisor or a parallel environment, like TrustZone or a smartcard.
2. **Secure storage**
   When a device is powered off, the state of it (and its software) has to be preserved securely, including secrecy, integrity, and freshness against replays, rollback attacks, etc. A normal OS provides secrecy and integrity through file system permissions. Freshness, however, requires a trusted counter or clock.

   iOS misses access-control for the device key. Android stores the data of the AccountManager in clear and only encrypts the file system by a user-secret which is entered during boot. In result, both systems lack secure storage.

3. **Remote attestation**
   Another security property is proof that a message came from a particular software module. To achieve this, the TCB of the program has to be attested to the remote party. Usually, the TCB consists of OS kernel and the application itself. One method to implement this is to create a private key managed by a small TCB which is certified by a trusted party (normally the device manufacturer).

   Note that attestation only verifies the loaded code and does not protect from run-time manipulations.
4. **Secure provisioning**
   Secure provisioning runs in the reversed direction compared to remote attestation. Instead of authenticating a sender, it authenticates the recipient. Thus, it enables a module to send data to a specific module on a specific device, secrecy and integrity protected.
5. **Trusted path**
   To ensure that the communication with the user is correct, the authenticity to a peripheral device has to be secured. Optionally, secrecy and availability can be guaranteed. For this, either the OS is part of the TCB or the device driver runs isolated from the OS. Additionally, the platform must provide low-level access-control for peripherals.

   ARM TrustZone distributes the state of the system (i.e. normal or secure world) to the bus to enable a trusted path.

After defining security properties, Vasudevan et al. analysed ARM TrustZone towards them:

1. The world model of TrustZone already splits the system (incl. interrupts) and its peripherals into two separated parts. TrustZone aware MMUs isolates the memory, but DMA from normal-world to secure-world needs to be blocked, too. Thus, TrustZone can achieve isolated execution but only with additional support.
2. Since there is no explicit root of trust, storage is insecure. If secure elements are available, they can replace TrustZone for this issue. Software secure elements and removable hardware secure elements (e.g. SD cards) exist to compensate for the insecure storage of TrustZone.
3. Remote attestation requires a registers to store cryptographic hashes and a private key which only the TCB can access. Most mobiles are equipped with a secure boot function, but all of them lack the required registers. According to Vasudevan et al., these registers can be emulated in software.
4. Mobiles support authentication via the hash of the public signing key which is stored in read-only memory. It is used for e.g. software updates but is only available to OEMs and carriers.

5. Some systems, e.g. M-Shield, implement a trusted path in hardware. A special communication path between peripherals and secure DMA ensures a trusted path. It is used especially in the DRM context, but it is uncertain if this feature is open or closed.

An issue with TrustZone is that, for complete security, all components have to be present (TrustZone aware DMA controller, interrupt controller, a special bus bridge, etc.). According to the paper [76], most of the mobiles today miss some of these features and some devices even disable the secure world at boot.

Another point is that OEM and carriers have different goals and concerns from developers and users. Carriers need to support developers and make security features available. This can either be done by running the developers module directly in the secure environment (Module-IEE) or giving them access to functions of the secure environment (App-IEE).

At the moment, the missing support from OEM and carriers and the cost for development tools prevent the open-source community from working with ARM chips. Vasudevan et al. conclude that OEM, carriers, and developers have different focuses but should consider the others' goals and work more together to improve TEEs like TrustZone.

Integrated TEE: Samsung Knox and Intel SGX

Information about Knox and SGX is very coarse, since Samsung Knox is closed source and SGX has not been released.

Samsung introduced the Knox architecture in a report [65] and on their website [66, 67]. Knox builds on TrustZone and enhances it with a secure boot using a tamper-resistant sector of the ARM processor. Critical resources are measured during boot and are constantly verified at run-time. The Security Enhancements (SE) for Android protect OS and enforce access control. This is centrally controlled by the "corporate security administrator". Apps are protected in so-called Knox containers, a complete Android environment isolated from the rest of the system. Mobile device management, "Samsung for Enterprise (SAFE)", allows an administrator to remotely access the device.

The only paper about Samsung Knox presented "TrustZone based Real-time Kernel Protection (TZ-RKP)". [8] It was implemented on Samsung mobiles and bridges between hypervisors and hardware approaches. While hypervisors offer control and flexibility, hardware techniques as the secure world of TrustZone have no control over the normal world guest OS: secure world operating systems cannot intercept critical events like page faults or system call instructions of the normal world. To overcome this issue, the authors suggest event-driven monitoring through withdrawing control over critical system resources from the guest OS. Required are (1) memory protection of the normal world memory, (2) changes to the guest OS kernel, (3) a secure bootloader, and (4) a TrustZone aware architecture. Azab et al.

emulated a control instruction and trapped on changes to the translation table. To protect the memory they blocked kernel data double mappings. [8]

Regarding SGX, a few white papers from Intel exist showing that SGX provides protected software containers to isolate applications and secure data. The next paragraphs will summarise these three papers.

McKeen et al. [50] focused on the isolated execution part of SGX. They ensured that isolated environments in SGX, so-called enclaves, are protected from accesses of normal and privileged software, e.g. VMM, BIOS, and OS. A memory encryption engine automatically encrypts (including integrity and replay protection) the RAM section, named Enclave Page Cache (EPC), of the enclave. Memory accesses to protected regions are treated as accesses to non-existent memory, even for DMA and other external sources like the graphic engine.

SGX also supports remote attestation and secure persistent storage. Enclaves are identified by a SHA256 hash of the internal log of the enclave initialisation. Local attestation is done via a Message Authentication Code (MAC) and SGX as a form of TTP. For remote attestation, the MAC is replaced by a cryptographic signature with a device specific asymmetric key from a special enclave, called quoting enclave. To seal storage, data is encrypted and integrity protected as well as protected against replay attacks with monotonic counters offered by the platform. [4]

Intel also demonstrates how this architecture can secure applications by implementing a few proof-of-concept programs. Hoekstra et al. [32] implemented a one-time password, an enterprise rights management, and a secure video conferencing application with SGX and Intel PAVP. Enterprise rights managements need to defend the document against encryption key theft, platform or application spoofing, and use-policy tampering. The main parts of video chatting are authentication, encryption, and trusted I/O. The authors also suggested to add a policy engine to prevent call logging. [32]

SGX seems very promising but since no hardware supporting it is available at the moment, detailed information is missing and this work can only give a very brief overview about SGX. First independent surveys will reveal more.

Hardware: New Architectures

Since integrated hardware techniques do not solve all problems related to trust and security, a few researcher tried to create own architectures or use additional devices, mainly phones.

AEGIS [70] is a draft of a tamper-evident and -resistant single chip processor which assumes the memory outside the CPU and the OS as untrusted. It protects external resources for integrity and confidentiality by its three components: a module to verify memory integrity based on Merkle trees, an encryption module, and a secure context manager to track running AEGIS processes. As proof of concept, certified execution as used in distributed gird computation and DRM was designed.

Lie et al. analysed Execute-Only Memory (XOM) to create tamper resistant software. They base their implementation on machines supporting “internal compartments” which is a sort of a boundary, blocking processes in one compartment from reading data of other compartments. [42]

To protect guest VMs from the underlying hypervisor, Szefer and Lee [40, 71] modified microprocessor and MMUs. To achieve confidentiality and integrity protected tables, the hardware was altered to store an additional register in the memory controller and a few check routines were added (in hardware). The tables are stored in a new protected part of the RAM together with two keys for encryption and hashing. Testing took place on an OpenSparc T1 simulator. All in all, this technique comes quite near to malware tolerance, apart from the trusted hardware.

Bastion [17] is another architecture to protect software in an untrusted software stack. Each software module has its own secure memory and secure storage. The architecture was expanded with registers for hashes of hypervisor, storage owner, and storage as well as a storage key and current module ID. Further, a memory encryption and hashing engine, random number generator, and module state tables were included. The entire system was also simulated on OpenSparc.

Noorman et al. designed Sancus [55], a “zero software approach” to isolate software modules based on the program counter. Secure communication and attestation was secured through a shared symmetric key between software provider and software module. With secure communication, a MAC of the module’s identity is sufficient to verify the module (remote attestation).

Hardware: External Devices

Already existent external devices were also used to form an independent entity. A phone served as an isolated entity to encrypt data using a public key [44], effectively hiding the long-term secret from the computer.

Since phones have a display that is independent from the computer, they can represent a secure monitor for a trusted path to the user. The devices work in combination with an encryption keyboard. The root of trust was provided by the Flicker TEE [47] for pre-processing before and post-processing after delivering the input to the OS. [48] Previously to this concept, the authors experimented with SRTM. [45] Both ideas seem to suffer from problematic usability due to confusion with the additional hardware.

In contrast to all the other techniques, Safeslinger [22] is focusing on online communication. It enables pairing of smart-phones when they are brought together physically. They are exchanging public keys and establish a secure channel for secrecy and authenticity.

While phones are an easily available external resource, they are not very secure. Researchers demonstrated how to attack a phone over USB [80] and how to compromise an USB stick on firmware level which endangers computers and phones. [54]

2.1.5 Solved Scenarios

Contrary to problematic scenarios, we want to give and overview of the already working schemes.

Online Banking

A positive example is online banking if it is implemented correctly. For this, transaction authentication has to be distinguished from entity authentication. While transaction authentication requires approval of the user for every single bank transaction, entity authentication only secures the creation of new recipients. In general, transaction authentication is more secure.
Multiple techniques exist as authentication mechanism [1, 37, 75, 9, 10, 58]:

1. The simplest approach only requires a login-name and a password but is vulnerable to most attacks (trojans, keylogger, Man-in-the-Middle, and Man-in-the-Browser).
2. After the first attacks, banks adopted the One-Time Password (OTP) approach. Each authentication is approved by another password. Initially, these passwords were sent to the clients as a list on paper, the so called "iTAN" (indexed TAN).
3. iTANplus is a development of iTAN and uses an additional CAPTCHA to check that a human entered the OTP.
4. Since these techniques are either not secure or usable, the mTAN (mobile Transaction Authentication Number (TAN)) or also smsTAN was invented. The OTP is sent to the mobile phone of the client.
5. Similarly, there are also automated calls to the mobile. A computer generated voice reads transaction and prompts the user to confirm the transaction with a code displayed on the computer screen.
6. More advanced techniques use security tokens. These generate TANs to authenticate an action.
7. As simple tokens are bound to an account or a person, the chipTAN or card-TAN was invented. It consists of a security token including a card-reader to insert the bank card. Thus, readers are generic and can be reused.
8. Lastly, there are also advanced embedded devices like BestSign[a] which is a USB-stick with a display and a button to approve or abort.

