Department of Computer Science

Trusted Execution Environments in Cloud Computing

Arseny Kurnikov

Aalto University

DOCTORAL DISSERTATIONS

Aalto University publication series
DOCTORAL DISSERTATIONS 171/2021

Trusted Execution Environments in Cloud Computing

Arseny Kurnikov

A doctoral dissertation completed for the degree of Doctor of Science (Technology) to be defended, with the permission of the Aalto University School of Science, at a public examination held at the lecture hall TU1 of the school on 21 December 2021 at 12am.

Aalto University
School of Science
Department of Computer Science
Secure Systems

Supervising professor
Professor N. Asokan, Aalto University, Finland

Thesis advisor
Doctor Andrew Paverd, Microsoft Research, England

Preliminary examiners
Professor Liqun Chen, University of Surrey, England
Doctor Raja Akram, University of Aberdeen, Scotland

Opponent
Professor Mika Ylianttila, University of Oulu, Finland

Aalto University publication series
DOCTORAL DISSERTATIONS 171/2021

ISBN 978-952-64-0618-3 (printed)
ISBN 978-952-64-0619-0 (pdf)
ISSN 1799-4934 (printed)
ISSN 1799-4942 (pdf)
http://urn.fi/URN:ISBN:978-952-64-0619-0

Unigrafia Oy
Helsinki 2021

Finland

NORDIC SWAN ECOLABEL
Printed matter
4041-0619

Aalto University

Abstract

Aalto University, P.O. Box 11000, FI-00076 Aalto www.aalto.fi

Author
Arseny Kurnikov

Name of the doctoral dissertation
Trusted Execution Environments in Cloud Computing

Publisher School of Science

Unit Department of Computer Science

Series Aalto University publication series DOCTORAL DISSERTATIONS 171/2021

Field of research Hardware security

Manuscript submitted 29 October 2021 **Date of the defence** 21 December 2021

Permission for public defence granted (date) 12 November 2021 **Language** English

☐ **Monograph** ☒ **Article dissertation** ☐ **Essay dissertation**

Abstract

Cloud technologies have become ubiquitous, being widely used by a variety of users: from individuals to large corporations. Without a doubt, clouds offer major benefits that make them attractive, such as elasticity, scalability, availability, and reliability. However, users' data privacy and security are two main concerns when data is processed in the cloud. Currently, users have to trust cloud providers for processing their data securely, not disclosing it to third parties maliciously or by mistake, and applying best security practices. Largely this trust is based on the cloud provider's reputation: it is in the interest of cloud providers to take security seriously; otherwise, they may lose customers. However this trust model does not provide the technical means of security assurance.

Trusted Execution Environments (TEEs) can be used to change the situation. It is a very promising technology that enables secure computation on a remote host. Applying TEEs to cloud environments changes the trust model: customers no longer need to rely on the reputation of the cloud provider. The main reason is that a TEE provides the ability to execute arbitrary code in isolation from the underlying operating system or the hypervisor. Additionally, it is possible to remotely verify what code is running inside a TEE. The trust is based on the hardware and technically it is moved from the cloud provider to the hardware manufacturer.

This work addresses the following challenges when developing TEE applications and deploying TEEs in the cloud. First, how to build scalable applications while maintaining TEE security guarantees? Second, how to reconcile hardware-based TEEs with existing cloud practices? Third, how to utilize TEEs to provide trustworthy resource measurements in the cloud?

The contributions of this dissertation are developing two scalable Trusted Applications for cloud environments, a TEE migration framework to support Virtual Machine migration in the cloud, and a secure resource measurement framework based on TEEs. The contributions form an important step forward towards a wider deployment of TEEs in the cloud that opens opportunities for cloud providers and their clients to benefit from assurance based on hardware and stronger security guarantees.

Keywords TEE, cloud computing, remote attestation

ISBN (printed) 978-952-64-0618-3 **ISBN (pdf)** 978-952-64-0619-0

ISSN (printed) 1799-4934 **ISSN (pdf)** 1799-4942

Location of publisher Helsinki **Location of printing** Helsinki **Year** 2021

Pages 133 **urn** http://urn.fi/URN:ISBN:978-952-64-0619-0

Preface

This dissertation is a result of the research conducted at the Department of Computer Science at Aalto University under the supervision of Prof. N. Asokan.

I would like to thank Prof. N. Asokan for his guidance, encouragement, patience, and support during these years. Also, I would like to thank Prof. Jörg Ott for giving me the opportunity to start the academic research at Aalto University. This work would not be possible without the great help of my advisor, Dr. Andrew Paverd. Our fruitful discussions on various ideas made solving research problems look easy.

I would like to express my gratitude to my co-authors: Fritz Alder, Klaudia Krawiecka, Prof. Mohammad Mannan, Dr. Michael Steiner. Working with them was a great learning experience, I have received a lot of help with implementation and testing the ideas in this work.

I thank my preliminary examiners, Dr. Raja Akram and Prof. Liqun Chen, for agreeing to review this dissertation and providing valuable feedback. I also thank Prof. Mika Ylianttila for being my opponent.

My gratitude goes to my colleagues who created a warm atmosphere at work and many nice moments at informal events.

If it were not for my friends with all their support and kindness, completing the dissertation would be ten times harder. Adriana, Anastasiia, Asel, Emilio, Gautam, Jayaprakash, Karthik, Kaustuv, Laurits, Lina, Maria, Michail, Neelabh, Olga, Pramod, Riju, Rohit - you were always believing in me, motivating and encouraging me.

Finally, I would like to express a warm word of appreciation to my family and relatives. To my aunt, my grandmother, my father, and my mother - thank you for looking after me, being there with me and for your love.

Helsinki, November 18, 2021,

Arseny Kurnikov

Contents

List of Publications

This thesis consists of an overview and of the following publications which are referred to in the text by their Roman numerals.

I Klaudia Krawiecka, Arseny Kurnikov, Andrew Paverd, Mohammad Mannan, N. Asokan. SafeKeeper: Protecting Web Passwords using Trusted Execution Environments. In *Proceedings of the 2018 World Wide Web Conference*, Lyon, France, pp. 349-358, April 2018.

II Arseny Kurnikov, Andrew Paverd, Mohammad Mannan, N. Asokan. Keys in the Clouds: Auditable Multi-device Access to Cryptographic Credentials. In *Proceedings of the 13th International Conference on Availability, Reliability and Security*, Hamburg, Germany, pp. 40:1-40:10, August 2018.

III Fritz Alder, Arseny Kurnikov, Andrew Paverd, N. Asokan. Migrating SGX Enclaves with Persistent State. In *IEEE/IFIP International Conference on Dependable Systems and Networks*, Luxembourg City, pp. 195-206, June 2018.

IV Fritz Alder, N. Asokan, Arseny Kurnikov, Andrew Paverd, Michael Steiner. S-FaaS: Trustworthy and Accountable Function-as-a-Service using Intel SGX. In *CCSW 2019: The ACM Cloud Computing Security Workshop*, London, UK, pp. 185-199, November 2019.

Author's Contribution

Publication I: "SafeKeeper: Protecting Web Passwords using Trusted Execution Environments"

Both student authors Klaudia Krawiecka and I are the main authors of the paper. Klaudia Krawiecka and Andrew Paverd developed the original idea. Klaudia Krawiecka implemented the client-side extension and the server-side password hashing functionality and carried out the user study. I designed and implemented the remote attestation functionality between the server and the client, evaluated the performance of the system, and wrote a few sections of the paper. I created the demo that was published as an additional paper in the same conference. My overall contribution to this research was approximately 50 per cent.

Publication II: "Keys in the Clouds: Auditable Multi-device Access to Cryptographic Credentials"

I am the main author and the sole student author of the paper. All 4 authors came up with the original idea. Andrew Paverd and I designed the main features of the application for storing private keys in the cloud. I implemented the application, performed its evaluation from performance and security perspectives, and wrote multiple sections of the paper. My overall contribution to this research was approximately 90 per cent.

Publication III: "Migrating SGX Enclaves with Persistent State"

Both student authors Fritz Alder and I are the main authors of the paper. Andrew Paverd and I developed the original idea and designed the migration framework for Intel SGX enclaves. Fritz Alder implemented

major parts of the framework. I implemented remote attestation between the framework's enclaves, participated in the evaluation of the framework, and wrote multiple sections of the paper. My overall contribution to this research was approximately 60 per cent.

Publication IV: "S-FaaS: Trustworthy and Accountable Function-as-a-Service using Intel SGX"

Both student authors Fritz Alder and I are the main authors of the paper. The original idea was proposed by Michael Steiner. Andrew Paverd, Fritz Alder, Michael Steiner and I designed the resource measurement framework for SGX enclaves. Fritz Alder implemented the key management, memory measurements, and Javascript sandboxing parts of the framework. I implemented the CPU measurements part using Intel TSX. Fritz Alder and I performed the evaluation tests, with work equally split between us. All authors participated in writing the paper. My overall contribution to this research was approximately 50 per cent.

Language check

The language of my dissertation has been checked by the Institute of Language Checks. I have personally examined and accepted/rejected the results of the language check one by one. This has not affected the scientific content of my dissertation.

List of Figures

Abbreviations

API Application Programming Interface

BFT Byzantine Fault Tolerance

CaaS Container-as-a-Service

CPU Central Processing Unit

DoS Denial-of-Service

EPC Enclave Page Cache

FaaS Function-as-a-Service

IaaS Infrastructure-as-a-Service

IAS Intel Attestation Service

HMAC Hash-Based Message Authentication Code

HSM Hardware Security Module

HTTP Hypertext Transfer Protocol

KDE Key Distribution Enclave

ME Migration Enclave

MEE Memory Encryption Engine

OS Operating System

PaaS Platform-as-a-Service

PKI Public Key Infrastructure

QE Quoting Enclave

RA Remote Attestation

RAM Random Access Memory

RoT Root of Trust

SGX Software Guard Extensions

SGX Software Guard Extensions

SSA Save State Area

SaaS Software-as-a-Service

TA Trusted Application

TCB Trusted Computing Base

TEE Trusted Execution Environment

TLS Transport Layer Security

TPM Trusted Platform Module

TSX Transactional Synchronization Extensions

VM Virtual Machine

1. Introduction

1.1 Motivation

The focus of this dissertation is to study how Trusted Execution Environments (TEEs) can be utilized in cloud environments, what benefits they provide, and what challenges arise when using them in the cloud setting. In this section, I will outline the security and privacy concerns in cloud computing and consider how TEEs can help address those concerns, along with the various challenges that must be overcome.

1.1.1 Cloud computing

Cloud computing became ubiquitous for outsourcing the management of IT infrastructures, due to the many advantages it provides including elasticity, scalability, robustness, and availability [81]. The computing resources in the cloud are virtualized, so it is easy to provision more resources when the demand is high and reduce unnecessary consumption when the demand is low. Failed components can be easily replaced due to the abundance of resources. Being often geographically distributed, clouds do not significantly increase the response time and delays because the computing infrastructure can be located in the region closest to the clients [2]. Thus the benefits of using cloud computing are considerable.

When using *public clouds*, data is processed remotely by the *cloud provider*. Clients of the cloud providers necessarily share their data with the cloud provider. The nature of the data depends on the required services, but ultimately the cloud provider and its client make an explicit or an implicit contract about how the data is processed and how it is protected [4]. The results of the data processing are then available to the client.

Sharing data with remote entities poses security risks. Privacy-sensitive data, such as medical records, genome data, or other private information, is crucial to protect from being accessed by unauthorized parties [1]. Usu-

ally, cloud providers pay a lot of attention to securing their premises and applying best security practices to data processing units [52]. However, *trusting* the cloud provider *itself* is typically based on the assumption that the cloud provider implements all the necessary measures to protect against external attacks and it has to maintain its reputation and hence cannot become an adversary [91].

By *trusting* the provider the clients make assumptions about what actions the provider will do. In particular, it is assumed that the clients' data will not be shared with unauthorized third parties and the providers will take care that the data is protected when it is processed. The providers are assumed to follow best practices in ensuring the security of their infrastructure. Finally, a provider should execute the computation that it is requested to execute. Since the clients do not generally have any assurance of what the provider will do, they have to *trust* the cloud provider based on its reputation.

The reputational trust assumption includes two aspects. The first one is that the cloud provider does not become a malicious adversary. Secondly, the best security practices that protect both from external attackers and insider threats must be in place. Hence, even if the cloud provider is not malicious (i.e. by stealing user data or performing other activities that can harm users in any way), it may not meet the security requirements by not following best security practices (e.g. not updating the software stack as often as needed) [58]. The second scenario is more realistic, but both scenarios break the reputational trust.

1.1.2 Trusted Execution Environments

If the reputational trust assumption does not hold, the fundamental question becomes how to establish trust in a remote entity. A promising approach is *remote attestation (RA)* in which a *prover* provides evidence to a *verifier* about some property that is true on the prover side [40]. This evidence could consist of a signed hash of the software running on the prover's device and this allows the verifier to make a trust decision about the prover's device software and hardware by verifying the signature. In some realizations, the verifier needs to communicate with the hardware vendor to verify the signature. In other types of architectures, the verifier can validate the signature by itself.

The verifier needs to know that the RA evidence is trustworthy. A prover should have a *Root of Trust (RoT)* – a small component that must be trusted. If the verifier trusts the RoT, then the verifier and the prover can build up a protocol to establish trust in other parts of the prover's system. In modern systems, the RoT is usually realized as a hardware component and software protected by this component because hardware is much harder to tamper with than software.

To build trust starting from the hardware requires that the platform contains special hardware that provides certain functionality to enable security features that cannot be achieved by software only. An early example of such platform security hardware is a Trusted Platform Module (TPM) [57]. A TPM can store cryptographic keys and provides an API to perform operations using these keys. It provides a way to record a tamper-evident log of all software executed on the system (RoT for measurement). It can also use this log during the remote attestation (RoT for reporting). The collection of RoTs bound to the same piece of hardware is called a *trust anchor* [97].

However, TPM-style attestation encompasses all software on the platform and might make it difficult for a verifier to make a trust decision [46]. Besides, the software can contain vulnerabilities that are not known to the cloud provider but can be used by adversaries.

More recently, TEEs have become prevalent [104, 80, 45]. They allow the execution of an arbitrary piece of code in isolation from all other software on the system. Applications running in a TEE are called trusted applications (TAs). In TEE attestation, only TAs are attested, not the whole software stack. It is easier to make trust decisions based on the TEE attestation because that codebase that must be analyzed is smaller. TEEs are becoming popular and there are cloud providers that support them [83].

1.1.3 TEE security

As TEEs are becoming more widespread, many attacks against TEEs are being discovered, as introduced in Section 2.2.2. In some cases, these attacks undermine the security guarantees provided by current TEE implementations. However, hardware manufacturers continue to implement defenses against the attacks and improve the TEEs implementations constantly **??**. Future TEE implementations are likely to be secure against these types of attacks.

1.1.4 TEEs on the server side

Traditionally, TEEs were used on the client side [112, 44]. An example scenario for TEE usage on the client side is the following. A device with a TEE is remotely attested to make sure that it has a genuine hardware-based TEE and is running a specific TA. For example, a bank can attest a client's smartphone and only after a successful attestation send a secret authentication token to the bank's TA on the client's device. Later, the client presents the token to the bank to authenticate itself.

In this work, I explore the idea of using TEEs in the context of cloud computing. The ability to securely execute arbitrary code opens new possi-

bilities for cloud applications, but its realization brings various challenges that give rise to the following research questions.

1.2 Research questions

In this dissertation, I address the challenges and contribute to the research in the area of supporting TEEs on the cloud provider side. This dissertation focuses on the following three important aspects of using TEEs in the context of cloud computing.

1.2.1 Scalable server-side TEE applications

There are two main considerations when building scalable server-side TEE applications: the scalability of the remote attestation and the scalability of the application itself across multiple TEEs.

Attestation scalability. Remote attestation is typically designed for client-side scenarios, where there is one verifier that attests many provers. In the example above, the bank is the verifier and the client devices are the provers. If the verifier needs to communicate with the hardware vendor to validate the attestation presented by the client devices (like, for example, in case of Intel SGX - see Section 2.2.1), this is possible, because one verifier can establish a relationship with the hardware vendor. When attesting server-side TEEs, the clients of the cloud provider or the users of the cloud-hosted application are the verifiers. Thus, there are many verifiers and it is not feasible for all verifiers to establish relationships with the hardware vendor, if this is needed to perform the attestation. Also, due to deployability reasons, the attestation protocol should be as lightweight as possible.

Application scalability. One of the main benefits of using the cloud infrastructure is achieving scalability by running a computation on many machines in parallel [120]. Hence the TAs should be developed to enable this type of horizontal scalability. When designing scalable TAs the security guarantees of the application should not be undermined, especially when applications are stateful because the replication of state might lead to undesired effects (such as double transactions in case of the bank application). Often the coordination between TEE instances should be explicit.

Attestation is crucial to establish trust in a remote entity, and scalability is important for cloud-based applications. The research question that is being investigated is:

Research Question 1: How to build scalable cloud-based TEE applications while still providing strong assurance via remote attestation? (**RQ1**)

1.2.2 Infrastructure support

Deploying TEEs on the cloud provider side encounters infrastructure challenges because the *virtualized* nature of the cloud conflicts with binding to the *hardware* trust anchor. Cloud practices such as virtual machine (VM) migration assume that the software can be moved between different physical machines seamlessly [89]. In cloud environments, migration is an essential technology that allows cloud providers to balance the load and provide robustness guarantees.

However, the TEE functionality breaks if the software is moved as is because TAs are bound to the hardware trust anchor. When a VM with a TA is migrated to another physical machine, the trust anchor changes and the TA's data cannot be accessed by the TA anymore. The research question to that end is:

Research Question 2: How to reconcile hardware-based TEEs with existing cloud practices such as VM migration? (**RQ2**)

1.2.3 Secure resource measurements

One important area of cloud computing is resource usage measurement [107] because the clients are billed based on the amount of resources they use. Currently, the measurements are provided by the cloud provider and clients have to rely on the provider's reputation to trust the measurements. If the cloud provider is not trusted to measure the resource consumption of the applications correctly, a mechanism to do it securely should be deployed. This applies especially to smaller cloud providers or new entrants into the cloud provider market that might not have enough time or resources to establish the reputational trust. In certain scenarios like edge computing where individuals share their spare computational resources, there is no single entity to trust to begin with [99].

TEEs can be utilized to not only protect the confidentiality of clients' data and the integrity of clients' computation, but also to support other aspects of cloud computing, such as resource measurements. I investigate the following research question:

Research Question 3: How to utilize TEEs in the cloud to produce trustworthy resource usage measurements? (**RQ3**)

1.3 Contributions

Table 1.1 describes the relationships between research questions, chapters and publications.

1.3.1 Chapter 3: Developing scalable TAs

To answer **RQ1**, **Publication I** and **Publication II** present the design, implementation, and evaluation of two applications that use TEEs in the cloud. The differences between server-side TEE attestation and client-side TEE attestation are analyzed. This leads to designing a novel server-side TEE attestation protocol presented in **Chapter 3** that takes into account scalability requirements when TAs are deployed in the cloud.

Both applications developed in **Publication I** and **Publication II** share common design principles for scalable TAs. These are general principles that can be applied to other cloud-based TAs as discussed in **Chapter 3**.

1.3.2 Chapter 4: Migrating TAs

To answer **RQ2**, **Publication III** focuses on the challenges of migrating TAs in cloud environments since VM migration is a crucial functionality for cloud providers. The publication presents a framework and protocols for migrating VMs containing TEEs. **Chapter 4** summarizes this framework and shows how it is a key enabler for using TEEs in the cloud.

1.3.3 Chapter 5: Trustworthy resource measurements

Publication IV serves to answer **RQ3** by describing the design, implementation, and evaluation of a novel resource measurement framework using TEEs. Although this publication considers the specific case of Function-as-a-Service (FaaS), the underlying principles can be used to build trustworthy resource measurement for other types of outsourced computation as explained in **Chapter 5**.

1.3.4 Chapter 6: Cross-cutting concerns in using TEEs in the cloud

Building on the above contributions, **Chapter 6** summarizes the dissertation results. It discusses cross-cutting concerns of using TEEs in the cloud

setting: server-side remote attestation, developing scalable secure applications, trust and risk management for hardware security technologies. The chapter analyses future challenges and discusses the directions for future work.

Table 1.1. Dissertation structure.

	Chapter 3	**Chapter 4**	**Chapter 5**
Publication I	**RQ1**, **RQ2**		
Publication II	**RQ1**		
Publication III		**RQ2**	
Publication IV			**RQ3**, **RQ1**

2. Background

This dissertation focuses on the use of Trusted Execution Environments (TEEs) technologies in cloud computing. In this chapter, I give the background for the topics and discuss work related to the use of TEEs in cloud computing, the challenges involved, and the possible solutions to those challenges that have been proposed.

2.1 Cloud computing

Cloud computing in general means that there is a *cloud provider* who owns computation resources and allows *clients* to rent them. Cloud computing emerged as a result of rapid advances in virtualization technology (both software and hardware) that enable the execution of many virtual machines (VMs) on the same physical host. Cloud providers build data centers and provide services at scale. The benefits to cloud providers' clients are many-fold: there is no need to maintain the infrastructure, while robustness and scalability are improved. Cloud providers' clients provide services to *end users*.

2.1.1 Cloud service architectures

The different types of cloud provider services are Infrastructure-as-a-Service (IaaS), Platform-as-a-Service (PaaS), Software-as-a-Service (SaaS), and more recently, Container-as-a-Service (CaaS) and Function-as-a-Service (FaaS) [16].

At the base level, IaaS cloud computing serves clients with bare computing nodes [5, 49, 82, 43]. This means that the clients are provided with the VMs and operating systems (OSes), but apart from that the rest of the software stack should be maintained by the clients themselves. On one hand, IaaS gives the clients a lot of flexibility in configuring the VMs and applications, but on the other hand, the full stack software management is the client's responsibility.

In the PaaS architecture, in addition to VMs and OSes, the cloud provider manages the *platform* for deploying applications [64]. This can include the software runtime stack for a particular programming language, web-servers, load balancers, etc. The clients can focus on developing applications without the need to maintain the software, though the clients depend on the cloud provider maintaining the stack and the applications should be developed in a certain fashion.

The SaaS architecture means that the cloud provides an application for end users to work with [86]. The users can configure applications but cannot install their own software in the cloud.

CaaS and FaaS architectures have recently emerged. CaaS is similar to IaaS but instead of VMs, the cloud provider establishes an infrastructure for launching containerized applications [53]. Containers can be seen as a lightweight alternative to VMs where isolation happens on the OS kernel level, and applications share the kernel but are isolated into namespaces. In FaaS, the clients develop *functions* to be executed by the end users [39].

A common trend in cloud computing is to move towards microservices instead of monolithic applications [94]. A microservice is a part of the applications that has a well-defined functionality and scope and exposes Application Programming Interface (API) to interact with it. The microservice architecture allows to develop, deploy and scale microservices independently. A microservice is often *stateless*, so that new instances can be easily deployed and terminated.

2.1.2 Resource measurements

Resource measurements are essential for cloud computing since they usually form the basis for billing. These measurements must be trustworthy and accurate from both the client's perspective and the cloud provider's perspective. Often clients must rely on the reputational trust of the cloud provider to accurately perform resource measurements.

In the IaaS architecture, the cloud provider tracks the uptime of the VMs, and the clients can verify the reported resource measurements by calculating the uptime themselves. However, verifying that the VM CPU frequency is as manifested by the cloud provider is more challenging. Besides, for example, cloud providers might deploy VMs in such a way that the required cumulative number of CPUs is more than the physical machine actually has. This can result in underperforming VMs on high loads, i.e. when all VMs are performing intensive computations and demand a lot of CPU resources.

In the PaaS architecture, resource accounting focuses on application resource consumption, so it is more granular than in the case of IaaS. The clients need to trust the cloud provider to run the applications on the machines with characteristics claimed by the provider.

In the SaaS scenario, the cloud provider is interested in providing a good user experience, so there is an incentive for the provider to run the application at a high performance level. Clients cannot verify performance characteristics and have to rely on the reputational trust model.

In the FaaS architecture, the FaaS functions can be small snippets of code and one execution might not consume a lot of resources, so the resource accounting mechanism should be precise and at a fine granularity. Currently, FaaS solutions report resources consumed at a coarse granularity (see Table 1, **Publication IV**).

2.1.3 Virtual machine migration

VMs need to be moved between physical machines due to various reasons, such as load balancing, machine failures, or performance requirements [65]. *Non-live migration* stops the VM on one physical host and starts it on the destination physical host [122].

During *live migration*, the VM does not need to be stopped [89]. During the migration procedure, the hypervisor reads the VM's memory and copies it and the CPU state of a VM from one physical machine to another. During the copy process, the modifications to the memory pages are monitored. When only modified pages remain, the machine is paused and they are copied as well. This procedure minimizes the downtime and makes the live migration transparent to applications running inside the VM.

The migration is controlled by the hypervisor [113, 78], so to copy the VM memory, the hypervisor runs at a higher privilege level that allows it to read the VM memory and CPU state.

2.2 Trusted Execution Environments

Trusted Execution Environment (TEE) is a technology that allows executing workloads in a securely isolated execution environment [10, 41]. Other applications, the OS, and the hypervisor do not have access to the applications running inside TEEs. Application memory is encrypted and the access is restricted on the hardware level. The code that is executed inside a TEE is called a trusted application (TA). The TEE and the TA form a Trusted Computing Base (TCB). Typically, the TCB needs to be minimized to reduce the attack surface.

Attestation is a process of verifying the state of a remote host. Two parties participate in attestation: a prover and a verifier. The goal of the verifier is to identify what software (and hardware) is running on the prover. The prover generates signed evidence (a *measurement*) that it runs a particular software. The measurement is bound to the Root of Trust (RoT) – the part of the system that has to be trusted in the first place and

on top of which the rest of the trustworthiness is built up. The verifier attests the prover by verifying the signature of the measurement.

In *software-based attestation*, the RoT is not protected by the hardware. For example, Google provides a framework called SafetyNet for Android-based devices that allows application developers to verify that the application is running on a device that was previously approved [54]. In *hardware-based attestation*, the RoT is bound to the hardware. In this dissertation, when the term remote attestation (RA) is used, it refers to hardware-based attestation unless mentioned otherwise.

Sealed storage is a functionality provided by TEEs to enable non-volatile secure storage of TA data. TAs request sealing and unsealing operations to store and load data in encrypted form and only the TA that has sealed some data will be able to unseal it later.

Monotonic counters are integer counters that can only be increased. I.e. if at some point in time the value of a monotonic counter is n then it is guaranteed that later it will never become smaller than n. Monotonic counters are used to protect sealed data from replay attacks (see 2.2.2).

2.2.1 Intel SGX

The publications in this dissertation use Software Guard Extensions (SGX), one of the TEE realizations by Intel, because it is widely available and suitable for research. SGX provides the following main features:

- isolated execution;
- remote attestation;
- sealed storage;
- monotonic counters.

The isolated execution is based on a concept called *a secure enclave* [41]. An enclave is part of the application that is protected by SGX. Enclaves can only be run in unprivileged CPU mode (ring 3). Each enclave has an identity **MRENCLAVE** calculated as a hash of its code. The same enclaves have the same MRENCLAVE. An enclave is also signed by the enclave's author and the hash of the author's public key is called **MRSIGNER**. Enclaves of the same author have the same MRSIGNER.

The enclave memory pages are allocated from the Enclave Page Cache (EPC). The CPU prevents non-enclave code from reading EPC memory and the Memory Encryption Engine (MEE) encrypts EPC memory when it leaves the CPU boundary.

The CPU securely handles the transitions of the execution from the

untrusted parts of the system to the enclave and vice versa. The transitions can be synchronous when the code itself requests the transition via special SGX CPU instructions: EENTER and EEXIT. The state of the CPU is saved into the Save State Area (SSA) at EEXIT and restored from SSA at EENTER. The OS can interrupt the enclave execution at any time and in this case, the transition is asynchronous, but the CPU handles the SSA in the same way.

In SGX, there are two types of attestations: local and remote. Local attestation occurs between enclaves running on the same physical host. Remote attestation (see Figure 2.1) is based on the local attestation between the enclave that is being attested and a special enclave, called the quoting enclave (QE). The measurement of an enclave is called a report [7]. The QE attests the enclave locally and signs the reports with a private key derived from a secret embedded into the hardware by the manufacturer, producing a *quote*. The quote can be verified by sending it to the Intel Attestation Service (IAS).

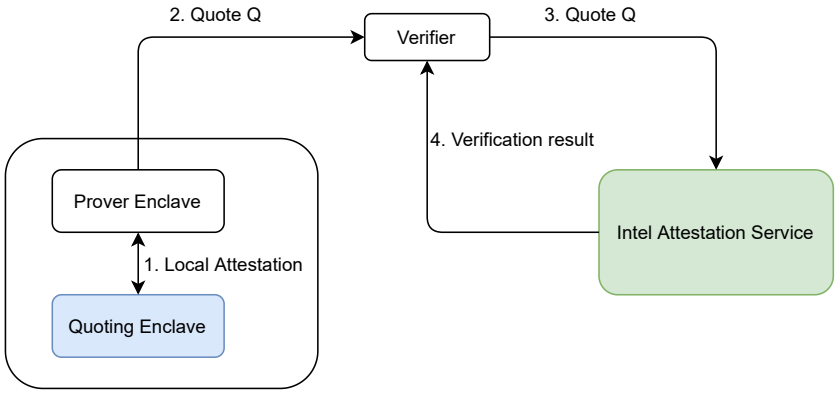

Figure 2.1. Intel SGX attestation.

SGX sealing is implemented in the following way. The enclave code requests the sealing keys from the CPU. The keys can be based on MRENCLAVE or MRSIGNER. If the sealing is performed based on MRENCLAVE, only the same enclave will be able to unseal the data. The data sealed with keys based on MRSIGNER can be unsealed by other enclaves of the same author. This can be used, for example, when there is a new version of the enclave that needs to access the data sealed by the previous version.

SGX supports 256 monotonic counters per enclave that are implemented as virtual counters in a special platform enclave. Since the monotonic counters rely on writing to non-volatile memory, the increment rate is limited by the firmware.

2.2.2 SGX security

Security guarantees (confidentiality and integrity) provided by Intel SGX, in theory, would protect the code and data of the enclave from a malicious application, OS, or hypervisor. However, when designs are implemented in real systems, the theoretical assumptions underlying the design may not hold. This has resulted in many attacks discovered against SGX.

Side-channel attacks. One class of attacks includes side-channel attacks that utilize implementation weaknesses to retrieve the information that otherwise should not be available to the attacker. For example, cache side-channel attacks are based on the timing differences when data is cached compared to when it has to be fetched from the memory [55]. Lee et al. [75] present another type of side-channel attack that infers the control flow inside an enclave via branch shadowing. Typical mitigation of side-channel attacks is to randomize access patterns [25].

Speculative execution attacks. Another wide range of attacks is based on the *transient execution* feature of modern processors. In short, transient execution means that some execution paths are executed in advance to improve performance. If a certain path would not be needed in the final result, then the results of the computation along this path are discarded. However, before they are discarded they might be leaked to the attacker through a side-channel. One example of these attacks is Meltdown [79], where the transient execution results are discarded because an exception occurs and a side-channel is cache-based. Another broad class of transient execution attacks is called Spectre Attacks [73] and its variants, which have been found to attack against Intel SGX as well [35, 119].

Fault-injection attacks. Another class of attacks is based on faults injection in the computation. Probably the most well-known attack of this type is a row hammer attack against volatile memory [106]. Fault injection in TEEs can lead to modifying the results of cryptographic computations. Plundervolt [87] utilizes voltage drops to bypass the security guarantees of Intel SGX.

Rollback, replay, and fork attacks. A *rollback attack* occurs when a system is reverted to an old state while preserving the confidentiality of the system. A *replay attack* occurs when a previously executed transaction is executed again. A *fork attack* occurs when the system instance is duplicated for malicious purposes. These attacks can result in an incorrect state of the system (for example, if it is a bank account, then it has to be protected from these types of attacks, otherwise some transactions might execute twice or be deleted).

It is important to make sure that applications do not introduce new attack surfaces. In particular, rollback and replay attacks on TEEs must be prevented by utilizing monotonic counters. The value of the monotonic

counter must be sealed along with the sealed data so that the enclave can detect when the data provided for unsealing is stale.