[a]http://seal-one.com

Electronic Voting

Another working example is Du-Vote [62, 29] which uses a simple security token to solve the scenario of electronic voting. A server, client computer, and the token are working together to cast a vote secretly and anonymously. The server hosts a bulletin board which also serves as verification for users of their vote.

Client computer and server communicate to create a table of hashes for the voter. These hashes are cryptographically secured to send particular information from the server to the token. For each possible candidate server and client computer create two hashes. The user randomly chooses one column of the hashes, picks his or her candidate in this column, and enters the hash of the candidate plus the entire other one of the two columns into the token. The computer is unable to identify which column the user chose and, based on this, the vote is hidden from the computer.

Encrypted votes are combined on the server utilising homomorphic encryption since the multiplication of El Gamal encrypted ciphertexts results in addition of the plaintext. Consequently, votes can be counted without decrypting them, only revealing the final amount of votes per candidate.

The system is secure if the token is honest as it reveals wrongdoing of computer and server. But, even if the token was compromised and server as well as PC try to change the vote, this will result in errors with other voters who then identify the attack. The only assumption here is, that an adversary cannot control the token in real time. In conclusion, electronic voting is secured against attacks of the client computer and the server.

Secure Centralised System

Li et al. [41] created an implementation that could also be called malware-tolerant. It is a three-fold approach consisting of a sandbox, a hypervisor, and a TEE like TrustVisor [49] or SGX [4]. While the TEE secures the hardware, the sandbox protects the operating system from malicious programs. The hypervisor acts as an isolation module to separate operating system and software, protecting each from the other (for example also an honest application from a malicious operating system). This technique is very well designed and seems to defend a lot of attacks. The only attack point is the hardware, as the draft consists of one physical system with one TCB.

2.1.6 Further Techniques

Lastly, we want to point to further areas of research which are related in a broader sense.

The term obfuscation is used for software that hides secrets in the program code. The desired general-purpose black-box obfuscation is impossible [11, 28, 12, 63] but indistinguishable obfuscation is achievable. For malware tolerance, this is of less importance as the computation of indistinguishable obfuscation is too slow and we desire higher security than only obfuscation.

Encryption is helpful for malware tolerance in general. Boneh et al. [14] created revocable encryption based on a so-called Security Mediator (SEM). This encryption splits the key into multiple parts which can be distributed among parties. This way, only cooperating parties can decrypt the secret.

There is research on the area of homomorphic encryption [16], an encryption that allows computations on the ciphertext. Recently, the field of searchable encryption developed. [15] It uses homomorphic encryption to search for certain data in the ciphertext.

2.2 Inadequacies of Previous Work

As we have seen in the literature review, most previous research projects focused on securing one centralised system. To our knowledge, there is no paper accepting an entirely compromised computer (apart from Du-Vote [62, 29]) and a lot of them concentrated on the desktop computer architecture ignoring helpful external devices as e.g. online banking tokens.

Furthermore, the hardware is very often assumed trusted which is an assumption that is not feasible in certain scenarios, especially considering firmware attacks.

2.2.1 Centralised System Approaches and Shortcomings

One of the biggest disadvantages of an individual device is its single point of failure. If that device is compromised due to implementation bugs or wrong setup, security is breached. In addition, one single device is also usually controlled by one person - be it the manufacturer, the programmer, or the administrator. An attack originating from the hardware is very powerful and, in case of the manufacturer being the adversary, has impact on a wide range of systems.

When working with a centralised system, the TCB needs to be secure. Vasudevan et al. [76] elaborated five properties a centralised system must fulfil (see also Fig. 1 and Chapter 2.1.4):

1. Isolated execution
2. Secure storage
3. Remote attestation
4. Secure provisioning
5. Trusted path

Most of these characteristics seems to be a precondition for secure (computer) systems in general, even for distributed systems.

Figure 1: Centralised System Architecture

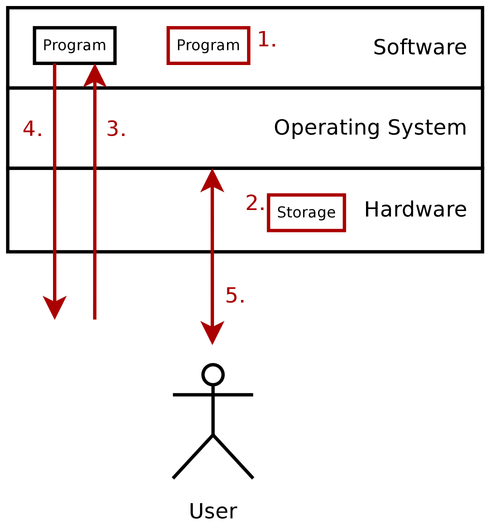


Fig. 1 highlights that all those properties depend on the hardware. That means for a centralised system approach, the hardware has to be trusted, verified, or secured. Since there are already firmware (BIOS rootkit) and hardware (hardware keylogger) attacks the architecture cannot be assumed trusted. Hardware verification on the other hand is very expensive and, thus, not feasible. Lastly, securing the hardware is challenging and error-prone if we only have one untrusted device. A warning example is the SRTM approach of the TPM. It had problems due to the code size of the TCB. The first version was vulnerable to bootloader attacks, TPM reset, and BIOS attacks [36] which lead to further development.

Another issue, that was raised by van Rijswijk-Deij and Poll [74], is the verifiablity for the user, namely local attestation. The main point is how a user can test if he or she is indeed communicating with the secure part of the system when the display is used by both, secure and normal applications. A malicious program could start a phishing attempt and only pretend that the secure system is running.

Adding this requirement, we end up with six essential properties for a secure centralised platform. Implementing of all of them results in a large TCB, a single point of failure.

1. Isolated execution
2. Secure storage
3. Remote attestation
4. Secure provisioning
5. Trusted path
6. Local attestation

As Sailer et al. [64] already pointed out, independent trust is essential. The available hardware features determine which level of trust is possible. This is problematic for a single device where the hardware is, at least mostly, untrusted.

2.2.2 Distributed Systems

While a centralised system consists of one architecture, a distributed system involves a variety of devices. In this sense, it is more reliable than a sole platform and also less vulnerable to attacks of people controlling one entire device (e.g. manufacturers).

Although this approach is not new, it was never examined under the assumption of whole devices being compromised. Network traffic (IDS) is commonly untrusted but the hardware of the computer is trusted.

Distributed systems were researched from the viewpoint of operating systems but, to our knowledge, never regarding a (partly) malicious computing base.

3 Examples

The basic idea of our research is to distribute trust over multiple (partly-trusted) devices to protect from attacks of one or more of those. As this is a new approach, we want to introduce a few practical examples first before elaborating our hypothesis and the underlying concepts.

As already stated earlier, we mainly consider two domains: messaging (including e-mails) and storage in a home or work environment. The following sections describe a few drafts. Our hypothesis is defined afterwards in Chapter 4.1.

3.1 Setting

When dealing with a potentially malicious device, we aim to limit malware to data that is shown on the screen. Hiding any displayed information is impossible, assuming the attacker has full control over the displaying device (e.g. computer or phone).

Usually, the attacker gets even more than only displayed information. An example: when we log in to our email account, an attacker controlling the computer gets our credentials and, thus, earns full access to our email account. He or she can read all old and new e-mails, can write new ones, and can delete emails; abilities that reach much wider than only displayed data.

To reduce the impact, we have to ensure that each action has to be independent from other actions.

3.2 Reducing Impact: Splitting Keys

We propose encrypting every item (e.g. an e-mail) with a different key. Then, getting access to one key only reveals one item (similar to OTP). Write accesses have to be controlled analogously.

Our idea is to use a smart-card to gain certain properties. We can, for example, assume that all its information is only available when the smart-card is present reducing the time period in which attacks are possible.

If we assume that the smart-card is honest, we only move the trust from the original device (e.g. phone or computer) to the smart-card. We could, of course, store a secret key on the smart-card but this would leave us vulnerable to theft or loss. Thus, decryption must depend on multiple devices instead of one (i.e. not only on the smart-card).

Our approach encrypts the data two times. Once to send it to a special entity (e.g. PGP key of the computer) and another time to establish a "session" over this protocol. The session key $k_{session}$ is either stored on the smart-card or encrypted with the public key of the smart-card. Thus, smart-card and computer have a key with both keys being necessary for decryption.

This way, an attacker controlling the smart-card can only obtain the PGP encrypted email and an attacker controlling the computer will only get emails that are approved by the smart-card. To gain full access, both devices have to be compromised by the same attacker or colluding attackers.

Figure 2: Authorisation by Smart-card

Blue:	trusted entity
Orange:	partly trusted entity (2 of 3 need to be honest)
Red:	untrusted entity

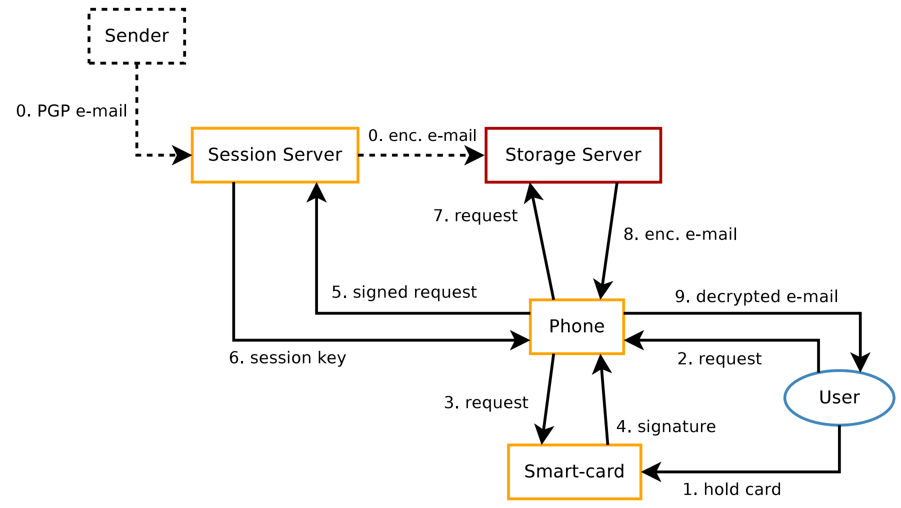


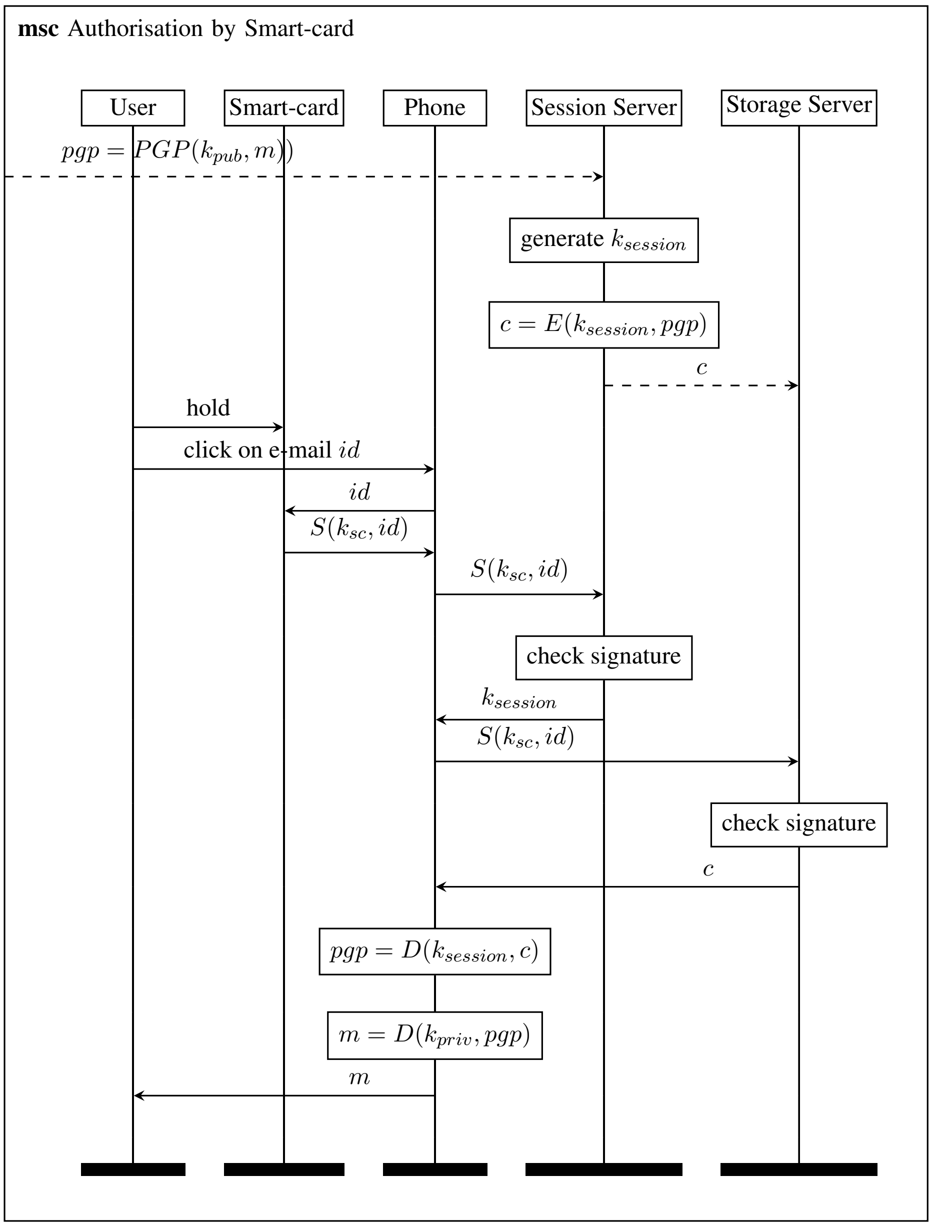
Fig. 2 and Fig. 3 (message sequence chart) show how to securely retrieve a message on a phone:

0. (Initialisation phase) A sender sends a Pretty Good Privacy (PGP) encrypted email. The “session” server receives it, encrypts it with a newly created session key $k_{session}$ and forwards it to the untrusted storage server.
1. The user wants to read a critical e-mail and holds his/her smart-card to the phone.
2. He/she clicks on the item to retrieve (= request of email id).
3. The phone requests a signature for that item from the smart-card.
4. The smart-card signs requests while also rate-limiting them.
5. By giving proof of the existence of the smart-card through its signature, the phone asks for the session key $k_{session}$.
6. The session server checks the signature and delivers the session key.
7. The phone requests the two-times encrypted e-mail from the storage server,
8. receives the e-mail,
9. decrypts it with the session key, decrypts it with the PGP key, and displays it to the user.

Figure 3: Authorisation by Smart-card: Sequence Diagram

This message sequence diagram shows a first draft of our smart-card authorisation protocol. A user holds the smart-card to the phone and clicks on the e-mail to retrieve, which is identified by id. With a signature of the smart-card, the phone can request e-mail and session key from the appropriate servers. The e-mail is decrypted and shown to the user. $S(k, m)$ denotes a signature of data m with key k, $E(k, m)$ is encryption, and $PGP(k, m)$ represents a PGP encrypted e-mail.

broken line:	initialisation phase



While the protocol already tolerates attacks from one of the three partly trusted devices (session server, phone, smart-card), it has some disadvantages. At the moment, replay attacks are possible (see Fig. 3). This is probably acceptable since retrieved messages are anyway assumed to be compromised.

Furthermore, commands sending or deleting messages have to rely on signatures of the smart-card. Imaginable are signatures for entire actions like *read email* id or *delete email* id.

Another idea is to use SEM enhanced revocable cryptography [14] which splits an RSA key into several sub-keys. In our protocol, this effectively replaces the double encryption with a single encryption (better performance) and possibly removes the session server improving usability and reducing the TCB.

Possibilities:

- The smart-card stores one part of the key and the another part is copied to the phone. Attacks are then only possible when the smart-card is connected but losing phone and smart-card (e.g. stolen/lost backpack) would grant the attacker full access.
- We split the key into three parts for smart-card, phone, and server (could be the company's server or a raspberry pi at home). This way, the server can revoke keys in case of loss or theft.

3.3 Trusted Input

Creating messages is more difficult than receiving them. When we assume an attacker obtains everything that is displayed, our messages will never remain secret. However, integrity protection seems feasible.

We use an encryption-capable keyboard in combination with a smart-card (see also Fig. 4 and 5) to create a protocol we call "sign-what-you-type" in contrast to the normal "sign-what-is-displayed". The process is as follows:

1. First, the user types a character m_i.
2. The keyboard sends the keystrokes one-by-one encrypted (stream cipher) to the smart-card,
3. which buffers them and sends the decrypted character back to the computer
4. to display.

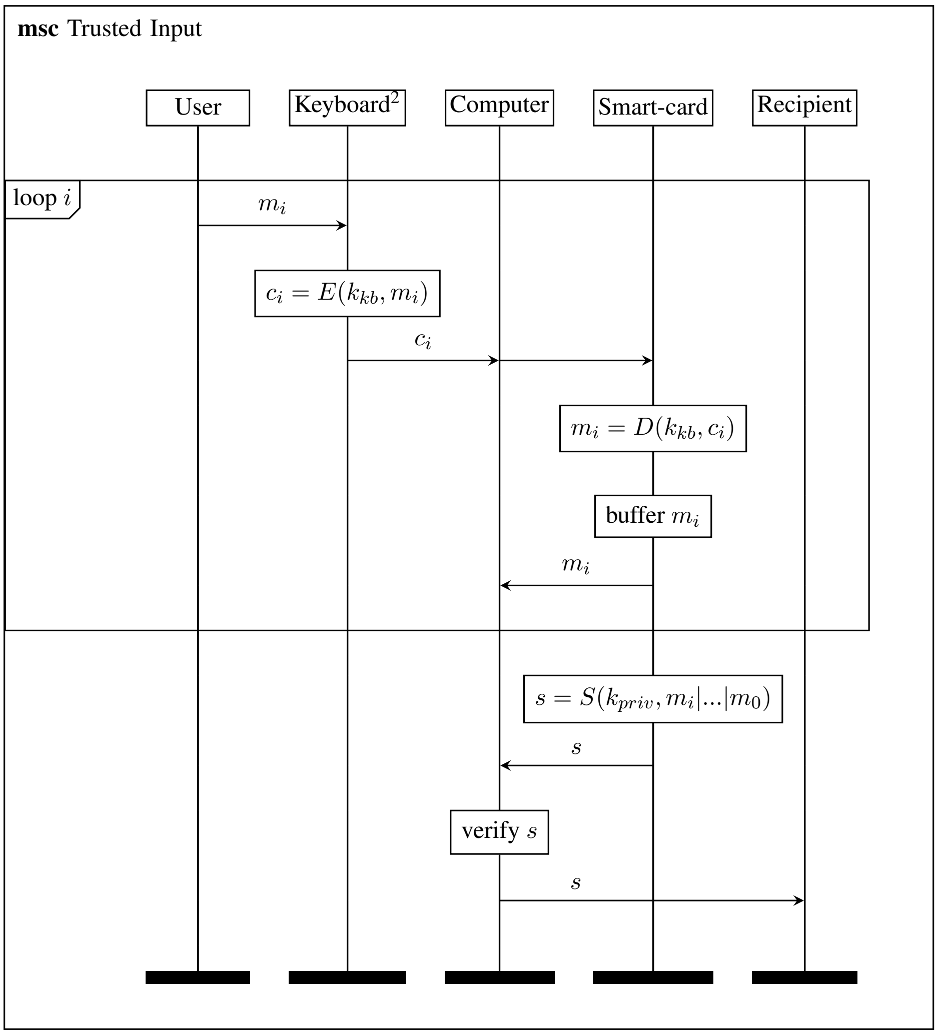
   (repeat steps 1. to 4.)

n. When finished typing, the smart-card signs what it received (not what the computer displayed).

   The computer has the chance to verify the signature before sending it to the server, since it knows clear text, signature, and public key of the smart-card. Thus, it is controlling the smart-card.