When developing TAs presented in this dissertation, side-channel, speculative execution, and fault-injection attacks were not included in the model, because they can be considered flaws in implementation. Intel provided mitigations against the attacks [67] and correct implementations that are resistant to side-channel attacks are possible. TAs presented in this dissertation are based on Intel SGX but in general, they could be developed for other TEE implementations.

2.3 Other hardware features

2.3.1 TPM

Trusted hardware is a common term that covers various technologies, such as Trusted Platform Modules (TPMs) and TEEs.

A TPM contains hardware-protected keys and can perform cryptographic operations using those keys [57]. A TPM contains various private keys that serve as the basis for verification procedures. Like TEEs, a TPM serves as an RoT as well. Results of the TPM cryptographic operations can be verified to originate from a particular TPM based on the RoT.

TPMs were proposed to be used in cloud environments. Bleikertz et al. [22] describe Cryptography-as-a-Service where TPM functionality is provided by the cloud. Memoir [8] presents a rollback protection framework for TPMs. Van Dijk et al. [118] describe an offline storage system that utilizes TPMs to detect rollback attacks. Danev et al. [42] describe a migration framework for virtualized TPMs.

2.3.2 TSX

Intel Transactional Synchronization Extensions (TSX) [62] are a set of CPU instructions designed to optimize parallel workloads. In TSX, a thread can start a *transaction* instead of relying on mutexes and locks for thread synchronization. Memory addresses that are read from during the execution of the transaction form its *read set*. Memory addresses that are written to during the execution of the transaction form its *write set*.

If one thread modifies memory that corresponds to a read set of another thread, then the transaction is aborted. Accordingly, if one thread tries to read the memory from another thread's write thread, the transaction is aborted.

This forms a communication channel between the threads which can be utilized for various notification events from one thread to another. One

application of this communication channel is to protect SGX enclaves from side-channel attacks [116, 108].

3. TEE assisted services

The main motivation for introducing TEEs in the cloud is to enhance the security of cloud applications. There are two broad categories of cloud applications that can benefit from the use of TEEs: *existing* applications where adding TEE support can improve the security guarantees that the applications provide, helping to defend against attacks by a malicious or compromised cloud provider; and *new* applications that can be developed only because the data is processed inside TEEs.

For example, an existing web application that requires users to authenticate needs to process users' passwords. If the cloud provider is not trusted, then introducing a TEE would allow users to attest the password processing TA to make sure that the passwords are not leaked.

New applications can heavily rely on the TEE to protect sensitive user data; without a TEE, a comparable level of security cannot be provided cost-effectively. For example, managing users' private keys with multiple devices is challenging. If a device is lost or compromised, the keys need to be revoked. Users' private keys can be stored in the cloud and users would request cryptographic operations using those keys. These keys are very high-value assets and users might not entrust them to the cloud provider without additional protection. Without a TEE, an application that stores users' private keys in the cloud would rely on cryptographic algorithms (e.g. homomorphic encryption) with very low performance [47].

Both existing applications and new applications share common design patterns. In particular, a common workflow is the following:

1. A user makes a trust decision about the TA via remote attestation (RA).

2. The user and the TA establish a secure channel between themselves.

3. The client sends input, requests operations from the TA, and gets results.

When developing TAs, the common challenges are scalability, minimizing

the Trusted Computing Base (TCB), and side-channel resistance.

Scalability. This chapter addresses **RQ1**: how to build scalable cloud-based TEE applications while still being able to provide strong assurance via RA? Taking scalability considerations into account when building TAs has to be done in a way that does not compromise security. In particular, RA guarantees must remain intact. In certain scenarios, this means that the RA protocol needs to be re-worked. Other scalability aspects such as synchronizing the state between the TA instances are discussed in this chapter as well.

Minimizing TCB. The TCB should be minimized because it reduces the attack surface and allows verifying the code that needs to be trusted more thoroughly [110]. There have been proposals to execute entire applications inside a TEE [105]. While this approach does protect the code and data from the cloud provider, and does not require developing parts of the application to work specifically inside a TEE, its main limitation is the large TCB. Without being able to verify the correctness of the whole application, it is not possible to ensure the security guarantees when the application is run inside a TEE because the application's behavior is not determenistic and it is not clear whether the application contains security vulnerabilities.

In this chapter, the TAs are developed from scratch, with best security practices in mind. However, the formal proof of TAs correctness is left out of scope.

Side-channel resistance. Side-channel resistance means that the operations performed inside a TEE should not leave traces outside of the TEE that could lead to potential information leakage [26, 92]. As discussed in Section 6.3, the security risks for new applications may be higher than for existing ones, because new applications may solely rely on the functionality provided by a TEE for security, while the existing applications may use TEEs only for additional security features.

This chapter focuses on two examples of TAs and describes common design patterns when building scalable TAs for the cloud. Also, the implications cloud-based TAs have on the RA process are discussed.

3.1 Protecting passwords in the web

In this dissertation, I focus on Intel SGX as one implementation of TEEs.

3.1.1 Adversary model

The adversary model for TAs in the cloud assumes that the cloud provider is not trusted. The cloud provider's hypervisor, host OS and other software running along with the TA on the same machine can be tampered with

by an adversary. Network communications can be monitored and altered by an adversrary. The adversary model does not include Denial-of-Service (DoS) attacks, because the cloud provider can always stop the TA execution.

3.1.2 Motivation

Let us assume that there is a cloud application that performs some operations on privacy-sensitive data. In SGX, system calls involve context switches that are expensive operations from a performance point of view [115]. Legacy applications might perform a lot of system calls, so porting unmodified legacy applications to run inside a TEE would drastically harm the performance. The goal of enhancing the application is to isolate parts of it that are directly working with sensitive data into TAs and establish interfaces between the isolated secure parts and the rest of the application. An example of such a design is presented in **Publication I**, where I study the case of enhancing an existing application.

The application concerns password-based authentication on the web. Password-based authentication is the most common form of authentication mechanisms due for several reasons [23]:

- Strong passwords provide good protection for user accounts.
- Passwords are easy to change in case of compromise.
- The mechanism is understood by most users.

Plain text passwords are generally not stored on the servers of web service providers. Instead, the passwords are hashed and the password hashes are stored along with the salt values (Figure 3.1). At login, the salt value is extracted from storage, the password hash is calculated and then compared to the value from storage (Figure 3.2).

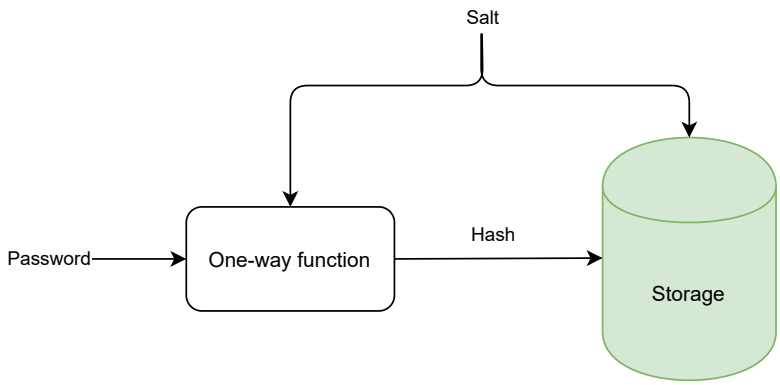

Figure 3.1. Password hash creation. Random salt protects against rainbow tables attacks.

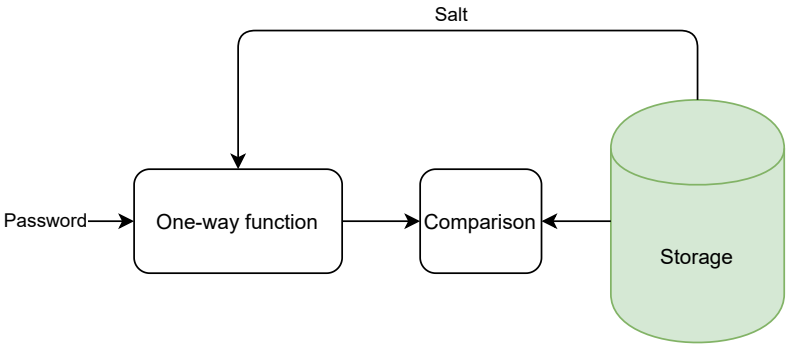

Figure 3.2. Password check at login. The salt values are stored along with the hashes and used when computing the hash at login time.

Salt values are used to protect against attacks using pre-computed rainbow tables on leaked password hashes: they ensure that even if two users have the same password, their corresponding salted hashes will be different. Hence the attacker has to mount a brute-force attack against each user, rather than trying the guesses against the whole database at once. However, since the salts are usually stored with the hashes, an adversary can perform a brute-force attack if they obtain a database of password hashes [98]. Obtaining a user password allows the adversary to gain access to other accounts of the same user because passwords are often re-used [68, 63].

Password databases are often stolen [121, 100]. Passwords can also be obtained via phishing attacks, where the user is tricked into entering the password on a malicious website [9]. Hence, a better mechanism for protecting user passwords is desirable. The challenge is how to strengthen password-based user authentication with minimal deployment cost.

3.1.3 Server-side TEE service

In **Publication I**, we improve the authentication application in two aspects: we use a TEE on the server to replace hashing of the password by computing a keyed one-way function; we encrypt the password on the user's side via a browser extension and send it directly to the TEE after attesting the TA. This improved authentication framework is called SafeKeeper.

On the server, the TA computes a keyed one-way function of the password and the resulting value is stored in the database. The TA inputs are the *user password* and the *salt value*, and the output is the Message-Authentication Code (MAC) of the password. Since the key is protected by the TEE, an adversary who obtained the database cannot perform a brute-force attack to get the original passwords. The SafeKeeper application has a small TCB and its correctness can be verified formally.

Secure channel. Performing a single cryptographic operation could

be done by the TPM as well. The advantages of using a TEE over a TPM include an end-to-end secure channel and additional features such as a rate-limiting mechanism, that can be implemented because TEEs are programmable. Typically, the secure channel is established via a Transport Layer Security (TLS) connection. In the case of TPMs, the TLS termination happens before a call to the TPM [32]. With TEEs, the TLS termination can happen inside the enclave, improving the security of the communication [13, 72].

Rate-limiting mechanism. Since the adversary can also launch the TA, the TA itself can act as a helper in brute-forcing the passwords. If the TA computes MACs without any rate limits, the attacker would be able to use it to obtain the passwords. Hence, the rate at which the MACs are computed should be limited.

From the deployability perspective, the application inputs must remain the same as for the original password hashing mechanism because then the TEE version can be easily deployed by substituting the call to the hashing algorithm with the call to the TA. These inputs are a password and a salt value. Hence, the rate limit cannot be based on the usernames, since they are not part of the input to the TA.

In **Publication I**, a novel idea is introduced to address this problem: to limit the rate of computing MACs based on the salt value. Indeed, the salt value should be randomly chosen and be unique to a particular user, so it can be seen as a replacement for the username. If the adversary generates the same salt values for all the users, it only limits the number of attempts further.

A related work by Birr-Pixton [21] suggests using TEEs to perform password hashing. SafeKeeper protects against rogues servers by utilizing remote attestation and offline attacks by introducing a rate-limiting mechanism.

3.1.4 Browser extension

On the end user's side, users can either be notified that the server supports TEE-protected password MACs, or the interaction with the TA can be transparent. From the usability perspective, it would be better if no user interaction was required. If users are not aware of the TEE on the server side, then legitimate servers still benefit from utilizing SafeKeeper to protect users' passwords (hashes are substituted by MACs resulting from the MAC function). However, this allows servers to claim that they support SafeKeeper without users being able to verify that, since the verification has to be done via remote attestation (RA) and web browsers do not support it. Hence, if there is no user-side software that supports RA, users might be giving their passwords to rogue servers, that either mishandle the passwords or were broken into and the password-handling code was

replaced by an adversary.

User awareness is achieved by the SafeKeeper browser extension that identifies if the website supports SafeKeeper via RA. The design of the

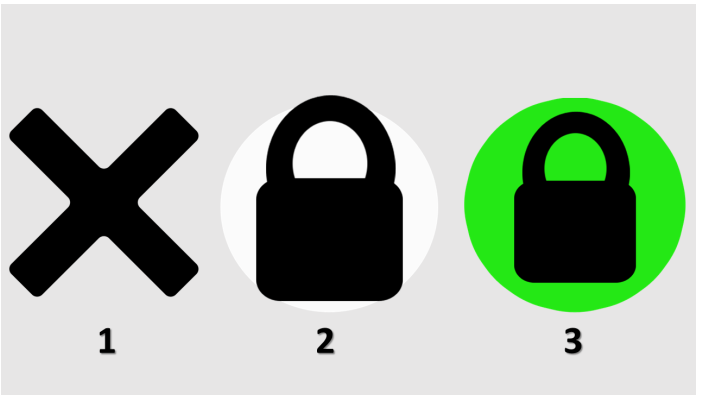

Figure 3.3. Figure 3 in Publication I. The states of SafeKeeper. 1. SafeKeeper is not supported or the attestation has failed. 2. SafeKeeper is supported and the attestation has succeeded. 3. Protected input fields are highlighted.

extension was driven by the requirement to secure against malicious websites.

If the RA fails or the website does not support SafeKeeper, the extension displays a corresponding icon (the first icon in Figure 3.3). If the RA succeeds, the extension shows a lock icon, that, on being clicked, highlights the fields on the website that will be sent to the TEE. A phishing site can claim that it supports SafeKeeper and can even successfully pass the RA. However, instead of encrypting the text from the password field for the TEE, it could encrypt some other field, and obtain the clear-text password. That is why the highlighting of the field that is going to be sent to the TEE is necessary.

If a malicious website attempts to spoof the highlighting, it can be detected by repeatedly clicking on the SafeKeeper icon because when the icon is clicked again the highlighting is disabled. Spoofing prevention could be also achieved by disabling Javascript on the page but that would harm usability.

Usability, performance, deployability. Security solutions should always take usability and performance requirements into consideration. Testing SafeKeeper on the server side showed that a TEE computes password MACs at a similar rate as the original hashing, and sometimes even faster. This can be explained by the fact that the hashing algorithms are slow by design to prevent brute-force attacks [98]. There is no need to slow down the MAC computation inside the TEE because without knowing the key used to compute the MAC, it is not possible to perform a brute-force attack. So SafeKeeper provides adequate performance.

From the deployment perspective, SafeKeeper can be deployed incrementally, because it only requires substituting one function call to execute a TA function instead of a normal hashing function. If not all the cloud provider machines support SGX, then the ones without SGX support can still rely on the normal hashing and upgrade later.

User study. The browser extension was the subject of a user study with 86 participants (Section 6.2 in **Publication I**) from 18 to 39 years old. The basic demographic information (Section 6.2.1 in **Publication I**) about participants is presented in Table 3.1. One group consisting of 64 participants was given instructions on how to use SafeKeeper in the browser and asked a question if the website provides extra security for the password. The results showed that 80% of the users understood and managed to use the browser extension successfully. The control group of 22 users who were not given any instructions at all managed to use the browser extension to detect if the webserver supports SafeKeeper in 74% cases. The user study showed that the SafeKeeper browser extension is usable by a majority of the users.

Table 3.1. User study participants demographic information.

Gender	
Male	70%
Female	30%
Education	
Education unspecified	14%
High school diploma	9%
Bachelor's degree	41%
Master's degree	34%
PhD's degree	2%

3.2 Cloud key store

SafeKeeper is an example of using cloud-hosted TEEs to strengthen the security of an existing usage scenario. But cloud-hosted TEEs can also enable applications for new usage scenarios that cannot be realized otherwise with the same level of security and performance. An example of such an application is the storage of private user keys in the cloud as presented in **Publication II**. Storing users' private keys on the server without TEE would not be possible if the cloud provider is not trusted. TEEs provide sufficient security guarantees to implement the centralized storage of user private keys.

3.2.1 Motivation

The main motivation for storing user private keys in the cloud is that there are usually multiple devices from which the user wishes to use the keys. Having multiple copies of private keys on multiple devices has security implications because it increases the probability of losing a private key. One solution is to use a hardware token [126] but it needs to be always with the user and if it is lost the key on the token must be revoked. Another solution is to copy the private keys on to all the devices of the user. But if there are multiple copies of a private key on user's devices, then if one device is lost or compromised the key has to be revoked. Moving the key to a newly acquired device is difficult as well because it must occur over a secure channel and the key should not leak during the process. Storing private keys in the cloud is beneficial both from usability and security perspectives.