Figure 4: Trusted Input: Sequence Diagram

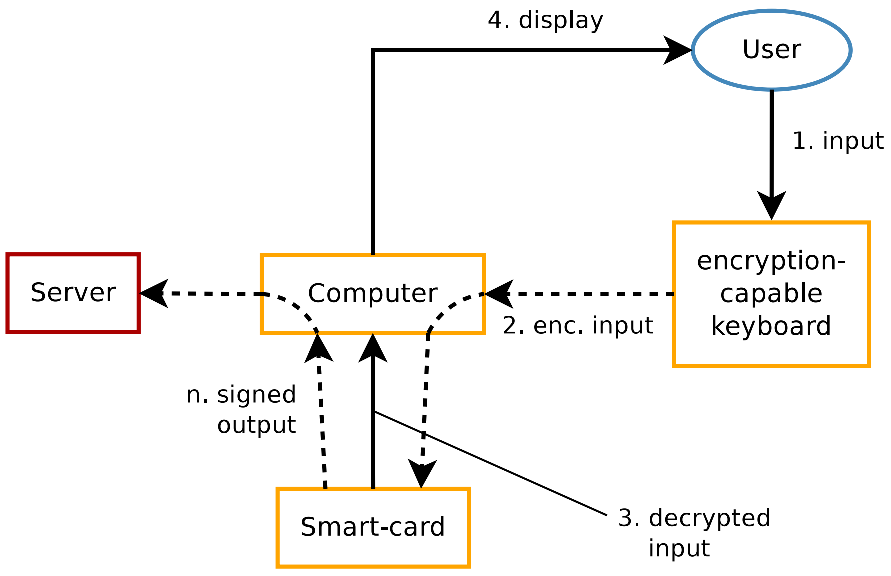
The following figure displays the steps for trusted input. Users type in keystrokes m_i which get encrypted and transmitted to the smart-card. The smart-card signs what it decrypts and, thus, signs-what-you-type. Before sending the signed keystrokes to the recipient (e.g. a server), the computer can verify the signature. $S(k, m)$ denotes a signature of data m with key k, $E(k, m)$ is encryption, and $m_i|...|m_0$ represents a concatenation.



[2]Encryption-capable keyboard

Figure 5: Trusted Input

Blue:	trusted entity
Orange:	partly trusted entity
Red:	untrusted entity



An informal first analysis shows that either computer and smart-card, or computer and keyboard have to be compromised in order to attack the system:

1. Honest computer
   Suppose the computer is honest, then the computer can prevent attacks of the other two devices. The computer is always the last entity in the protocol before the data is sent to the recipient (e.g. a server) and can, thus, reject wrong signatures.

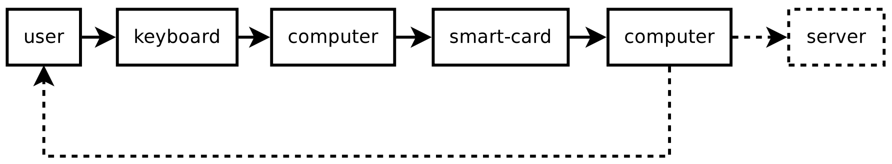
   The smart-card only has two possibilities for attacks:
   It could either sign something different from what was shown to the user or it could deliver something different than the user types. In the first case, the signature would not verify and the computer can safely reject it. The second case will be detected by the user as a mismatch between input and display will be noticed quickly.

   Since the data of the keyboard is only forwarded to the smart-card, malicious behaviour will trigger the same two cases as with dishonest smart-card. In conclusion, the computer can always prevent tampered output.
2. Malicious computer
   If the computer is malicious, the technique only works if smart-card and keyboard are both honest. The keyboard would send the correct keystrokes to the smart-card which will sign them. The computer could randomly tamper

> or swap encrypted keystrokes but this would with high probability produce data without meaning.
>
> When the keyboard is malicious in addition to the computer, then it could send the correct keystrokes to the computer unencrypted while sending the tampered keystrokes to the smart-card. The computer would display the correct input but the smart-card would sign the tampered ones, even if it is honest.
>
> On the other hand, if the keyboard is honest but the smart-card is malicious, the smart-card could simply leak it's keys to the computer. The computer would then be in full control over the whole system and could create signatures arbitrarily.

In summary, the technique works if either computer, or smart-card and keyboard are honest. This becomes clear when we unroll the graph and represent it as a sequence of events (see Fig. 6).

Figure 6: Trusted Input: Sequence of Events



Each single step involves the user twice, once at the beginning and once at the very end, enabling him or her to control the rest of the chain. Users only have limited capabilities and cannot check a signature for example, but in some scenarios this is possible. In our protocol, the user only has to verify if the typed character matches the screen. Commonly, when the screen does not match the expectations, a user would stop typing instantly and search for the error.

An advantage of the technique is that the user-experience does not change as the smart-card only needs to be installed once (e.g. inserting a smart SD-card). Input and display remain the same. While typing, the user can detect manipulations in real-time by comparing keystrokes to the screen. This solves the problem of how the user can identify correct behaviour when dealing with a shared display (an issue mentioned in [74]).

However, the secrecy part of the trusted input needs to be improved. This clearly requires weakening the assumptions as the parts involved in displaying have to be trusted. A TEE could be a useful extension to the smart-card and provide secure output.

To port this solution to a smart-phone without losing usability, we want to swap the encryption-capable keyboard for TrustZone. This approach is less secure as the hardware vendor is trusted. We plan to experiment with TrustZone and smart SD-cards to further improve the technique.

3.4 Enforcing Detection

A smart-card can not only store keys but can also enforce certain policies like writing a log file (Merkle tree) or starting a TEE.

A Merkle tree [72, 61] is a cryptographic append-only data structure, providing proofs that an element exists and proofs for correct extension of the tree. It can serve as a log file.

Suppose a malicious computer rejects to display information to the user, then a Merkle tree could act as a proof of output. A smart-card could only cooperate after receiving a proof that the string to output was written to a Merkle tree. Afterwards, the user is able to view this log file, from another computer if necessary.

If the client is honest, it would write to the Merkle tree in any case (even with malicious smart-card). If the computer is malicious, an honest smart-card would enforce a correct tree. Supplementary, an attacker has to control two independent devices (computer and smart-card) at the same time to tamper with the output.

A smart-card could also exclusively work together with a TEE, e.g. TrustZone, on the computer or phone and enforce the device to start the TEE (instead of imitating it to trick the user). If the TEE is not present, setting up communication between smart-card and TEE (pre-shared key) would fail and the smart-card would block accesses to its secrets.

4 Proposed Research

4.1 Hypothesis: Separation of Trust in a Distributed System

Our hypothesis is that distributed systems can not only prevent computation faults, as e.g. seen in Byzantine Fault Tolerance (Chapter 2.1.1) and computation redundancy, but can also tolerate attacks of devices by dividing security between multiple systems.

Current solutions mostly focus on one sole technique and assume its Trusted Computing Base (TCB) secure. In contrast, we do not assume the basis of any method trusted. Instead, we use several independent TCBs to guarantee that security holds provided at least one of them is honest. We presume with a large enough amount of TCBs the task of compromising all TCBs is infeasible, especially if users randomly choose different manufacturers. Existing approaches are not built for this setup and new techniques are necessary.

4.2 Research Plans

When using several devices, we do not exclusively have to trust one single machine which would leave us vulnerable to the hardware vendor, if not even more potential adversaries.

A second device (mobile, online banking token, etc.) or a second channel (phone line, post, direct contact) hardens the infrastructure against malware as seen in e.g. online banking and electronic voting. It would be beneficial to tolerate attacks in this manner for messaging and (cloud-) storage in particular.

We will have a look at scenarios in the following sections.

4.2.1 Adversary and Threat Model

The adversary in these domains is usually very powerful. E.g. e-mails are target to surveillance by mail providers, Internet Service Providers (ISPs), or governments; electronic voting is nearly purely connected to governments; and information-secrecy (i.e. storage) is subject to secret service and companies as it can give competitive advantage towards others.

Big companies should not be underestimated. Google for example has control over e-mails with gmail, browsers with Chrome, and phones with Android. Same counts for Mozilla with Firefox, Thunderbird, and Firefox OS. That means for certain users, some companies are able to take over online-banking accounts (even though mostly not desired).

Critical infrastructures, as e.g. nuclear power plants (Stuxnet [39]), should not rely on the honesty of manufacturers and companies as recent cases indicate. (Superfish [79])

In these domains, there are two attack routines, mass and targeted attacks (including Advanced Persistent Threats (APTs)). Depending on the underlying scenario, one of both is more likely. E-mails for example are more target to mass surveillance while information secrecy (i.e. spying on data) in a company's context is prone to targeted attacks.

In summary, our adversaries are the most powerful entities: big companies, hardware manufacturers, governments (not limited to the own one or only externals), secret services, cloud providers, ISPs, and more.

We initially assume the adversary has full control over one device (that includes manufacturing, physical hardware access, and software control). The other devices in the system have to compensate for this. Ideally, they compensate for this (1) without knowing that the device is compromised, (2) independent from which device is compromised, and (3) even for more than one compromised device. This is only possible if one device is still honest. If all devices are entirely controlled by an attacker in real-time, security is impossible since the TCB is non-existent.

We will start from the initial assumption and explore to what extend a practical system can fulfil the ideal properties.

4.2.2 Scenarios

A rough classification of scenarios in general can be found in Table 2. Note that availability does not only include virtual availability, i.e. DoS, but also physical availability (see Stuxnet [39]).

As we see in the table, information-secrecy and -manipulation are the main problems among all scenarios. In our work, we will mainly target messaging and e-mail as well as (cloud) storage.

There are further scenarios which one can categorise in this manner. Online-banking is a classical example, but the adversary model is different: Governments do not have to exploit online banking to access bank accounts and big companies mostly already have the payment details (e.g. online shops) or are not interested in them. In effect, the adversary model is different.

If implemented correctly using a hardware token for transaction authentication, integrity is protected to a degree that people are willing to accept. Transaction authentication should not be confused with entity authentication which does not protect the transaction but only verifies the recipient (see Chapter 2.1.5 for a more detailed explanation).

Similar to online banking, electronic voting was designed [62, 29] using hardware tokens. The solution seems to be solid, achieves anonymity, and secures privacy.

Electronic tax return [21] should also be named here. Governments do not have any interest in attacking their own system, removing them from the list of adversaries. Furthermore, strong encryption with a pre-shared key is used to secure the communication which seems to be secure enough for this purpose.

Table 2: Brief Classification of Critical Scenarios

Scenario	Confidentiality	Integrity	Anonymity	Availability
Messaging/E-mail	x	x		
(Cloud) storage	x	x		x
Online banking	x	x		
Electronic voting	x	x	x	x
E-tax return	x	x		

Messaging

Sending a message requires input into a possibly infected device. Our idea is to secure typed data independently from the device. At the same time, we want to enable users to detect modifications in real-time by comparing (keyboard-)input and displayed data.

For receiving messages, we aim to develop a protocol that guarantees the user to limit the impact of malware to messages which have been displayed on the infected machine.

Steps:

1. Research how to deal with input into a (mostly untrusted) distributed system.
2. Explore how a smart-card in combination with an encryption-capable keyboard could help securing the input (see Fig. 5).
3. Further search for combinations of technologies that could help (e.g. Trusted Computing (TC) like SGX and TrustZone).
4. If SGX is available[3]: analyse it towards its possible usage for malware tolerance. Otherwise, examine TrustZone.

Storage

Storing data encrypted with only one key presents attackers with a clear target: the device storing the key. Our goal is to develop an approach without a single point of failure that only decrypts data when the user agrees.

Updating or more general writing data is similar to sending authenticated data to the storage but with additional issues. The party checking the authentication has to be secured and removing data is critical as well.

Steps:

1. Explore the idea of two times encrypted data (see Fig. 2). This way, one identity could control the second one, splitting the original single target of the attacker into two.
2. Check the suitability of a smart SD-card or NFC smart-card as external TEE, storage for a key, etc.
3. Find further ways in dealing with storage. Helpful techniques are searchable encryption, homomorphic encryption, and SEM based [14] revocable encryption.
   - Examine the scenario of writing and removing data of the storage.
   - Try to remove dependent untrusted devices (e.g. the session server) and minimise TCBs.
4. Consider the same techniques as for messaging (e.g. SGX, TrustZone, encryption-capable keyboard, etc.) and further, if applicable.

[3]Intel SGX was not yet released publicly.

4.2.3 Research Questions

Our research focuses on working with an infected system. To address this, we will use multiple devices including partly trusted architectures like smart-cards, TEE, encryption-capable keyboards, etc.

The following questions should be answered by this research:

1. How to split trust into multiple devices such that security is guaranteed provided certain subsets are not colluding or are verifiable trustworthy? (e.g. a valid assumption would be that, at the moment, Microsoft and Huawei are not colluding nor their governments or secret services)
    - How can smart-cards and additional devices help securing an untrusted system in this context?
    - Is it possible to secure input and output on an untrusted machine?
        - How can we restrict attacks to the displayed information only?
        - How can we detect wrong output? How to prevent it?
        - How to secure integrity and privacy of input?
    - How can TrustZone or SGX (if available) contribute to a solution in those scenarios?
        - In case of TrustZone: how to make it secure? (it has no built-in hardware bootloader and all devices connected to the bus have to respect the state of the world)
        - A "reverse-sandbox" would be useful to protect trusted applications from the architecture and the OS. Can that be implemented in SGX (if released) or TrustZone? How to implement it?
2. When trust is distributed over several systems: Can honest and dishonest devices identify each other or each alike?
    - Interception
      Malicious-but-cautious systems only reveal their intentions if they are sure that the other system is also malicious. How to disable dishonest systems from identifying each other? (prevent them from informing each other and, thus, the adversary)
    - Identification
        - Can we identify the honesty of a system if it is potentially malicious? If so, how? (this corresponds to remote attestation and seems impossible without trusted hardware)
        - How to verify (possibly with help of the user or a technician) the honesty of a system?

4.2.4 Contributions and Goals

The aim of this project is to draft a malware-tolerant, distributed system. Three parts are required: (1) trusted input, (2) secure processing, and (3) trusted output. Depending on the scenario and device, secure processing consists of trusted execution and/or secure storage. Vasudevan et al. also considered remote attestation and secure provisioning as vital security properties (see Chapter 2.1.4) but these are unnecessary in our model since we do not trust other devices.

In summary, we strive to make the following contributions by our research:

- A draft of a system, including several devices, that is tolerant to malware and attacks

  This should be a combination of distributed systems, intrusion/fault tolerance, TEE (TrustZone and/or, if possible, SGX), additional devices (phone, smart-card, encryption-capable keyboard, online-banking tokens, ...), software techniques (e.g. Merkle trees) and/or further techniques.

  Note that firmware programmers, manufacturers, service providers, and similar must also be considered as potential attackers and shall not be assumed trusted by default. However, assumptions that different manufacturers and companies do not collude are reasonable if chosen correctly. E.g. Google and Huawei are dependent since Google produces Android which runs on Huawei phones, but Microsoft and Huawei are (to our knowledge) currently independent. Assuming that they are not colluding seems sensible.

  Trusting a single provider might be possible, too, if we can (1) control the results or actions, or (2) verify honesty of device or technique.

- A method for secure, i.e. at least integrity protected, input and output

  Input is naturally only involving one device. Distributing input over multiple devices, like an online banking token and a PC, might be secure but heavily impacts usability. Such a token is suitable for infrequent tasks, like voting and banking, but is inappropriate for frequent tasks, as e.g. messaging.

  In contrast to input, output can be realised in a distributed fashion. An e-mail is read on several devices: e.g received on the computer at work and read again on the phone when being out of office. Another possibility for distributed output is a central secure data structure (e.g. a Merkle tree) which all devices can, at least, read. An output to this data structure can then be displayed by any associated device (distributed output).

- An examination of integrated hardware (e.g. TrustZone or SGX) as support for untrusted distributed systems

  When assuming most of the devices untrusted, TCBs become the building blocks of a system. Since TEEs are a popular source for a TCB, we will examine if and how they can contribute to security in our scenarios.

4.3 Timetable

The next step will be to elaborate the idea of sign-what-you-type (see Table 3). We will implement a proof of concept with a keyboard, an embedded device, a smart-card, and a computer. Afterwards, we plan to examine double encryption and TrustZone or SGX (if available). The next section (Chapter 4.4) introduces our planned papers.

Table 3: Milestones

2014-03-14	Start
2014-05-19	**Report 1**
2014-09-20	**Report 2**
2014-10	Explored "Storage" under the two assumptions
2014-12	Idea: double encryption [Former: Develop a mechanism to only display needed information]
2015-01	Researched formal representation of distributed systems, Idea: Trusted input ("sign-what-you-type")
2015-02/-03	**Report 3:** Thesis proposal
2015-04	Expand trusted input idea, implement a proof of concept
2015-05	**Paper 1:** sign-what-you-type
2015-06	A: Examine TrustZone, try to replace encryption-capable keyboard, research private input
2015-09-20	**Report 4**
2015-10	B: Improve double encryption idea and enhance with smart-card, TrustZone, encryption-capable keyboard, etc.
2015-12	**Paper 2:** TrustZone or double encryption
2016-04-03	**Report 5**
2016-04	Further research A/B depending on their results. Evaluate Intel SGX if it is available. Examine trusted output.
2016-07	**Paper 3:** (possibly SGX or trusted output)
2016-07	Begin writing the dissertation
2016-10-02	**Report 6**
2017-03-13	Planned submission
2017-04-09	Report 7
2017-10-08	Report 8
2018-03-13	Submission deadline

4.4 Outline of First Papers

4.4.1 Trusted Input

As seen in Table 3, we plan to create a paper on trusted input in the next months.

Main parts of the technique we named sign-what-you-type (see also Fig. 5) are

- an encryption-capable keyboard,
- a smart-card,
- and the computer itself.

The encryption-capable keyboard sends each keystroke encrypted to the smart-card which decrypts and buffers it before sending it back to the computer in clear text. When typing is finished, the smart-card signs the data it received and delivers signature and text to the computer. The signature can be verified at any step thereafter. That includes the computer itself which effectively controls the honesty of the smart-card.

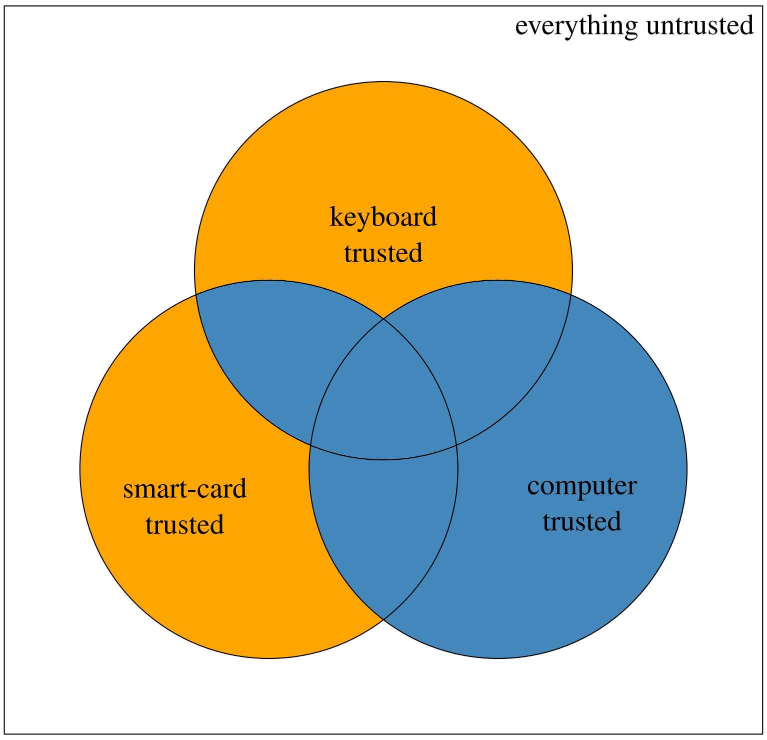
As a result, computer and the combination of keyboard and smart-card are both each one TCB. An attacker has to compromise both TCBs to break integrity, i.e. computer and any of the other two. The technique holds if only one device is malicious or if the computer is honest. In other words, it is secure unless keyboard and computer; or smart-card and computer collude. Fig. 7 shows this graphically. Each circle stands for the device being trusted, that means the intersection between all circles is equal to all devices are honest while the outer area represents that all parts are malicious.

We hope to further improve the untrusted combinations. Our goal is to guarantee integrity if one device is honest. That is, the (untrusted) orange areas in Fig. 7 should be secured, too. In order to harden our infrastructure against colluding devices, we will analyse if a malicious-but-cautious device (out of keyboard, smart-card, and computer) can detect whether any of the other two is also malicious.

Figure 7: Trust Model for Sign-What-You-Type

Circles indicate that the named device is trusted. Intersections stand for more devices are trusted. E.g. the very centre of the diagram means that all devices have to be trusted. The intersection between keyboard and computer without smart-card refers to an untrusted smart-card with trusted keyboard and trusted computer.

Blue:	combination is secure
Orange:	not secure yet



Without trusted output, privacy cannot be protected if we display the data. However, this works for passwords, since we do not need to display them. Another idea to secure confidentiality is to use a TEE like TrustZone to achieve trusted output. When the TEE (and thus the output) is secure, privacy of the input is possible. The smart-card would simple re-encrypt the keystrokes with the key of the TEE and send them to the computer. With a secure TEE, decryption is secure and displaying the data is also secure. The assumption that the entire TEE is trusted is broad and should be improved with better techniques for trusted output. To ensure confidentiality of our idea, we only need secure output, not necessarily a complete TEE.