Using TEEs to store private keys was proposed in [33]. Barbican [33] is a key-management solution for VMs in the cloud utilizing TEEs to seal cryptographic keys.

Cloud Key Store (CKS), introduced in **Publication II**, is the cloud application that stores user private keys and allows users to perform cryptographic operations, such as signing or decrypting, using those keys. Each user is authenticated with their credentials: a username and a password. Essentially, CKS allows performing cryptographic operations that require strong secrets (such as private keys) with access control based on a weak secret (the password). For the end users, working with CKS looks the same as working with private keys on a smartcard.

3.2.2 Application workflow and features

The session with the CKS server starts with RA. When the end user's device verifies the quote and establishes a secure channel, it sends the username and the password to the server. The user also sends the requested operation and the required input for the operation. The CKS TA authenticates the user, and if the authentication succeeds, the CKS TA performs the operation and returns the results to the user.

The authentication attempts are rate-limited, similar to the SafeKeeper rate-limiting mechanism. In CKS, the rate-limiting can be based on the username because the protocol between the end user and the server is developed from scratch. The TA gets both the username and the password and makes the authentication decision based on that.

3.2.3 Cloud-hosted keystore novel features

Storing private keys in the cloud enables new features that are not possible to implement when the keys are stored locally. These features are: auditing the key usage and key delegation.

Audit. The key usage logging is an important feature of CKS. Whenever a cryptographic operation is performed, CKS logs it so that it can be audited later. This helps to detect password compromises and allows to change the password in case of compromise. Another advantage of storing keys centrally is the ability to log all operations, since they are performed on the server. Also, in case of password compromise, only the password needs to be changed and the keys do not need to be revoked.

Key delegation. Since all the users are managed centrally, implementing other features such as key delegation is easy to enable. A user can delegate the use of their private key to another user and limit the usage by specifying a key usage policy. The policy can specify a period and/or the allowed number of uses.

3.2.4 CKS scalability.

One instance of the CKS TA might not be enough to handle the load on the server. If there are several instances, they should be aware of each other, otherwise an attacker would be able to increase its password-guessing rate by running multiple instances of CKS. Indeed, assuming two CKS instances serve the same set of users, the rate-limiting cap should be split between the two. When a new CKS instance is launched, it can join the cluster by sending a request to an existing instance. After performing mutual attestation, the existing instance can reduce its maximum number of attempts per user and authorize a new instance to serve the users with a given maximum number of attempts. When the instance goes offline, it can be revoked, and the maximum number of attempts of existing instances in the cluster can be adjusted accordingly.

A more sophisticated rate-limiting protocol is presented in [109]. It utilizes the Raft protocol [93] to achieve consensus between TA replicas on the number of guesses left. A rate-limiting mechanism is essential for the operation of SafeKeeper, CKS, and similar applications.

The rate-limiting mechanism does not hinder usability because it is developed to protect against malicious servers, not against remote password guessing attempts. An honest server will implement an additional rate-limiting mechanism against remote password guessing. The retry count of a TA is much higher than the retry count of the remote password guessing protection. This means that users would never reach the CKS internal retry count and even when there are multiple CKS instances, the retry count of any individual instance would not prevent users from

authenticating.

To summarize, CKS is a TA that stores users' private keys, allowing them to perform cryptographic operations with those keys by authenticating with a password, with logging and key delegation features. The CKS scalability is a challenge that should be explicitly taken into account.

3.3 Discussion

In general, both SafeKeeper and CKS follow the same design pattern: perform a remote attestation, establish a secure channel between the end user's device and the TA, and execute the requested operation. SafeKeeper and CKS address research question **RQ1** by employing novel techniques for remote attestation and state management to ensure that the scalability requirements are met.

3.3.1 Remote attestation

Remote attestation is the main mechanism that enables establishing trust in remote TEEs. In the SGX attestation protocol, during the attestation process, a secure channel between the end user's device and the TA is established based on Diffie-Hellman ephemeral key exchange protocol. All the sensitive data is protected by the secure channel when being transferred between the verifying application and the TA, and it is protected by the TEE when being processed on the server side.

Server-side remote attestation (both SafeKeeper and CKS are running on the server side) imposes new requirements compared to client-side remote attestation. First of all, in the case of client-side remote attestation, there is usually one verifier that is attesting many devices. With server-side remote attestation, many users verify one server-side application. The original protocol for SGX RA better suited client-side TEE scenarios (i.e. when the verifier is the service provider and the prover's TEE is on the client-side). Indeed, first the verifiers have to establish relationships with Intel by requesting certificates to communicate with Intel Attestation Service. It does not scale when the verifier is not the service provider but users of the service.

Secondly, the original SGX RA protocol consists of four messages, resulting in two round trips. In web communication, the attestation itself should ideally be a zero round trip protocol. The quote should be sent back with the server response, and the verifier should be able to verify the quote without being registered with Intel.

In SafeKeeper, we introduce a novel two-message protocol (described in full detail in the technical report version of the publication [74]). It does not compromise security if the server uses a static Diffie-Hellman key instead

of an ephemeral one. Hence the server key does not need to be generated each time a client connects to the server. Instead, the server includes the public part of its static key into the quote. The client can verify that it is indeed the genuine TA that owns the corresponding private key.

The quote is sent to the client along with the server response (a web page in the case of SafeKeeper). The original HTTP request-response protocol does not need to be changed to incorporate remote attestation.

Since it is not viable for every web browser user to establish a relationship with Intel, the attestation in SafeKeeper is achieved via an attestation proxy. Only the proxy needs to be registered with Intel Attestation Service (IAS, see Section 2.2.1). The proxy does not need to be trusted by verifiers since the IAS returns a signed result of the verification. The verifiers can check the Intel signature on the resulting message of the attestation. However, using the proxy is not the only possible solution. Another possibility is for the quote to be sent to the IAS by another enclave that is running on the same machine as the TA being attested. This way, there is no need for the proxy because the verifiers will get the IAS response directly from the TA instead of the quote.

3.3.2 State Management

It is a common trend in cloud computing to move towards stateless microservices. Cloud TAs should be stateless, too, because it is easier to manage applications if they can be easily deployed and terminated. Crashes and failures can be dealt with by simply restarting the application.

However, both SafeKeeper and CKS have to maintain a state: the number of login attempts for the rate-limiting mechanism. Additionally, CKS stores user private keys. Rate limiting is an important aspect of an authentication mechanism when it comes to weak secrets (like passwords). Authentication based on strong credentials like public keys and certificates does not need to implement the rate-limiting mechanism. The authenticating party will need to store a database where common names from the certificates define the users' access rights. Thus, it can be assumed that any application that requires authentication will have to maintain a state. Even if the state data is stored in a central location, applications will need access to the keys to authorize themselves within the central storage.

More specifically, if there are multiple instances of SafeKeeper or CKS running in the cloud, they have to be aware of each other. Otherwise, an attacker could simply deploy more TAs to increase his password-guessing rate. So, if there is one instance of a rate-limiting application running, and a cloud provider wants to deploy another instance, the rate limit should be split between the instances. This can be achieved by mutual attestation between the TEEs when a new instance is deployed, as described in the technical report version of **Publication I** [74]. In an application-specific

protocol, the TA instances agree on how to split the remaining retry counts. When an instance is removed, the remaining TA instances can run the same protocol to re-distribute the retry counts among themselves.

3.3.3 Conclusion

This chapter answers **RQ1** by describing two TAs that solve the problems of scalable remote attestation and state management with multiple TA instances. A novel remote attestation protocol can be utilized in server-side scenarios. One possible solution for state management between several TA instances is described. These solutions can be applied to other TAs: for example, various databases can be protected with TEEs [15, 71, 96], and the state synchronization is an important aspect for these TAs.

State management is necessary even for the simplest applications. The infrastructure implications of using TEEs in the cloud and maintaining a state are discussed in Chapter 4. In particular, since TAs have to maintain a state, the migration of TAs must take this into account. Simply moving a TA instance from one physical machine to another will break the TA, because the migrated instance will not have access to the state that was sealed on the physical source machine. The detailed problem description and the solution proposed are described in the next chapter.

4. Cloud infrastructure TEE challenges

To deploy TAs in the cloud, the cloud infrastructure must provide suitable TEEs. However, since TEEs are a hardware-backed mechanism, TA execution is bound to a particular physical machine. This binding contradicts one of the main cloud practices: virtualization. Virtual machines (VMs) do not have direct access to the hardware, rather, they run on top of hypervisors that provide VMs with hardware abstractions [101]. This allows the VM to be run on and migrated between different physical machines.

There are virtualization technologies for trusted hardware as well [18, 19]. But even though applications inside a VM get access to trusted hardware functionality, the VM itself cannot be migrated seamlessly from one physical host to another. The migration is not possible because the virtualized trusted hardware relies on the hardware trust anchor. For example, TAs can utilize the virtualized trusted hardware to encrypt data with keys that are bound to the hardware trust anchor. Decrypting the data encrypted with these keys is not possible if the trust anchor changes due to migration.

VM migration is a core cloud functionality and disabling it by binding to a particular physical host prevents TA deployment in the cloud. Before TAs can be widely used in the cloud, the problem of migrating VMs with TAs from one physical host to another should be solved.

This chapter addresses **RQ2**: how to reconcile hardware-based TEEs with existing cloud practices such as VM migration? It presents a migration framework that meets the following requirements. Firstly, the migration solution should provide the same security guarantees as the original trusted hardware does. Secondly, it should have acceptable performance and a minimal increase in Trusted Computing Base (TCB).

4.1 Motivation

4.1.1 Adversary model

The adversary model for the VM migration is similar to the TAs adversary model: the cloud provider's hypervisor, host OS and other VMs and applications are not trusted. The TA itself that needs to be migrated is assumed to be trusted (see Section 4.3.3).

4.1.2 VM migration

The VM migration process involves moving a VM from one physical host to another. During non-live migration, the VM is stopped (e.g. shut down) on the source physical machine, its persistent state (e.g. virtual disk) is copied to the destination machine, and the VM is restarted. Stateful applications have to persist their state on the persistent storage, like the VM's disk. Since the VM is shut down, the contents of its Random Access Memory (RAM – non-persistent state) are lost and not copied from the source to the destination. The VM may also be shut down when not in use to reduce costs and save resources. After the VM is restarted, there is no guarantee that it will start on the same physical machine. This is a form of non-live migration with additional time between shut down and restart.

During live migration, the VM does not need to be stopped [37, 89, 124]. Its memory is copied from the source machine to the destination while the machine is running, and the VM is only paused for a short time to finish the copying process. The data that is stored on disks inside a VM is copied as well.

4.1.3 Challenges of TA migration

There are three main challenges with the migration of a TA.

Encrypted memory. When a TA is running inside a VM, its migration is not possible as per the normal live migration process. In SGX, the memory of the TA cannot be read by the hypervisor to copy it to the destination machine because it is encrypted by the CPU. If the memory is copied without re-encryption, the destination machine CPU will not be able to decrypt it because the encryption keys are bound to the hardware trust anchor. Hence, migrating a VM that contains a TA from one physical machine to another will result in the TA not being able to continue execution.

Sealed data. Even in the case of non-live migration, TAs that persist state on disk (i.e. *sealed data*) will not function properly when the VM is migrated. Since data is sealed using a key bound to the hardware trust anchor, the migrated TA will not be able to unseal it on the destination ma-

chine. Ignoring this aspect results in a Denial-of-Service for the migrated TA.

Architectural support. In certain TEE architectures, software TCB might be split between the TA itself and architectural parts that reside on the machine. For example, in the case of SGX, there is a platform services enclave that maintains rollback-protected counters [41]. If a TA relies on this functionality (and TAs that seal data have to include the counter into the sealed data to protect against rollback attacks), and this is not taken into account, then migration would allow forking and rollback attacks.

In the *fork attack* [24], the goal of the attacker is to get two copies of the enclave running at the same time. Migration might make this easier for the attacker by starting an enclave on the destination machine and not stopping the enclave on the source. This would result in two instances of the TA that might have undesirable side effects. For example, if the enclave performs some transactions, then the attacker could first route the user to one copy of the enclave and later show another copy, tricking the user to perform the same transaction again.

In the *rollback attack*, the goal of the attacker is to roll back the state of the enclave [24]. Since the TA's state should be persisted during the migration, the attacker could resume the execution of the TA from an older state. Since the state is obsolete, the attacker can benefit, for example, from repeating operations that were supposed to be executed just once, using revoked resources, and so on.

4.1.4 Applications and persistent state.

Even though there might exist TAs that do not require persistence of state, most of the time an application does need to store data at rest. For example, both SafeKeeper and Cloud Key Store have to maintain the counter for the rate-limiting mechanism. These counters represent the architectural state that has to be maintained during migration.

To tackle the problem of maintaining the application state (that exists even without TAs, for example in container-based cloud solutions [70]), cloud providers introduce the cloud pattern for managing storage that includes a centralized storage server. This can be a distributed storage system over multiple physical machines, but the access is centralized for simplicity from the storage client's perspective [125]. In this case, VMs or containers do not store data locally but upload it to the centralized storage. However, this still does not allow for completely stateless VMs, because to mutually authenticate with the centralized storage VMs need to use some secret authentication credential. This secret is essentially the state that needs to be maintained along with the VM.

For TAs, the problem of using centralized storage includes not only mutual authentication but also maintaining a key for secure storage (to

encrypt data before it is written out to centralized persistent storage) and having access to a monotonic counter to ensure the freshness of the data coming from the storage service. So a TA utilizing the centralized storage needs to maintain some state as well, as long as the centralized storage is not trusted. Fully trusted centralized storage is not a reasonable assumption in cloud environments since this requires trusting the cloud provider [90].

4.2 Migrating TEEs with architectural state

4.2.1 Migration framework

Publication III defines a set of security and design requirements for TA migration mechanisms. From the security perspective, the migration framework should maintain the guarantees provided by the TEE while making TAs migratable, as well as prevent possible fork and rollback attacks. This is achieved by providing migratable versions of the TEE primitives that are based on the underlying TEE functionality, such as remote attestation, data sealing, and monotonic counters.

Other design requirements include performance and usability. The migration overhead should be small when the VM contains a TA, both during the migration process and when the VM is running. The development of TAs with migration capability should not be significantly harder than developing TAs without such capability. Ideally, it should be transparent to the TA developers and the migratable versions of the TEE primitives should look and work the same as the original primitives.

In **Publication III**, a framework for migrating TAs that meets the above security and design requirements is presented. The migration framework depicted in Figure 4.1 is designed for SGX TAs, but in principle, the design can be applied with other trusted hardware technologies.

The framework consists of two parts: the migration library and the migration enclave (ME). MEs are deployed on the physical machines and communicate with each other during the migration process. MEs attest to each other and the migrating enclaves, and during the attestation process, they establish the secure channel to copy the migration data over from the source machine to the destination.

The migration library is part of the enclave that needs to be migrated. The library provides migratable primitives for trusted hardware operations that are otherwise dependent on the hardware trust anchor.

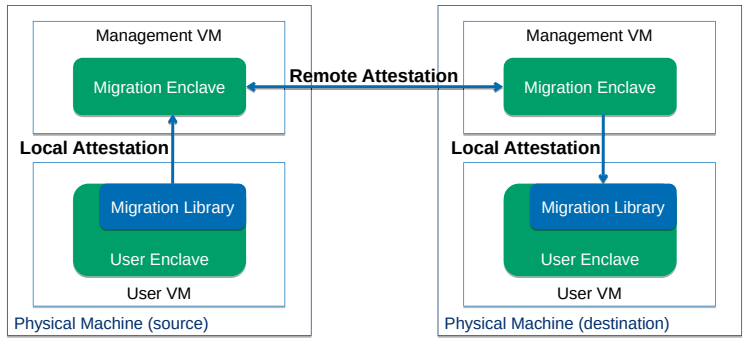

Figure 4.1. Figure 1 in Publication III. Migration framework.

4.2.2 Migratable sealing

Instead of using the platform data sealing functionality, the migrating enclave relies on the migratable sealing provided by the migration library. Migratable sealing uses the key that is generated by the migration library. This key is protected by encrypting it with the original TEE data sealing key, but since the migratable sealing key is not bound to the hardware trust anchor, it is possible to transfer it to the destination migration library via the MEs so that the TA data can be accessed by the TA on the destination machine. For example, in SafeKeeper and Cloud Key Store, the number of remaining authentication attempts should be sealed with the migratable sealing key, so that an attacker will not be able to reset it by migrating the TAs to another physical machine.

4.2.3 Migratable counters

Another primitive that is provided by the library is the support for monotonic counters. The library keeps the state of counters used by the application and transfers it to the destination machine along with the data sealing key. One possible approach to restore the state of counters on the destination machine would be to increase the hardware-backed monotonic counters on the destination machine from zero to the value used by the application on the source machine at every migration. However, increasing the hardware-backed monotonic counter is a slow operation. The library optimizes the handling of the counters by introducing an offset and adding this offset to the hardware-backed counters when returning the library counter. The state of the library counters is rollback-protected by the

platform monotonic counters. As one possible usage scenario, in SafeKeeper and Cloud Key Store, the number of remaining authentication attempts should be rollback-protected with migratable counters.

4.2.4 Migration process

During the migration process presented in Figure 4.2 - Figure 2 in **Publication III**, MEs mutually attest to each other and establish a secure channel to transfer the migration data. The migrating enclave attests the local ME on the source machine and also establishes a secure channel with it. This way, the enclave data is always protected while being transferred from the source to the destination, and there is no need for the hypervisor to read the encrypted memory or to copy it from one machine to another. During the migration process, the migrating enclave has to be stopped on the source machine and restarted on the destination; however, the downtime is small, because the state that needs to be copied is constant: the data sealing key and an array of counter values. In combination with the approach from [59], the memory of the enclave can be copied as well, so that the enclave can be paused and resumed without the need to restart.

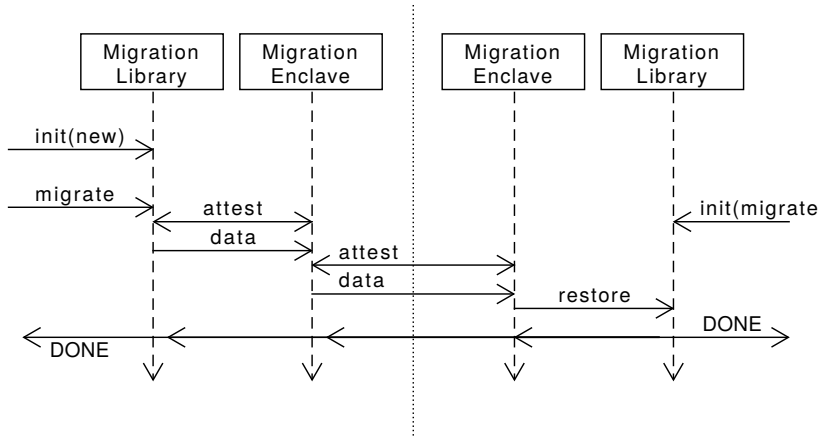

Figure 4.2. Figure 2 in Publication III. Migration process. TA's data is copied from the source to the destination by MEs over a secure channel after a successful attestation.

4.2.5 Framework evaluation

The proposed migration framework addresses the challenges of TA migration while providing the same security guarantees as the underlying TEE does and introducing small overhead during the migration and TA

execution. The maximum overhead is observed for counters operations and they are 12% slower than the platform native counters. The TA migration process takes on average $0.47(\pm 0.035)$ seconds more compared to the VM migration without TAs due to attestation, secure channel establishment, and sending of the enclave data to the destination. Since the migration library provides a similar API analogous to the platform primitives, the development effort of including the library and making the enclave migratable is small.

4.2.6 Related work

One of the first approaches to migrating SGX enclaves is presented in [59]. The main idea of the approach is to pause the source enclave by entering into busy loops with all its worker threads. Then, a special control thread can transfer the enclave memory to the destination. In the destination enclave, the execution resumes by breaking out of busy loops.

This method addresses the first challenge of TA migration by reading and transferring the enclave memory from within the enclave. However, if the VM contained sealed data of the enclave, it will not be able to decrypt it on the destination machine. Additionally, this approach does not take into account the state of other architectural components, such as monotonic counters.

TEEnder [61] describes a framework utilizing Hardware Security Modules (HSM) that makes the migration process more efficient. Emotion [95] proposes to extend the SGX architecture to support the migration operation.

Containers as a more lightweight alternative to VMs can be run inside TEEs [11]. Liang et al. [76] present a solution for migrating SGX containers.

4.3 Discussion

4.3.1 Trusted storage

If the storage provider is trusted, there is no need to rely on counters for freshness guarantees. However, applications still need to persist some data to mutually authenticate with the storage provider. Essentially, the authentication data needs to be migratable, thus even if the storage is trusted, the migration framework is still needed.

4.3.2 Migrating to a trusted destination

It should not be possible for an attacker to migrate an enclave from the cloud provider machine to an attacker-controlled machine. Even if the attacker is migrating to another genuine SGX-capable machine, taking control over the enclave significantly increases the attack surface. For example, the attacker-controlled machine may not have the latest security patches, and the attacker can perform offline attacks on the sealed data.

Thus, the destination of the migration should be able to prove that it belongs to the same cloud provider as the source machine. The attestation process does not include the *authorization* of the destination: the destination should be verified by other means. One possible mechanism to perform the authorization of the destination is based on verifying a certificate belonging to the cloud provider.

When a new physical machine is added to the cluster of the cloud provider, the migration enclave of this machine should generate a key pair. The certificate that contains the public key should be transferred to other migration enclaves so that they become aware of the new machine. Alternatively, the migration enclaves themselves could have the cloud provider's public key included with their code. This way, the source ME can provide its certificate at migration time, and the destination ME can verify that this was signed by the cloud provider.

Clients might use several cloud providers at the same time. To support migration between different clouds cloud providers can agree to trust certificates issued by one another. An alternative solution can be based on the client performing the authorization. For example, a client can provide its own certificate to be included into the list of certificates trusted by migration enclaves. The corresponding private key can be provisioned to the client's enclave over a secure channel.

Note that the migration enclave itself should not be migratable, i.e. it should not contain the migration library and it should be bound to a particular physical machine. The generated private key has to be sealed with SGX primitives to prevent the migration enclave from executing on other physical machines.

To securely generate the key pair and transfer the certificate to another migration enclave, a trusted setup phase can be assumed when the new physical machine is disconnected from the network and is provisioned by the software verified by the cloud provider.

4.3.3 Non-cooperative TEE

The migration framework assumes that the migrating enclave does not try to attack the migration library. A malicious enclave can pretend to be migratable and secure by including the migration library, but then

it could ignore the migration library's primitive restrictions and allow a rollback attack to be executed. For example, in the case of a malicious operator in either SafeKeeper or Cloud Key Store who is willing to reset the rate-limiting counter, the operator could include the migration library but after the migration, ignore the counter offset (see Section 4.2.3) and perform a rollback attack.

However, the problem of malicious enclaves (that try to mislead the verifier that they are checking the counter values but allow rollback attacks) does not get worse by introducing the migration library. Indeed, an enclave might pretend to store the counter values with the data, but then "forget" to check those values, thus opening themselves to rollback attacks. Not checking the counter value is equivalent to ignoring the counter offset when the migration library is present. Both situations are detectable from the attestation. It should not be assumed that if the enclave includes the migration library it uses it in a way that prevents rollback attacks.

In general, the problem of isolating parts of a TA from each other is important, since a TA might be executing third-party code that cannot be verified beforehand. A possible solution to this problem is described in the next chapter.

4.3.4 Conclusion

This chapter answers **RQ2** by introducing a framework for TA migration. Reconciling cloud practices with hardware binding nature of TEEs required developing migratable alternatives of the TEE primitives, such as sealing and monotonic counters. An RA-based protocol verifies that both source and destination machines run genuine TEEs and belong to the same cloud provider. Developing migratable TAs does not require significant developer efforts, and the performance overhead for migratable TAs is acceptable.

Currently, major cloud providers do not support live migration of TAs [83, 50]. But migration is not the only cloud practice that needs to be taken into account when deploying TAs in the cloud. For example, deploying multiple instances of a TA at scale is addressed in [111]. It is important to resolve infrastructure challenges so that TEEs in the cloud would see a wider adoption.

5. TEE-supported cloud resource consumption measurements

Different ways of providing cloud services offer different levels of infrastructure abstractions: from full-stack applications to bare metal machines. Generic computation services let the clients run arbitrary code on cloud provider premises. Several types of cloud architectures realize different variants of remote computing. The main difference between these types is the software level at which the clients are provided access to the cloud provider machines.

There is a tendency to reduce the amount of code considered as a single execution unit. Virtualization enabled cloud providers to host many clients' VMs on the same physical machine [69]. Container technologies allowed for isolation within a VM [20]. One of the more recent developments, called Function-as-a-Service (FaaS), utilizes containers to execute a single function from the client.Examples of FaaS are Amazon Lambda [6], Google Cloud Functions [51], and Microsoft Azure Functions [84].

The business model of cloud providers is based on measuring the computational resources that are consumed by the clients: for example, the type of VM and the amount of time for which it is running in IaaS setups. Resource measurements are essential for cloud providers and their clients as they serve as the basis for the business relationships between the parties, and therefore must be trustworthy from both the cloud provider's and clients' persepctives. In FaaS setups, the resource measurements should not only be trustworthy, but must also have finer granularity than in traditional settings.

Trustworthy resource measurements are especially important in cases where cloud providers cannot rely on reputational trust. In theory, even individuals could give access to their computational resources for small computations and get paid based on the resources consumed. For example, a peer-to-peer cloud architecture was proposed in [14] and Gridcoin [56] project rewards users that share their computation resources for scientific projects. In this case, the clients have to be sure that the reported resource consumption was not tampered with by the provider.

This chapter addresses **RQ3**: how can cloud providers use TEEs to pro-

duce trustworthy resource usage measurements? It describes a TEE-based framework that enables fine-grained trustworthy resource measurements in the clouds. Additionally, executing functions inside TEEs provides a secure isolated execution environment. Practically, when TEEs are deployed in the cloud, this creates a new cloud offering for the clients: secure computing. TEE functionality, isolated execution, and attestation enable new features that would not be possible without TEEs.

5.1 Motivation

5.1.1 Adversary model

The cloud provider is not trusted to provide trustworthy resource measurements or to even execute the function submitted by the client. TEEs can provide guarantees of secure remote computing when the cloud provider is adversarial.

Since cloud providers are billing clients based on consumed resources, both parties need to obtain trustworthy resource measurements. If the cloud provider is adversarial, then its goal is to increase the reported resources consumed by the clients without actually performing the requested amount of work, whereas the goal of the adversarial clients is to tamper with the resource measurement mechanism of the cloud provider to decrease the reported consumed resources. Hence, proper isolation between the resource measurement mechanism and the client's code should be in place.

5.1.2 Resource measurements

Resources that should be measured usually include CPU and memory consumption. Depending on the nature of the executed computation, network bandwidth could also be one of the resources to measure.

Currently, the granularity of the resource measurements is coarse. For example, cloud providers measure the execution time of the function and the *allocated* memory. By multiplying these two values an integral representing both CPU and memory consumption is computed. This measurement serves as the basis for the billing plan. The number of function invocations also contributes to the cost. Table 5.1 shows what quantities different cloud providers use for billing.

More precise and trustworthy resource measurements would be beneficial both for cloud providers and clients by reducing the costs, and allowing better resource utilization for cloud providers. TEEs enable more granular, attestable resource measurement mechanisms. **Publication IV**

Table 5.1. Table 1 in Publication IV. Billing policies of some current FaaS services. Calculating the memory time-integral implicitly requires measuring compute time (indicated by ○).

Service Provider	Runs	Time	Memory	Network
Amazon [6]	✓	○	✓	
Microsoft [84]	✓	○	✓	
Google [51]	✓	✓	✓	✓
IBM [66]	✓	○	✓	

introduces S-FaaS (Secure Function-as-a Service) that applies TEEs for providing generic FaaS computation and allows performing fine-grained resource measurements.

Brenner et al. [27] propose Javascript-based frameworks for sandboxing and evaluate them from a performance perspective, but does not address the resource measurements aspect of secure FaaS. Goltzsche et al. [48] introduce WebAssembly-based sandbox with resource accounting. It requires code instrumentation to count the number of WebAssembly instructions executed by the workload.

5.2 FaaS with TEE

5.2.1 S-FaaS design

S-FaaS utilizes Intel SGX to execute functions provided by the clients inside secure enclaves. The functions are written in JavaScript, but in general, they can be developed in any interpreted language. As described in Section 2.1, cloud providers' clients provide services to end users. In FaaS architectures, end users invoke the functions provided by the clients.

When end users invoke the function, they provide the inputs for the function. Since cloud providers are not trusted, both the inputs of the function and the results of the invocation have to be encrypted so that they are accessible only within the enclave to protect their confidentiality and integrity. The S-FaaS framework consists of several components and seeks to address the following two main technical challenges.

Fast, scalable attestation. Since in FaaS scenarios it is not known beforehand what particular worker node will execute the workload, traditional SGX attestation does not scale well. It is infeasible to attest the worker enclave before every function invocation, therefore, a different attestation mechanism is necessary to attest many worker nodes and scale well with an increasing number of worker nodes.

Isolated resource measurements. On one hand, it is necessary to

isolate the resource measurements from the clients' workloads so that the clients cannot tamper with the resource measurements. On the other hand, the resource measurement mechanism should be able to track when the client's workload is interrupted by the OS so that the client is charged only for the time when the workload is executing.

To address the above challenges, S-FaaS makes two main contributions: transitive attestation, and trustworthy TEE resource measurement, as explained in the following subsections.

5.2.2 Transitive attestation

The architecture of S-FaaS is presented in Figure 5.1.

When an end user executes a function, they need to attest the enclave that is going to execute the function and establish a secure channel with the enclave to provide function inputs. Since it is not viable to perform the standard SGX remote attestation for each function invocation, S-FaaS introduces a transitive attestation scheme. There is a centralized enclave called Key Distribution Enclave (KDE) that attests worker enclaves and serves as a key management service. The clients attest KDE and KDE attests the worker enclaves, so the clients can be sure that the function will be invoked by a trusted worker thread and establish an end-to-end trust.

KDE generates the following key pairs for worker enclaves and the client.

- Clients use a **key agreement key** to generate the symmetric encryption key for the input to the worker enclave.
- The worker enclave uses an **output signing key** to sign the output of the function.
- A **resource measurement signing key** is used to sign the resource measurement receipt.

5.2.3 TEE resource measurement

The main challenge in measuring the CPU consumption of a TA is the fact that the enclave can be interrupted by the OS at any time. These interruptions are transparent to the TA. Obtaining time measurements by calling APIs like `sgx_get_trusted_time` is not secure within our adversary model because the calls can be arbitrarily delayed by an untrusted OS. The resource measurement mechanism of S-FaaS is based on Intel Transactional Synchronization Extensions (TSX) that allow the resource measurement mechanism and the workload executor to synchronize and

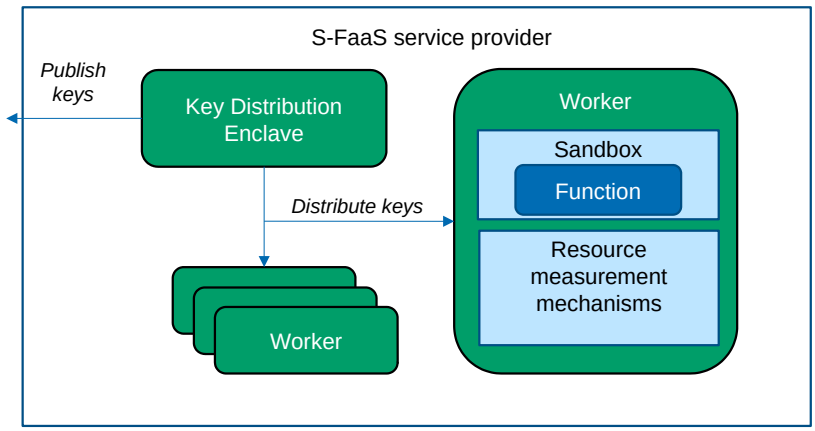

Figure 5.1. Figure 2 in Publication IV. S-FaaS architecture.

be aware of OS interruptions.