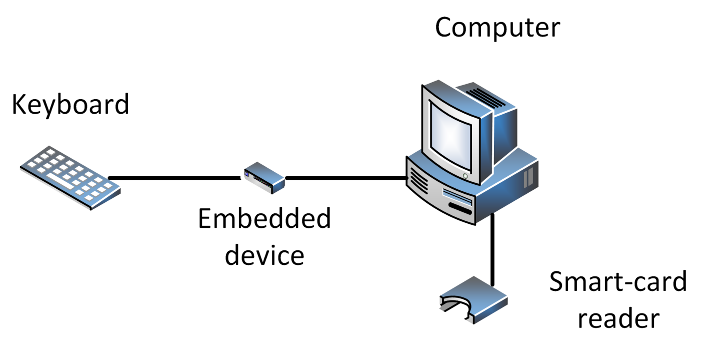
We strive to implement a proof of concept with slight changes to the architecture. To be more flexible, we simulate the encryption-capable keyboard with a normal keyboard and an embedded device, as e.g. a Raspberry Pi, FriendlyARM Mini 6410 SBC[4], or similar. This will not change the approach much but offers flexibil-

[4]The FriendlyARM Mini 6410 SBC apparently does not disable TrustZone. [83]

ity for programming the encryption of, originally, the keyboard.

Our setup will look like the following (see also Fig. 8). A normal keyboard is connected to an embedded device which handles the encryption and is coupled with the computer. We will either use a smart-card reader or a smart-SD-card together with the computer.

Figure 8: Trusted Input: Setup



To test it, we thought of trying to intercept and change keystrokes. We expect that reading keys will be possible when no TEE is present but altering keystrokes shall never succeed.

4.4.2 Ideas for a Second and a Third Paper

Since we plan to implement the trusted input in combination with an embedded device running an ARM chip (Raspberry Pi, FriendlyARM Mini 6410 SBC), the next step will be to examine TrustZone. We hope to either produce a survey of the TrustZone architecture, as there is not much literature on this, or enhancing the chosen scenarios with the TrustZone TEE.

The third idea, double encryption, requires elaboration before it is suitable for publication. A TEE like TrustZone could replace the smart-card or session-server in our idea improving usability and security. We will not only focus on TEE but also SEM enhanced revocable cryptography [14] and search for further technologies.

5 Conclusion

This work shall be the first step into a different direction. Instead of defending against malware, we aim to accept that some devices are infected but nevertheless want to use them for security critical scenarios.

Our next steps will be to examine the ideas of double-encrypted data and trusted input, implement a proof of concept, and see how TrustZone can contribute.

5.1 Challenges and Problems

Since the idea of malware tolerance is a new way to approach the problem, it is not clear if it is possible in general scenarios. This work will examine this hypothesis.

A challenge, even though we do not particularly aim at it, is usability. When using a token to secure online banking, we achieve high security at the cost of usability. Assuming we port this technique to e-mails, we might be able to secure them, but no one would accept creating a TAN for every single e-mail. This might be possible for very critical e-mails, even though we doubt this. As a result, our systems also need "acceptable" usability.

The trustworthiness of TEEs is another issue since most of them are produced by the same manufacturer, like e.g. SGX, IPT, and TrustZone. In our suspicious adversary model, those chips might be equal to the bare device without TEE, depending on manufacturers and how the parts are assembled. However, in any case TEE might prevent other adversaries like software or firmware attacks.

5.2 Possible Outcome

We are confident, that our idea of trusted input can be extended to a publication. The next steps will be to realise this and further improve the idea by examining TEEs and additional techniques for their possible benefit. We are keen to analyse TrustZone as it seems to be omitted in previous research.

References

[1] Manal Adham, Amir Azodi, Yvo Desmedt, and Ioannis Karaolis. How to attack two-factor authentication internet banking. In *Financial Cryptography and Data Security*, pages 322–328. Springer, 2013. URL `http://link.springer.com/chapter/10.1007/978-3-642-39884-1_27`. accessed: 2014-06-26.

[2] Pieter Agten, Raoul Strackx, Bart Jacobs, and Frank Piessens. Secure compilation to modern processors. In *Computer Security Foundations Symposium (CSF), 2012 IEEE 25th*, pages 171–185. IEEE, 2012. URL `http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=6266159`. accessed: 2014-07-24.

[3] Bas Alberts. Dr linux 2.6 rootkit released. online, September 2008. URL `http://seclists.org/dailydave/2008/q3/215`. accessed: 2015-02-16.

[4] Ittai Anati, Shay Gueron, Simon P Johnson, and Vincent R Scarlata. Innovative technology for cpu based attestation and sealing. In *Proceedings of the 2nd International Workshop on Hardware and Architectural Support for Security and Privacy, HASP*, volume 13, 2013. URL `https://software.intel.com/en-us/articles/innovative-technology-for-cpu-based-attestation-and-sealing`. accessed: 2014-04-17.

[5] ARM. Building a secure system using trustzone technology. Technical report, ARM Security Technology, April 2009. URL `http://infocenter.arm.com/help/topic/com.arm.doc.prd29-genc-009492c/PRD29-GENC-009492C_trustzone_security_whitepaper.pdf`. accessed: 2015-02-26.

[6] ARM. *ARM Architecture Reference Manual ARMv8, for ARMv8-A architecture profile*. ARM Limited, 110 Fulbourn Road, Cambridge, England CB1 9NJ, v8 edition, December 2013. URL `https://silver.arm.com/download/ARM_and_AMBA_Architecture/AR150-DA-70000-r0p0-00bet3/DDI0487A_b_armv8_arm.pdf`. accessed: 2014-06-24.

[7] ARM. Arm trustzone website. online, 2014. URL `http://www.arm.com/products/processors/technologies/trustzone/index.php?tab=Hardware+Architecture`. accessed: 2014-06-24.

[8] Ahmed M Azab, Peng Ning, Jitesh Shah, Quan Chen, Rohan Bhutkar, Guruprasad Ganesh, Jia Ma, and Wenbo Shen. Hypervision across worlds: Real-time kernel protection from the arm trustzone secure world. In *Proceedings*

of the 2014 ACM SIGSAC Conference on Computer and Communications Security, pages 90–102. ACM, 2014. URL `https://dl.acm.org/citation.cfm?id=2660350`. accessed: 2015-02-17.

[9] Banks Germany. mtan / sms tan. online, 2014. URL `http://banks-germany.com/information/internet-banking/mtan-sms-tan`. accessed: 2015-03-13.

[10] Banks Germany. Chip-tan procedure. online, 2014. URL `http://banks-germany.com/information/internet-banking/chip-tan-procedure`. accessed: 2015-03-13.

[11] Boaz Barak, Oded Goldreich, Rusell Impagliazzo, Steven Rudich, Amit Sahai, Salil Vadhan, and Ke Yang. On the (im)possibility of obfuscating programs. In *Advances in Cryptology—CRYPTO 2001*, pages 1–18. Springer, 2001. URL `http://link.springer.com/chapter/10.1007%2F3-540-44647-8_1`. accessed: 2014-04-02.

[12] Boaz Barak, Sanjam Garg, Yael Tauman Kalai, Omer Paneth, and Amit Sahai. Protecting obfuscation against algebraic attacks. Technical report, Cryptology ePrint Archive, February 2014. URL `http://eprint.iacr.org/2013/631.pdf`. accessed: 2014-04-02.

[13] Ravi Bhargava, Benjamin Serebrin, Francesco Spadini, and Srilatha Manne. Accelerating two-dimensional page walks for virtualized systems. *ACM Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 42(2):26–35, 2008. URL `https://dl.acm.org/citation.cfm?id=1346286`. accessed: 2014-11-24.

[14] Dan Boneh, Xuhua Ding, Gene Tsudik, and Chi-Ming Wong. A method for fast revocation of public key certificates and security capabilities. In *USENIX Security Symposium*. Usenix, 2001. URL `http://static.usenix.org/publications/library/proceedings/sec01/full_papers/boneh/boneh.ps`. accessed: 2015-02-17.

[15] Christoph Bösch, Pieter Hartel, Willem Jonker, and Andreas Peter. A survey of provably secure searchable encryption. *ACM Computing Surveys (CSUR)*, 47(2):18, 2014. URL `https://dl.acm.org/citation.cfm?id=2636328`. accessed: 2015-01-29.

[16] Claus Boyens and Oliver Günther. Using online services in untrusted environments: a privacy-preserving architecture. In *ECIS*, pages 239–250, 2003. URL `sdaw.info/asp/aspecis/20040017.pdf`. accessed: 2014-04-07.

[17] David Champagne and Ruby B Lee. Scalable architectural support for trusted software. In *High Performance Computer Architecture (HPCA), 2010 IEEE 16th International Symposium on*, pages 1–12. IEEE, 2010. URL `http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=5416657`. accessed: 2014-07-24.

[18] Fangzhe Chang, Ayal Itzkovitz, and Vijay Karamcheti. User-level resource - constrained sandboxing. In *Proceedings of the 4th USENIX Windows Systems Symposium*, volume 91, 2000. URL `https://www.usenix.org/legacy/events/usenix-win2000/full_papers/chang/chang.pdf`. accessed: 2014-04-17.

[19] Yueqiang Cheng, Xuhua Ding, and Robert H Deng. Driverguard: A fine-grained protection on i/o flows. In *Computer Security–ESORICS 2011*, pages 227–244. Springer, Berlin, Heidelberg, 2011. URL `http://link.springer.com/chapter/10.1007/978-3-642-23822-2_13`. accessed: 2014-11-24.

[20] Yves Deswarte, Laurent Blain, and J-C Fabre. Intrusion tolerance in distributed computing systems. In *Research in Security and Privacy, 1991. Proceedings., 1991 IEEE Computer Society Symposium on*, pages 110–121. IEEE, 1991. URL `http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=130780`. accessed: 2014-04-08.

[21] ELSTER. Information zu den sicherheitsverfahren (information about the security [of elster]). online, 2014. URL `https://www.elsteronline.de/eportal/Sicherheit.tax`. accessed: 2014-08-25.

[22] Michael Farb, Manish Burman, G Chandok, Jon McCune, and Adrian Perrig. Safeslinger: An easy-to-use and secure approach for human trust establishment. Technical report, Technical Report CMU-CyLab-11-021, Carnegie Mellon University, 2011. URL `http://dl.acm.org/citation.cfm?id=2500428`. accessed: 2014-04-24.

[23] Massimo Ficco and Massimiliano Rak. Intrusion tolerance of stealth dos attacks to web services. In *Information Security and Privacy Research*, pages 579–584. Springer, 2012. URL `http://link.springer.com/chapter/10.1007/978-3-642-30436-1_52`. accessed: 2014-04-08.