The main idea of S-FaaS for resource measurements is to run two threads inside the enclave. One of the threads, called *the worker thread*, executes the client's workloads. Another thread, *the timer thread*, is dedicated to resource measurements. The timer thread runs a busy loop and the measurements are based on the number of cycles the loop is executed, while the worker thread executes the workload. TSX is utilized to enable "notifications" from the worker thread to the timer thread about worker thread interruptions by OS.

The enclave needs to obtain information about interruptions. When the OS interrupts a thread running inside an enclave, the SGX runtime stores the state of the CPU registers in a memory region called the State Save Area (SSA) inside the enclave. By monitoring this area from another thread in the enclave, it is possible to determine when the interruption occurs.

The monitoring mechanism is based on TSX. As described in Chapter 2, TSX was originally developed to optimize concurrent execution solutions, so that locking mechanisms are not needed. A transaction has a read-set – a memory region, writing to which from another thread aborts the transaction. This can be used as a communication channel from one thread to another. If the timer thread includes the SSA of the worker thread into a read-set of the transaction, and the transaction aborts, that means that SSA was written to, hence the worker thread got interrupted. At this point, the timer thread stops counting its busy loops.

When the worker thread resumes its execution, it executes a resume handler. The default resume handler restores the CPU state from SSA and continues the execution from the next instruction. The timer thread needs

to be notified about the resuming of the worker thread execution. At the time of interruption, the timer thread substitutes the SGX default resume handler with a custom version to get a notification when the worker thread continues the execution. This way, the timer thread can restart counting CPU cycles when the worker thread resumes.

5.3 Discussion

Publication IV, presenting S-FaaS, constitutes an affirmative answer to **RQ3**. The performance overhead of the framework is below 7%, which is an acceptable price for authenticated and integrity-protected resource measurements. The framework provides accurate resource measurements (when the parameters are chosen correctly, see 5.3.3 below). This work has raised the following fundamental questions, and while **Publication IV** provides possible answers for each, there is scope for further research in each of the following areas.

5.3.1 In-enclave sandboxing

The worker thread in S-FaaS uses a Javascript interpreter as a sandbox for functions submitted by the client. Since there is a sandbox inside the enclave, the enclave itself does not need to be sandboxed. Therefore, it is no longer necessary from a security perspective to run the enclaves inside separate VMs or containers. This can result in reducing the overhead of executing enclaves.

However, potential bugs in the interpreter could result in functions escaping the sandbox. In this case, the cloud provider could introduce worker enclaves that only execute a specific function or functions from a specific client (see Section 8.1.4 in **Publication IV**).

5.3.2 Units of work

CPU consumption can be measured in CPU cycles, or in the number of instructions that are executed. For example, S-FaaS measures work in terms of CPU cycles, whereas in [127], the work is measured in terms of the number of instructions executed.

Different approaches can result in different measurements of the same workload. Modern CPUs are not deterministic, meaning that one instruction can take different numbers of CPU cycles based on many factors. For example, an instruction can spend CPU cycles waiting for cached data. Compilers attempt to re-order instructions to optimize the execution time. CPU features, such as speculative execution and branch prediction can also affect the number of CPU cycles that a particular instruction takes.

The client is interested in executing instructions, not in how many CPU cycles they take. However, the cloud provider sees the resources spent as the number of CPU cycles because practically CPUs consume energy proportionally to the number of executed cycles. Hence, if the client submits a workload that is not optimized, requiring more CPU cycles than ncessary, it is logical that the client should be penalized. But if the cloud provider executes many workloads from different (or the same) clients that compete for cache access, then the clients are paying more for CPU cycles that are caused by the cloud provider actions.

The cycle counting and instruction counting approaches can be combined. This would give clients the average number of CPU cycles per each instruction in the workload. Using this scaling factor clients can measure the reference time it takes to execute their workload and compare the measurements provided by the cloud provider with the reference time to detect the situations when the cloud provider executes many workloads from the clients. The cloud provider Service Level Agreement (SLA) can have a statement that the cloud provider does not hinder the client's workload's performance. If the cloud provider does hinder the client's performance and this is detected by the client, the situation can be resolved through a business process.

5.3.3 Measurements granularity

The granularity of the measurements can be adjusted by changing the maximum value of the timer thread's busy loop. To calibrate measurements, cloud providers could execute an example workload on an unloaded machine.

If the value is too high, then it is more likely that the worker thread gets interrupted in the middle of a timer thread loop, and this increases the probability of underreporting because S-FaaS does not count partially-completed timer thread transactions. In this way, the cloud providers cannot benefit from interrupting the worker thread. The opposite choice (to include the partially-completed timer thread transactions into the final measurement) would result in over-reporting. If the value is too small, then the overhead of setting up the TSX transaction might also result in under-reporting.

Hence under-reporting can occur for different reasons, as explained above. However, there is an optimal maximum value for a timer thread's busy loop that minimizes under-reporting. The cloud provider should find the value that produces time estimates closest to how much time the computation actually takes (see Figure 5 in **Publication IV**).

If the timer thread and the worker thread execute with different CPU clock frequencies, the resulting measurements can be different. So it is important to make sure that the worker thread and the timer thread share

the same CPU core. It is also beneficial to execute a trusted sibling thread on the same core as the main enclave thread to defend against several attacks on SGX enclaves [92, 36]. Utilizing such a dedicated thread for the resource measurement does not waste one thread of execution.

5.3.4 Conclusion

This chapter answers **RQ3** by introducing a resource accounting framework for TAs in the cloud. Resource accounting is an essential part of the cloud providers' business model, and the S-FaaS framework provides a mechanism to obtain trustworthy resource measurements when the reputational trust assumption does not hold. The framework has an acceptable performance overhead and does not require code instrumentation by the clients.

6. Discussion

This chapter discusses three cross-cutting aspects that arise from the research presented in the preceding three chapters:

- Server-side remote attestation.
- Developing scalable secure applications.
- Trust and risk management for hardware security technologies.

6.1 Remote attestation

6.1.1 Remotely attesting clients and servers

Client-side TEE. Traditionally, remote attestation was used in scenarios where there are multiple provers and a single verifier. A service provider that had distributed many devices containing TEEs to end-users had to verify each device. In this case, there is one verifier and many devices to be attested, i.e. provers.

The service provider that acts as a verifier has to make sure that the user's device is genuine. For example, the provider can provision the device with secret data only after a successful attestation. Each device does not have to be individually authenticated by the service provider. Group signature schemes can serve the purpose of attesting a device without individually authenticating it [28, 29].

In the case of SGX, Intel has designed the attestation so that the verifier has to be registered with Intel, because only Intel Attestation Service (IAS) can verify the quotes from SGX enclaves. This emphasizes the fact that there is assumed to be one verifier that is registered with IAS.

On the other hand, the provers need to authenticate the verifier. For

example, the devices have to be sure that they are obtaining secrets from (or enrolling secrets to) the correct service provider. Typically, this is done via asymmetric cryptography. The service provider deploys its public key to the devices and they run a challenge-response protocol to achieve authentication of the service provider. This works well if there is one verifier with a known public key and many devices to be attested where this key can be deployed to.

Server-side TEE. When deploying TEEs on the server, the situation is different. The clients need to attest the TEE running in a server. In this case, there are many verifiers and one TEE to be attested. In many client-server use cases, the server does not need to authenticate the users (e.g. a public website). For example, in SafeKeeper, it is enough that the users authenticate the server, because the TEE is only used to calculate the password hash and the users do not provide the TEE with any data that can change its behavior.

Since there is no requirement for the TEE to authenticate the verifier, the challenge-response protocol used in the client-based attestation is not needed. Thus, the server-based remote attestation does not need two round-trips to complete. Indeed, the measurements of the TEE can be provided to the user along with the TEE response to the user's request.

A novel contribution of **Publication I** is to pre-generate the Diffie-Hellman key pair when the enclave starts up. Though there is no challenge-response protocol run to authenticate the verifier, during the attestation protocol a secure channel between the verifier and the prover is established. The quote provided by the prover includes its pre-generated Diffie-Hellman public key. The verifier has to generate a fresh Diffie-Hellman key pair each time it attests the prover to mitigate replay attacks.

Taking into account deployability considerations, the fact that there is no need for a challenge-response protocol provides ways to extend current client-server architectures with TEE-based server applications gradually. In particular, the remote attestation can run on top of the most widely used client-server HTTP protocol by including the quote into HTTP headers. This does not require changes to existing clients and servers at the same time – they can be updated independently of each other.

6.1.2 Transparency of the attestation

Another aspect related to remote attestation is its transparency. When attesting a server-side TEE that resides on some physical machine of the cloud provider, the cloud provider would not want to reveal which exact physical machine houses the TEE. Group signatures schemes with unlinkable signatures serve this purpose. However, it is still necessary to ensure that the TEE in question belongs to the correct cloud provider.

Binding to the cloud provider. The clients should be able to verify

that the TEE is running on a machine that belongs to the cloud provider. For example, in **Publication III**, when migrating a TEE from one physical platform to another, both the source and destination platforms must be able to establish that they are controlled by the same cloud provider to prevent an attacker from migrating enclaves into or out of the cloud provider's data center.

To provide machine identities, the cloud provider can provision each physical machine in its data center with a private key and certificate which asserts that the machine is part of the data center. Other schemes based on group signatures [31] are possible as well, but in general they share the same principle: a TA must contain a secret to prove that it belongs to the cloud provider.

One possible machine identities scheme is for each TEE to generate a key pair and the cloud provider to generate a certificate verifying the public key with the cloud provider's Certificate Authority (CA). During an attestation a TA signs a nonce value generated by the verifier. Then the verifier can check the signature against a public key of the TA and verify the certificate against the cloud provider's CA.

Signature-based revocation. Another related TEE cloud infrastructure problem is signature-based revocation. When an enclave running on one physical machine gets migrated, the quotes that it generates change. If later on the source physical machine is compromised and needs to be revoked, currently it is not possible to track where the enclaves that were running on it at some point were migrated. It is necessary for the cloud provider to explicitly track the locations of the running enclaves. The enclave migration and remote attestation thus have to fulfill contradictory requirements:

- Transparent to verifying clients.
- Belong to the same cloud provider.
- Track the history of where the enclave was running.

Perhaps the cloud providers do not have to support the last requirement since if one machine is compromised, it might be necessary to revoke others as well because there is no guarantee that other machines in the network are not compromised. In general, after a compromise, no assumptions can be made about what an attacker was able to achieve on a particular compromised machine as well as what consequences the attack had on other machines – it depends on the level of privileges that and attacker was able to gain and the auditing capabilities of the cloud provider [123].

6.1.3 Transitive attestation

Directly attesting a TEE might not be always possible. As already mentioned, in server-based attestation there are many verifiers and one TEE to be attested. But due to scalability requirements, it might be necessary to deploy a TA to several TEEs in the cloud. In this case, the client requests might get dynamically load-balanced to different TEEs. Attesting a TEE each time can take too long: ideally, for the clients it should look like the application is running in a single TEE. To deal with such a situation, there is an approach that introduces several layers of attesting parties.

In *layered attestation* or *transitive attestation*, two entities participate in the attestation process. First, the verifiers attest the *parent* entity to make sure that it will correctly perform the attestation of the *children* [12]. The parent is then used to provide children with secret data.

The verifiers can be sure that the data is processed by the correct children's TEEs because the parent TEE passes the children's measurements back to the verifiers when provisioning children. Thus, the attestation is transitive but the trust is not. However, if the parent TEE fails to pass the attestation, its children become untrusted as well: the distrust is transitive, too.

6.2 Scalability

6.2.1 Horizontal explicit scalability

The horizontal scalability of applications is achieved by deploying them to multiple machines. If the application is stateless, then this is a straightforward process. It is enough to load balance requests from the clients to several application instances. However, in developing stateful secure applications, scalability requirements must be considered.

If an application has a state of some sort, it is not possible to simply deploy multiple instances of it to several machines. The state has to be synchronized. For example, both SafeKeeper and Cloud Key Store need to maintain a counter of login attempts from each user. To synchronize the state between instances, they need to be aware of each other. This means that scalability has to be explicitly taken into account when developing TAs.

In recent years, cloud computing has moved from monolithic applications to microservices. One the key features of microservices is that they are stateless. It is assumed that the services can accept requests and serve them no matter how many other service replicas exist. But still, even in this architecture, there is usually a service (typically, a database) that

contains the persistent state.

To authenticate to that service, other applications, including TAs that need to persist state, need to have credentials for the authentication. But credentials themselves should be persisted, because they must be provisioned to the TA by the TA owner. Another alternative would be to hardcode the credentials into the TA code, but it is not a viable option from a security perspective. Hence, stateless applications are stateless only to some extent (see Section 1 in **Publication III**).

6.2.2 Scalability and security

Replicating the state of the TAs must take into account security requirements. In particular, rollback protection is important. When there are multiple copies of the application state, rollback protection becomes more challenging. Indeed, for example, when the enclave is migrated from one machine to another, and then migrated back, the migration process must ensure that the old state was invalidated.

Rollback protection is not the only issue to take care of when allowing multiple instances of the trusted application. In the example of a rate-limiting application, the maximum number of allowed attempts has to stay the same regardless of the number of application instances. This means that the application instances should be aware of one another. When a new instance is created, it should explicitly join the other instances. When the application is scaled down, the remaining instances should re-distribute the load. The deleted instances can be revoked to make sure that an attacker cannot utilize them to increase the guessing rate. This protocol is described in more detail in the technical report version of **Publication I** [74], but there is still scope for further research on this topic.

6.3 Trust and risk management

6.3.1 Trusting a TEE manufacturer

TEEs are not guaranteed to be absolutely secure. First, there are several attacks against TAs that can be used to retrieve secrets from TEEs. Second, even if the application is carefully designed to mitigate the attacks, the platform itself can contain bugs that leave all TAs vulnerable to attacks. For example, SGX side-channel attacks described in Section 2.2.2 break SGX security guarantees of integrity and confidentiality. Section 6.3.3 describes this scenario and discusses its implications. Furthermore, if a hardware manufacturer introduces a security vulnerability into the

hardware it is much harder to detect and mitigate [114]. In the case of Intel SGX, remote attestation was provided by Intel. There are third-party attestation verifiers as well [85], but technically, Intel or a third party could verify malicious enclaves, or introduce backdoors into the implementation.

It is a valid assumption that the manufacturer producing TEEs is incentivized to increase the security of its product. There are proposals on how to develop processors that are more robust to speculative execution attacks [60]. To mitigate the risk of trusting only one hardware manufacturer, solutions from multiple hardware manufacturers can be combined [3].

Without using a TEE, one has to trust both the cloud provider and the hardware manufacturer. Introducing a TEE is an example of separation of privileges: in order to attack a specific client a malicious cloud provider needs to cooperate with a malicious hardware manufacturer. The hardware manufacturer does not have the information about cloud provider's clients, but the cloud provider does, hence it is beneficial to remove the cloud provider from the TCB. Introducing TEEs will not decrease the level of security guarantees anyway, because the hardware manufacturer is trusted regardless of whether TEEs are used or not.

6.3.2 Designing trusted applications

There are SGX research proposals to run entire unmodified applications inside a TEE [105, 34]. The benefits of this approach are that it reduces the development time and effort and provides TEE security guarantees for legacy applications. However, it goes against one of the main concepts of a TEE: the goal is to minimize the Trusted Computing Base (TCB). If the entire operating system or an operating system library is run inside a TEE, then the TCB includes the code of the OS or the library. First, it is hard to attest complicated software. Second, the software may be vulnerable to side-channel attacks because it was not designed specifically to run inside a TEE. So the TAs need to be designed carefully and provide specific security-critical functionality.

Designing and implementing TAs is challenging [38]. Not only should the application not contain bugs, but it should be side-channel resistant. It was also shown that TEE SDKs themselves are often vulnerable to various attacks [117]. Minimizing application functionality can potentially lead to more secure applications. Smaller applications are easier to attest as well.