[24] Atanas Filyanov, Jonathan M McCuney, A-R Sadeghiz, and Marcel Winandy. Uni-directional trusted path: Transaction confirmation on just one device. In *Dependable Systems & Networks (DSN), 2011 IEEE/IFIP 41st International Conference on*, pages 1–12. IEEE, 2011. URL `http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=5958202`. accessed: 2014-11-25.

[25] Miguel Garcia, Alysson Bessani, Ilir Gashi, Nuno Neves, and Rafael Obelheiro. Os diversity for intrusion tolerance: Myth or reality? In *Dependable Systems & Networks (DSN), 2011 IEEE/IFIP 41st International Conference on*, pages 383–394, 2011. URL http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=5958251. accessed: 2014-04-08.

[26] Miguel Garcia, Alysson Bessani, Ilir Gashi, Nuno Neves, and Rafael Obelheiro. Analysis of operating system diversity for intrusion tolerance. *Software: Practice and Experience*, 44:735–770, January 2013. doi: 10.1002/spe.2180. URL http://onlinelibrary.wiley.com/doi/10.1002/spe.2180/full. accessed: 2014-04-08.

[27] Tal Garfinkel, Ben Pfaff, Jim Chow, Mendel Rosenblum, and Dan Boneh. Terra: A virtual machine-based platform for trusted computing. In *ACM SIGOPS Operating Systems Review*, volume 37, pages 193–206, 2003. URL http://dl.acm.org/citation.cfm?doid=1165389.945464. accessed: 2014-04-17.

[28] Sanjam Garg, Craig Gentry, Shai Halevi, Mariana Raykova, Amit Sahai, and Brent Waters. Candidate indistinguishability obfuscation and functional encryption for all circuits. In *Foundations of Computer Science (FOCS), 2013 IEEE 54th Annual Symposium*, pages 40–49. IEEE, July 2013. URL http://eprint.iacr.org/2013/451.pdf. accessed: 2014-04-02.

[29] Gurchetan S. Grewal, Mark D. Ryan, Liqun Chen, and Michael R. Clarkson. Du-vote: Remote electronic voting with untrusted computers. In *In Proceedings of the 28th IEEE Computer Security Foundations Symposium (CSF 2015)*, 2015.

[30] Guofei Gu, Phillip Porras, Vinod Yegneswaran, Martin Fong, and Wenke Lee. Bothunter: Detecting malware infection through ids-driven dialog correlation. In *Proceedings of 16th USENIX Security Symposium on USENIX Security Symposium*, page 12. USENIX association, 2007. URL http://static.usenix.org/legacy/events/sec07/tech/full_papers/gu/gu_html/. accessed: 2014-04-17.

[31] Matthew Hoekstra. Intel sgx for dummies (intel sgx design objectives). online, September 2013. URL https://software.intel.com/en-us/blogs/2013/09/26/protecting-application-secrets-with-intel-sgx. accessed: 2015-02-16.

[32] Matthew Hoekstra, Reshma Lal, Pradeep Pappachan, Vinay Phegade, and Juan Del Cuvillo. Using innovative instructions to create trustworthy software solutions. In *Proceedings of the 2nd International Workshop on Hardware and Architectural Support for Security and Privacy, HASP*, volume 13, 2013. URL https://software.intel.com/sites/default/files

/article/413938/hasp-2013-innovative-instructions-for-trusted-solutions.pdf. accessed: 2014-04-17.

[33] Grégoire Jacob, Hervé Debar, and Eric Filiol. Behavioral detection of malware: from a survey towards an established taxonomy. In *Journal in computer Virology - Springer* Willems et al. [81], pages 251–266. doi: http://dx.doi.org/10.1109/MSP.2007.45. URL http://link.springer.com/article/10.1007/s11416-008-0086-0. accessed: 2014-04-17.

[34] James E Just, James C Reynolds, Larry A Clough, Melissa Danforth, Karl N Levitt, Ryan Maglich, and Jeff Rowe. Learning unknown attacks - a start. In *Recent Advances in Intrusion Detection*, pages 158–176. Springer, 2002. URL http://link.springer.com/chapter/10.1007/3-540-36084-0_9. accessed: 2014-04-08.

[35] Kaspersky. Teamwork: How the zitmo trojan bypasses online banking security. online, October 2011. URL http://www.kaspersky.com/about/news/virus/2011/Teamwork_How_the_ZitMo_Trojan_Bypasses_Online_Banking_Security. accessed: 2015-02-16.

[36] Bernhard Kauer. Oslo: Improving the security of trusted computing. In *Proceedings of the USENIX Security Symposium*, volume 24, page 173, 2007. URL http://static.usenix.org/event/sec07/tech/full_papers/kauer/kauer_html/. accessed: 2014-04-17.

[37] Spencer Kelly. Hackers outwit online banking identity security systems. online, February 2012. URL http://www.bbc.co.uk/news/technology-16812064. accessed: 2015-03-13.

[38] Liwei Kuang and Mohammad Zulkernine. An intrusion-tolerant mechanism for intrusion detection systems. In *Availability, Reliability and Security 2008*, pages 319–326. IEEE, 2008. URL http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=4529353. accessed: 2014-09-03.

[39] Ralph Langner. Stuxnet: Dissecting a cyberwarfare weapon. *Security & Privacy, IEEE*, 9:49–51, 2011. URL http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=5772960. accessed: 2014-08-06.

[40] Ruby B Lee, Peter CS Kwan, John P McGregor, Jeffrey Dwoskin, and Zhenghong Wang. Architecture for protecting critical secrets in microprocessors. In *Computer Architecture, 2005. ISCA'05. Proceedings. 32nd International Symposium on*, pages 2–13. IEEE, 2005. URL http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1431541. accessed: 2014-07-24.

[41] Yanlin Li, Jonathan McCune, James Newsome, Adrian Perrig, Brandon Baker, and Will Drewry. Minibox: A two-way sandbox for x86 native code.

In *2014 USENIX Annual Technical Conference*. USENIX Association, 2014. URL `https://128.2.134.25/group/pub/liatc14.pdf`. accessed: 2014-08-12.

[42] David Lie, Chandramohan Thekkath, Mark Mitchell, Patrick Lincoln, Dan Boneh, John Mitchell, and Mark Horowitz. Architectural support for copy and tamper resistant software. *ACM SIGPLAN Notices*, 35(11):169–177, 2000. URL `http://dl.acm.org/citation.cfm?id=357005`. accessed: 2014-07-24.

[43] David J Malan. Cs50 sandbox: secure execution of untrusted code. In *Proceeding of the 44th ACM technical symposium on Computer science education*, pages 141–146. ACM, 2013. URL `https://dl.acm.org/citation.cfm?id=2445242`. accessed: 2015-03-11.

[44] Mohammad Mannan and Paul C van Oorschot. Using a personal device to strengthen password authentication from an untrusted computer. In *Financial Cryptography and Data Security*, pages 88–103. Springer, 2007. URL `link.springer.com/chapter/10.1007/978-3-540-77366-5_11`. accessed: 2014-04-07.

[45] Jonathan M McCune, Adrian Perrig, and Michael K Reiter. Bump in the ether: A framework for securing sensitive user input. In *Proceedings of the annual conference on USENIX'06 Annual Technical Conference*, pages 17–17. USENIX Association, 2006. URL `http://dl.acm.org/citation.cfm?id=1267376`. accessed: 2014-08-12.

[46] Jonathan M McCune, Bryan Parno, Adrian Perrig, Michael K Reiter, and Arvind Seshadri. Minimal tcb code execution. In *Security and Privacy, 2007. SP'07. IEEE Symposium*, pages 267–272. IEEE, 2007. URL `http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=4223231`. accessed: 2014-08-12.

[47] Jonathan M McCune, Bryan J Parno, Adrian Perrig, Michael K Reiter, and Hiroshi Isozaki. Flicker: An execution infrastructure for tcb minimization. In *ACM SIGOPS Operating Systems Review*, volume 42, pages 315–328. ACM, 2008. URL `http://dl.acm.org/citation.cfm?id=1352625`. accessed: 2014-04-30.

[48] Jonathan M McCune, Adrian Perrig, and Michael K Reiter. Safe passage for passwords and other sensitive data. In *Proceeding of the 16th Annual Network and Distributed System Security Symposium*, February 2009. URL `http://www.cs.unc.edu/~reiter/papers/2009/NDSS.pdf`. accessed: 2014-11-26.

[49] Jonathan M McCune, Yanlin Li, Ning Qu, Zongwei Zhou, Anupam Datta, Virgil Gligor, and Adrian Perrig. Trustvisor: Efficient tcb reduction and attestation. In *Security and Privacy (SP), 2010 IEEE Symposium*, pages 143–158. IEEE, 2010. URL `http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=5504713`. accessed: 2014-08-12.

[50] Frank McKeen, Ilya Alexandrovich, Alex Berenzon, Carlos V Rozas, Hisham Shafi, Vedvyas Shanbhogue, and Uday R Savagaonkar. Innovative instructions and software model for isolated execution. *HASP*, 133:10, 2013. URL `http://css.csail.mit.edu/6.858/2013/readings/intel-sgx.pdf`. accessed: 2014-04-17.

[51] Manoranjan Mohanty, Viktor Do, and Christian Gehrmann. Media data protection during execution on mobile platforms – a review. Technical report, SICS Swedish ICT AB, 2014. URL `http://soda.swedish-ict.se/5685/1/Media_Data_Protection_during_Execution_on_Mobile_Platforms_%E2%80%93_A_Review.pdf`. accessed: 2014-09-01.

[52] Steven J Murdoch. Introduction to trusted execution environments (tee). online, 2014. URL `https://www.cl.cam.ac.uk/~sjm217/talks/rhul14tee.pdf`. accessed: 2014-04-17.

[53] Arvind Narayanan, Vincent Toubiana, Solon Barocas, Helen Nissenbaum, and Dan Boneh. A critical look at decentralized personal data architectures. *arXiv preprint (CoRR Journal)*, abs/1202.4503, February 2012. URL `http://arxiv.org/abs/1202.4503`. accessed: 2014-05-16.

[54] Karsten Nohl and Jakob Lehl. Badusb–on accessories that turn evil. online (Black Hat 2014), 2014. URL `https://srlabs.de/blog/wp-content/uploads/2014/07/SRLabs-BadUSB-BlackHat-v1.pdf`. accessed: 2015-02-09.

[55] Job Noorman, Pieter Agten, Wilfried Daniels, Raoul Strackx, Anthony Van Herrewege, Christophe Huygens, Bart Preneel, Ingrid Verbauwhede, and Frank Piessens. Sancus: Low-cost trustworthy extensible networked devices with a zero-software trusted computing base. In *USENIX Security*, pages 479–494, 2013. URL `https://www.usenix.org/sites/default/files/conference/protected-files/noorman_sec13_slides.pdf`. accessed: 2014-07-24.

[56] Bryan Parno, Jacob R Lorch, John R Douceur, James Mickens, and Jonathan M McCune. Memoir: Practical state continuity for protected modules. In *Security and Privacy (SP), 2011 IEEE Symposium*, pages 379–394. IEEE, 2011. URL `http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=5958041`. accessed: 2014-08-12.

[57] Marco Patrignani, Dave Clarke, and Frank Piessens. Secure compilation of object-oriented components to protected module architectures. In *Programming Languages and Systems*, pages 176–191. Springer, 2013. URL `http://link.springer.com/chapter/10.1007/978-3-319-03542-0_13`. accessed: 2014-07-24.

[58] PC Welt. Sicheres online-banking - so geht's. online, October 2011. URL `http://www.pcwelt.de/ratgeber/Anzeige-Sicheres-Online-Banking-so-funktioniert-es-3449782.html`. accessed: 2015-03-14.

[59] Martin Pirker and Daniel Slamanig. A framework for privacy-preserving mobile payment on security enhanced arm trustzone platforms. In *Trust, Security and Privacy in Computing and Communications (TrustCom), 2012 IEEE 11th International Conference on*, pages 1155–1160. IEEE, 2012. URL `http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=6296107`. accessed: 2015-02-17.

[60] Vassilis Prevelakis and Diomidis Spinellis. Sandboxing applications. In *USENIX Annual Technical Conference, FREENIX Track*, pages 119–126. Citeseer, 2001. URL `http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.22.4424&rep=rep1&type=pdf`. accessed: 2014-03-26.

[61] Mark D. Ryan. Enhanced certificate transparency and end-to-end encrypted mail. In *Network and Distributed System Security (NDSS) Symposium*. University of Birmingham, UK, CloudTomo Ltd., February 2014. URL `http://dx.doi.org/10.14722/ndss.2014.23379`. URL was not accessable. Paper received directly from Prof. Ryan.

[62] Mark D. Ryan, Gurchetan Grewal, Michael Clarkson, and Liqun Chen. Du-vote: Remote electronic voting with untrusted computers (invited talk). talk; abstract online, June 2014. URL `http://ceur-ws.org/Vol-1158/fms_proceedings.pdf`. accessed: 2015-03-16.

[63] Amit Sahai and Brent Waters. How to use indistinguishability obfuscation: Deniable encryption, and more. Technical report, IACR Cryptology ePrint Archive, 2013. URL `http://eprint.iacr.org/2013/454.pdf`. accessed: 2014-04-02.

[64] Reiner Sailer, Xiaolan Zhang, Trent Jaeger, and Leendert Van Doorn. Design and implementation of a tcg-based integrity measurement architecture. In *USENIX Security Symposium*, volume 13, pages 16–33, 2004. URL `https://www.usenix.org/legacy/events/sec04/tech/full_papers/sailer/sailer.pdf`. accessed: 2014-04-17.

[65] Samsung. Samsung knox - white paper : An overview of samsung knox. Technical report, Samsung Electronics Co. Ltd., September 2013. URL `https://www.samsungknox.com/en/system/files/whitepaper/files/MP_KNOX_Overview_Whitepaper_V1.0_20131023.pdf`. accessed: 2014-06-23.

[66] Samsung. Samsung knox (website). online, October 2013. URL `https://www.samsung.com/uk/business/solutions-services/mobile-solutions/security/samsung-knox`. accessed: 2014-06-23.

[67] Samsung. Samsung knox - technical details. online, 2014. URL `https://www.samsungknox.com/en/solutions/knox/technical`. accessed: 2014-06-23.

[68] Nuno Santos, Himanshu Raj, Stefan Saroiu, and Alec Wolman. Using arm trustzone to build a trusted language runtime for mobile applications. In *Proceedings of the 19th international conference on Architectural support for programming languages and operating systems (ASPLOS)*, pages 67–80. ACM, 2014. URL `https://dl.acm.org/citation.cfm?id=2541949`. accessed: 2015-02-17.

[69] Raoul Strackx and Frank Piessens. Fides: Selectively hardening software application components against kernel-level or process-level malware. In *Proceedings of the 2012 ACM conference on Computer and communications security*, pages 2–13. ACM, 2012. URL `http://dl.acm.org/citation.cfm?id=2382200`. accessed: 2014-07-24.

[70] G Edward Suh, Dwaine Clarke, Blaise Gassend, Marten Van Dijk, and Srinivas Devadas. Aegis: architecture for tamper-evident and tamper-resistant processing. In *Proceedings of the 17th annual international conference on Supercomputing*, pages 160–171. ACM, 2003. URL `http://dl.acm.org/citation.cfm?id=782838`. accessed: 2014-07-24.

[71] Jakub Szefer and Ruby B Lee. Architectural support for hypervisor-secure virtualization. In *ACM SIGARCH Computer Architecture News* Lee et al. [40], pages 437–450. URL `http://dl.acm.org/citation.cfm?id=2151022`. accessed: 2014-07-24.

[72] Michael Szydlo. Merkle tree traversal in log space and time. In *Advances in Cryptology-EUROCRYPT 2004*, pages 541–554. Springer, 2004. URL `http://link.springer.com/chapter/10.1007/978-3-540-24676-3_32`. accessed: 2015-03-18.

[73] JD Tygar and Bennet S Yee. Strongbox: A system for self securing programs. *CMU Computer Science: 25th Anniversary Commemorative. Addison-Wesley*, 199(1):163–197, 1991. URL

http://www.eecs.berkeley.edu/~tygar/papers/System_for_self_securing_programs.pdf. accessed: 2014-04-17.

[74] Roland van Rijswijk-Deij and Erik Poll. Using trusted execution environments in two-factor authentication: comparing approaches. In *Open Identity Summit 2013, OID 2013*, volume P-223 of *Lecture notes in informatics*, pages 20–31, Bonn, Germany, September 2013. URL http://doc.utwente.nl/91957/. accessed: 2015-02-26.

[75] Vasco. Card reader digipass. online, 2015. URL https://www.vasco.com/products/client_products/card_reader_digipass/connectable-card-reader-digipass.aspx. accessed: 2015-03-13.

[76] Amit Vasudevan, Emmanuel Owusu, Zongwei Zhou, James Newsome, and Jonathan M McCune. *Trustworthy Execution on Mobile Devices: What security properties can my mobile platform give me?*, volume 7344 of *Trust and Trustworthy Computing, CyLab, Carnegie Mellon University, USA*. Springer, November 2012. doi: 10.1007/978-3-642-30921-2_10. URL https://www.cylab.cmu.edu/files/pdfs/tech_reports/CMUCyLab11023.pdf. accessed: 2014-06-23.

[77] Amit Vasudevan, Sagar Chaki, Limin Jia, Jonathan McCune, James Newsome, and Anupam Datta. Design, implementation and verification of an extensible and modular hypervisor framework. In *Security and Privacy (SP), 2013 IEEE Symposium*, pages 430–444. IEEE, 2013. URL http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=6547125. accessed: 2014-08-12.

[78] Giuliana Santos Veronese, Miguel Correia, Alysson Neves Bessani, Lau Cheuk Lung, and Paulo Verissimo. Efficient byzantine fault-tolerance. *Computers, IEEE Transactions on*, 62(1):16–30, 2013. URL http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=6081855. accessed: 2014-04-08.

[79] Jane Wakefield. Lenovo taken to task over 'malicious' adware (bbc). online, February 2015. URL http://www.bbc.co.uk/news/technology-31533028. accessed: 2015-02-20.

[80] Yhaohui Wang and Angelos Stavrou. Exploiting smart-phone usb connectivity for fun and profit. In *Proceedings of the 26th Annual Computer Security Applications Conference*, pages 357–366. ACM, 2010. URL delivery.acm.org/10.1145/1930000/1920314/p357-wang.pdf. accessed: 2014-04-07.

[81] Carsten Willems, Thorsten Holz, and Felix Freiling. Toward automated dynamic malware analysis using cwsandbox. *IEEE Security and Privacy*, 5

(2):32–39, 2007. doi: http://dx.doi.org/10.1109/MSP.2007.45. URL http://dl.acm.org/citation.cfm?id=1262675. accessed: 2014-04-17.

[82] Johannes Winter. Trusted computing building blocks for embedded linux-based arm trustzone platforms. In *Proceedings of the 3rd ACM workshop on Scalable trusted computing*, pages 21–30. ACM, 2008. URL https://dl.acm.org/citation.cfm?id=1456460. accessed: 2015-02-26.

[83] Johannes Winter. Experimenting with arm trustzone–or: How i met friendly piece of trusted hardware. In *Trust, Security and Privacy in Computing and Communications (TrustCom), 2012 IEEE 11th International Conference on*, pages 1161–1166. IEEE, 2012. URL http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=6296108. accessed: 2015-02-17.

[84] Gao Xiaopeng, Wang Sumei, and Chen Xianqin. Vnss: a network security sandbox for virtual computing environment. In *Information Computing and Telecommunications (YC-ICT), 2010 IEEE Youth Conference on*, pages 395–398. IEEE, 2010. URL http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=5713128. accessed: 2014-04-17.

[85] Zongwei Zhou, Virgil D Gligor, James Newsome, and Jonathan M McCune. Building verifiable trusted path on commodity x86 computers. In *Security and Privacy (SP), 2012 IEEE Symposium on*, pages 616–630, 2012. URL http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=6234440. accessed: 2014-11-13.

Appendices

A Workshops

The following table (Table 4) shows a list of conferences and workshops I attended:

Table 4: Workshops

Date	Workshop/Conference	Location
7th May 2014	Google Hack Day (Certificate Transparency)	Google London
19th-23rd May 2014	Academic Writing Seminar	University of Birmingham
27th-28th May 2014	CryptoForma 2014	University of York
14th-18th Jul 2014	Enterprise Summer School	University of Birmingham
23rd-24th Jul 2014	Publishing Academic Journals, Editing your writing	University of Birmingham
22nd Oct 2014	Time Management	University of Birmingham
Oct 2014 - Jan 2015	Research Skills Seminar	University of Birmingham
18th Nov 2014	Speed Reading	University of Birmingham
24th Nov 2014	Note Taking	University of Birmingham
2014	Cryptography 1	University of Stanford (Coursera)
2014 - 2015	Writing in the Sciences	University of Stanford (Coursera)
14th-15th Jan 2015	CryptoForma 2015	University of Kent