Even when the application is small and designed and implemented to be resistent to side-channel attacks, there is still a possibility that it will be vulnerable to other attacks. In this case, deploying a TA should be considered more from a risk management perspective. This means that running an application inside a TEE decreases the risks of data exposure, but does not eliminate them. Depending on the type of data that a TA is processing, utilizing a TEE might be a sufficient measure to protect

the data. If the risk is still considered too high, other activities should take place, for example, isolating the machine from external connectivity altogether. Another approach would be to run multiple instances of a TA and use a Byzantine fault-tolerant consensus protocol between them [102].

6.3.3 What if a TEE fails?

There are several ways in which a particular TEE can fail. When one physical machine turns out to be vulnerable and it is detected, the solution is straightforward: the machine should be revoked. The enclaves that were running on it can no longer be trusted. If the enclaves were provisioned with sensitive data it should be assumed that the data has been leaked. The revocation process should take into account the enclaves that were running on the vulnerable machine and those that were migrated. Additionally, if a machine contained enclaves that were instances of a scaled application, the other instances should be notified of the failure to re-synchronize the state.

On the other hand, new attacks against TEE implementations, and SGX in particular, emerge [30]. When we consider the possibility that a TEE might fail, there are different degrees of damage that applications can take. There are two distinct classes of applications.

TEE-enhanced applications. In one type of application, the usage of TEEs provides additional security. Essentially, the same application can be run without a TEE but then the security level would be lower. SafeKeeper is an example of such an application because it enhances the functionality of the existing application and only moves its functionality inside a TEE. If the TEE fails, this type of application falls back to the same level of security it had without the TEE.

TEE-dependent applications. On the other hand, another class of applications includes the ones designed specifically for TEEs. In particular, the Cloud Key Store (CKS) is one such application. Indeed, without a TEE, a user would probably store the key locally on the device. Moving the key to the cloud is secure only if the TEE works correctly. If the TEE fails, the applications and the users are left in a worse situation compared to TEE-enhanced applications. For example, in the case of CKS, if the TEE is compromised users' keys are leaked and need to be revoked.

Understanding potential risks and identifying the application type is important when providing TAs to end-users. If one application is riskier than the other, it should be explicitly stated, and the users should be able to make the right risk management decision. Server-side TAs are arguably more high-risk than client-side TAs because they could be centralized repositories of credentials (e.g. SafeKeeper and CKS). This should be factored into the risk calculation when deciding whether a particular TEE technology is suitable for server-side use.

6.4 Conclusion

Utilizing TEEs in the cloud is a promising approach to improve security and privacy guarantees provided by the cloud. Many applications have been developed in the recent years: TEE-enhanced databases [15, 71, 96], utilizing TEEs in blockchains [77, 103], securing network protocols [17, 88]. In this dissertation, we showed what design considerations need to be taken into account when developing TAs.

Major cloud providers started offering TEE-based services [83, 50]. Often infrastructure challenges are not considered and it limits the usage of TEEs in some scenarios. For example, live migration of TEE-enabled VMs is not supported. As the usage of TEEs in the cloud becomes more widespread, these challenges need to be addressed. Ideally, they should be addressed by TEE vendors, so that they introduce frameworks, similar to the one presented in the dissertation to support TAs live migration.

An example of how to obtain trustworthy resource measurements with TEEs in the cloud shows that TEEs in the cloud can be beneficial to both cloud providers and their clients. Future work can identify other areas where TEEs can enhance services provided by the cloud providers.

References

[1] Mete Akgün, A Osman Bayrak, Bugra Ozer, and M Şamil Sağıroğlu. Privacy preserving processing of genomic data: A survey. *Journal of biomedical informatics*, 56:103–111, 2015.

[2] A. Albeshri, C. Boyd, and J. G. Nieto. GeoProof: Proofs of geographic location for cloud computing environment. In *2012 32nd International Conference on Distributed Computing Systems Workshops*, pages 506–514, 2012.

[3] Fritz Alder. TEE-Squared - Combining Trusted Hardware to Enhance the Security of TEEs, 2018.

[4] M. Alhamad, T. Dillon, and E. Chang. SLA-based trust model for cloud computing. In *2010 13th International Conference on Network-Based Information Systems*, pages 321–324, 2010.

[5] Amazon. Amazon Web Services, 2021. `https://aws.amazon.com`. Accessed: 18-06-2021.

[6] Amazon. AWS Lambda, 2021. `https://aws.amazon.com/lambda/`. Accessed: 18-06-2021.

[7] Ittai Anati, Shay Gueron, Simon Johnson, and Vincent 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*, volume 13, 2013.

[8] Jay and Lorch, John (JD) Douceur, , and Jonathan M. McCune. Memoir: Practical state continuity for protected modules. In *Proceedings of the IEEE Symposium on Security and Privacy*, 2011.

[9] APWG.org. Phishing activity trends report (2nd quarter), 2020. `http://docs.apwg.org/reports/apwg_trends_report_q2_2020.pdf`. Accessed: 18-06-2021.

[10] ARM. TrustZone. `https://developer.arm.com/ip-products/security-ip/trustzone`. Accessed: 18-06-2021.

[11] Sergei Arnautov, Bohdan Trach, Franz Gregor, Thomas Knauth, Andre Martin, Christian Priebe, Joshua Lind, Divya Muthukumaran, Dan O'Keeffe, Mark L. Stillwell, David Goltzsche, Dave Eyers, Rüdiger Kapitza, Peter Pietzuch, and Christof Fetzer. SCONE: Secure Linux containers with Intel SGX. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, pages 689–703. USENIX Association, 2016.

[12] N. Asokan, Ferdinand Brasser, Ahmad Ibrahim, Ahmad-Reza Sadeghi, Matthias Schunter, Gene Tsudik, and Christian Wachsmann. SEDA: Scalable embedded device attestation. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, CCS '15, page 964–975. Association for Computing Machinery, 2015.

[13] Pierre-Louis Aublin, Florian Kelbert, Dan O'keeffe, Divya Muthukumaran, Christian Priebe, Joshua Lind, Robert Krahn, Christof Fetzer, David Eyers, and Peter Pietzuch. TaLoS: Secure and transparent TLS termination inside SGX enclaves. *Imperial College London, Tech. Rep*, 5, 2017.

[14] Ozalp Babaoglu, Moreno Marzolla, and Michele Tamburini. Design and implementation of a P2P cloud system. In *Proceedings of the 27th Annual ACM Symposium on Applied Computing*, SAC '12, page 412–417, New York, NY, USA, 2012. Association for Computing Machinery.

[15] Maurice Bailleu, Jörg Thalheim, Pramod Bhatotia, Christof Fetzer, Michio Honda, and Kapil Vaswani. SPEICHER: Securing LSM-based key-value stores using shielded execution. In *17th USENIX Conference on File and Storage Technologies (FAST 19)*, pages 173–190. USENIX Association, 2019.

[16] Ioana Baldini, Paul Castro, Kerry Chang, Perry Cheng, Stephen Fink, Vatche Ishakian, Nick Mitchell, Vinod Muthusamy, Rodric Rabbah, Aleksander Slominski, and Philippe Suter. *Serverless Computing: Current Trends and Open Problems*, pages 1–20. Springer Singapore, 2017.

[17] Johannes Behl, Tobias Distler, and Rüdiger Kapitza. Hybrids on steroids: SGX-based high performance BFT. In *Proceedings of the Twelfth European Conference on Computer Systems*, EuroSys '17, 2017.

[18] Stefan Berger, Ramón Cáceres, Kenneth A. Goldman, Ronald Perez, Reiner Sailer, and Leendert van Doorn. vTPM: Virtualizing the trusted platform module. In *Proceedings of the 15th Conference on USENIX Security Symposium - Volume 15*, USENIX-SS'06, pages 305–320. USENIX Association, 2006.

[19] Stefan Berger, Ramón Cáceres, Dimitrios Pendarakis, Reiner Sailer, Enriquillo Valdez, Ronald Perez, Wayne Schildhauer, and Deepa Srinivasan. TVDc: managing security in the trusted virtual datacenter. *ACM SIGOPS Operating Systems Review*, 42(1):40–47, 2008.

[20] David Bernstein. Containers and cloud: From LXC to docker to kubernetes. *IEEE Cloud Computing*, 1(3):81–84, 2014.

[21] Joseph Birr-Pixton. Using SGX to harden password hashing, 2016. `https://jbp.io/2016/01/17/using-sgx-to-hash-passwords`.

[22] Sören Bleikertz, Sven Bugiel, Hugo Ideler, Stefan Nürnberger, and Ahmad-Reza Sadeghi. Client-controlled Cryptography-as-a-Service in the cloud. In *Proceedings of the 11th International Conference on Applied Cryptography and Network Security*, ACNS'13, pages 19–36. Springer-Verlag, 2013.

[23] J. Bonneau, C. Herley, P. C. v. Oorschot, and F. Stajano. The quest to replace passwords: A framework for comparative evaluation of web authentication schemes. In *2012 IEEE Symposium on Security and Privacy*, pages 553–567, 2012.

[24] Marcus Brandenburger, Christian Cachin, Matthias Lorenz, and Rüdiger Kapitza. Rollback and forking detection for Trusted Execution Environments using lightweight collective memory. In *47th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, DSN 2017*, pages 157–168, 2017.

[25] Ferdinand Brasser, Srdjan Capkun, Alexandra Dmitrienko, Tommaso Frassetto, Kari Kostiainen, and Ahmad-Reza Sadeghi. DR.SGX: Automated and adjustable side-channel protection for SGX using data location randomization. In *Proceedings of the 35th Annual Computer Security Applications Conference*, ACSAC '19, page 788–800. Association for Computing Machinery, 2019.

[26] Ferdinand Brasser, Urs Müller, Alexandra Dmitrienko, Kari Kostiainen, Srdjan Capkun, and Ahmad-Reza Sadeghi. Software Grand Exposure: SGX cache attacks are practical. In *11th USENIX Workshop on Offensive Technologies (WOOT 17)*. USENIX Association, 2017.

[27] Stefan Brenner and Rüdiger Kapitza. Trust more, serverless. In *Proceedings of the 12th ACM International Conference on Systems and Storage*, SYSTOR '19, page 33–43. Association for Computing Machinery, 2019.

[28] E. Brickell and J. Li. Enhanced privacy ID from bilinear pairing for hardware authentication and attestation. In *2010 IEEE Second International Conference on Social Computing*, pages 768–775, 2010.

[29] Ernie Brickell, Jan Camenisch, and Liqun Chen. Direct anonymous attestation. In *Proceedings of the 11th ACM Conference on Computer and Communications Security*, CCS '04, page 132–145, New York, NY, USA, 2004. Association for Computing Machinery.

[30] Jo Van Bulck, Marina Minkin, Ofir Weisse, Daniel Genkin, Baris Kasikci, Frank Piessens, Mark Silberstein, Thomas F. Wenisch, Yuval Yarom, and Raoul Strackx. Foreshadow: Extracting the keys to the Intel SGX kingdom with transient out-of-order execution. In *27th USENIX Security Symposium (USENIX Security 18)*, page 991–1008. USENIX Association, 2018.

[31] Jan Camenisch and Markus Stadler. Efficient group signature schemes for large groups. In *Advances in Cryptology — CRYPTO '97*, pages 410–424, Berlin, Heidelberg, 1997. Springer Berlin Heidelberg.

[32] Emanuele Cesena, Hans Löhr, Gianluca Ramunno, Ahmad-Reza Sadeghi, and Davide Vernizzi. Anonymous authentication with TLS and DAA. In *Trust and Trustworthy Computing*, pages 47–62. Springer Berlin Heidelberg, 2010.

[33] Somnath Chakrabarti, Brandon Baker, and Mona Vij. Intel SGX enabled key manager service with OpenStack Barbican, 2017. `https://arxiv.org/abs/1712.07694`.

[34] Chia che Tsai, Donald E. Porter, and Mona Vij. Graphene-SGX: A practical library OS for unmodified applications on SGX. In *2017 USENIX Annual Technical Conference (USENIX ATC 17)*, pages 645–658. USENIX Association, 2017.

[35] G. Chen, S. Chen, Y. Xiao, Y. Zhang, Z. Lin, and T. H. Lai. SgxPectre: Stealing Intel secrets from SGX enclaves via speculative execution. In *2019 IEEE European Symposium on Security and Privacy (EuroS P)*, pages 142–157, 2019.

[36] G. Chen, M. Li, F. Zhang, and Y. Zhang. Defeating speculative-execution attacks on SGX with HyperRace. In *2019 IEEE Conference on Dependable and Secure Computing (DSC)*, pages 1–8, 2019.

[37] Christopher Clark, Keir Fraser, Steven Hand, Jacob Gorm Hansen, Eric Jul, Christian Limpach, Ian Pratt, and Andrew Warfield. Live migration of virtual machines. In *Proceedings of the 2nd conference on Symposium on Networked Systems Design & Implementation-Volume 2*, pages 273–286, 2005.

[38] Tobias Cloosters, Michael Rodler, and Lucas Davi. TeeRex: Discovery and exploitation of memory corruption vulnerabilities in SGX enclaves. In *29th USENIX Security Symposium (USENIX Security 20)*, pages 841–858. USENIX Association, 2020.

[39] Cloudflare. What is FaaS?, 2021. https://www.cloudflare.com/learning/serverless/glossary/function-as-a-service-faas/. Accessed: 18-06-2021.

[40] George Coker, Joshua Guttman, Peter Loscocco, Amy Herzog, Jonathan Millen, Brian O'Hanlon, John Ramsdell, Ariel Segall, Justin Sheehy, and Brian Sniffen. Principles of remote attestation. *International Journal of Information Security*, 10(2):63–81, 2011.

[41] Victor Costan and Srinivas Devadas. Intel SGX explained. *IACR Cryptology ePrint Archive*, 2016(086):1–118, 2016.

[42] Boris Danev, Ramya Jayaram Masti, Ghassan O. Karame, and Srdjan Capkun. Enabling secure VM-vTPM migration in private clouds. In *Proceedings of the 27th Annual Computer Security Applications Conference*, ACSAC '11, pages 187–196. ACM, 2011.

[43] Digital Ocean. Digital Ocean Cloud Computing, 2021. https://www.digitalocean.com. Accessed: 18-06-2021.

[44] Jan-Erik Ekberg, Kari Kostiainen, and N. Asokan. Trusted Execution Environments on mobile devices. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, pages 1497–1498, 2013.

[45] Jan-Erik Ekberg, Kari Kostiainen, and N Asokan. The untapped potential of trusted execution environments on mobile devices. *EEE Security & Privacy*, 12(4):29–37, 2014.

[46] Paul England. Practical techniques for operating system attestation. In *Trusted Computing - Challenges and Applications*, pages 1–13, 2008.

[47] Craig Gentry. Fully homomorphic encryption using ideal lattices. In *Proceedings of the Forty-First Annual ACM Symposium on Theory of Computing*, STOC '09, page 169–178. Association for Computing Machinery, 2009.

[48] David Goltzsche, Manuel Nieke, Thomas Knauth, and Rüdiger Kapitza. AccTEE: A WebAssembly-based two-way sandbox for trusted resource accounting. In *Proceedings of the 20th International Middleware Conference*, Middleware '19, page 123–135. Association for Computing Machinery, 2019.

[49] Google. Google Cloud, 2021. https://cloud.google.com. Accessed: 18-06-2021.

[50] Google. Google Cloud Confidential Computing, 2021. https://cloud.google.com/confidential-computing. Accessed: 18-06-2021.

[51] Google. Google Cloud Functions, 2021. https://cloud.google.com/functions. Accessed: 18-06-2021.

[52] Google. Google Infrastructure Security Design Overview, 2021. https://cloud.google.com/security/infrastructure/design/. Accessed: 18-06-2021.

[53] Google. Google Kubernetes Engine, 2021. https://cloud.google.com/kubernetes-engine. Accessed: 18-06-2021.

[54] Google. SafetyNet Attestation API, 2021. https://developer.android.com/training/safetynet/attestation.html. Accessed: 18-06-2021.

[55] Johannes Götzfried, Moritz Eckert, Sebastian Schinzel, and Tilo Müller. Cache attacks on Intel SGX. In *10th European Workshop on Systems Security*, 2017.

[56] Gridcoin. Rewarding Volunteer Distributed Computing, 2021. https://gridcoin.us/. Accessed: 18-06-2021.

[57] Trusted Computing Group. TPM main specification Level 2, version 1.2, revision 116. Technical report, National Institute of Standards and Technology, 2011.

[58] Nils Gruschka and Meiko Jensen. Attack surfaces: A taxonomy for attacks on cloud services. In *2010 IEEE 3rd International Conference on Cloud Computing*, pages 276–279, 2010.

[59] J. Gu, Z. Hua, Y. Xia, H. Chen, B. Zang, H. Guan, and J. Li. Secure live migration of SGX enclaves on untrusted cloud. In *2017 47th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pages 225–236, 2017.

[60] Marco Guarnieri, Boris Köpf, Jan Reineke, and Pepe Vila. Hardware-software contracts for secure speculation, 2020. https://arxiv.org/abs/2006.03841.

[61] João Guerreiro, Rui Moura, and João Nuno Silva. TEEnder: SGX enclave migration using HSMs. *Computers & Security*, 96:101874, 2020.

[62] Per Hammarlund, Alberto J Martinez, Atiq A Bajwa, David L Hill, Erik Hallnor, Hong Jiang, Martin Dixon, Michael Derr, Mikal Hunsaker, Rajesh Kumar, et al. Haswell: The fourth-generation Intel core processor. *IEEE Micro*, 34(2):6–20, 2014.

[63] Weili Han, Zhigong Li, Minyue Ni, Guofei Gu, and Wenyuan Xu. Shadow attacks based on password reuses: A quantitative empirical view. *IEEE Transactions on Dependable and Secure Computing*, 2016.

[64] Heroku. Cloud Application Platform, 2021. https://www.heroku.com. Accessed: 18-06-2021.

[65] Michael R. Hines, Umesh Deshpande, and Kartik Gopalan. Post-copy live migration of virtual machines. *SIGOPS Oper. Syst. Rev.*, page 14–26, 2009.

[66] IBM. IBM Cloud Functions, 2021. https://cloud.ibm.com/functions/. Accessed: 18-06-2021.

[67] Intel. Speculative Execution Side Channel Mitigations, 2021. https://www.intel.com/content/www/us/en/developer/articles/technical/software-security-guidance/technical-documentation/speculative-execution-side-channel-mitigations.html. Accessed: 28-10-2021.

[68] David Jaeger, Chris Pelchen, Hendrik Graupner, Feng Cheng, and Christoph Meinel. Analysis of publicly leaked credentials and the long story of password (re-)use. In *Conference on Passwords*, Bochum, Germany, 2016.

[69] Raj Jain and Subharthi Paul. Network virtualization and software defined networking for cloud computing: a survey. *IEEE Communications Magazine*, 51(11):24–31, 2013.

[70] H. Kang, M. Le, and S. Tao. Container and microservice driven design for cloud infrastructure DevOps. In *2016 IEEE International Conference on Cloud Engineering (IC2E)*, pages 202–211, 2016.

[71] Taehoon Kim, Joongun Park, Jaewook Woo, Seungheun Jeon, and Jaehyuk Huh. ShieldStore: Shielded in-memory key-value storage with SGX. In *Proceedings of the Fourteenth EuroSys Conference 2019*, EuroSys '19. Association for Computing Machinery, 2019.

[72] Thomas Knauth, Michael Steiner, Somnath Chakrabarti, Li Lei, Cedric Xing, and Mona Vij. Integrating remote attestation with transport layer security, 2019.

[73] P. Kocher, J. Horn, A. Fogh, D. Genkin, D. Gruss, W. Haas, M. Hamburg, M. Lipp, S. Mangard, T. Prescher, M. Schwarz, and Y. Yarom. Spectre attacks: Exploiting speculative execution. In *2019 IEEE Symposium on Security and Privacy (SP)*, pages 1–19, 2019.

[74] Klaudia Krawiecka, Arseny Kurnikov, Andrew Paverd, Mohammad Mannan, and N. Asokan. SafeKeeper: Protecting web passwords using trusted execution environments, 2017. https://arxiv.org/abs/1709.01261.

[75] Sangho Lee, Ming-Wei Shih, Prasun Gera, Taesoo Kim, Hyesoon Kim, and Marcus Peinado. Inferring fine-grained control flow inside SGX enclaves with branch shadowing. In *USENIX Security Symposium*, 2017.

[76] H. Liang, Q. Zhang, M. Li, and J. Li. Toward migration of SGX-enabled containers. In *2019 IEEE Symposium on Computers and Communications (ISCC)*, pages 1–6, 2019.

[77] Joshua Lind, Ittay Eyal, Peter R. Pietzuch, and Emin Gün Sirer. Teechan: Payment channels using Trusted Execution Environments, 2016. https://arxiv.org/abs/1612.07766.

[78] Linux. Kernel Virtual Machine, 2021. https://www.linux-kvm.org/page/Main_Page. Accessed: 18-06-2021.

[79] Moritz Lipp, Michael Schwarz, Daniel Gruss, Thomas Prescher, Werner Haas, Anders Fogh, Jann Horn, Stefan Mangard, Paul Kocher, Daniel Genkin, Yuval Yarom, and Mike Hamburg. Meltdown: Reading kernel memory from user space. In *27th USENIX Security Symposium (USENIX Security 18)*, 2018.

[80] Brian McGillion, Tanel Dettenborn, Thomas Nyman, and N. Asokan. OpenTEE–an open virtual Trusted Execution Environment. In *2015 IEEE Trustcom/BigDataSE/ISPA*, volume 1, pages 400–407, 2015.

[81] Peter M. Mell and Timothy Grance. SP 800-145. The NIST Definition of Cloud Computing. Technical report, National Institute of Standards and Technology, Gaithersburg, MD, USA, 2011.

[82] Microsoft. Azure, 2021. https://azure.microsoft.com/en-us/. Accessed: 18-06-2021.

[83] Microsoft. Azure Confidential Computing, 2021. https://azure.microsoft.com/en-us/solutions/confidential-compute/. Accessed: 18-06-2021.

[84] Microsoft. Azure Functions, 2021. https://azure.microsoft.com/en-us/services/functions/. Accessed: 18-06-2021.

[85] Microsoft. Microsoft Azure Attestation, 2021. https://docs.microsoft.com/en-us/azure/attestation. Accessed: 18-06-2021.

[86] Microsoft. What is SaaS?, 2021. https://azure.microsoft.com/en-us/overview/what-is-saas/. Accessed: 18-06-2021.

[87] Kit Murdock, David Oswald, Flavio D. Garcia, Jo Van Bulck, Daniel Gruss, and Frank Piessens. Plundervolt: Software-based fault injection attacks against Intel SGX. In *41st IEEE Symposium on Security and Privacy (S&P'20)*, 2020.

[88] Yoshimichi Nakatsuka, Andrew Paverd, and Gene Tsudik. PDoT: Private DNS-over-TLS with TEE support. In *Proceedings of the 35th Annual Computer Security Applications Conference*, ACSAC '19, page 489–499. Association for Computing Machinery, 2019.

[89] Michael Nelson, Beng-Hong Lim, Greg Hutchins, et al. Fast transparent migration for virtual machines. In *USENIX Annual Technical Conference*, pages 391–394, 2005.

[90] NIST. Zero Trust Architecture. `https://csrc.nist.gov/publications/detail/sp/800-207/final`. Accessed: 18-06-2021.

[91] Talal H Noor, Quan Z Sheng, Lina Yao, Schahram Dustdar, and Anne HH Ngu. CloudArmor: Supporting reputation-based trust management for cloud services. *IEEE Transactions on Parallel and Distributed Systems*, 27(2):367–380, 2015.

[92] Oleksii Oleksenko, Bohdan Trach, Robert Krahn, Mark Silberstein, and Christof Fetzer. Varys: Protecting SGX enclaves from practical side-channel attacks. In *2018 USENIX Annual Technical Conference (USENIX ATC 18)*, pages 227–240. USENIX Association, 2018.

[93] Diego Ongaro and John Ousterhout. In search of an understandable consensus algorithm. In *2014 USENIX Annual Technical Conference (USENIX ATC 14)*, pages 305–319. USENIX Association, 2014.

[94] Michael P. Papazoglou, Paolo Traverso, Schahram Dustdar, and Frank Leymann. Service-oriented computing: State of the art and research challenges. *Computer*, 40(11):38–45, 2007.

[95] Jaemin Park, Sungjin Park, Brent Byunghoon Kang, and Kwangjo Kim. eMotion: An SGX extension for migrating enclaves. *Computers & Security*, 80:173 – 185, 2019.

[96] C. Priebe, K. Vaswani, and M. Costa. EnclaveDB: A secure database using SGX. In *2018 IEEE Symposium on Security and Privacy (SP)*, pages 264–278, 2018.

[97] Graeme Proudler, Liqun Chen, and Chris Dalton. *Trusted Platform Architecture*, pages 109–129. Springer International Publishing, 2014.

[98] Niels Provos and David Mazieres. A future-adaptable password scheme. In *USENIX Annual Technical Conference, FREENIX Track*, pages 81–91, 1999.

[99] Ioannis Psaras. Decentralised edge-computing and IoT through distributed trust. In *Proceedings of the 16th Annual International Conference on Mobile Systems, Applications, and Services*, pages 505–507, 2018.

[100] Have I Been Pwned. Pwned websites. `https://haveibeenpwned.com/pwnedwebsites`. Accessed: 18-06-2021.

[101] M. Rosenblum and T. Garfinkel. Virtual machine monitors: current technology and future trends. *Computer*, 38(5):39–47, 2005.

[102] Mark Russinovich, Edward Ashton, Christine Avanessians, Miguel Castro, Amaury Chamayou, Sylvan Clebsch, Manuel Costa, Cédric Fournet, Matthew Kerner, Sid Krishna, Julien Maffre, Thomas Moscibroda, Kartik Nayak, Olya Ohrimenko, Felix Schuster, Roy Schwartz, Alex Shamis, Olga Vrousgou, and Christoph M. Wintersteiger. CCF: A Framework for Building Confidential Verifiable Replicated Services. Technical Report MSR-TR-2019-16, Microsoft, April 2019.

[103] Mark Russinovich, Edward Ashton, Christine Avanessians, Miguel Castro, Amaury Chamayou, Sylvan Clebsch, Manuel Costa, Cédric Fournet, Matthew Kerner, Sid Krishna, Julien Maffre, Thomas Moscibroda, Kartik Nayak, Olya Ohrimenko, Felix Schuster, Roy Schwartz, Alex Shamis, Olga Vrousgou, and Christoph M. Wintersteiger. CCF: A framework for building confidential verifiable replicated services. Technical Report MSR-TR-2019-16, Microsoft, 2019.

[104] Mohamed Sabt, Mohammed Achemlal, and Abdelmadjid Bouabdallah. Trusted Execution Environment: what it is, and what it is not. In *2015 IEEE Trustcom/BigDataSE/ISPA*, volume 1, pages 57–64, 2015.

[105] Felix Schuster, Manuel Costa, Cédric Fournet, Christos Gkantsidis, Marcus Peinado, Gloria Mainar-Ruiz, and Mark Russinovich. VC3: Trustworthy data analytics in the cloud using SGX. In *Proceedings of the 2015 IEEE Symposium on Security and Privacy*, SP '15, pages 38–54. IEEE Computer Society, 2015.

[106] Mark Seaborn and Thomas Dullien. Exploiting the DRAM rowhammer bug to gain kernel privileges. *Black Hat*, 15:71, 2015.

[107] Vyas Sekar and Petros Maniatis. Verifiable resource accounting for cloud computing services. In *Proceedings of the 3rd ACM workshop on Cloud computing security workshop*, pages 21–26, 2011.

[108] Ming-Wei Shih, Sangho Lee, Taesoo Kim, and Marcus Peinado. T-SGX: Eradicating controlled-channel attacks against enclave programs. In *Network and Distributed System Security Symposium 2017 (NDSS'17)*. Internet Society, 2017.

[109] Signal. Secure value recovery. `https://signal.org/blog/secure-value-recovery`. Accessed: 18-06-2021.

[110] Lenin Singaravelu, Calton Pu, Hermann Härtig, and Christian Helmuth. Reducing TCB complexity for security-sensitive applications: Three case studies. In *Proceedings of the 1st ACM SIGOPS/EuroSys European Conference on Computer Systems 2006*, EuroSys '06, page 161–174. Association for Computing Machinery, 2006.

[111] C. Soriente, G. Karame, W. Li, and S. Fedorov. ReplicaTEE: Enabling seamless replication of SGX enclaves in the cloud. In *2019 IEEE European Symposium on Security and Privacy (EuroS P)*, pages 158–171, 2019.

[112] Sandeep Tamrakar. *Applications of Trusted Execution Environments (TEEs)*. Doctoral thesis, School of Science, 2017.

[113] The Linux Foundation. Xen Project, 2021. `https://xenproject.org/`. Accessed: 18-06-2021.

[114] Ken Thompson. Reflections on trusting trust. *Commun. ACM*, 27(8):761–763, August 1984.

[115] Dan Tsafrir. The context-switch overhead inflicted by hardware interrupts (and the enigma of do-nothing loops). In *Proceedings of the 2007 Workshop on Experimental Computer Science*, ExpCS '07. Association for Computing Machinery, 2007.

[116] J. Van Bulck, N. Weichbrodt, R. Kapitza, F. Piessens, and R. Strackx. Telling your secrets without page faults: Stealthy page table-based attacks on enclaved execution. In *USENIX Security Symposium*, 2017.

[117] Jo Van Bulck, David Oswald, Eduard Marin, Abdulla Aldoseri, Flavio D. Garcia, and Frank Piessens. A tale of two worlds: Assessing the vulnerability of enclave shielding runtimes. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, CCS '19, page 1741–1758. Association for Computing Machinery, 2019.

[118] Marten van Dijk, Jonathan Rhodes, Luis F. G. Sarmenta, and Srinivas Devadas. Offline untrusted storage with immediate detection of forking and replay attacks. In *Proceedings of the 2007 ACM Workshop on Scalable Trusted Computing*, STC '07, pages 41–48. ACM, 2007.

[119] Stephan van Schaik, Andrew Kwong, Daniel Genkin, and Yuval Yarom. SGAxe: How SGX fails in practice. `https://sgaxeattack.com/`, 2020.

[120] Luis M Vaquero, Luis Rodero-Merino, and Rajkumar Buyya. Dynamically scaling applications in the cloud. *ACM SIGCOMM Computer Communication Review*, 41(1):45–52, 2011.

[121] Vigilante.pw. The breached database directory. `https://vigilante.pw`. Accessed: 18-06-2021.

[122] William Voorsluys, James Broberg, Srikumar Venugopal, and Rajkumar Buyya. Cost of virtual machine live migration in clouds: A performance evaluation. In *Cloud Computing*, pages 254–265. Springer Berlin Heidelberg, 2009.

[123] Brent R Waters, Dirk Balfanz, Glenn Durfee, and Diana K Smetters. Building an encrypted and searchable audit log. In *Network and Distributed System Security Symposium (NDSS)*, volume 4, pages 5–6, 2004.

[124] Timothy Wood, Prashant J Shenoy, Arun Venkataramani, Mazin S Yousif, et al. Black-box and gray-box strategies for virtual machine migration. In *Symposium on Networked Systems Design and Implementation (NSDI)*, volume 7, pages 17–17, 2007.

[125] J. Wu, L. Ping, X. Ge, Y. Wang, and J. Fu. Cloud storage as the infrastructure of cloud computing. In *2010 International Conference on Intelligent Computing and Cognitive Informatics*, pages 380–383, 2010.

[126] Yubico. YubiKey Strong Two Factor Authentication, 2021. `https://www.yubico.com/`. Accessed: 18-06-2021.

[127] Fan Zhang, Ittay Eyal, Robert Escriva, Ari Juels, and Robbert Van Renesse. REM: Resource-efficient mining for blockchains. In *26th USENIX Security Symposium (USENIX Security 17)*, pages 1427–1444. USENIX Association, 2017.

Aalto-DD 171/2021

ISBN 978-952-64-0618-3 (printed)
ISBN 978-952-64-0619-0 (pdf)
ISSN 1799-4934 (printed)
ISSN 1799-4942 (pdf)

Aalto University
School of Science
Department of Computer Science
www.aalto.fi

BUSINESS + ECONOMY

ART + DESIGN + ARCHITECTURE

SCIENCE + TECHNOLOGY

CROSSOVER

DOCTORAL DISSERTATIONS